L. Conway
Archives

i. 1

ARCHIVE OF DOCUMENTS AND REFERENCE MATERIALS REGARDING THE IBM ACS-1 MACHINE

Lynn Conway*
February 16, 1999

This volume contains documents and reference materials that I have compiled regarding the IBM Advanced Computing Systems ACS-1 supercomputer. These are copies of original documents dating back to the ACS project itself. Taken together, they may be sufficient to disclose many of the system architectural innovations of the ACS architecture team.

The front-matter for the archive contains a brief, but important overview, of each document, including some details regarding the document's context within the ACS project. Also included is my initial letter to Dr. Mark Smotherman of Clemson University regarding the possibilities of reconstruction of many details of the ACS-1 machine.

CONTENTS: **PAGE:**

* My name was legally changed to Lynn Conway on January 30, 1969. Since I am widely known under my new name, we've chosen to use it on my earlier papers in this archive.

L. Conway
Archives

1. "Dynamic Instruction Scheduling", February 23, 1966:
L. Conway, B. Randell, D. Rozenberg, D. Senzig

The background on this paper is as follows. Sometime in late '65, I suddenly visualized a solution to the general multi-issuance and conflict-resolution problem. I quickly compiled block diagrams and notes to capture the ideas, and during the next few days I presented these ideas in staff meetings in the architecture group. There was a rapid, very positive reaction. I was tasked to document the ideas in more detail, to incorporate one of the branching schemes then under study, and to turn the scheme into an architectural "proposal".

Since I was quite junior and had little experience with coordinating and writing ACS proposals, I worked with a number of ACS staff members, including Don Rozenberg, Brian Randell, Don Senzig and others to produce the resulting paper. There was a sense that these weren't just ordinary ideas, and we worked hard to frame the concepts in a tutorial form, so that they would be clear to team members. Brian Randell in particular came up with some wonderful articulations about the DIS schemes, in his inimitable British manner. We hoped to be able to publish the ideas openly later on.

But things then moved fast, and within a year the ideas in the paper had became the basis for, and were implemented within, a fully revised ACS-MPM architecture.

Although the original dynamic instruction scheduling ideas were mine alone, the paper was a team effort. As inventor, I was the lead author, and was followed by Brian Randell, Don Rozenberg and Don Senzig. I think Ed Sussenguth and Herb Schorr gave useful feedback too; had the paper gone on to publication they might have been included as co-authors.

The dynamic instruction scheduling paper is labeled "[DRAFT]". I believe that by late February '66, we saw this paper as a work in progress towards formal publication. The ideas were already, in parallel, being evaluated for use in the actual machine. Thus in this draft I think we stepped back from revealing thinking on exactly how the ideas might be applied in the machine, as, for example, by using dual instruction windows.

But by then we also needed a tutorial on the ideas for those outside the architecture group, such as the logic designers, to use as a reference. Thus this "draft" version of 2-23-67 was released within ACS. After that date, no further work was done on the paper. It was completely overtaken by the escalating events surrounding adoption of this scheme for use in the ACS machine. Thus the invention itself then became quite "secret".

Interestingly, the name "dynamic instruction scheduling" never really entered into the team's "lingo". Instead, the relevant structures were usually just called "instruction queues", or "instruction buffers", or "contender stacks" for short, as is seen in all the later documents. It's possible that many ACS vets won't recall the specific title of the paper. Could that perhaps explain why no one from the team has ever come forward and mentioned this work?

On the other hand, it is very likely that copies of this paper surreptitiously passed into circulation outside IBM during the late 60's and early 70's, providing a path for transfer of this knowledge, and its name, into computer architecture circles outside of IBM.

L. Conway
Archives

2. "ACS Simulation Technique", Mar. 15, 1966: D. Rozenberg, L. Conway, R. Riekert

This paper documents the methods used to build the ACS MPM register-transfer level simulator. This paper may prove valuable by helping later analysts better understand and interpret the source code and the output results of the "MPM Timing Simulator".

The simulator was built in FORTRAN IV. Thus it is relatively easy to "read the code" that defines the workings of each module and functional unit. The simulation methods were also aimed at being fast enough to support long runs involving many, many variations of the machine architectural parameters.

The simulator was initially used to take quick looks at architectural variants, watch code passing through them, and figure out why things got blocked or didn't work as expected. Later it was used to gather data on the performance of many serious MPM variants running lots of real code, and then to "balance and tune" the emerging ACS-1 machine.

Notice the use of a "memory queue" function as the tutorial example in this paper. I believe that by this time in '66, we were already doing basic simulator implementations and evaluations of various "instruction queuing" structures and controls, as part of our explorations of dynamic instruction scheduling methods. I think we may have just simplified and then "reused" some of that code to create the example in this paper.

Don Rozenberg was lead author, I was second and Bob Riekert was third. Bob had done important work on the simulation methods at Yorktown, but didn't go west with ACS.

3. "Dual Arithmetic on ACS-1", May 1, 1967: T. C. Chen

This paper is an internal proposal from Tien Chi (T. C.) Chen to Jack Bertram regarding methods for implementing dual floating point arithmetic in ACS-1. It contains interesting references to dual arithmetic on the ILLIAC IV machine.

I include this paper as a good example of an ACS "proposal", though I do not recall right now the details of how this particular one turned out.

Note that the data-path register-transfer-level details of the arithmetic-functional units were an independent architectural dimension of the project that had to meet logic design/machine-cycle constraints on the one hand, and bussing/pipelining/issuance-control/architectural constraints on the other.

Thus only the timings of the ACS-1's arithmetic units, and not those units' internal functional details, were modeled in the timing simulator. (An "unroller" processed assembly code input instructions to produce the input instruction stream to the timing simulator). This was in contrast to the OP fetch, Bussing, OP interlocking and issuance, SKIP, Branch and Exit functioning, etc., which were fully modeled in the timing simulator.

L. Conway
Archives

4. "Architecturally Critical Paths in the MPM", May 12, 1967: E. Sussenguth

This is an important internal memo from Ed Sussenguth to Herb Schorr that summarizes the results of detailed MPM architectural design studies during the spring of 1967. It pins down the final list of critical paths that must be insured against any performance slippage in any later design iterations.

In each particular case, the critical path functions are identified as needing to be completed within a certain number of machine cycles. Then, for each of these functions, there would have been related critical logic design exposures, wherein specific logic functions had to be completable within a machine cycle .

This memo was the result of an intense period of simulation and tradeoff studies to tune and balance the MPM mechanisms for OP fetching, Bussing, OP interlocking and issuance, SKIP, Branch and EXIT mechanisms, functional unit timings, etc.

Together with the other documents, this paper shows that the near-final form of ACS-1 machine architecture was completed and was being fine-tuned during the spring of '67; thus it supports the inference that generalized dynamic instruction scheduling must have been incorporated into the revised ACS machine architecture sometime in the latter part of '66.

The details in this memo about MPM critical paths should really help during efforts at interpreting other ACS documents, and reconstructing the MPM's architecture.

5. "MPM Timing Simulation", August 25, 1967 (ACS AP #67-115) : L. Conway

This paper is a gold mine of detail on the system architecture of the ACS-1 MPM. It was originally intended as a users' manual that others could reference, in order to submit simulator input and interpret simulator output. I was sole author of this paper.

The simulator was written in FORTRAN IV (H), and ran on an IBM S/360 Mod 75 under OS/360. It operated at a rate of approximately 10 simulated instructions per second; typical programs thus ran at a rate of about 20 instructions per second.

By this date, the simulator was the de facto formal description of the structure and functions of the timing and controls of the ACS-1 MPM. All architecture team members coordinated their work with the making of modifications to the evolving versions of this simulator. Detailed functional modifications were seen to work or not, by whether they functioned as expected during simulation runs.

By the time this document was written, a lot of experience had been gained in the effects on machine performance of variations in machine parameters. In particular, it was clear by then that the 3 out of 8 issuance scheme for A-Ops was near optimal in terms of mean OPs/cycle while meeting the logic-level and machine cycle-time constraints. This paper uses that 3 out of 8 scheme in a very detailed example, including detailed timing diagrams and the corresponding simulator input and output listings.

L. Conway
Archives

Therefore, this paper provides a peek inside an ACS-1 MPM actually running code, enabling the reader to see how the OP fetching, Bussing, instruction scheduling, Branch and Exit functions, functional unit timings, etc., all worked together.

The paper defines and elaborates on the mnemonics of all those machine facilities, enabling readers to make detailed interpretations of timing diagrams and simulator output listings. Those mnemonics were used widely within ACS by this date, so these definitions will be helpful in interpreting other ACS documents. This paper includes a list of all instruction mnemonics, but, unfortunately, no detailed descriptions of the instructions themselves.

This manual, together with the detailed "Timing Simulator Notebook" and the "Timing Simulator Source Code Listing", provides sufficient information to possibly enable later analysts to reconstruct a running version of the ACS timing simulator.

This document, with all its details of how the ACS-1 processed instructions, may also have passed into circulation outside of IBM, and thus helped to propagate ACS architectural concepts into the computer architecture community.

6. MPM Architecture and Simulator Notebook, August 1967: L. Conway

This notebook contains my working documentation of the ACS-1 machine architecture, and materials regarding translation of that architecture into the MPM Timing Simulator. It contains very detailed information on the ACS-1 as of late August 1967, which was a mature point in the machine's evolution, and the design point for which important benchmarks have been described elsewhere. The notebook consists of about 120 pages of flowcharts, tables and notes, in addition to the ACS AP #67-115 paper.

Unfortunately, these notes do not contain a description of the OP set itself, as it was documented in a separate memo that, I believe, was entitled "ACS-1 MPM Instruction Manual" (we should really try to find a copy of that one, if one still exists). However, many important details regarding the OP set, including the OP Tags, are included in these notes. A listing of the contents of this notebook is included on the following page.

L. Conway Archives

Listing of contents of the Timing Simulator Notebook (draft listing, as of 1-21-99) :

L. Conway
Archives

7. Timing Simulator Source Code Listings, August 1967: L. Conway

This notebook contains a set of listings of the source code for the near-final version of the ACS machine's register-transfer level timing simulator. There are about 5000 lines of FORTRAN IV (H) source code in these 100 or so pages of listings. This is probably the version of the code used to generate the examples in the ACS AP #67-115 paper.

By mid-67, the timing simulator was the de facto formal description of the overall team-coordinated details of the evolving ACS-1 architecture. Therefore, these listings, when taken together with the Timing Simulator Manual and the additional diagrams, flowcharts and other details in the Timing Simulator Notebook, provide a very detailed account of the ACS-1 system architecture.

8. "ACS Logic Design Conventions: A Guide for the Novice", Nov. 29, 67: L. Conway

On joining ACS, I found that there was no single convenient source for this information. Some of the information was not documented in any available references. Since most of the logic designers used different notations and conventions, it proved to be a time consuming and confusing process to learn the precise details of this very simple, basic material. Many of the designers related to me that they had had similar initial experiences.

At the time I made some notes for my own personal use, and later formed these notes into this memorandum in the hope that it might prove useful to newcomers to ACS. This memo may prove useful in ACS retrospectives and reconstructions by enabling more precise analysis of original ACS DRKS design records.

9. "A Proposed ACS Logic Simulation System", Oct. 31, 1967: L. Conway

This memo proposes an LSS to provide a means for debugging the logic design of the ACS machine. Included is a means to extract design partitions from DRKS files and run simulations on the partitions based on interface signals extracted from the equivalent partition of the system-level (MPM timing) simulator. Considerable detail in the form of block diagrams, flow-charts and calculations are included to clarify interfaces and interaction in the overall system. One requirement for such a system to work would be formal acceptance of the system-level simulator as the formal description of machine structure and functions, and forcing of logic design partitions to implement the functions of the equivalent system-level partitions. This seemed feasible at the time, since the MPM Timing Simulator had already become the de-facto formal description of the machine. This memo may provide useful insights into various practical aspects of ACS logic design and engineering at the time.

L. Conway
Archives

10. "The Computer Design Process: A Proposed Plan for ACS", Aug 6, 68: L. Conway

This memo builds on item 9, and proposes a detailed design for the overall ACS machine design process, including system architecture, logic design and engineering, physical specification and process automation, and maintenance. The thesis is that proper design of the design process is as important as proper design of the machine itself. It exploits the System-level Simulator as the overall machine specification, and discusses the overall integration and protocols for use of that simulator with the LSS, DRKS, Physical Specification and Process Automation tools. It addresses many concerns, such as the fact that design phases do not follow serially but overlap in time, that some partitions may be far along in specification while others may be quite tentative, and that later design phases constantly feedback feasibility or cost issues to earlier (higher-level) phases. This proposal was fairly widely circulated and had gained considerable support just before the project was cancelled. This memo provides useful insights into practical aspects of ACS system architecture, logic design, engineering, physical specification and process automation at the time. [Also, taken together with the other materials, all this work substantially informed my later explorations at Xerox PARC on VLSI design and implementation methodologies].

LYNN CONWAY
PROFESSOR OF ELECTRICAL ENGINEERING
AND COMPUTER SCIENCE
THE UNIVERSITY OF MICHIGAN
COLLEGE OF ENGINEERING

170 ATL BUILDING
ANN ARBOR, MICHIGAN 48109-2110
313 763-5509 FAX 313 763-1260
INTERNET: conway@engin.umich.edu

L. Conway
Archives

2 January 1999

Dr. Mark Smotherman
Department of Computer Science
Box 341906
Clemson University
Clemson, SC 29634-1906

Dear Dr. Smotherman:

When I came upon your web site identifying the IBM-ACS machine as "the First Superscalar" computer, many past events came rushing back into my mind. I had been at ACS, first at Yorktown Heights, then in Sunnyvale and then up on Sand Hill Road, during the period when the exciting architectural work was being done there.

There were publications and talks, by Herb Schorr in the early 70's and later by John Cocke and others, that hinted at the scope of the ACS innovations. But these early retrospectives lacked detail about the system's architecture and lacked a context in which to embed the ideas so as to fully convey their significance. Many computer architects sensed that amazing things had happened at ACS, but few could be sure quite what, or why it even mattered.

As modern VLSI superscalars emerged into widespread application, and details of their architectures were described, I became aware that important early ACS innovations had transferred directly into those machines. Even the early ACS name for one of those innovations, dynamic instruction scheduling, is now used by superscalar architects, and is described as such in modern computer architecture textbooks.

More than thirty years after the original work, modern superscalars now at last provide a context for understanding and appreciating the value of the early ACS innovations. For some time now, I've hoped that someone from the ACS team might step forward and point towards the sources of those concepts. However, no one has come forward.

When I read the ACS retrospective on your web site, I began thinking about why such claims haven't been made before. The sudden elimination of the project, followed by exits and transfers of the architecture team members, must have meant that few, if any, original ACS documents were saved by anyone. Thus the machine seemed to have just "vanished", and there was little material evidence on which to base any retrospectives.

It vanished almost everywhere, that is, except in a notebook, documents and computer listings that I compiled and kept stored away all these years.

L. Conway
Archives

Hopefully, the materials that I have saved can be used to reconstruct many details of ACS machine architecture, and more fully document the accomplishments of the ACS team. I'm interested in helping with such an effort, and in helping contact other ACS alums who might have original artifacts and personal knowledge of events there.

The years I spent at IBM-ACS were among the most intellectually exciting of my life. It was an incredible opportunity for me to be able to work with John Cocke, Herb Schorr, Fran Allen, Ed Sussenguth, Don Rozenberg and all the others upon just finishing my graduate work at Columbia. Reflections on my experiences at ACS, and the documents relating to my work there, may help you and others reconstruct the overall story.

When I joined ACS, the team was based at IBM Research in Yorktown Heights N.Y., and the effort went by the code name "Project Y". I joined in a support role to build the register-transfer timing simulator for the emerging supercomputer. In that role, I had ongoing access to almost all the team's architectural discussions and debates.

During the early phases of the project, I became fascinated with John Cocke's "open questions" about computer architecture. By an amazing stroke of luck, I hit upon a pretty good general solution to one of those questions, namely the problem of multiple issuance. The team was very democratic and open to suggestions and proposals from any member, at any level. They listened to my ideas, and then acted on them.

We initially called the resulting invention "dynamic instruction scheduling". It went on to play an important role in the overall system architecture of the ACS main processing module (MPM). Fortunately, among my documents are those describing this invention, and showing how it was exploited in the ACS-MPM. These documents are identified in an annotated list attached to this letter.

Included in the attached list are my reference notebook, the source code and a detailed user's manual for the MPM timing simulator. During 1967, the timing simulator became the de facto formal description of much of the machine's architecture. Therefore, these materials can be used to reconstruct many details of ACS machine architecture. It's even conceivable that a running timing simulator could be reconstructed someday, based on these materials.

Given the significance and impact of superscalar computers, I really do feel the need to set the story straight, namely that the ACS machine, a long forgotten "orphan", was never really dead. ACS lives on after all, as the original source of many fundamental innovations that have since passed on into modern machines.

I commend you on your efforts to reconstruct events at ACS and to document details of ACS machine architecture. The independent, detailed context that you have already established, together with my materials, should at least confirm the origins of generalized dynamic instruction scheduling. That invention is one of the coolest ideas I've hit upon. It would mean a very great deal to me for its origins in my ACS work to be acknowledged.

L. Conway
Archives

I'm not sure how to best proceed from here, but I do suggest that initially we try to acquire more materials, contact more ACS alums, work on a project timeline, etc., before releasing further preliminary conclusions. Also, by putting more ACS materials on a web site, we could perhaps clarify that a lot of materials do still exist, and thereby interest others in participating in reconstruction efforts.

Many of the events surrounding ACS were shaped by internal IBM politics that I and most of my colleagues were unaware of at the time. The sudden demise of the project completely stunned us. I never understood why the decision had been made that ACS must be 360 compatible. However, it was clear right away that the 360 decision meant that the ACS architectural innovations were going to be shelved.

You can imagine what the project's demise meant to those who had done the creative work there. Sure, John Cocke went on to become famous among the cognoscenti in computing. Indeed, four members of the early ACS architecture team, including John Cocke, Fran Allen, Ed Sussenguth and myself, were later elected to the National Academy of Engineering for a variety of other contributions. But imagine how much it would have meant to John and the rest of us if the ACS designs at least had been saved, and approved for later publication. Instead, almost all that wonderful work was discarded, as if it had never existed.

Since I'm not sure what sensitivities remain regarding theories about the project's cancellation, I'd like to proceed carefully when gathering information on the overall story. It is certainly important to try to contact ACS team members named in the various documents in advance of any public uses of those documents. Efforts should also be made to involve as many ACS alums as possible, so that a wider set of perspectives can be gained and a more thorough history compiled.

I really enjoyed talking with you recently about ACS. I look forward to interacting with you further on this interesting project.

Sincerely,

Lynn Conway

Lynn Conway

Professor of EECS, Emerita
University of Michigan, Ann Arbor, MI

Attachment: Annotated list of reference materials regarding the ACS-1 machine

IBM CONFIDENTIAL

ADVANCED COMPUTING SYSTEMS
SAN JOSE
February 23, 1966

MEMORANDUM TO: File

SUBJECT: DYNAMIC INSTRUCTION SCHEDULING (DRAFT)

L. Conway
B. Randell
D. P. Rozenberg
D. N. Senzig

001

L. Conway Archives

DYNAMIC INSTRUCTION SCHEDULING

INTRODUCTION

The order in which the instructions comprising a program are to be excecuted is normally assumed to be given by the order in which the instructions are held in program storage and by the sequencing control indicated by transfer and conditional transfer instructions. However a programmer, or compiler, can produce many different but equivalent versions of a program merely by making minor alterations to the sequence in which instructions are placed. Normally the actual choice among these alternative sequences will be somewhat arbitrary, though careful programming or compilation often involves an attempt to design a program whose detailed sequences are tailored to make best use of a computer's control and functional capabilities. This can be particularly worthwhile for computers whose internal organization has been designed to attempt to overlap the use of its various functional capabilities.

Take, for example, a computer which initiates execution of instructions in strict sequence, without necessarily awaiting the completion of one instruction before execution of the next instruction, provided that the operands of the second instruction are ready, and the necessary busses and functional units are available. On such a computer the sequence (written here for convenience in a 3-address format)

$$R_1 + R_2 \rightarrow R_3$$

$$R_1 \times R_4 \rightarrow R_5$$

$$R_6 + R_2 \rightarrow R_7$$

$$R_3 \times R_6 \rightarrow R_8$$

might well be preferable to

$$R_1 + R_2 \rightarrow R_3$$

$$R_6 + R_2 \rightarrow R_7$$

$$R_1 \times R_4 \rightarrow R_5$$

$$R_3 \times R_6 \rightarrow R_8$$

if the adder and multiplier were independent functional units.

L. Conway Archives

Thus if really effective use is to be made of the internal capabilities of such a computer, careful attention must be paid to the detailed sequencing of instructions in frequently executed portions of a program. This 'scheduling' can be done by an ambitious optimizing compiler, or an extremely conscientious hand-coder. There is often, however, a difficulty in achieving really optimum sequencing by such means--that of the effects of memory interference, which if present will cause variations in the times which operands take to reach the arithmetic and control unit from storage. The effects of such memory interference will not usually be calculable in advance of program execution, particularly if the interference is caused by autonomous I/O units using the memory. Thus there is often cause to consider the possibility of supplementing (or even replacing) the static scheduling performed by coder or compiler by dynamic scheduling performed by the computer as it executes a program. In this paper we describe a technique of dynamic scheduling permitting non-sequential instruction execution. Furthermore, the technique presented is shown to be capable of controlling the simultaneous execution of two or more instructions at a time on machines with sufficiently generous bussing and functional capabilities. In any actual computer design care would of course have to be taken to ensure that any possible gains achieved by such dynamic scheduling were not offset by the cost (both in speed and in circuits) of the extra hardware necessary to perform the scheduling.

The scheme presented uses a very general, but conceptually simple, method of controlling non-sequential instruction execution, and of identifying groups of instructions which are mutually independent and can be executed simultaneously. Brief descriptions of earlier schemes for achieving some of these aims have been given by Amdahl [1], Chen [2], and Thornton [3].

003

L. Conway Archives

NON-SEQUENTIAL INSTRUCTION EXECUTION

In this section we restrict our attention to the sequencing of straight line coding comprised of instructions, the locations of whose operands and results can be determined directly from the instructions themselves, rather than needing any address computation to be performed.

The sequence in which a series of instructions have been written implies the total effect that these instructions are intended to have when executed. Each separate instruction contributes to this total effect by performing its operations on the contents of certain registers (accumulators, index registers, indicators, etc.) and setting its results into other registers. A dynamic scheduling technique has to insure that any instructions obeyed out of sequence do not change the contents of any registers which are to be used by any instructions whose execution has been delayed temporarily.

A simple set of rules for determining if a given instruction can be obeyed out of sequence is as follows:

(i) The required busses and functional units are available.

(ii) The instruction must not use any registers which are used as result registers by instructions whose execution has been initiated but not yet completed.

(iii) The instruction must not use as result registers any registers which are used as operand registers by any preceding instructions which have not yet been initiated.

(iv) The instruction must not use any registers (either as result or operand registers) which are used as result registers by any preceding instructions which have not yet been initiated.

These checks can be made in a systematic fashion using what are here called 'sequencing matrices'. Two matrices are used, namely a 'source matrix' (S) and a 'destination matrix' (D). At each cycle, when the machine is attempting to choose an instruction to be executed, rows in these matrices are set up corresponding to each of the instructions which are being considered by the scheduling machanism. (The cycle referred to above is a clock cycle, which corresponds to the maximum rate at which instructions can be initiated, and will presumably be much shorter than a storage cycle.) The elements in each row of the matrices indicate whether a given register is being used, or will be affected, by the corresponding instruction.

L. Conway Archives

The element $S_{i,j}$ is set to one if the i^{th} instruction uses the contents of register j as an operand. The element $D_{i,j}$ is set to one if execution of the i^{th} instruction will cause the contents of register j to be replaced.

Take, for example, a very simple machine with eight registers and a 3-address format, using a scheduling mechanism that processes four instructions per cycle. A typical situation would be:

Instruction	Source Matrix								Destination Matrix							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
1. $R_3 + R_4 \rightarrow R_7$			1	1											1	
2. $R_7 \times R_2 \rightarrow R_4$		1					1					1				
3. $R_1 + R_2 \rightarrow R_5$	1	1											1			
4. $R_8 \div R_1 \rightarrow R_8$	1							1								1

Fig. 1

Thus each row has been set up by processing the register address fields of the corresponding instructions, and converting these addresses into unary form. However in more realistic machines the setting up of the matrix elements would not be so straightforward. Almost certainly it would involve decoding the operation code part of the instruction to determine what implied registers are used by an instruction in addition to those indicated by address fields.

In addition to the matrices, which provide a conveniently coded form of indicating the register requirements of instructions awaiting execution, a 'busy vector' (B) is used to indicate the current status of the machine registers. The length of the vector is equal to the number of registers. The element B_j is set to one when execution of an instruction which will cause the contents of register j to be replaced is initiated; it is reset to zero when the replacement has been completed.

Once the sequencing matrices and the busy vector have been set up as described, the basic algorithm for choosing an instruction to be executed can be described as follows. Starting with the top row of the matrices, each instruction is checked--instruction i can be executed if:

(i) The required busses and functional units are available.

(ii) The elements of B corresponding to the non-zero elements of the i^{th} rows of S and D are zero.

005

L. Conway Archives

(iii) The elements above row i of the columns of D corresponding to the non-zero elements of row i of S contain only zeroes.

(iv) The elements above row i of the columns of S and D corresponding to non-zero elements of row i of D contain only zeroes.

Returning to the previous example, with the busy vector set up to indicate that certain registers, 3 and 6 for instance, are still to have their contents replaced, by the action of previously initiated instructions

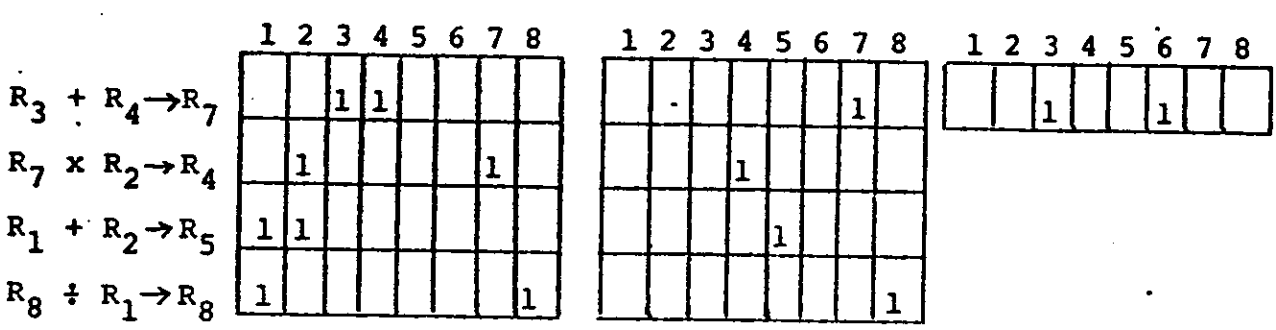


Fig. 2

Instruction 1 cannot be executed because of rule (ii)

Instruction 2 cannot be executed because of rules (iii) and (iv)

However instruction 3 can be executed, provided that the necessary bussing and functional capabilities are available.

Each cycle, while the scheduling mechanism is attempting to choose an instruction to initiate, a decoding mechanism could be processing a further instruction, taken from the address in the instruction store given by an instruction counter. In contrast to a conventional instruction counter, this counter does not indicate which instruction is currently being executed, but rather which instruction is next in line for processing by the scheduling mechanism. With non-sequential instruction sequencing it is not possible to have a conventional instruction counter. This can in certain circumstances be a disadvantage of the system, and is discussed further below.

At the end of a cycle, if an instruction has been chosen (it is of course possible that none of the instructions can be initiated until some of the non-zero elements of the busy vector become zero), the rows corresponding to the instruction are removed from the matrices. The remaining rows are then pushed upwards.

006

L. Conway Archives

to fill in any gap, the bottom row of the matrix is replenished using the instruction which has just been decoded, and the instruction counter is incremented. All is then ready for the scheduling mechanism to again scan the matrices in an attempt to choose another instruction to initiate.

In the above example, the situation at the start of the next cycle might be (assuming that registers 3 and 6 have still not had their contents replaced) as shown in Fig. 3. During this cycle the Divide instruction will be chosen for execution.

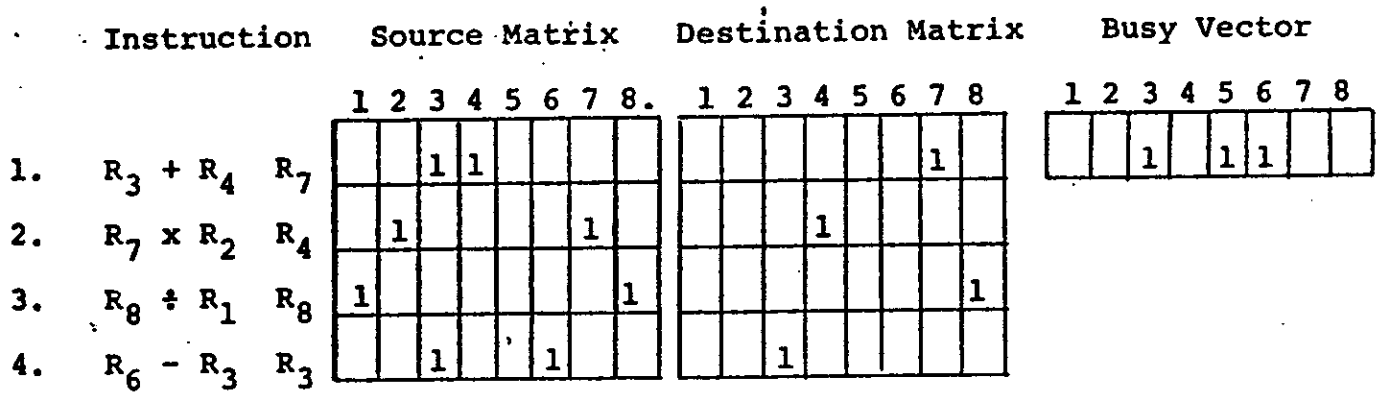


Fig. 3

In the above general description of the proposed technique for non-sequential instruction execution the discussion has been limited to the scheduling of straight-line coding composed of instructions whose register requirements can be determined immediately from inspection of the instructions. The next two sections of this paper deal with the effect of unconditional and conditional branch instructions, and with a technique for scheduling instructions which refer to indexed addresses in storage.

007

L. Conway Archives

UNCONDITIONAL AND CONDITIONAL BRANCHING

There is one kind of branch instruction, namely the unconditional branch to an explicit instruction address, which can be handled very simply, without recourse to the sequencing matrices. The instruction is executed as soon as it has been decoded, causing the appropriate modification to the instruction counter which indicates the location from which the sequencing matrices are to replenished.

The other types of branch instructions, where the branch address and/or the question of whether the branch is to be taken cannot be determined directly from the instruction, but rather depend on the contents of one or more registers, cause rows to be entered into the sequencing matrices in the usual way. However refilling of the matrices then stops until the branch instruction has been executed and any necessary modification has been made to the instruction counter. Thus once such a branch instruction has entered into the matrices, the matrices will gradually empty until the execution of the branch instruction permits refilling to begin. This means that every effort should be made to initiate execution of the branch instruction as soon as possible, and that once the branch instruction has been executed, empty rows of the matrix should be replenished as quickly as possible. Otherwise, the matrices will spend much of their time only partly full, and the chances of finding an executable instruction each cycle will be considerably reduced.

Since a scan of the matrices enables all the executable instructions to be identified, what is required is to ensure that a branch instruction is given priority over any other executable instructions. The simplest way of doing this, since there can never be any instructions in the matrices below a branch instruction, is to always choose the lowest executable instruction, whether or not this is a branch instruction. However it could be argued that this is taking unnecessary liberties with the sequencing of a program, which will cause undue complications in program debugging. The alternative is to arrange some system whereby if there is an executable branch instruction it is initiated, but that otherwise the highest executable instruction is chosen.

The second requirement, that of speedy replenishment of the matrices once a branch instruction has been executed, required decoding facilities operating in parallel on several instructions. The alternative of relying solely on the normal decoding and replenishment mechansim, which fills only one row each cycle, is unlikely to be adequate.

An 'Execute' instruction, which can be regarded as a temporary branch for the duration of a single instruction, involves only slight extensions to the above system. Filling of the matrices is halted once an Execute instruction has been reached, until it can be obeyed and the instruction which it specifies can be fetched

L. Conway Archives

and placed in the matrices. Unless this is another Execute instruction, or a branch instruction, filling of the matrices can then be resumed, starting with the instruction following the original Execute instruction.

009

L. Conway Archives

THE SEQUENCING OF STORAGE ACCESSES

Another area where dynamic scheduling can be of value is the sequencing of accesses interleaved storage. Such storage is characterised by the fact that access to one of the autonomous memory units, or of which the storage is comprised does not have to await the completion of previous accesses to other boxes. Rather, storage accesses can be made at the rate at which they can be accepted by the bussing system, provided that repeated accesses to the same box are sufficiently separated. Thus the problem of sequencing storage accesses can be regarded as having similarities to that of sequencing instructions, with boxes taking the place of registers, and 'bus slots' the place of clock cycles.

The particular box involved in a storage access is determined from the effective address of the location to which access is being made (typically a group of the least significant digits of the address is used). Such an address will normally be the result of a calculation involving the contents of one or more registers. Thus the box used by a storage access requested by a register load or store instruction cannot be determining directly by examination of the instruction, it being necessary to wait until the effective address can be calculated.

Though one can conceive of a single scheduler being used for sequencing both instructions and storage accesses, it seems more reasonable to have a second scheduler just for sequencing storage accesses, operating in conjunction with the instruction scheduler. The storage access scheduler could operate according to the same general principles as the instruction scheduler, using source and destination matrices (S_A and D_A, say), and a busy vector (B_A), whose respective columns and elements correspond to the various boxes. It would receive requests for storage accesses both from the instruction scheduler, on behalf of load and store instructions, and from the instruction fetch mechanism which is used to replenish the instruction scheduler.

The instruction scheduler described above is designed on the assumption that once an instruction is removed from the matrices and issued, it no longer has any demands on the registers that it uses for its operands. Therefore, a set of buffer registers are included in the storage access scheduling mechanism to hold the contents of registers which are to be stored, until the required storage access can be initiated.

Certain constraints must be placed on the order in which storage access requests can be issued to the storage access scheduler from the instruction scheduler. For example, a store request must not be issued to the storage access scheduler before any preceding load request. Only when the boxes involved in these requests have been determined will it be possible for the storage access scheduler to

L. Conway Archives

perhaps make such modifications to the sequencing of storage access requests. In fact what is necessary is for the instruction scheduler to treat the store as a single extra register. Therefore an additional column is added to the S and D matrices, and an element is added to the busy vector. However this extra busy vector element is not set to one unless the storage access scheduler is unable to accept any further storage access requests. All load instructions have the extra element in their row of the S matrix set to one; all store instructions have the extra element in their row of the D matrix set to one. The normal sequencing rules will then apply the necessary constraints to the issuing of access requests.

Figure 4 demonstrates the setting of the matrices and busy vectors of the two schedulers on a machine with 4 registers and 4 storage boxes. The instruction scheduler processes six instructions per cycle; the storage access scheduler processes four access requests per bus slot. Instructions are either 3-address format, or specify single-indexed loads and stores. The vector B indicates that registers R_1 and R_3 are still involved with previously initiated instructions, and that the storage access scheduler has capacity for further storage access requests. The storage access scheduler contains only three access requests--a load of register R_3 from address 53 in box 1, and a store of the literal 91 (the contents of some register) in address 29 of box 2, and a load of register R_1 from address 25 of box 3. The vector B_A indicates that box 1 is still involved in some earlier access request.

When the instruction scheduler initiates execution of a load or store instruction the rows corresponding to the instruction are removed from the S and D matrices, and the B vector (except for the last element, corresponding to the store) is updated in the usual way. The effective address is calculated, and it and the address of the register to be loaded or stored are transmitted to the storage access scheduler (together with the contents of the register, in the case of a store instruction). This storage access request causes the highest unoccupied row of the matrices S_A and D_A to be set up so as to indicate the box requirements of the request.

L. Conway Archives

INSTRUCTION SCHEDULER

1. $R_1 + R_2 \rightarrow R_3$
2. $S[R_1 + 2] \rightarrow R_3$
3. $S[R_2 - 1] \rightarrow R_4$
4. $R_2 \rightarrow S[R_1 + 1]$
5. $R_1 \times R_3 \rightarrow R_1$
6. $R_4 - R_1 \rightarrow R_2$

S

	1	2	3	4	S
1	1	1			
2		1			1
3		1			1
4	1	1			
5	1		1		
6	1			1	

D

	1	2	3	4	S
1			1		
2			1		
3				1	
4					1
5	1				
6		1			

B

	1	2	3	4	S
1	1		1		

STORAGE ACCESS SCHEDULER

1. 1:53 R_3
2. '91' 2:29
3. 3:25 R_1
4. - -

S_A

	1	2	3	4
1	1			
2				
3			1	
4				

D_A

	1	2	3	4
1				
2		1		
3				
4				

B_A

	1	2	3	4
1	1			

Fig. 4 Example of a 4 Register, 4 Storage Box Machine

012

L. Conway Archives

The matrices S_A and D_A are scanned each bus slot time, in order to choose an access request which can be issued ahead of any preceding requests which are held up, and which does not involve a box indicated by the vector B_A as being still involved with a previous access. The corresponding to this request are removed from the matrices, the rows are pushed up to fill in the gap, and the busy vector updated. When a storage access to a box has been completed the corresponding element of B_A is made zero once again. If this access was on behalf of a load instruction, the appropriate element of B is made zero when loading of the register has been completed.

Returning to the example demonstrated in Fig. 4, the situation after one machine cycle and bus slot time is shown in Fig. 5. The third instruction, a load instruction, has been chosen for execution, the effective address specified by it has been calculated to be location 57 of box 4, and it has been issued as an access request to the storage access scheduler. Meanwhile the second storage access request has been issued, the preceding request being still blocked because the required box is still involved in an earlier access.

L. Conway Archives

INSTRUCTION SCHEDULER

1. $R_1 + R_2 \quad R_3$
2. $S[R_1 + 2] \quad R_3$
3. $R_2 \quad S[R_1 + 1]$
4. $R_1 \times R_3 \quad R_1$
5. $R_4 - R_1 \quad R_2$
6. $R_2 \quad S[9]$

S

1	2	3	4	S
1	1			
	1			1
1	1			
1		1		
1				1
	1			

D

1	2	3	4	S
		1		
		1		
				1
1				
	1			
				1

B

1	2	3	4	S
1		1	1	

STORAGE ACCESS SCHEDULER

1. 1:53 R_3
2. 3:25 R_1
3. 4:57 R_4
4. - -

S_A

1	2	3	4
1			
		1	
			1

D_A

1	2	3	4

B_A

1	2	3	4
1	1		

Fig. 5. The Example of Fig. 4 One Cycle and One Bus Slot Later

L. Conway Archives

There are many possible variations on this scheme for sequencing storage accesses. For instance, one can dispense with extra buffer registers and continue to hold quantities in the working registers until the appropriate memory unit can be accessed. What is required to avoid unessential slowing down of the instruction scheduler is that the registers used in the calculation of the effective address be released before the instruction is necessarily removed from the matrix. This introduces a new complexity. Previously an instruction was not modified in the matrices, except for its possible bubbling towards the top, until its complete removal from the matrices.

The bits in the source matrix corresponding to those components of the effective address calculation would beset to zero as soon as they are used. This at least releases those registers for use in further calculations. One might further refine interlocking on register usage so that effective address calculations were performed before the contents of the register to be loaded or stored were available.

Indirect addressing can be handled in much the same way as branch and execute instructions. If the various levels of indirect addressing use new indexing registers at each step then no instruction can be permitted to be executed which may result in any register modification. Unless memory read buffers are present this effectively means that indirect addressing will stop instruction initiation though matrix replenishment can proceed. If indirect addressing does not require new indexing registers but simply generates new memory store access requests then only succeeding store instructions must be inhibited until the indirect addressing chain is terminated.

L. Conway Archives

SIMULTANEOUS EXECUTION OF INSTRUCTIONS

The instruction scheduling method described above uses the sequencing matrices in order to detect which instructions can be obeyed out of sequence. As a byproduct it automatically detects which instructions can be initiated simultaneously, at least in so far as register usage is concerned. Thus, given sufficient functional capabilities and sufficient busses between registers and functional units, the scheduling scheme can be used to control the simultaneous initiation of instruction execution. The matrix scanning algorithm would remain unchanged, though from a hardware point of view if not conceptually the procedure for compressing the remaining rows in the matrices upwards to fill in any gaps becomes more complex.

We assume that the machine has a number of independent functional units in addition to the memory and branch control units. Typical additional independent specialized functional units are floating point add/subtract, multiply, and divide units. We make the further assumptions that each functional unit has a buss connecting with the registers and that there is only one functional unit of each type. The complexities that arise when these assumptions are removed will be discussed below.

The requirements for simultaneous initiation of instruction execution is the addition of a bit to the busy vector for each functional unit that cannot accept operands every cycle and a column appended to the destination matrix for every functional unit.

The busy vector bit corresponding to the functional unit is turned on by the initiation of execution of an instruction in the c corresponding funtional unit. The busy vector bit is turned off when the functional unit is able to accept a new operand pair.

Rule (i) of the sequencing algorithm given informally above can here be stated as: the elements of B representing the functional units must have zeros corresponding to non-zero elements in the i^{th} row of D. The elements above row i of the columns of D corresponding to the non-zero elements of D contain only zeros.

The operation code portion of the instruction is decoded to the extent that it is known which functional unit is going to execute the instruction. This information sets a one in the bit position whose row index corresponds to the instruction and whose column index corresponds to the functional unit.

Going back to the example used in Fig. 2 and assumming that the functional units are an add/subtractor that can accept a new pair of operands every cycle, a multiplier and a divider that cannot accept a new pair of operands every cycle, and a branch controller, we have the situation shown in Fig. 5.

L. Conway Archives

As in Fig. 2, Instruction 1 cannot be executed because of rule (ii). Instruction 2 cannot be executed because of rules (iii) and (iv). In addition Instruction 2 cannot be executed because of rule (i), i.e., because the multiplier is busy. The execution of Instructions 3 and 4 can be initiated--they violate none of the rules on register usage and the appropriate functional units are free.

As is done in the sequential case, at the end of the cycle, instructions that have been chosen for execution are removed from the matrix. The remaining rows are pushed up to fill in the gaps, and new instructions are inserted at the bottom of the matrix to replace those which have been initiated, and the instruction counter is incremented.

In the above example (Fig. 5) the situation during the next cycle might be as shown in Fig. 6. The instructions 1 and 2 are inhibited by the same reasons as before. Since the Busy vector bit corresponding to the Branch unit is zero (indicating no Branch instructions in the matrix) new instructions can be entered. The new instruction 3 ($R_6 - R_3 \rightarrow R_3$) is inhibited by rules (ii) and (iv).

The new fourth instruction specifies a branch to the memory locations specified by the contents of register R_1 plus 71 if register R_2 contains a zero. Since all of the registers used by this instruction are free this instruction can be initiated. Since we still can have but one branch instruction in the matrix at a time no Branch column on the Destination matrix is needed though the equivalent may be needed by the replenishing mechanism. The Branch bit on the Busy Vector is needed to inhibit the matrix replenishing hardware.

In the case of the sequential control the point was made that preference should be given to branch instructions. Here, because one can say that each functional unit is looking for work, no special priority need be given to a branch instruction.

L. Conway Archives

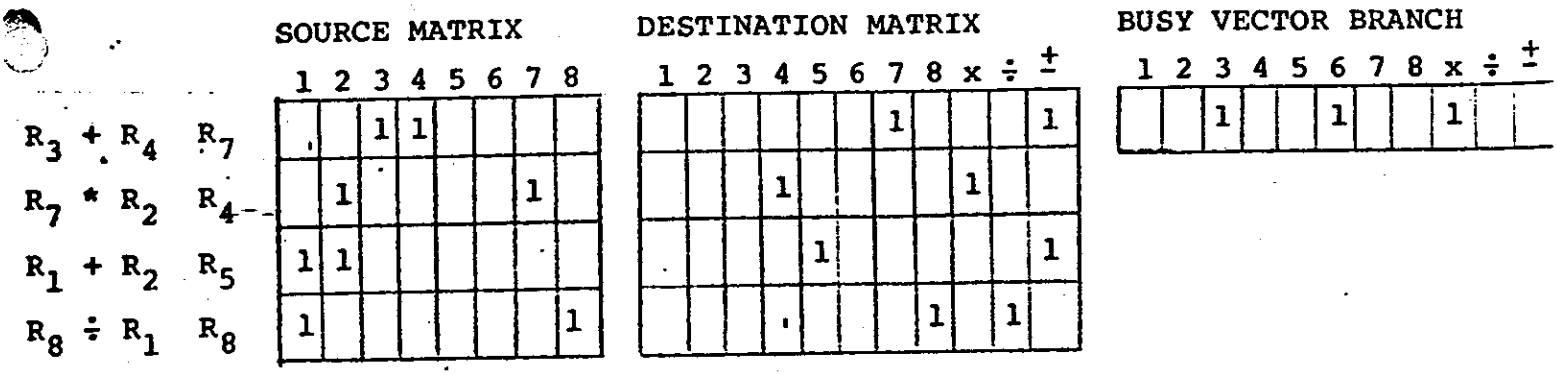


Fig. 5. Example of Multiple Instructions per Cycle Initiation--Cycle 1..

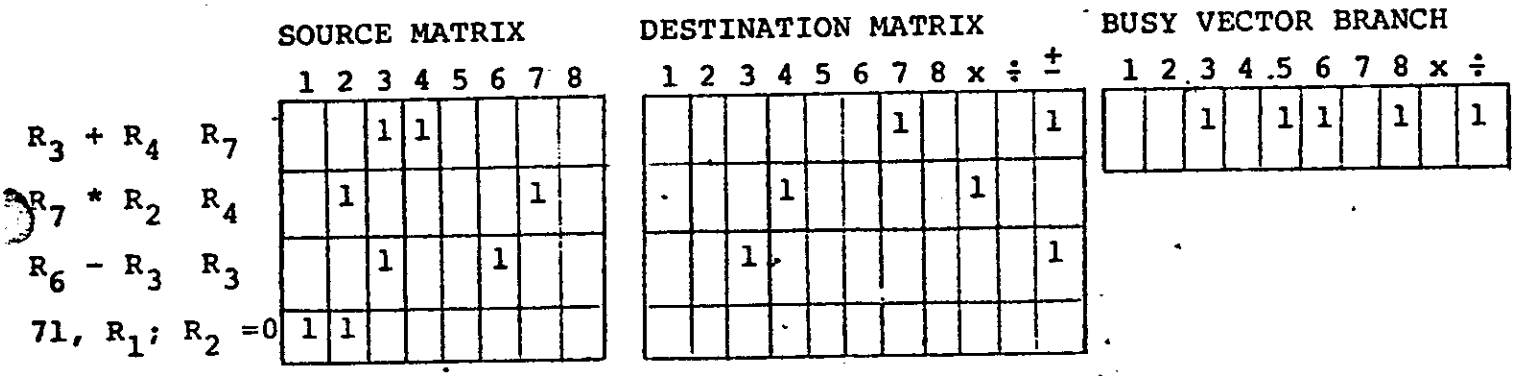


Fig. 6. Example of Multiple Instructions per Cycle Initiation--Cycle 2.

L. Conway Archives

If more than one functional unit of a given type exists but each has its own busses then it is necessary to add a bit to the busy vector corresponding to the new functional unit. No additional columns are added to the Destination Matrix.

In the discussions above it has been tacitly assumed that the functional units were completely passive since the scheduler dispenses operands to the functional units for execution. If instead one takes the approach that the functional units are active, and that the sequencing matrixes are used by the functional units to provide the necessary interlock information then the handling of multiple functional units of a given type is perhaps easier to envision. The functional unit then executes the uppermost instruction that has a one in the column of the Destination Matrix corresponding to the functional unit and has its registers free. With multiple functional units the individual functional units must in addition check the status of all life functional units.

If the number of instructions that can be initiated per cycle is restricted by the number of busses, i.e., one has fewer busses than functional units or rows in the sequencing matrices, one can then take the approach that each instruction uses a functional unit called buss in addition to the functional unit explicitly requested by the instruction.

019

L. Conway Archives

CONCLUSION

In this paper we have described a dynamic scheduling mechanism for providing a look-ahead capability which enables the execution of instructions to be initiated out-of-sequence. In addition the mechanism is capable of controlling the simultaneous initiation of two or more instructions.

The generality of register and functional unit interlocking provided by the mechanism may well be in excess of what is necessary for a given computer design. The modifications to suit any particular design will usually be reasonably obvious and are beyond the scope of this paper.

020

L. Conway Archives

REFERENCES

L.
1. G. M. Amdahl. Engineering Aspects of Large, High-Speed Computer Design; Part II--Logical Organization. Paper presented at the Office of Naval Research Symposium on High-Computer Hardware, November 17-18, 1964, Washington D. C.

2. T. C. Chen. The Overlap Design of the IBM System/360 Model 92 Central Processing Unit. AFIPS Conference Proceedings Vol. 25, Part 2. 1964 Spring Joint Computer Conference. Spartan Books, Washington D. C. (1964) pp. 73-80.

3. J. E. Thornton. Parallel Operation in the Control Data 6600. AFIPS Conference Proceedings Vol. 25, Part 2 Spring Joint Computer Conference. Spartan Books, Washington D. C. (1964) pp. 33-40.

021

L. Conway Archives

IBM CONFIDENTIAL

ADVANCED COMPUTING SYSTEMS
SAN JOSE
March 15, 1966

MEMORANDUM TO: File

SUBJECT: ACS Simulation Technique

D. P. Rozenberg

L. Conway

R. H. Riekert

LC/DP/RR:kp

cc: Architectural Distribution List

022

L. Conway
Archives

INTRODUCTION

A brief description of computer simulation of physical systems in general and the features of current simulation languages is given.

A technique is then described for simulation using FORTRAN IV, which maintains the essential features of current simulation languages with a great improvement in run times and core requirements.

This technique may be useful in the production of very large simulation programs where run times and core requirements are such that programming in existing simulation languages may not be feasible.

L. Conway
Archives

SYSTEM SIMULATION

Assume that it is of interest to study the behavior of complex systems or automata. If the level of complexity is such that the number of states of the system and the possible sequences of states is very large, then a logical approach to such a study is to simulate the system using a digital computer.

Physical systems are usually described in terms of laws or logical rules relating causes and effects; i.e., a given state together with inputs to the system causes or determines the state (or the probability of selecting the state) at some future time. The "behavior" of the system is thus the sequence of states of the system during the passage of time, in response to a specific input sequence.

A computer simulation thus consists of identifying variables which determine the states of a system and the rules for future state selection (the cause and effect relation) and implementing this model with a computer program. Thus it is possible to artificially experiment with the system, and to study the sequence of states for chosen sequences of inputs, with time as an independent variable of the simulation.

In simulating a system it is necessary to perform a computation only at those times when a state or an input has changed since it is only at such times that a future change of state can be caused. It is therefore not necessary to examine the system at regular clocked intervals. Indeed, this may be vastly more efficient, than examining a system at clocked intervals of simulated time if the time interval between changes of state varies over a wide range of values.

Thus it is found that digital simulation languages may provide the programmer with utility routines for (1) providing a means of causing at future times those effects determined by past and present system states and inputs, and (11) advancing time, as an independent variable of the simulation, to the next scheduled effect (change of state) or to the next change of the input sequence, and (111) passing control to that subroutine which simulates the effect and which itself may cause future effects. Perhaps the best known simulation languages of this type are SIMSCRIPT and GPSS . These languages have in addition to the above features a number of utilities which (1) ease the specification of variables and events, (2) ease the writing of the simulation model description, and (3) simplify the production of output routines.

L. Conway Archives

For many purposes these additional utilities are not essential. Indeed, they may cost a high overhead in terms of core space and running time.

GPSS has eased the writing of the simulation to the point where one often cannot specify sufficiently complex tests for branching. Thus, it does not document well a complex description. SIMSCRIPT is sufficiently general but a high cost is extracted in storage and running time because it attempts to simplify the handling of variables.

So, to have a powerful simulation language or technique without all the unnecessary utility features of existing languages, it was decided to write utility routines to perform the basic simulation requirements. A decision had to be made on the language in which to write the simulator utilities routines and also the simulated system description.

If it is important to use the program listings as documentation of the model, a high level language may be necessary. Otherwise, an assembly language might give slight time and storage advantages. In either case, it is desirable to use a common language which runs under a reasonably powerful operating system.

Since in most detailed simulations, the exact model description and documentation can only reside in the simulation program listings, a high level language was chosen as the basic language.

Thus, the utility routines described in the following sections and the model descriptions are all written in FORTRAN IV which is a common high level language running under IBSYS. IBSYS is an operating system which is sufficiently powerful so as to be a valuable aid in running and debugging programs.

L. Conway Archives

THE FORTRAN IV SIMULATION ROUTINES

A general description of the simulation utility routine written in FORTRAN IV follows.

The central idea in the operation of the simulation program is the ordered placement of event notices into a calender of future events as the related cause statements are encountered. The calender is ordered according to increasing time of occurrence. When an event terminates (i.e., the event subroutine terminates), the ordering of the calender indicates the most imminent event and its scheduled time of occurrence. Thus time can be advanced to that scheduled time and the appropriate event subroutine can be called.

A set of arrays, located in blank common, comprise the calender. An event notice consists of one element from each array with the same index. Each notice contains linking information, the scheduled time of occurrence, an indication of the event routine to be called, and three parameters, to be used by the event routine. An event notice will be said to occupy a row of the calender.

During the execution of an event routine, conditions may call for the causing of an event. This is implemented by calling utility subroutine CAUSE with the parameter set: Name of event routine being caused, the time at which the event is to occur, and zero to three parameters to be passed to the event routine. The utility subroutine CAUSE will place in the calender the appropriate event notice. An event may cause any number of events including itself to occur at a future time.

After completion of an event routine, control is returned to routine MAIN. MAIN then calls the utility TSTEP which extracts the next most imminent event from the calender, sets simulated time to the scheduled time of that event, and transfers control to the appropriate event routine. Upon completion of one event routine, control is passed to next most eminent event routine which will then have the capacity for causing additional events.

In some instances it is desirable to cancel an event which may have been scheduled for the future. To accomplish this a utility subroutine REMOVE is included. It is called with the name of the event to be canceled as a parameter and its function is to search the calender for the first instance of an event notice having the name of the event to be canceled. That event notice is then removed.

The routine package for any given simulation would contain MAIN, CAUSE, REMOVE, and TSTEP plus all of the event subroutines necessary for specifying the model being simulated. CAUSE, REMOVE, and TSTEP are all utility subroutines which are invariant from one simulation to another. MAIN varies from one simulation to another only in that

L. Conway Archives
026

it is desirable to have MAIN contain common statements which include all the systems variables and initializations of system variables.

Included in COMMON are the special variables and the system variables. The special variables include the calender arrays; TIME- the current value of simulated time; IPAR 1, IPAR 2, and IPAR 3 - the parameters associated with the current event; and ISL and ITL - pointers utilized in the calender manipulation. The system variables are those variables in terms of which the programmer describes his simulation model.

The calender consists of six single indexed arrays which are indexed by the same pointer. Thus the calender will be referred to as though it were a two dimensional array with six columns. Column 1 contains linkage for the ordering of event notices. Column 2 contains the time of occurrence while Column 3 contains the reference to the event routine. The remaining columns contain parameters to be passed to the event routine; associated with the event are two pointers - ITL which specifies the next most imminent event and ISL which specifies the row to be filled by the next call of subroutine CAUSE.

As part of the initialization in MAIN, the linkage in the calender is set up. The first row is linked to the second, the second to the third, and so forth. The link in the last row is set to zero to indicate the end of the chain. The first row of the calender is set to indicate an event with a very large value of scheduled time. (This simplifies the calender searching in CAUSE. Finally, ITL is set to 1 and ISL is set to 2.

To schedule an event (i.e., place an event notice in the calender) the time of occurrence, the event routine reference, and the three input parameters are stored in positions 2 through 6 of the row indexed by ISL. Following this, ISL is set equal to the value of the link in the same row. Next, column 2, the time of occurrences, is searched beginning with the row designated by ITL in the order given in column 1, the linkage column. The object of that search is to find an event row k with a time of occurrence which is greater than the occurrence time of the event being scheduled. When such a row is found, the links are adjusted to schedule the new event ahead of the event in row k. The initial event in row 1 guarantees that we find a row k.

Whenever TSTEP is called, position 2 is stored in the COMMON variable TIME, and positions 4, 5, and 6 are stored variables IPAR 1, IPAR 2, and IPAR 3. In addition, the old value of position 1 goes into ITL, the old value of ITL goes into ISL, and the old value of ISL goes into position 1. Finally, the event routine reference is used to call the appropriate event.

An example will now be given to illustrate calender manipulation. Assume that the calender is in the state given in figure 1.

027

L. Conway Archives

Calender

Index	Link	Time	Event Reference	Par 1	Par 2	Par 3
1	0	10^{30}				
2	4	1.0	Event 1			
3	6	2.0	Event 17			
4	3	1.5	Event 3			
5	1	4.0	Event 9			
6	5	3.0	Event 5			
7	8					
8	9					
9	10					

ISL=7, ITL=2

FIGURE 1

L. Conway Archives

Assume that the next encountered utility call is

CALL CAUSE (EVENT 12, 3.25, 0, 0, 0).

The result is shown is figure 2.

Index	Link	Time	Event Reference	Par 1	Par 2	Par 3
1	0	10^{30}				
2	4	1.0	Event 1			
3	6	2.0	Event 17			
4	3	1.5	Event 3			
5	1	4.0	Event 9			
6	7	3.0	Event 5			
7	5	3.25	Event 12			
8	9					
9	10					

ITL=2, ISL=8

FIGURE 2

029
L. Conway Archives

If the next encountered utility call is:

```
CALL   TSTEP
```

The result is given in figure 3.

Index	Link	Time	Event Reference	Par 1	Par 2	Par 3
1	0	10^{30}				
2	8					
3	6	2.0	Event 17			
4	3	1.5	Event 3			
5	1	4.0	Event 9			
6	7	3.0	Event 5			
7	5	3.25	Event 12			
8	9					
9	10					

ITL=4, ISL=2

FIGURE 3

030
L. Conway Archives

The final subject of this section is the transfer of control to the intended event subroutine when the statement CALL TSTEP is encountered in MAIN. Two satisfactory methods have been used. The first method utilizes FORTRAN IV in a perfectly straight forward manner and is the method to be described in this report. The other method (Method 2) has the advantage of being simpler and easier to use than Method 1, but has the disadvantage of depending on specific characteristics of the IBM 7090/94 IBSYS compiler.

In using Method 1 a variable in a block of named common is defined for each event routine. This variable is the event reference mentioned earlier and is thought of as the event name while the event subroutine name consists of the same FORTRAN N symbol prefixed with an X. For example, a particular simulation model might consist of the following five events. The corresponding subroutine names are also given below.

Event Names	Subroutine Names
MOVE	X MOVE
GENER	X GENER (A)
DELAY	X DELAY
PROC	X PROC (X,Y,Z)
FINIS	X FINIS

Further, it is required that the event names be assigned unique integer values from 1 thru N where N is the number of events. The initialization of event names may be done in routine MAIN.

The organization of MAIN could be as follows:

```
      COMMON 11
      . . .
      COMMON /NAMES/      MOVE, GENER, DELAY, PROC, FINIS

      INTEGER, GENER, DELAY, PROC, FINIS


C     SYSTEM INITIATION STATEMENTS

C     CALENDER INITIALIZATION STATEMENTS

      MOVE = 1
      GENER = 2
      DELAY = 3
      PROC = 4
      FINIS = 5
```

031
L. Conway Archives

```
C     PLACE INITIAL EVENT NOTICE
      CALL CAUSE (MOVE, 1.0, 0, 0, 0)

1000  CALL TSTEP (NEVENT)
      GO TO (1, 2, 3, 4, 5), NEVENT

    1 CALL X MOVE
      GO TO 1000

    2 CALL X GENER (IPAR 1)
      GO TO 1000

    3 CALL X DELAY
      GO TO 1000

    4 CALL X PROC (IPAR 1, IPAR 2, IPAR 3)
      GO TO 1000

    5 CALL X FINIS
      GO TO 1000

      END
```

Thus TSTEP returns as the event reference the event number defined in the initialization of event names. The event number is then used to branch to the statement which calls the intended event subroutine.

Method 2 requires less bookkeeping on the part of the programmer. The event subroutine names are the same as the event name and are not included in COMMON. Further, the statements for entering the event subroutines are unchanged from one simulation to another as contrasted to Deck MAIN of Method 1 which must be modified whenever an event is added or deleted. However, one special variable MYSELF is located in COMMON. Its use will be developed later.

Referring to the above example, assume that it is desirable to have event MOVE cause event DELAY T units of time in the future. Subroutine MOVE will contain the two following statements:

```
      Subroutine MOVE

      EXTERNAL DELAY
        .
        .
        .
      CALL CAUSE (DELAY, TIME + T, 0, 0, 0)
        .
        .
        .
      RETURN

      END
```

032

L. Conway Archives

When subroutine CAUSE is entered the address associated with the parameter DELAY is the address of the entry point in subroutine DELAY. Therefore, what gets stored in column 3 of the calender is the first executable instruction in subroutine DELAY. Thus, the event references mentioned above are the first instructions of the event subroutines. As will be apparent below, this Method 2 mechanism works because the first instruction of a subroutine is always a transfer to the prolog of the subroutine.

In deck MAIN, the subroutine selection statements are:

```
1000  MYSELF  =  NEVENT (ITL)
      CALL TSTEP (MYSELF)
      GO TO 1000
```

When statement 1000 has been executed MYSELF contains the first instruction of the event routine to be entered. Following that, subroutine TSTEP is called with the address of MYSELF as the parameter address.

The form of TSTEP is:

```
      SUBROUTINE TSTEP (DUMMY)
        .
        .
      IPAR 1 =
      IPAR 2 =
      IPAR 3 =

      CALL DUMMY (IPAR 1, IPAR 2, IPAR 3)

      RETURN

      END
```

The address of DUMMY is, remember, the address of MYSELF. The CALL DUMMY is translated into the following instructions:

```
      TSX   MYSELF, 4

      TXI      3

      PZE

      PZE   IPAR 1

      PZE   IPAR 2

      PZE   IPAR 3
```

033

L. Conway
Archives

The TSX MYSELF, 4 instruction causes the control to transfer to a location in COMMON with linkage established in index register 4. As mentioned above the first instruction of a FORTRAN IV subroutine compiled by IBSYS is always of the form:

```
TRA  Prolog
```

Therefore, after the TSX transfers control to the location of MYSELF, the value of MYSELF transfers contol to the prologs of the desired event without modifying the return or parameter linkage. This is precisely the desired transfer.

The variable MYSELF serves one other important function. Because FORTRAN does not allow a routine to contain an EXTERNAL statement which contains the name of that routine, event routine MOVE cannot contain a statement of the form:

```
CALL CAUSE (MOVE, . . . .).
```

However, the desired effect will be obtained using:

```
CALL CAUSE (MYSELF, . . . ).
```

The complete listings of the utilities routines required for both Method 1 and Method 2 are given in the appendices.

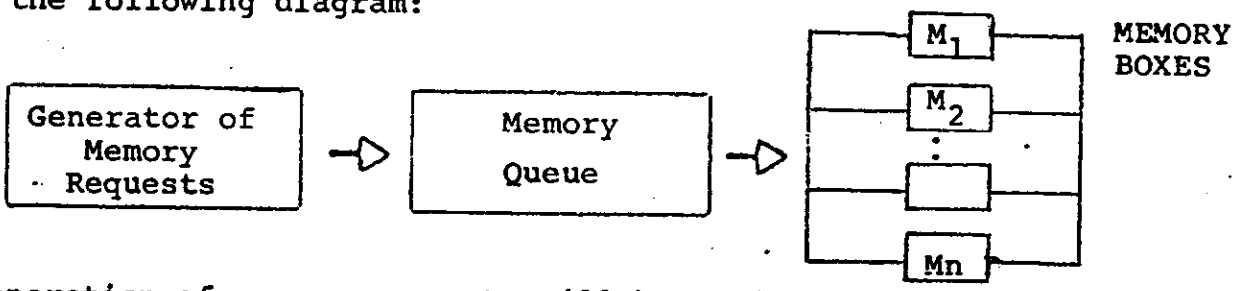
L. Conway
Archives

EXAMPLE

An example will now be described which illustrates the details of implementation of a system simulation in FORTRAN IV.

The system under study will be a simple computer memory queue. Suppose a computer has several independent memory boxes. We may thus queue up memory requests and each computer cycle examine the queue and the memory boxes to see if there is a request on the queue for some non-busy box. A simulation will enable us to experimentally determine such things as average time on queue, average queue length, etc., as a function of request generation rate, number of memory boxes, and the memory cycle time.

The system may be roughly described as consisting of three parts, as in the following diagram:



The generation of memory requests will be artificially modeled by forming either no request or one request per computer cycle, according to some probability, with the box number of the request chosen at random. A generated request will be placed on the queue, if there is space for it. Every cycle, the queue will be scanned for the first request for a free memory box. If one is found it will then cause the memory box to be set busy for the cycle time.

A detailed description of the simulation now follows, with the FORTRAN IV event subroutines separately listed and described.

GENER

The simulation of the generation of requests is performed by GENER, a routine which first causes itself one cycle later and thus runs every cycle. GENER causes a request to be generated if a random number, uniformly distributed between 0 and 1, is found to be less than the specified probability of generating one request in a cycle. If the request is to be generated, a random number is then used to select a memory box for the request. If there is room on the queue, the request is caused to arrive at the queue.8 units of time later, at the "end" of the machine cycle. The listing of GENER follows.

L. Conway Archives

```
$IBFTC GENER
      SUBROUTINE XGENER
      COMMON _ _ _
C     GENERATE MEMORY REQUEST
      CALL CAUSE (GENER, TIME + 1.0)
      CALL RANDOM (R)
      IF(R.GT. PROB1) RETURN
      CALL RANDOM (R)
      BOXNO = (R * FLOAT (NBOX)) + 1.0
      IF(QMPNT.EQ.NQM) RETURN
      INUMB = INUMB + 1
      CALL CAUSE (QBUSY, TIME + .8, BOXNO, INUMB)
      RETURN
      END
```

QBUSY

The event routine QBUSY simulates the arrival of a request on the queue. This is done by incrementing the queue input pointer QMPNT, and placing the instruction number and memory box number into the queue array QM.

```
$IBFTC QBUSY
      SUBROUTINE XQBUSY (BOXNO, INSTR)
      COMMON _ _ _
C     PLACE REQUEST ON QUEUE
      QMPNT = QMPNT + 1
      QM (QMPNT, 1) = INSTR
      QM (QMPNT, 2) = BOXNO
      RETURN
      END
```

QMCON

The simulation of the control of the queue is performed by QMCON. This event first causes itself to run again one cycle later. Then a scan pointer QMSCAN is initialized to one. The queue entry indicated by QMSCAN is then examined to see

L. Conway Archives

if the indicated memory box is busy. If it is, the scan pointer is advanced and the next entry similarly examined. If the box is not busy, the memory request is issued by causing the events MBUSY and QUEMP at .8 units of time later (at the "end" of the cycle), and by causing the event MCYCC at a time .8 + CYCT later.

```
$IBFTC QMCON
      SUBROUTINE XQMCON
      COMMON _ _ _
C     QUEUE CONTROL, SCAN QUEUE AND
C     SEND OUT MEMORY REQUEST, IF POSSIBLE
      CALL CAUSE (QMCON, TIME + 1.0)
      QMSCAN = 1
10    IF(QMSCAN, GT, QMPNT) RETURN
      BOXNO = QM(QMSCAN, 2)
      INSTR = QM(QMSCAN, 1)
      IF(MEMBSY (BOXNO).EQ. 1) GO TO 20
      CALL CAUSE (MBUSY, TIME + .8, BOXNO, INSTR)
      CALL CAUSE (QUEMP, TIME + .8, QMSCAN)
      CALL CAUSE (MCYCC, TIME + .8 + CYCT, BOXNO)
      RETURN
20    QMSCAN = QMSCAN + 1
      IF(QMSCAN.GT.NQM) RETURN
      GO TO 10
      END
```

MBUSY

This event sets the indicated memory box busy by placing INSTR into position BOXNO the array MEMBSY.

```
$IBFTC MBUSY
      SUBROUTINE XMBUSY(BOXNO, INSTR)
      COMMON _ _ _
```

037

L. Conway Archives

```
      C    PLACE REQUEST INSTR IN MEMORY BOXNO
           MEMBSY (BOXNO) = INSTR
           RETURN
           END
```

QUEMP

This event removes the indicated entry from the queue, "moves up" any following entries, and decrements the input pointer.

```
      $IBFTC QUEMP
           SUBROUTINE XQUEMP (QMSCAN)
           COMMON _ _ _
      C    REMOVE REQUEST AT QMSCAN FROM QUEUE
           J = NQM - L
           DO 9 L = 1, 10
           DO 7 K = QMSCAN, J
      7    QM(K,L) = QM(K + 1, L)
      9    QM (NQM,L) = 0
           QMPNT = QMPNT - 1
           RETURN
           END
```

MCYCC

This event simulates the completion of the memory cycle by resetting the memory busy indicator of the specified memory box.

```
      $ IBFTC MCYCC
           SUBROUTINE XMCYCC (BOXNO)
           COMMON _ _ _
      C    AT MEMORY CYCLE COMPLETION, FREE BOX
           MEMBSY (BOXNO) = 0
           RETURN
           END
```


L. Conway Archives

STATS

Included in the list of events is one called STATS which is an output routine. STATS causes itself one cycle later, and outputsthe current system status. The run stops if a specified value of simulated time MAXT is exceeded.

```
$IBFTC STATS

      SUBROUTINE XSTATS

      COMMON _ _ _

C     STATS IS THE OUTPUT ROUTINE

      CALL CAUSE (STATS, TIME + 1.0)
     ////////////////////
      COLLECT AND OUTPUT SYSTEM STATUS
     ////////////////////
      IF(TIME .GE.MAXT) STOP

      RETURN

      END
```

RANDOM

Random is a random number generator. The statement CALL RANDOM(R) returns R to the calling routine a value between 0 and 1 with uniform distribution.

CAUSE

CAUSE is one of the simulation utility subroutines previously specified in this report. It is called to place an event into the calender.

TSTEP

TSTEP is one of the simulation utility subroutines previously specified in this report. It is called from MAIN to advance simulated time to that of the next event in time, and get the parameters and number of that event.

MAIN

MAIN is the first entered and "main" routine of the simulation program and performs a number of functions. First it initializes the common variables to zero. Then the run parameters are read into the appropriate common variables. The calender is then initialized with the proper linkage and starting events are placed into the calender with CAUSE statements. Following and

039

L. Conway Archives

including the statement number 1000 in MAIN are the instruction necessary to cycle thru the events in the calender.

Assume that the following COMMON and specification statements are included in every routine described, and indicated by the statement: COMMON _ _ _

```
      COMMON TIME, IPAR 1, IPAR 2, IPAR 3, ID, ISL, ITL,
     1  LINK (200),CTIME (200), NEVENT (200), KOLI (200),
     2  KOL2(200), KOL3(200), NBOX, NQM, CYCT, MAXT,
     3  PROB1, QM (32,2), MEMBSY(64), QMPNT, INUMB
      INTEGER QM,QMPNT
      REAL    MAXT
      COMMON / NAMES / GENER, QBUSY, QMCON, MBUSY
     1  QUEMP, MCYCC, STATS
      INTEGER GENER, QBUSY, QMCON, QUEMP, STATS
```

The listing of MAIN follows.

```
$IBFTC MAIN
      COMMON _ _ _
      EQUIVALENCE (COM(1), TIME), (X, CTIMEC(1))
C     MAIN INITIALIZES COMMON TO ZEROES. READS IN
C     SYSTEM PARAMETERS, SETS UP THE CALENDER, INITIALIZES
C     THE EVENT VALUES, PLACES STARTING EVENTS INTO THE
C     CALENDER AND THEN CONTROLS THE SEQUENCING OF EVENTS
      DO 101 I = 1,3000
  101 COM (I) = 0
      READ  PROB1, CYCT, NQM, NBOX, MAXT
      TIME = 0.0
      DO 92 ITL = 2,199
   92 LINK (ITL) = ITL + 1
      ISL = 2
      ITL = 1
      X = 1.0E30
      GENER = 1
```

040

L. Conway Archives

```
      QBUSY = 2
      QMCON = 3
      MBUSY = 4
      QUEMP = 5
      MCYCC = 6
      STATS = 7
      CALL CAUSE (STATS, TIME + 1.0)
      CALL CAUSE (QMCON, TIME + 1.1)
      CALL CAUSE (GENER, TIME + 1.1)
1000  CALL TSTEP (EVENT)
      GO TO (1, 2, 3, 4, 5, 6, 7), EVENT
   1  CALL GENER
      GO TO 1000
   2  CALL XQBUSY (IPAR 1, IPAR 2)
      GO TO 1000
   3  CALL XQMCON
      GO TO 1000
   4  CALL XMBUSY (IPAR 1, IPAR 2)
      GO TO 1000
   5  CALL XQUEMP (IPAR 1)
      GO TO 1000
   6  CALL XMCYCC (IPAR 1)
      GO TO 1000
   7  CALL XSTATS
      GO TO 1000
      END
```


L. Conway Archives

REFERENCES

1. K. Blake and G. Gordon, "Systems Simulation with Digital Computers," IBM Systems Journal, Vol. 3, No. 1, 14 (1964).

2. R. Efron and G. Gordon, "A General Purpose Digital Simulator and Examples of its Application: Part I Description of the Simulator, " IBM Systems Journal, Vol. 3, No. 1, 22 (1964).

3. "General Purpose Systems Simulator II," Form B20-6346-1, International Business Machines Corporation.

4. B. Dimsdale and H. M. Markowitz, "A Description of the SIMSCRIPT Language," IBM Systems Journal, Vol. 3, No. 1, 57 (1964).

5. H. M. Markowitz, B. Hausner, and H. W. Karr, "SIMSCRIPT, A Simulation Programming Language," The RAND Corporation, 1963, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

L. Conway Archives

Appendix A

Listings of utility routines for Method 1

043

L. Conway
Archives

CAUSE - EFN SOURCE STATEMENT - IFN(S) -

```
      SUBROUTINE CAUSE(IEV,T,IP1,IP2,IP3)
      COMMON TIME,IPAR1,IPAR2,IPAR3,ID,ISL,ITL,
     XLINK(200),CTIME(200),NEVENT(200),KOL1(200),KOL2(200),KOL3(200)
C               CAUSE ENTERS EVENTS ONTO CALENDAR
C               ITL IS LOCATION OF FIRST EVENT IN CALENDAR
C               ISL IS LOCATION OF FIRST AVAIL ROW IN CALENDAR
      NEXT=ITL
      GO TO 20
C  10           LOOP UNTIL GIVEN TIME IS LESS THAN NEXT ENTRY IN CALENDAR
   10 LAST=NEXT
      NEXT=LINK(NEXT)
   20 IF (T .GT. CTIME(NEXT)) GO TO 10
      ID=ISL
      ISL=LINK(ISL)
      LINK(ID)=NEXT
C               SEE IF THIS EVENT WILL BE THE FIRST ON THE LIST
      IF (NEXT.EQ. ITL)GO TO 40
      LINK(LAST)=ID
   30 CTIME (ID)=T
      NEVENT(ID)=IEV
      KOL1(ID)=IP1
      KOL2(ID)=IP2
      KOL3(ID)=IP3
      RETURN
   40 ITL=ID
      GO TO 30
      END
```


L. Conway Archives

REMOVE - EFN SOURCE STATEMENT - IFN(S) -

```
      SUBROUTINE REMOVE(EVENT,STIME,I,J,K)
      COMMON TIME,IPAR1,IPAR2,IPAR3,ID,ISL,ITL,
     XLINK(200),CTIME(200),NEVENT(200),KOL1(200),KOL2(200),KOL3(200)
      INTEGER EVENT
      NEXT=ITL
      IF(NEVENT(ITL).EQ.EVENT) GO TO 20
   10 LAST=NEXT
      NEXT=LINK(NEXT)
      IF(NEXT.EQ.0) GO TO 30
      IF(NEVENT(NEXT).NE.EVENT) GO TO 10
C               WE FOUND EVENT
      STIME=CTIME(NEXT)
      I=KOL1(NEXT)
      J=KOL2(NEXT)
      K=KOL3(NEXT)
      LINK(LAST)=LINK(NEXT)
      LINK(NEXT)=ISL
      ISL=NEXT
      RETURN
C               EVENT IS FIRST IN LIST
   20 CONTINUE
      STIME=CTIME(NEXT)
      I=KOL1(NEXT)
      J=KOL2(NEXT)
      K=KOL3(NEXT)
      ITL=LINK(ITL)
      LINK(NEXT)=ISL
      ISL=NEXT
      RETURN
C               EVENT NOT PRESENT
   30 CONTINUE
      STIME=TIME
      I=0
      J=0
      K=0
      RETURN
      END
```


L Conway Archives

TSTEP - EFN SOURCE STATEMENT - IFN(S) -

```
      SUBROUTINE TSTEP(IEVENT)
      COMMON TIME,IPAR1,IPAR2,IPAR3,ID,ISL,ITL,
     XLINK(200),CTIME(200),NEVENT(200),KOL1(200),KOL2(200),KOL3(200)
C               SUBROUTINE TO STEP EVENTS IN CALENDAR
C               ITL IS LOCATION OF FIRST EVENT IN CALENDAR
C               ISL IS LOCATION OF FIRST AVAIL ROW IN CALENDAR
      ID=ITL
      ITL=LINK(ID)
      LINK(ID)=ISL
      ISL=ID
      TIME=CTIME(ID)
      IPAR1=KOL1(ID)
      IPAR2=KOL2(ID)
      IPAR3=KOL3(ID)
      IEVENT=NEVENT(ID)
      RETURN
      END
```


L. Conway Archives

Appendix B

Listings of utility routines for Method 2

L. Conway
Archives

```
      SUBROUTINE CAUSE(IEV,T,IP1,IP2,IP3)
      COMMON TIME,IPAR1,IPAR2,IPAR3,ID,MYSELF,ISL,ITL,
     XLINK(200),CTIME(200),NEVENT(200),KOL1(200),KOL2(200),KOL3(200)
C               CAUSE ENTERS EVENTS ONTO CALENDAR
C               ITL IS LOCATION OF FIRST EVENT IN CALENDAR
C               ISL IS LOCATION OF FIRST AVAIL ROW IN CALENDAR
      NEXT=ITL
      GO TO 20
C  10           LOOP UNTIL GIVEN TIME IS LESS THAN NEXT ENTRY IN CALENDAR
   10 LAST=NEXT
      NEXT=LINK(NEXT)
   20 IF (T .GT. CTIME(NEXT)) GO TO 10
      ID=ISL
      ISL=LINK(ISL)
      LINK(ID)=NEXT
C               SEE IF THIS EVENT WILL BE THE FIRST ON THE LIST
      IF (NEXT.EQ. ITL)GO TO 40
      LINK(LAST)=ID
   30 CTIME (ID)=T
      NEVENT(ID)=IEV
      KOL1(ID)=IP1
      KOL2(ID)=IP2
      KOL3(ID)=IP3
      RETURN
   40 ITL=ID
      GO TO 30
      END
```


L. Conway Archives

```
      SUBROUTINE REMOVE(EVENT,STIME,I,J,K)
      COMMON TIME,IPAR1,IPAR2,IPAR3,ID,MYSELF,ISL,ITL,
     .XLINK(200),CTIME(200),NEVENT(200),KOL1(200),KOL2(200),KOL3(200)
      INTEGER EVENT
      NEXT=ITL
      IF(NEVENT(ITL).EQ.EVENT) GO TO 20
   10 LAST=NEXT
      NEXT=LINK(NEXT)
      IF(NEXT.EQ.0) GO TO 30
      IF(NEVENT(NEXT).NE.EVENT) GO TO 10
C                WE FOUND EVENT
      STIME=CTIME(NEXT)
      I=KOL1(NEXT)
      J=KOL2(NEXT)
      K=KOL3(NEXT)
      LINK(LAST)=LINK(NEXT)
      LINK(NEXT)=ISL
      ISL=NEXT
      RETURN
C               EVENT IS FIRST IN LIST
   20 CONTINUE
      STIME=CTIME(NEXT)
      I=KOL1(NEXT)
      J=KOL2(NEXT)
      K=KOL3(NEXT)
      ITL=LINK(ITL)
      LINK(NEXT)=ISL
      ISL=NEXT
      RETURN
C               EVENT NOT PRESENT
   30 CONTINUE
      STIME=TIME
      I=0
      J=0
      K=0
      RETURN
      END
```


L. Conway Archives

```
      SUBROUTINE TSTEP(DUMMY)
      COMMON TIME,IPAR1,IPAR2,IPAR3,ID,MYSELF,ISL,ITL,
     XLINK(200),CTIME(200),NEVENT(200),KOL1(200),KOL2(200),KOL3(200)
C              SUBROUTINE TO STEP EVENTS IN CALENDAR
C              ITL IS LOCATION OF FIRST EVENT IN CALENDAR
C              ISL IS LOCATION OF FIRST AVAIL ROW IN CALENDAR
      ID=ITL
      ITL=LINK(ID)
      LINK(ID)=ISL
      ISL=ID
      TIME=CTIME(ID)
      IPAR1=KOL1(ID)
      IPAR2=KOL2(ID)
      IPAR3=KOL3(ID)
      CALL DUMMY(IPAR1,IPAR2,IPAR3)
      RETURN
      END
```


L. Conway Archives

IBM CONFIDENTIAL

Date: May 1, 1967
From (location & U.S. mail address): Advanced Computing Systems, Menlo Park, California
Dept. & Bldg.: 985
Telephone Ext.: 275

IBM

Subject: Dual Arithmetic on ACS-1

Reference: S.J.C.C., 1967 and our recent conversation

To: Dr. J. E. Bertram

One of the more formidable features of the ILLIAC IV is dual arithmetic, where a pair of floating point numbers are made to interact with another pair, yielding a pair of independent results:

$$\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \begin{pmatrix} \phi \\ \phi \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \end{pmatrix} \rightarrow \begin{pmatrix} C_1 \\ C_2 \end{pmatrix} \equiv \begin{pmatrix} A_1 \ \phi \ B_1 \\ A_2 \ \phi \ B_2 \end{pmatrix}$$

The scheme is useful on the ILLIAC IV for the following reasons:

1. The 64-bit word length is adequate for a pair of hex-floating numbers, each with 8-bit exponent and 24-bit hex-fraction.

2. Significant time savings can be achieved in the PE by using the already-wide data paths for dual arithmetic. There may be an extra shift cost of 2 cycles per instruction comparing with single 64-bit operations, this extra cost is something like 33% on floating adds (8 cycles rather than a possible 6) and may be more than offset in multiplies because of the shorter fractions.

3. For usual partial differential equations even 16 fraction bits may be adequate because of the sizable discretizing error. Parts of computation which call for longer lengths can be localized without serious effort.

4. Many problems do exhibit low-order parallelism exploitable by this feature. This even includes Monte Carlo computations, where the precision demand is low; radar signal analysis, and pattern analysis in general. Where parallelism is lacking, the two components in the packed word can be detached for individual attention at low timing cost.

051

L. Conway Archives

Dual Arithmetic on ACS-1

With the dual arithmetic feature, the ILLIAC IV PE can claim to be an 8-MIPS machine. Their weather program (NCAR model) by the full 4-QUAD machine is said to achieve 600x6600, with upper and lower hemispheres treated "dually".

The proper way to counteract this claim is to install dual arithmetic ourselves. There are several difficulties:

1. The 48-bit word length is not adequate for an independent pair of floating point numbers each with 12-bit exponent. The fraction would have only 12 bits, small even by the most optimistic advocates of short precision arithmetic.

2. Unless one performs at a rate of two operations per cycle, the saving in time is invisible. The shifting cost would be a major handicap.

3. Excessive hardware to achieve dual arithmetic is more likely on a pipeline machine, where the "fixed-time duration" requirement is compounded by a "uniform flush rate" requirement.

4. The operation code repertoire is already near the 256 "limit".

I would like to advocate a limited form of dual arithmetic in which one exponent is shared by two fractions. This "block-normalization" philosophy is quite acceptable for partial differential equations and matrix computations (Cf. discussions in an earlier memo to file, "Mixed floating add operations" by T. C. Chen, dated March 14, 1967). The following advantages of the new dual arithmetic are apparent, many are unique to the block normalizing format.

1. Parallel comparison shifting with one single shifter.

2. Parallel add with one 48-bit adder (with, however, added extra sign detection, overflow detection, and perhaps extra partial recomplementation features).

3. Parallel post-shifting (normalizing usually just one of the fractions).

L. Conway
Archives

Dual Arithmetic on ACS-1

4. Parallel multiply (with added hardware blocking of carries).

5. Only one exponent handling mechanism is needed.

6. TWO OPERATIONS PER CYCLE PER UNIT.

(It is suspected that the ILLIAC IV dual operations will turn out to be "block normalized" also, to reduce the circuit count.)

There are still some problems. With exponent unaltered, the fraction length is only 17 bits + sign, adequate only for very limited computations such as the weather problem and radar signal analysis. A better deal might be the format

$S_1\ E\ F_1\ ;S_2\ F_2$ or $S_1\ E\ F_1\ ;F_2\ S_2$

with

1 bit for S_1 ,
8 bits for E , (7090 size!)
19 bits for F_1;
1 bit for S_2 ,
19 bits for F_2;

which will have roughly the same fraction capacity as the hex-fraction of 24 bits.

There ought to be a reasonably full dual-instruction set, including packing and unpacking (but perhaps no pipelined divide). I feel dual arithmetic to be more useful than double multiply and double divide, and am again advocating their removal to make room for the dual instructions.

Tien Chi Chen

TCC:va
cc: Dr. G. M. Amdahl
Mr. G. F. Nielsen
Mr. R. E. Pickett
Dr. H. Schorr
Dr. E. H. Sussenguth
SADL

053

L. Conway Archives

IBM CONFIDENTIAL

Date: May 12, 1967
From (location & mail address): Advanced Computing Systems, Menlo Park, 986/031
Telephone Ext.:

IBM

Subject: Architecturally Critical Paths in the MPM

Reference:

To: Dr. H. Schorr

Attached is a list of critical timing paths within the MPM from an architectural point of view. Degradation in any of these paths would have a major detrimental effect on overall MPM performance. By overall is meant a global effect, rather than a local effect such as slippage in divider performance. Of the twelve points noted, those involving the contender stacks and interlocking are by far the most critical.

E. H. Sussenguth

EHS:slb

cc: SADL

054

L. Conway Archives

I. Effective address path: (7 cycle path)

ea generation (three input add)	1
bus to BLCU	1/2
BLCU interference resolution	1
storage delay including bussing	3
BLCU decision per tag entry	1
bus to MPM	1/2
internal MPM bus to functional unit	0

II. By-pass from functional unit output to input (0 cycle path)

1. Full bypassing is eminently desirable.
2. If specialized bypassing is necessary the following groupings are the most important:

add to add
add to mpy
mpy to add
mpy to mpy
add to cmp
mpy to cmp

mixed mpy to d. p. add
d. p. add to d. p. add

integer add (with respect to carry register)

shift to shift
shift to logic
logic to shift
logic to logic
shift to cmp
logic to cmp

index add to ea add
index add to cmp

cmp to branch/skip control

III. A-unit interlock control

When an instruction satisfies its interlock constraints, it must be logically removed from contention so that other instructions dependent on it (because of destination-source interlocks or bus conflict interlocks, for example) can start execution on the next cycle.

L. Conway Archives

IV. X-unit interlock control

When an X-contender stack position is vacated, it is refilled with another instruction so that the new instruction can be interlocked and vacated on the next cycle.

The X-unit register data is bussed to the functional units simultaneously with the interlock determination. If the interlocks fail, the functional unit action is logically stopped in such a way that it can restart on the next cycle. (In particular, a unit with a pipeline rate of 2 or more, must not be "busy" working on the illegitimate data.)

V. Instruction start-up path (3 cycle path in X-unit)

Storage bus to IB's	0 (bypass to dispatcher?)
IB to dispatch register	1
Dispatch to contender	1
Contender to functional unit	1 (2 in A-unit)

VI. Effective branch address path

The worst case timing situation occurs when an EXIT has been detected (in the X-dispatch registers) and the BRANCH instruction has not been executed (is in the X-contender stack).

The computation path is:

interlock tests on BRANCH	cycle 1
compute eba, successful/unsuccessful	cycle 2
test top DO table entry:	
if DO entry is correct:	
next instructions to dispatchers	cycle 3
if DO entry is incorrect:	
correct DO table	cycle 3
next instructions to dispatchers	cycle 4

VII. DO Table alteration

On each cycle both A- and X-pointers can be moved, an old entry be deleted, and a new entry be accepted.

L. Conway Archives

VIII. DO table control of instruction flow

The table entries indicate the number of cycles required to validate DO table entries and permit movement of new instructions to dispatch registers.

	if top DO entry is	correct	incorrect	
	if required instructions in	IB	IB	storage
unsuccessful branch exit		1	2	1 + access
successful branch exit		1	2	1 + access
no exit (normal sequence)		1	2*	1 + access*

*pathological case (hence unimportant)

IX. Next-fetch mechanism

On each cycle the next-fetch mechanism must search IB addresses, send an address to BLCU, search PSC registers, increment its contents by 8, and accept an override signal from the branch control.

X. Computation dependent SKIPs

The following sequence of instructions illustrates the problem

$A^3 \leftarrow$ any A instruction
$C_2 \leftarrow A^3 \geq A^{10}$
SKIP if C_2 or C_{30}
* any A instruction

The data/control sequence is

end of computation (A-unit)	cycle 1
result to compare unit (A-unit)	cycle 2
compare result to condition bit	
condition bits to skip test unit	cycle 3
compute skip condition (X-unit)	
skip condition to A-unit interlocks	
start bussing on NOP the *-ed op (A-unit)	cycle 4

057

L. Conway Archives

The sequence noted (A-unit compare, SKIP, * on A-op) is probably the worst case as the path involves A-to-X and X-to-A communication and is a relatively frequent occurrence in code. The dual sequences are:

(X-cmp, SK, * on A): X-unit skew should alleviate this

(X-cmp, SK, * on X): no inter-unit paths (but important in X-unit)

(A-cmp, SK, * on X): one inter-unit path, of less programming significance

XI. Computation dependent branches

A discussion similar to VIII obtains. An illustrative sequence is:

$A^3 \leftarrow$ any A instruction
$C_2 \leftarrow A^3 \geq A^{10}$
BRANCH if C_2 or C_{30}
EXIT

XII. Functional unit performance

The current performance of the functional units are noted below

Floating point, 48-bit	ADD	3/1 and 4/1
	MPY	3/1
	DIV	10/7 or 10/8
	CMP	1/1
Floating point, 96-bit	ADD	4/1
	MPY	5/3
	DIV	17/14
	CMP	1/1 (maybe 2/1)
Floating point, mixed	MPY	3/1
	DIV	10/7
Integer	ADD	2/1
	MPY	4/2
Index integers	ADD	1/1
	MPY	4/2 (improve to 3/1)
	DIV	13 max, 8 avg (improve to 8 max)
	CMP	1/1
Shift, logic, moves (A and X)		1/1

058
L. Conway Archives

IBM CONFIDENTIAL

Date: August 25, 1967
From (location): Advanced Computing Systems
U.S. mail address): Menlo Park
& Bldg.: 986/031
Telephone Ext.:

ACS AP #67-115

IBM

Subject: MPM Timing Simulation

Reference:
1. ACS AP #66-022, ACS Simulation Technique
2. ACS-1 MPM Instruction Manual
3. ACS AP #67-068, MPM-Instruction Sequencing

To: File

L. Conway

LC:slb

cc: SADL

059

L. Conway Archives

CONTENTS

L. Conway Archives

INTRODUCTION

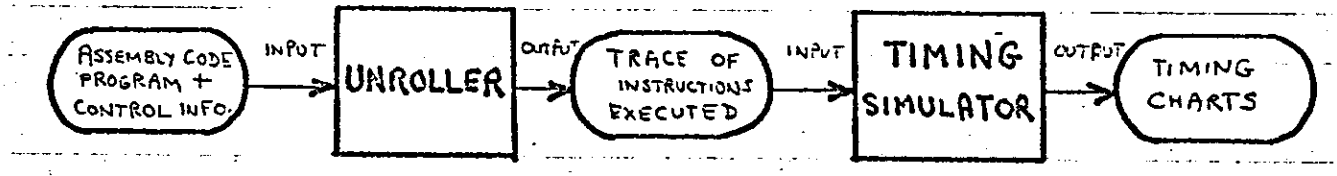
This memo describes the programs which perform MPM timing simulation. It is primarily a "users manual" for these programs.

Two programs, the Unroller and the Timing Simulator, are run consecutively in order to time the MPM's execution of a user's input program.

The Unroller program accepts an ACS assembly language program and control information concerning branch and skip execution, and "unrolls" the program to produce a trace of the instructions executed by the MPM when running the program. The trace is the sequence of instructions along with their addresses, register fields, and certain other information.

The Timing Simulator then operates on the trace of instructions executed by the MPM and produces timing charts indicating the timing of the activities initiated by these instructions in the various hardware components of the MPM.

The following diagram illustrates the functions and relationships of these two programs.



In the following sections of the memo, these programs are separately described with examples given illustrating preparation of input and interpretation of output.

The job running procedures for using the programs is described, and the MPM ops currently implemented in the Timing Simulator are listed.

Since the programs are currently undergoing changes, the current and planned changes are described to assist users in their planning.

Criticisms and suggestions from potential users are welcome and will be helpful in making the Timing Simulator useful to ACS.

061

L. Conway Archives

THE UNROLLER PROGRAM (Prog. by J. Novicki, CSC)

The Unroller program produces the input trace to the Timing Simulator from an ACS assembly code program plus control information.

In the past an Execution Simulator, which performed a detailed simulation of the execution of an input program, was used to generate the instruction trace. It was found to be inconvenient to use an execution simulator for this purpose because that requires the accurate programming of all the tests and computations which determine the desired path of execution through the program. It often proved to be difficult and time consuming to write a correctly executing program even though the path to be followed was easily described.

The Unroller program was written to solve this problem. Given an ACS assembly language program, explicit indicators are placed on the branch and skip instructions of the program to determine the path of instruction execution. For example a branch op might be followed by (3 BEGIN, *) to indicate that the first three times the branch is executed it is successful with the branch being to the instruction labelled BEGIN, and the fourth time the branch is executed it is unsuccessful.

This program and control information is processed by the Unroller to yield the trace of instructions executed, which may then be used as input to the Timing Simulator.

Input Language, Card Input Format

Input cards may contain a label, an op code and operands. The Branch and Skip instructions may contain additional control information. A free form format is used with no fixed starting columns for each of these fields but with certain delimiter restrictions. An asterisk in column 1 indicates a comment card.

Label: A label can be up to 8 characters maximum and must start with one of the characters A through Z or $. A label can contain no imbeded blanks and must be terminated by a delimiting colon.

Op Code: An op code can be up to 6 characters long with no embedded blanks. It may be immediately followed by an asterisk to indicate the skip flag. At least one blank column must be between the op code and its operand fields.

062

L. Conway Archives

Operands: The operand fields can contain information for the i, j, k, and h fields of the instruction. Two fields must be separated by a comma and a missing field will be indicated by two consecutive commas. The first blank column terminates the operand fields. The i, j, and k fields may be one of the following formats:

(i) Ldd
(ii) dd

where "L" is the letter A for Arithmetic Register or the letter X for Index Register or the letter C for Condition Register or the letter S for Special Register. "dd" is a decimal number from 00 to 31 (leading 0 may be omitted). The h field may contain a symbolic label or a decimal number (up to 5 digits).

Branch Parameters: A string of control parameters may be listed after a branch instruction to determine the path of instruction sequencing. The parameters indicate if the branch is successful or unsuccessful for each time it is executed. The branch parameter information must begin with a left parenthesis and end with a right parenthesis and contains no imbedded blanks. Two parameters in the list must be separated by a comma. The parameter format is:

(i) dL for a successful branch
(ii) d* for an unsuccessful branch

where d is an optional digit indicating the number of times the branch is successful or unsuccessful, L is the symbolic label of the instruction branched to, and * is an indicator for an unsuccessful branch. For example, if we have the instruction

BEQ C1, C2, X4 (3ABC, *, XY)

the program would be expanded to reflect the branch execution as follows:

(i) first three executions of branch are successful and branch is to instruction labelled ABC
(ii) fourth execution of branch is unsuccessful
(iii) fifth execution of branch is successful - to XY

Skip Parameters: A string of control parameters may be listed after a skip instruction to determine the effect of that instruction on the sequence of skip states. The parameters indicate whether the skip is taken or not taken each time it is executed. The parameter string

L. Conway Archives

has the same format as the branch parameter string with any dummy label serving to indicate that the skip is taken, an * indicating the skip is not taken. For example, if we have

SKØR C1, C2 (2*, LABEL, *)

the Unroller would set the skip state in the trace to reflect the execution of the skip as follows:

(i) first two times skip is executed it is not taken
(ii) third time skip is executed it is taken
(iii) fourth time skip is executed it is not taken

Output of Unroller

Corresponding to the sequence of execution of the instructions of the input program the Unroller produces the standard input trace for the Timing Simulator: a card deck which is described in detail in Section 2. One card is produced for each instruction executed. The card contains the op, i, j, k, h fields, branch and skip states, instruction and data reference addresses and certain other fields.

The Unroller also lists the input program and output trace. Certain diagnostic messages may be listed:

(i) Too many input cards (300 maximum)
(ii) Operand Field error
(iii) Error on following card (i. e. label information error)
(iv) Op code on next card not implemented

Example: On the following page are the listings of a simple input program deck and the trace deck produced by the Unroller from that input deck. Note that the branch parameter list specifies branch successful two times then branch unsuccessful. Thus we make 3 passes through the loop. The branch and skip states in the trace (see trace format Section 2) reflect the branch and skip execution. Note: the OP "STOP" terminates unrolling, and the pseudo op "END" marks the end of the unroller input deck.

L. Conway Archives

EXAMPLE: UNROLLER INPUT DECK - - - - - - - - -

```
LOOP:   CGEX    2,4,3
        BAND    2,2,0,0     (2LOOP,*)
        CGEN    1,1,2
        AXK     3,3,0,1
        AN      1,1,8
        LA      8,0,0,1000
        AN      2,2,9
        LA      9,0,0,2000
        SKOR    1,1         (*,2DUMMY)
        MN*     1,1,2
        EXIT
        STA     1,0,0,1000
        STOP
        END
```

CORRESPONDING UNROLLER OUTPUT DECK - - - - - - - - - - - -

```
 0 CGEX    02 04 03  00000   000  00000    1   87  1
 1 BAND    02 02 00  00000   100  00000    3  139  2
 3 CGEN    01 01 02  00000   100  00000    4   79  1
 4 AXK     03 03 00  00000   100  00000    6   76  2
 6 AN      01 01 08  00000   100  00000    7  166  1
 7 LA      08 00 00  01000   100  01000    9    7  2
 9 AN      02 02 09  00000   100  00000   10  166  1
10 LA      09 00 00  02000   100  02000   12    7  2
12 SKOR    01 01 00  00000   100  00000   13  150  1
13 MN*     01 01 02  00000   101  00000   14  178  1
14 EXIT    00 00 00  00000   100  00000    0  199  1
 0 CGEX    02 04 03  00000   000  00000    1   87  1
 1 BAND    02 02 00  00000   100  00000    3  139  2
 3 CGEN    01 01 02  00000   100  00000    4   79  1
 4 AXK     03 03 00  00000   100  00000    6   76  2
 6 AN      01 01 08  00000   100  00000    7  166  1
 7 LA      08 00 00  01000   100  01000    9    7  2
 9 AN      02 02 09  00000   100  00000   10  166  1
10 LA      09 00 00  02000   100  02000   12    7  2
12 SKOR    01 01 00  00000   100  00000   13  150  1
13 MN*     01 01 02  00000   111  00000   14  178  1
14 EXIT    00 00 00  00000   110  00000    0  199  1
 0 CGEX    02 04 03  00000   010  00000    1   87  1
 1 BAND    02 02 00  00000   010  00000    3  139  2
 3 CGEN    01 01 02  00000   010  00000    4   79  1
 4 AXK     03 03 00  00000   010  00000    6   76  2
 6 AN      01 01 08  00000   010  00000    7  166  1
 7 LA      08 00 00  01000   010  01000    9    7  2
 9 AN      02 02 09  00000   010  00000   10  166  1
10 LA      09 00 00  02000   010  02000   12    7  2
12 SKOR    01 01 00  00000   010  00000   13  150  1
13 MN*     01 01 02  00000   011  00000   14  178  1
14 EXIT    00 00 00  00000   010  00000   15  199  1
15 STA     01 00 00  01000   010  01000   17    9  2
```


L. Conway Archives

THE TIMING SIMULATOR (Prog. by L. Conway, J. F. Parsons)

For the purpose of MPM hardware or program evaluation we may need detailed timing of the execution of a program by the MPM. The MPM is sufficiently complex that hand-timing of all but trivial programs is a very tedious process. The Timing Simulator is a program written to perform this timing by simulating in complete detail the hardware controls of the MPM.

The Timing Simulator is written in FORTRAN IV (H) and runs on a S/360 under OS, requiring an H level machine. The simulation technique is similar to SIMSCRIPT but uses simpler utility routines which are written in FORTRAN. Reference 1 provides a complete description of the simulation technique.

The level of hardware modelling performed by the Timer is best described as being an "architectural" level. Individual hardware triggers are included when they serve an individual control function, but buses, registers, etc., are modelled as logical entities rather than simulated to the bit level. Thus the timer does not model the detailed engineering implementation of the MPM. It does model all control algorithms in all sections of the MPM, to accurately simulate the timing of instruction execution by the MPM.

The Timer currently operates on a MOD 75 at a rate of approximately 10 simulated machine cycles per second. Typical programs are thus simulated at a rate of 20 inst./sec.

A detailed description of either the Timing Simulator program or the MPM model simulated is beyond the scope of this memo. Users may assume that the program reflects the latest specification of the MPM. This model is documented at an architectural level in Reference 3 and other similar references soon to be issued. Those who are familiar with the hardware design of the MPM and have specific questions about the details of the simulation model should contact the author.

The remainder of this section on the Timer is concerned with the practical problems of preparing input and interpreting the output timing charts.

The input to the timer is a "trace" of the instructions actually executed by the program to be timed. The trace consists of the sequence of instructions executed along with certain control information. This input is prepared by running an ACS assembly code program through the Unroller program (see Section 1).

066

L. Conway Archives

Certain job controlling cards including a specification of the hardware parameters for the run are added to the trace deck to form the input deck.

The output of the Timer is a series of timing charts which illustrate the activities initiated by the instructions of the input program trace in the various hardware components of the MPM as a function of time.

A detailed description of the input and output formats and output interpretation is given on the following pages. Examples are given which follow the paths of individual instructions through the various sections of the MPM as a function of time.

Timing Simulator Input Preparation

Input Trace Cards: The Unroller program is used to produce the input trace card decks for the Timing Simulator. An ACS assembly code program is run on the Unroller and a trace deck is produced as output. Refer to Section 1 for information on this program. The trace deck produced by the Unroller is an instruction by instruction record of those instructions actually executed by the program to be timed. Each instruction of the trace is present on a separate card. The format of these cards is specified in Fig. 2-1.

Timer Input Deck Format: Each program to be timed is formed into one deck beginning with a machine parameter card, followed by the trace cards for the program, and ending with a card containing 999 in cols 55, 56, 57 (a "STØP" card). A number of such input decks may be stacked and timed during one execution of the Timer. An example of this stacked job deck structure is illustrated in Fig. 2-2.

Parameter Card: The first card of each input program deck is a parameter card which specifies certain MPM hardware parameter values and certain parameters for the running of the job (maximum simulated time, etc.). These parameters are the following:

JOBNAME: Up to six characters identifying program

NABUF, NATEST, NAGØ: The number of A Buffers, the number tested each cycle for OP issuance, the maximum number of OP which may be issued for execution each cycle from the A Buffers (A Contending Stack).

NXBUF, NXTEST, NXGØ: Similarly for X unit Contending Stack.

L. Conway Archives

NQBUF, NQTEST, NQGØ: Similarly for Data Memory Queue.

NBØX: Number of memory boms.

NBBUF, NSBUF: Number of Exit History Table positions, number of Skip Table positions.

NØDØT: Number of DØ Table positions.

NØPSC: Number of PSC registers.

NDBUS: Number of Dispatcher Buses.

NADSP: Maximum number of OPS which may be dispatched to the A Buffer per cycle.

NXDSP: Similarly for X dispatching.

MXTIME: Run control parameter. Maximum simulated time allowed for run (in machine cycles). Run terminated if this time is exceeded.

MEMDLY: Memory Delay Time. See example of arithmetic load G7 on page 2-13 for exact definition.

ØUTLVL: One of four output levels may be chosen. Level 0 is most detailed, Level 3 is least detailed (and fastest cunning). Level 1 is normally used and is level shown in the examples at the end of this section.

FSTADD: Starting address of the input program.

Fig. 2-3 specifies the format of the parameter card. Minimum, typical, and maximum values of the parameters are given. The TYP values represent the "most likely" values of the hardware parameters.

There are other machine parameters not controlled by the parameter card which may be easily varied by changing certain initialization tables in the Timer. An example of this is the busing and facility characteristics in the A and X execution units. These structures are listed in the output for each run (see output portion of this section). If changes in these machine parameters are desired for a particular timing study, contact the author.

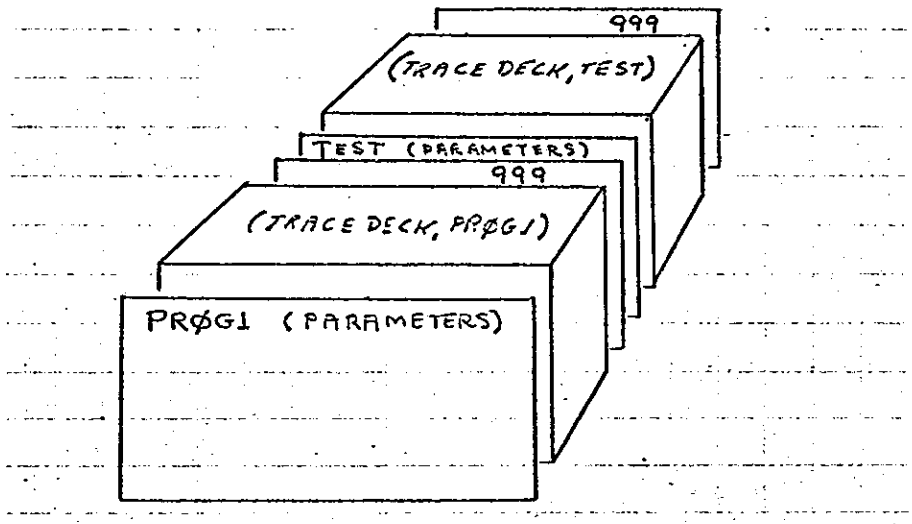
L. Conway Archives

Figure 2-1. Timer Input Track Card Format

		COLS
1.	Instruction Address	2-6
2.	Op Code Mnemonic (left justified)	8-14
3.	I (Dec)	16-17
4.	J (Dec)	19-20
5.	K (Dec)	22-23
6.	H (Dec)	26-30
7.	Branch Successful bit. Indicates result of branch op. Applies from and including branch op to and including EXIT op.	35
8.	Skip Flagged ops bit. Indicates skip state. Applies to op after skip to and including next skip	36
9.	Skip Flag	37
10.	Effective address accessed (LOAD/STORE)	41-45
11.	Address of next instruction to be executed	48-52
12.	Numeric Op Code	55-57
13.	Long Op = 2, Short Op = 1	60

Figure 2-2. Timer Input Deck Format

Example: Two PROGRAMS PRØG1 and TEST to be timed:



L. Conway Archives

Figure 2-3. The Parameter Card Format

PARAMETER	MIN	TYP	MAX	COLS
JØBNAME				1-6
NABUF	1	8	12	9-10
NATEST	1	8	NABUF	11-12
NAGØ	1	3	3	13-14
NXBUF	1	3	12	15-16
NXTEST	1	3	NXBUF	17-18
NXGØ	1	3	3	19-20
NQBUF	1	8	16	21-22
NQTEST	1	8	16	23-24
NQGØ	1	2	NBØX	25-26
NBØX	1	8	16	27-28
NBBUF	1	3	8	29-30
NSBUF	1	4	8	31-32
NØDØT	1	6	16	33-34
NØPSC	0	8	8	35-36
NDBUS	1	2	2	37-38
NADSP	1	4	NABUF	39-40
NXDSP	1	3	NXBUF	41-42
MXTIME		300.0		60-66 (F7.1)
MEMDLY	2.0	5.0		68-71 (F4.1)
ØUTLVL	0	1	3	73-74
FSTADD	0	0		76-80

070
L. Conway Archives

Timing Simulator Output Interpretation

For each input job, a deck headed by a parameter card and terminated by a 999 card, an output listing is produced of the following form:

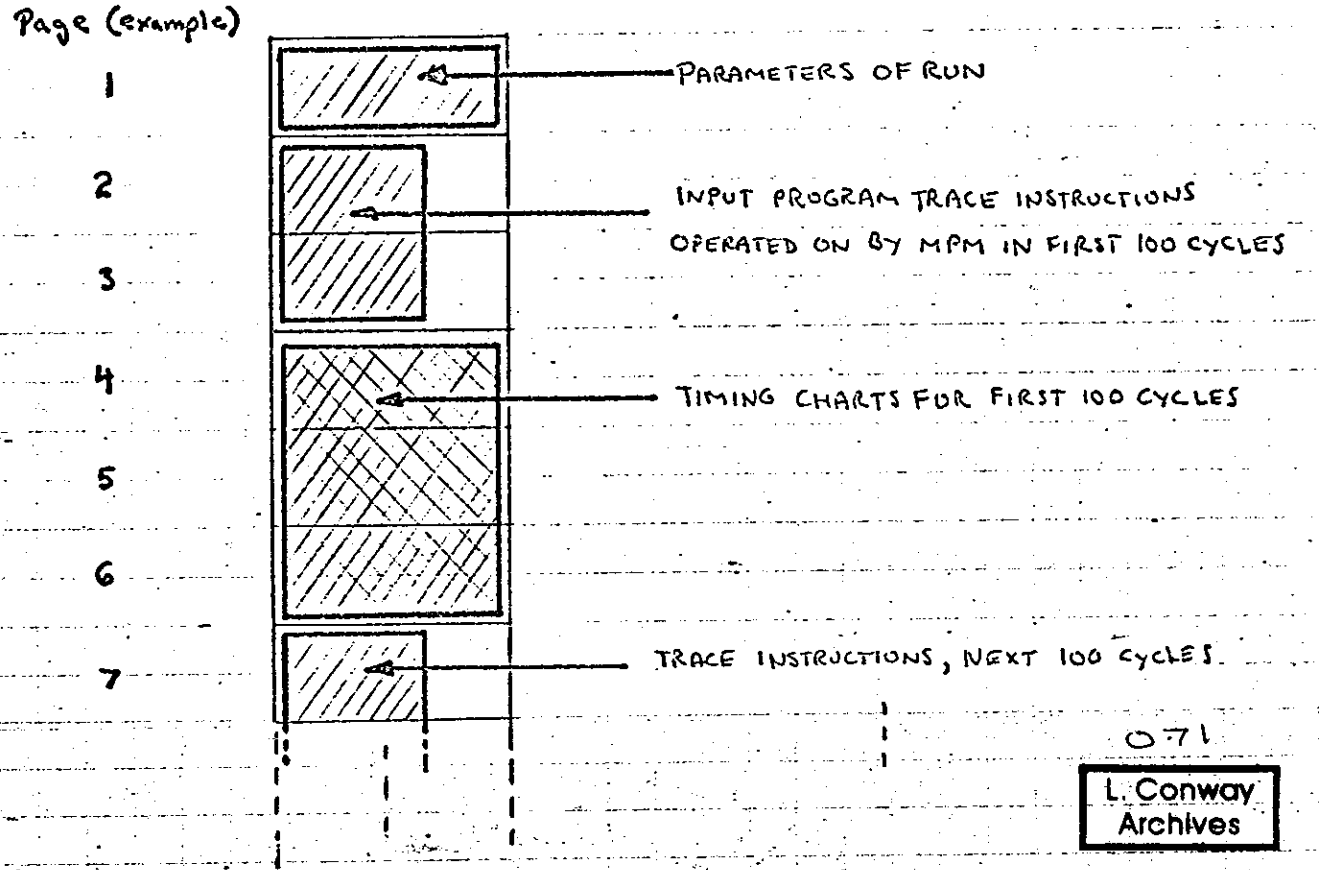
(i) The first page lists the job name and all parameters of the run including the busing and facility structure.

(ii) This is followed by a listing of those input trace instructions operated upon by the MPM during the first 100 simulated cycles of time.

(iii) This is followed by a listing of timing charts indicating the activities initiated by those instructions of (ii) during the first 100 simulated cycles.

(iv) Items (ii) and (iii) are repeated for successive 100 cycle periods till the run stops or is terminated by MXTIME.

Figure 2-4. Overall Form of Output Listings



071
L. Conway Archives

We will now examine the general characteristics of these three components of the output. A sample output listing is included at the end of the section for reference while studying these general descriptions.

- Some specific examples will then be developed which illustrate the progression of instruction activity through the different sections of the MPM. These examples are referenced by markers on the sample output listings.

Parameters of Run: This page lists the job name, date and time of run, and the MPM hardware parameters for the run. Many of these parameters are those specified on the input parameter card, described earlier in this section. The A and X unit busing and facility structures are printed for reference in a table with the following entries:

1. The abbreviated name of the facility (FA1 = floating adder 1).

2. The Rep Time of the facility - the number of cycles an operation keeps the facility busy.

3. The Delay Time of the facility - the number of cycles the facility requires to perform operation.

4. INBUS - the numbers assigned indicate which facilities share a common inbus.

5. BOX - the numbers assigned show which facilities share circuitry and cannot be simultaneously busy.

6. OUTBUS - the numbers indicate which facilities share a common outbus.

Input Program Trace: For each block of 100 cycles of simulated time the Timer prints the instructions of the input trace which have been operated upon by the MPM during that time. This is used to reference the timing charts for that period of time. The input program trace printed is a copy of the input cards with five fields added:

(i) Time markers are placed indicating the time (approx.) that the instruction entered an IB.

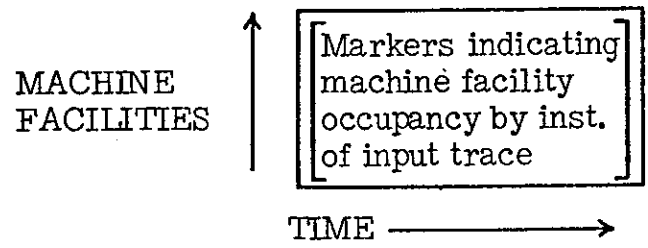
(ii) A letter is assigned to each instruction by decoding the instruction address MOD 26. This letter is then used as the marker for that instruction in the timing charts.

L. Conway Archives

(iii), (iv) Bits are set indicating whether the op is to be dispatched to the A unit, X unit or both.

(v) The number of the IB into which the instruction was fetched. This along with (i) will locate the instruction marker's first appearance on the timing charts (in a dispatch register).

The Timing Charts: A set of timing charts are produced for each 100 cycle period of simulated time. The general form of these charts is as follows:



The time axis has markers every cycle and number indicating 10, 20,...,90 cycle points in the 100 cycle period. The time of the period is listed at the top of the page (ex.: SIMULATED TIME = 300 TO 399).

The machine facilities included in the timing charts are identified as follows:

DSPX1, DSPX2, DSPA1, DSPA2: These are the dispatch registers X1, X2, A1, A2. The IB number and DO table entry are listed which correspond to the contents of the dispatch register. The eight 24-bit instruction fields are shown for each register with markers indicating which instructions of the input trace are currently present.

BRANCH CONTROLS: These are hardware triggers controlling the branching process. ER1, ER2, ER3, BE1, BE2, BE3, ET1, ET2, ET3 are the exit resolved, branch executed, and exit taken entries in the Exit History Table (EHT). BRXP, BRAP are the X and A pointers to the EHT. The description of the other listed controls is beyond the scope of this introductory memo.

SKIP CONTROLS: Skip state triggers with SKXP, SKAP, the X and A unit pointers to the triggers.

L. Conway Archives

A BUFFER, X BUFFER: These are the A and X unit contender stacks where ops are tested for interlocks before issuance to the functional units. This is the point where ops may be issued out of order if the appropriate interlocks are satisfied. The instruction occupancy of the buffer positions is indicated by markers.

A FACILITIES, X FACILITIES: These are the various functional units such as adders, multipliers, shifters, logic units, etc.

The instruction markers are placed in a facility position for that period of time during which the instruction actually has the facility busy for interlocking purposes. Note that an op keeps a facility busy for a number of cycles equal to the REP TIME of that facility.

MEMORY QUEUE (D): The data memory queue. This is the queue which holds data loads and stores after issuance from the contender stacks and before issuance to memory. This queue roughly approximates the timing effects of the BLCU with no paging activity. If appropriate interlocks are satisfied the requests may go out of order. An instruction is indicated by its marker.

MEMORY QUEUE (I): Instruction fetch memory queue. This queue holds the instruction fetch requests prior to issuance to memory. The markers are the IB destination number of the fetch. Four markers are placed corresponding to the four pieces of one request. When all have been issued a new set may enter.

MEMORY: Here we can observe the relative timing of loads, stores and instruction fetches as their markers indicate busy memory BOMS. The marker for an instruction is placed on the second of the two cycles that the op is activating the BOM--noting that the memory BOM REP TIME is one cycle.

A REGS BUSY: When an OP is issued from the A contender stack to a functional unit, the A destination register of the OP is marked busy with the OP marker. This is used to interlock the issuance of other OPS in the contender stack (which use that destination register) until the result arrives at the register (or is available for bypassing to the input of another facility).

L. Conway Archives

ABU REGS BUSY: The A Back-Up Registers are the destination registers for A loads and X to A moves (instructions issued from the X unit contender stack). At the time of issuance the op marker is placed in the ABU REGS BUSY position corresponding to the op destination and remains till the load or move is completed.

X REGS BUSY: The busy bits for the X Registers, similar to the A REGS BUSY described above.

Example of Timing Simulator Output

At the end of this section is a copy of the output listing for a typical run of the Timing Simulator. The parameter page is followed by 3 pages listing the input trace for the first 100 cycle period of time. Then 4 pages are listed containing the timing charts for the first 100 cycles.

The program being timed is a version of Crout Reduction. In this case the MPM is active for only 58 simulated machine cycles--a starting transient is followed by three passes through the inner loop of the program.

The interpretation of the timing charts can be somewhat complex. In this memo only a few simple illustrative examples are given which follow the paths of certain instructions of the sample program through the various sections of the machine.

A thorough knowledge of the MPM hardware controls and considerable practice are necessary for a complete interpretation of the timing charts. However, certain subsets of the charts may be studied with a detailed knowledge of only that section of the MPM. For example, someone interested in compiler scheduling of instructions could focus his attention on the performance of his input programs in the A and X BUFFERS and A and X FACILITIES, observing the effects of various schedulings on the timing through these units. A knowledge of the interlocking rules of the contender stacks and of the busing and facility structure would be sufficient to get a start at this.

Certain simple observations may yield useful measures of MPM performance on the input program. The overall time of the run is easily determined. It is given as the upper time limit on the last set of pages listing timing charts for the run. In our example this overall run time is 58 cycles. Another measure which is often useful is the time taken to execute a program loop. If the input program is of the type

075
L. Conway Archives

which repetitively executes a loop, the loop pattern will be obvious in the A and X FACILITY busy markers on the timing charts. This is because a given op has the same marker symbol each time the loop is executed (the marker is determined by the instruction address). Thus the loop time is found by measuring from marker to similar marker in the A FACILITIES for example. In our sample output we find that the MPM executes the program loop 3 times in the FLOATING MULTIPLIER between cycle 33 and cycle 52. The pattern has not yet settled down to a repetitive one in the example, but the loop time is seen to be approximately 8 cycles.

Some detailed examples follow. Refer to the sample listings at the end of this section.

Instruction Fetching: At time = 1 an instruction fetch request to fill IB(1) has been placed on the MEMORY QUEUE (I). It is issued to MEMORY in the next cycle and (after some busing time) we observe at time = 4 that MEMORY BOMS 1, 2, 3, 4 are busy servicing this request. The fetched instruction is then bused to IB(1) (not indicated in output). At time = 8 we observe that DSPX1 and DSPA1 have been loaded from IB(1). The instructions which were fetched are seen to be A, C, E, G, which are X OPS and in DSPX1, and G which is an A OP and in DSPA1.

Notice that instruction fetching occurs up to time = 33. After this time the loop has been contained in the IB's and no further instruction fetching is required to run the problem.

Multiply Instruction E37: At time = 37 we find the instruction MN 13, 5, 6, which is marked by an "E", in the instruction trace section of the output.

Let us follow the activity of this instruction through the MPM. We observe from the trace that E was fetched into IB(8). At time = 38 we notice that IB(8) → DSPA2 and we find E in DSPA2(1). At time = 38 only two positions are free in the A BUFFER so the OPS X and Y in DSPA1 move to the A BUFFER at time = 39 but E remains in the dispatchers, moving up to DSPA1(1).

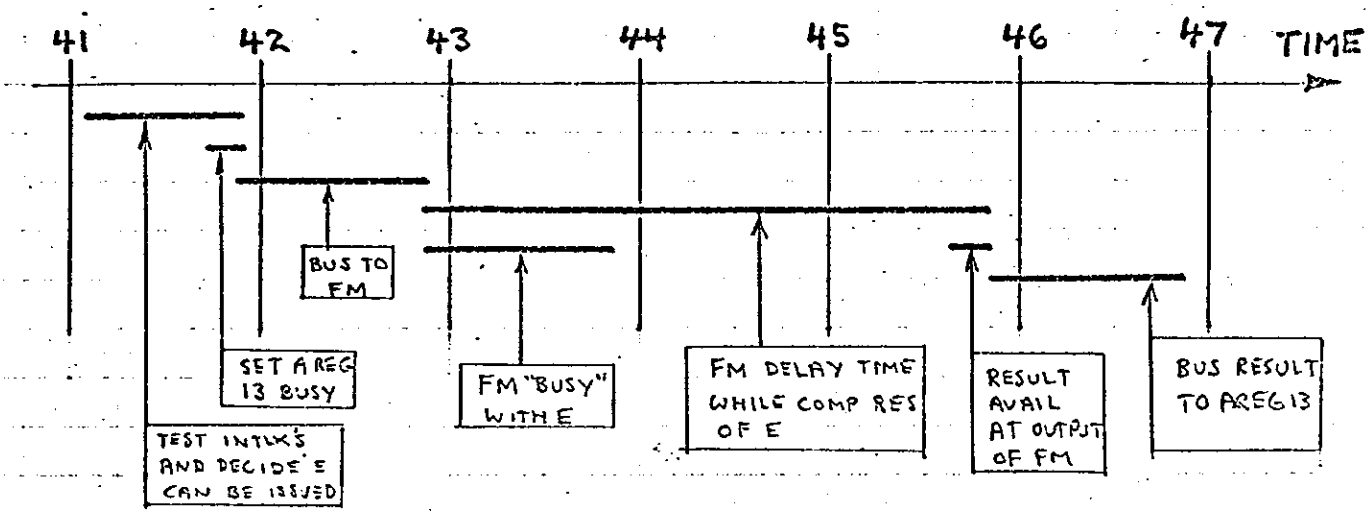
At time = 39, the A BUFFER has two free positions so at time = 40 instruction E along with F are bused to the A BUFFER. We find E in A BUFFER (4) at a time = 40.

L. Conway Archives

Now at time = 40 another multiply instruction, P, is present in the A BUFFER and ahead of E. This multiply, interlocking E, is issued the next cycle while E remains present at time = 41 in A BUFFER (3). At this time there are no ops ahead of it in the buffer which interlock it so it is issued for execution and is not present in A BUFFER at time = 42. Notice that A REG BUSY (13) goes on with the marker E at time = 42 to interlock any OPS following E which use A REG (13) as a source or destination.

The multiplier FM under A FACILITIES is found busy with E at cycle time = 43 (one cycle of busing required from A BUFFER to A FACILITIES). Then at time = 44 the A REG BUSY (13) is no longer marked by E indicating that the result of E will be available (for bypassing) at the output of the multiplier at cycle time = 46. Note that the delay time of the FM is 3 cycles, the multiply E taking cycles 43, 44, 45, with the result actually back at register 13 at cycle 47. But the multiplier is only "busy" with E for one cycle (the REP TIME of FM) so the multiplier could handle a new op every cycle. The timing of the busing and multiplication are illustrated in Fig. 2-5, for the specific example instruction E37.

Figure 2-5. Timing of Example Instruction E37



L. Conway Archives

Arithmetic Load Instruction G7: At time = 7 we find the instruction LAT 9, 0, 31, 136 which is marked by a "G", in the instruction trace section of the output. We observe from the trace that G was fetched into IB(1). It is both an AOP and an XOP and will be dispatched to both units.

At time = 8, we observe from the timing charts that IB(1) → DSPX1, IB(1) → DSPA1. At that time G is present in DSPX1(7), DSPX1(8), and in DSPA1(7), DSPA1(8). G is a long OP and takes two of the 24-bit positions in the dispatchers.

Let us follow the A unit activity of G first. We note that at time = 8 G is the first AOP to enter the dispatchers and thus it is bused to the A BUFFER the next cycle. At time = 9 we find G in A BUFFER (1). This part of G is a "replace" operation and is issued the next cycle, causing A REG BUSY (9) (the destination of the load) to be marked busy with a G at time = 10. This sets the "front" register busy waiting for the "back-up" register to be loaded by the X-unit.

Now let us follow the X unit activity of G. Since three other X OPS precede G in DSPX1 at time = 8, and at most 3 ops may be dispatched to the X BUFFER per cycle, G remains in DSPX1 at time = 9. At time = 10 it is bused to X BUFFER (2), for it is the next op to be dispatched to the X BUFFER and both A and C leave the X BUFFER at time = 10 allowing G to enter.

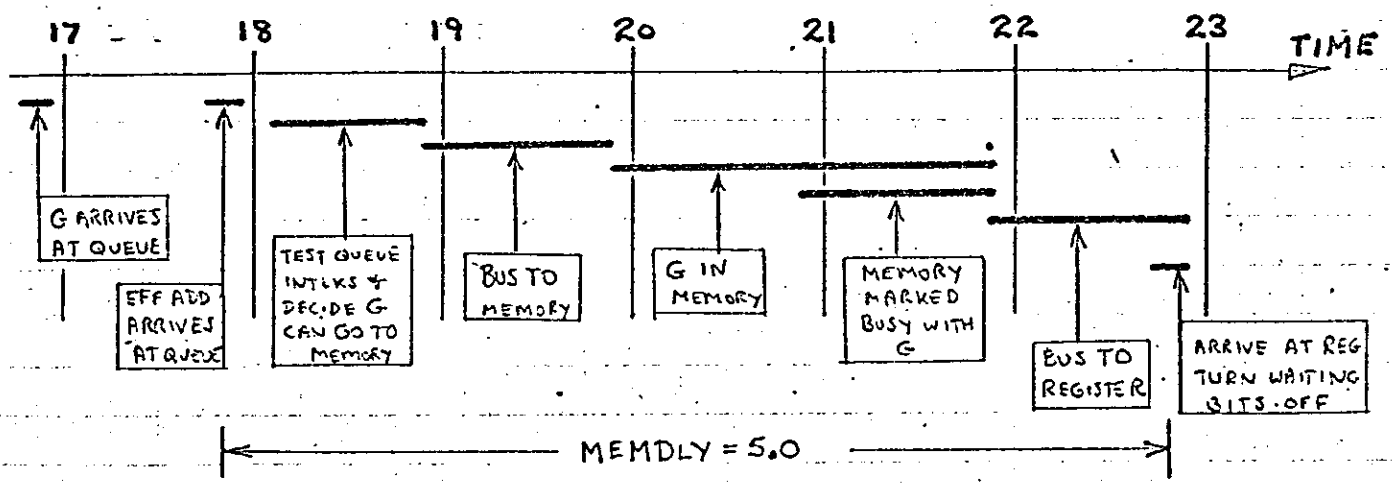
We now find that G remains in the X BUFFER through time = 16. This is because it uses X REG (31) as an index and X REG (31) is busy through time = 15 waiting for a load to arrive.

At time = 16 G finally satisfies the contender stack interlocks and at time = 17 its execution is initiated by (i) starting effective address computation in X FACILITY EA1, (ii) placing an entry in the MEMORY QUEUE (D), (iii) marking the ABU REG BUSY (9) with G. The queue entry waits on the queue another cycle for the effective address to arrive, and then is issued to memory. We note that at time = 21, MEMORY (1) is marked busy with G, and at time = 23 the busy bits on ABU (9) and A(9) are turned off indicating that the load has arrived at ABU (9) and then moved immediately to the waiting A(9).

The detailed timing of this memory activity is illustrated in Fig. 2-6.

L. Conway Archives

Figure 2-6. Timing of Memory Activity of Example G7



L. Conway Archives

- ACS-1 MPM SIMULATION PROGRAM - - - -

INPUT PROGRAM FOR THIS RUN = CR-FS

TIME/DATE OF RUN = 4D72CBFE C067194F

MACHINE PARAMETERS FOR THIS RUN - - -

NUMBER OF A BUFFERS = 8 NUMBER OF X BUFFERS = 3 NUMBER OF Q BUFFERS = 8

NUMBER A OPS TESTED = 8 NUMBER X OPS TESTED = 3 NUMBER Q OPS TESTED = 8

MAX A OPS ISS/CYCLE = 3 MAX X OPS ISS/CYCLE = 3 MAX Q OPS ISS/CYCLE = 2

MINIMUM Q-MEM DELAY = 5.0

NUMBER OF BOMS = 8

NUMBER BRANCH REGS = 3 NUMBER OF SKIP REGS = 4 SIZE OF DO TABLE = 6

NUMBER OF PSC REGS = 8

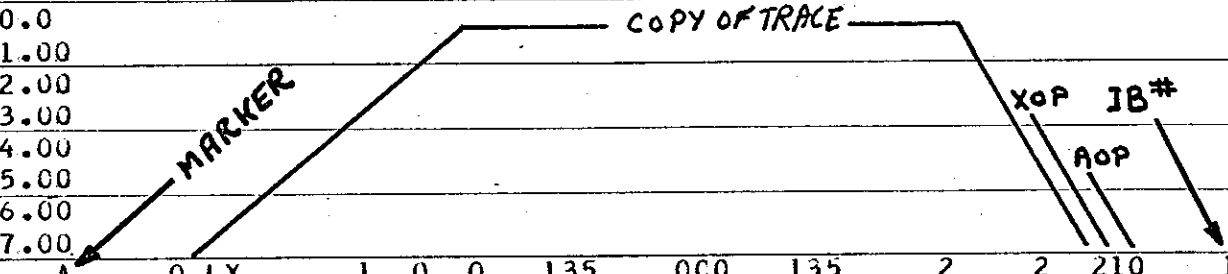
NUMBER DISP BUSES = 2

MAX A OPS DSP/CYCLE = 4 MAX X OPS DSP/CYCLE = 4

A FACILITIES - -	FA1	FA2	FM	FD	IA	IM	ID	C	L	S
REP TIME =	1	1	1	7	1	2	10	1	1	1.
DELAY TIME =	3	4	3	9	2	5	15	1	1	1
INBUS =	2	1	3	1	1	2	2	1	2	3
BOX =	1	2	3	4	2	4	4	5	6	7
OUTBUS =	2	1	4	3	2	4	4	6	1	3

X FACILITIES - -	EA1	EA2	L	S	M	D	XA	C
REP TIME =	1	1	1	1	2	8	1	1
DELAY TIME =	1	1	1	1	4	8	1	1
BOX =	1	2	3	4	5	5	6	7
OUTBUS =	5	6	1	3	2	2	7	10

L. Conway Archives



TIME= 0.0												
TIME= 1.00												
TIME= 2.00												
TIME= 3.00												
TIME= 4.00												
TIME= 5.00												
TIME= 6.00												
TIME= 7.00												
	A	0	LX	1	0	0	135	000	135	2	2	210	1
	C	2	LX	31	0	0	130	000	130	4	2	210	1
	E	4	LX	30	0	0	128	000	128	6	2	210	1
	G	6	LAT	9	0	31	136	000	136	8	15	211	1
TIME= 8.00													
	I	8	LAT	7	0	30	136	000	136	10	15	211	2
	K	10	LAT	5	0	31	196	000	196	12	15	211	2
	M	12	LAT	1	0	0	132	000	132	14	15	211	2
	O	14	LAT	2	0	0	126	000	126	16	15	211	2
TIME= 9.00													
	Q	16	LAT	3	0	30	196	000	196	18	15	211	3
	S	18	MXK	1	1	0	30	000	58	20	77	210	3
	U	20	MXK	4	31	0	30	000	30	22	77	210	3
	W	22	MXK	3	30	0	30	000	30	24	77	210	3
TIME= 10.00													
	Y	24	AXK	5	31	0	30	000	30	26	76	210	4
	A	26	LAT	8	0	4	78	000	78	28	15	211	4
	C	28	LAT	4	0	4	80	000	80	30	15	211	4
	E	30	AXK	2	30	0	30	000	30	32	76	210	4
TIME= 11.00													
	G	32	LAT	10	0	3	78	000	78	34	15	211	5
	I	34	LAT	6	0	3	80	000	80	36	15	211	5
	K	36	AXK	3	3	0	1	000	1	38	76	210	5
	M	38	AXK	4	4	0	1	000	1	40	76	210	5
TIME= 12.00													
	O	40	AX	1	1	31	0	000	840	41	71	110	6
	P	41	MN	11	9	10	0	000	0	42	178	101	6
	Q	42	LAT	9	0	5	196	000	256	44	15	211	6
	S	44	LAT	10	0	3	80	000	82	46	15	211	6
	U	46	AXK	5	5	0	60	000	90	48	76	210	6
TIME= 13.00													
TIME= 14.00													
	W	48	MN	12	7	8	0	000	0	49	178	101	7
	X	49	LAT	7	0	2	196	000	256	51	15	211	7
	Z	51	LAT	8	0	4	80	000	82	53	15	211	7
	B	53	CGEX	1	1	5	0	000	930	54	87	110	7
	C	54	BAND	1	1	0	41	100	881	56	139	210	7
TIME= 15.00													
	E	56	MN	13	5	6	0	100	90	57	178	101	8
	F	57	LAT	5	0	5	136	100	316	59	15	211	8
	H	59	LAT	6	0	3	82	100	84	61	15	211	8
	J	61	AN	2	12	2	0	100	30	62	166	101	8
	K	62	AN	1	11	1	0	100	840	63	166	101	8
	L	63	MN	14	3	4	0	100	2	64	178	101	8
TIME= 16.00													
	M	64	LAT	3	0	2	256	100	316	66	15	211	9
	O	66	LAT	4	0	4	82	100	84	68	15	211	9
	Q	68	AN	2	14	2	0	100	30	69	166	101	9
	R	69	AN	1	13	1	0	100	840	70	166	101	9
	S	70	AXK	4	4	0	2	100	3	72	76	210	9
TIME= 17.00													
TIME= 18.00													
TIME= 19.00													
TIME= 20.00													

J. Conway Archives

L. Conway Archives

TIME= 22.00
TIME= 23.00
TIME= 24.00
TIME= 25.00
TIME= 26.00
TIME= 27.00
TIME= 28.00
TIME= 29.00
TIME= 30.00
TIME= 31.00
TIME= 32.00

U	72	AXK	3	3	0	2	100	3	74	76	210	A
W	74	AXK	2	2	0	60	100	90	76	76	210	A
Y	76	EXIT	0	0	0	0	100	0	41	199	111	A
P	41	MN	11	9	10	0	000	0	42	178	101	A

TIME= 33.00
TIME= 34.00

Q	42	LAT	9	0	5	196	000	376	44	15	211	6
S	44	LAT	10	0	3	80	000	86	46	15	211	6
U	46	AXK	5	5	0	60	000	150	48	76	210	6

TIME= 35.00

W	48	MN	12	7	8	0	000	0	49	178	101	7
X	49	LAT	7	0	2	196	000	376	51	15	211	7
Z	51	LAT	8	0	4	80	000	86	53	15	211	7
B	53	CGEX	1	1	5	0	000	990	54	87	110	7
C	54	BAND	1	1	0	41	100	881	56	139	210	7

TIME= 36.00
TIME= 37.00

E	56	MN	13	5	6	0	100	150	57	178	101	8
F	57	LAT	5	0	5	136	100	436	59	15	211	8
H	59	LAT	6	0	3	82	100	88	61	15	211	8
J	61	AN	2	12	2	0	100	90	62	166	101	8
K	62	AN	1	11	1	0	100	840	63	166	101	8
L	63	MN	14	3	4	0	100	6	64	178	101	8

TIME= 38.00

M	64	LAT	3	0	2	256	100	436	66	15	211	9
O	66	LAT	4	0	4	82	100	88	68	15	211	9
Q	68	AN	2	14	2	0	100	90	69	166	101	9
R	69	AN	1	13	1	0	100	840	70	166	101	9
S	70	AXK	4	4	0	2	100	5	72	76	210	9

TIME= 39.00
TIME= 40.00

U	72	AXK	3	3	0	2	100	5	74	76	210	A
W	74	AXK	2	2	0	60	100	150	76	76	210	A
Y	76	EXIT	0	0	0	0	100	0	41	199	111	A
P	41	MN	11	9	10	0	000	0	42	178	101	A

TIME= 41.00
TIME= 42.00
TIME= 43.00

Q	42	LAT	9	0	5	196	000	496	44	15	211	6
S	44	LAT	10	0	3	80	000	90	46	15	211	6
U	46	AXK	5	5	0	60	000	210	48	76	210	6
W	48	MN	12	7	8	0	000	0	49	178	101	7
X	49	LAT	7	0	2	196	000	496	51	15	211	7
Z	51	LAT	8	0	4	80	000	90	53	15	211	7
B	53	CGEX	1	1	5	0	000	1050	54	87	110	7
C	54	BAND	1	1	0	41	100	881	56	139	210	7

TIME= 44.00

E	56	MN	13	5	6	0	100	210	57	178	101	8
F	57	LAT	5	0	5	136	100	556	59	15	211	8
H	59	LAT	6	0	3	82	100	92	61	15	211	8
J	61	AN	2	12	2	0	100	150	62	166	101	8
K	62	AN	1	11	1	0	100	840	63	166	101	8
L	63	MN	14	3	4	0	100	10	64	178	101	8

TIME= 45.00

	M	64	LAT	3	0	2	256	100	556	66	15	211	9
	O	66	LAT	4	0	4	82	100	92	68	15	211	9
	Q	68	AN	2	14	2	0	100	150	69	166	101	9
	R	69	AN	1	13	1	0	100	640	70	166	101	9
	S	70	AXK	4	4	0	2	100	7	72	76	210	9

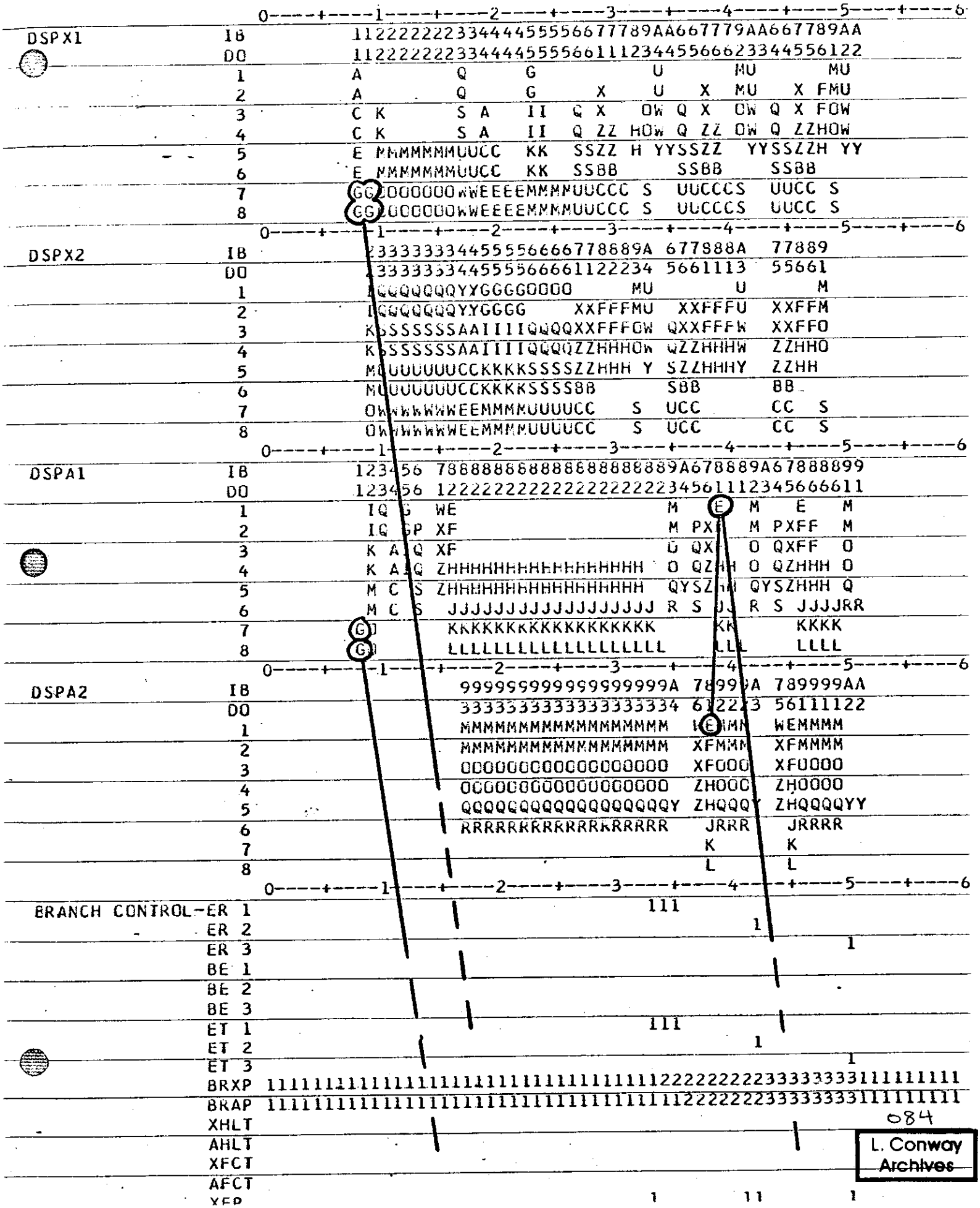
TIME= 46.00
TIME= 47.00
TIME= 48.00

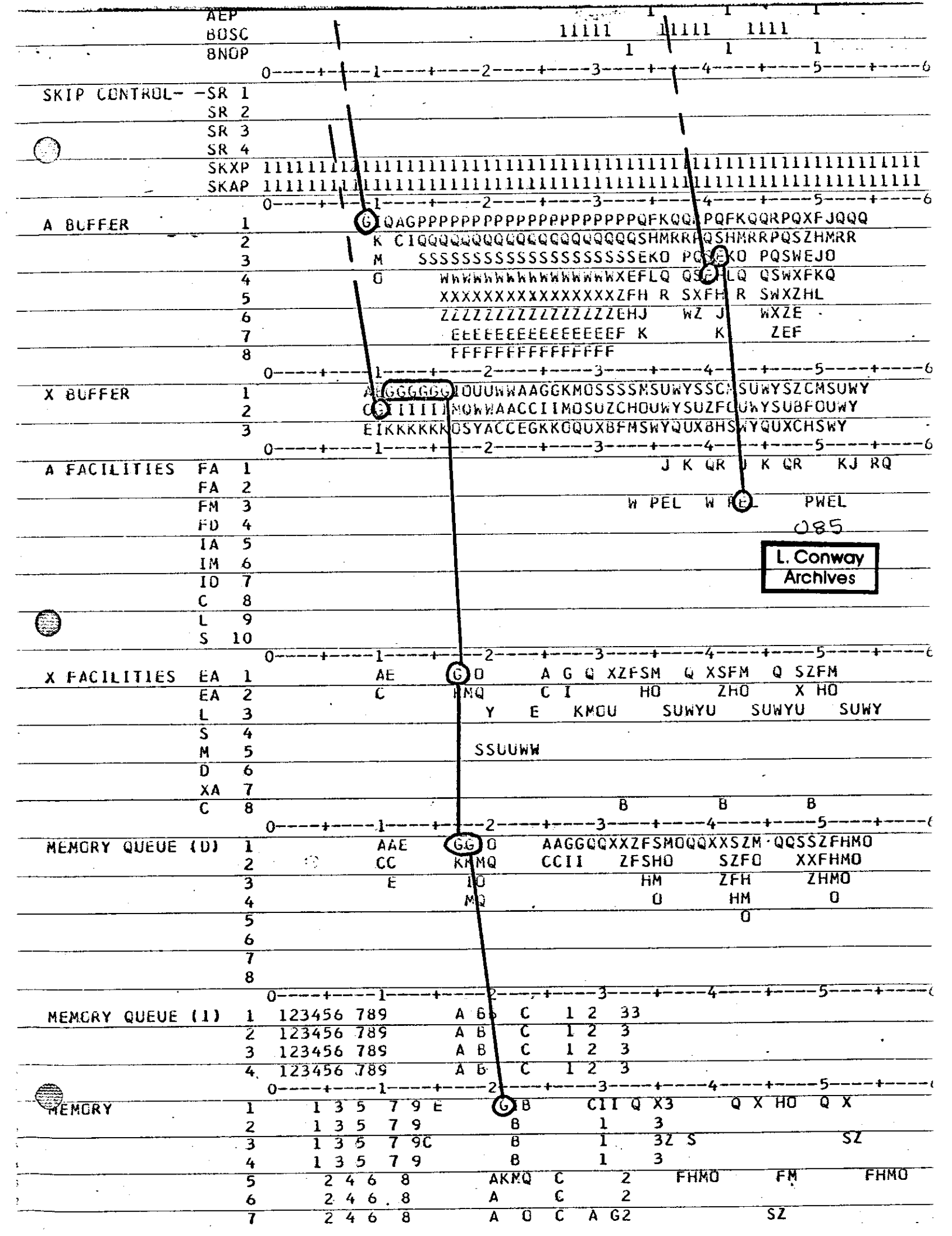
U	72	AXK	3	3	0	2	100	7	74	76	210	A
W	74	AXK	2	2	0	60	100	210	76	76	210	A
Y	76	EXIT	0	0	0	0	100	0	41	199	111	A
	0		0	0	0	0	000	0	0	999	000	A

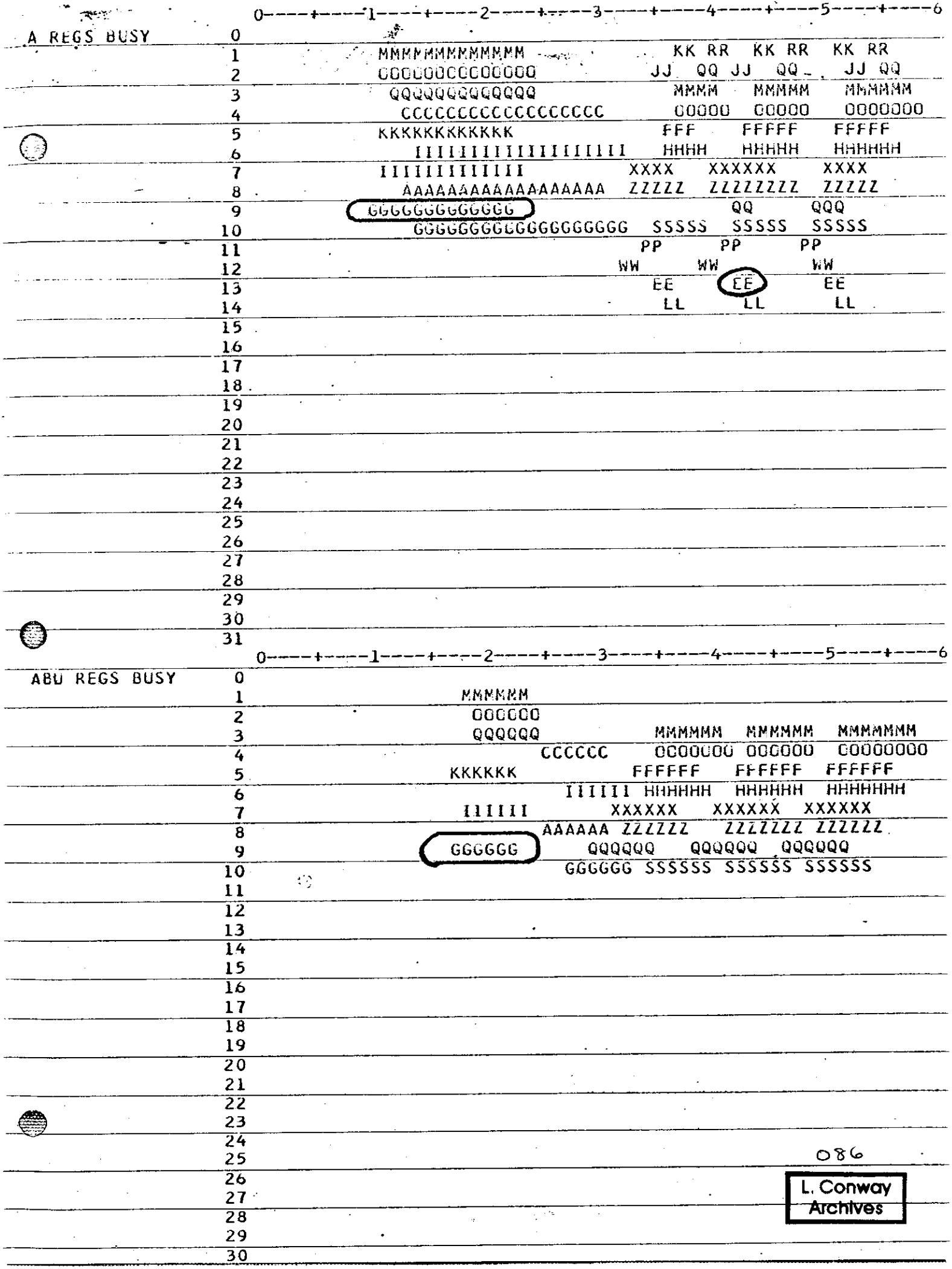
TIME= 49.00
TIME= 50.00
TIME= 51.00
TIME= 52.00
TIME= 53.00
TIME= 54.00
TIME= 55.00
TIME= 56.00
TIME= 57.00
TIME= 58.00
TIME= 59.00

L. Conway Archives

SIMULATED TIME = 0 TO 58 INPUT PROGRAM = CR-FS

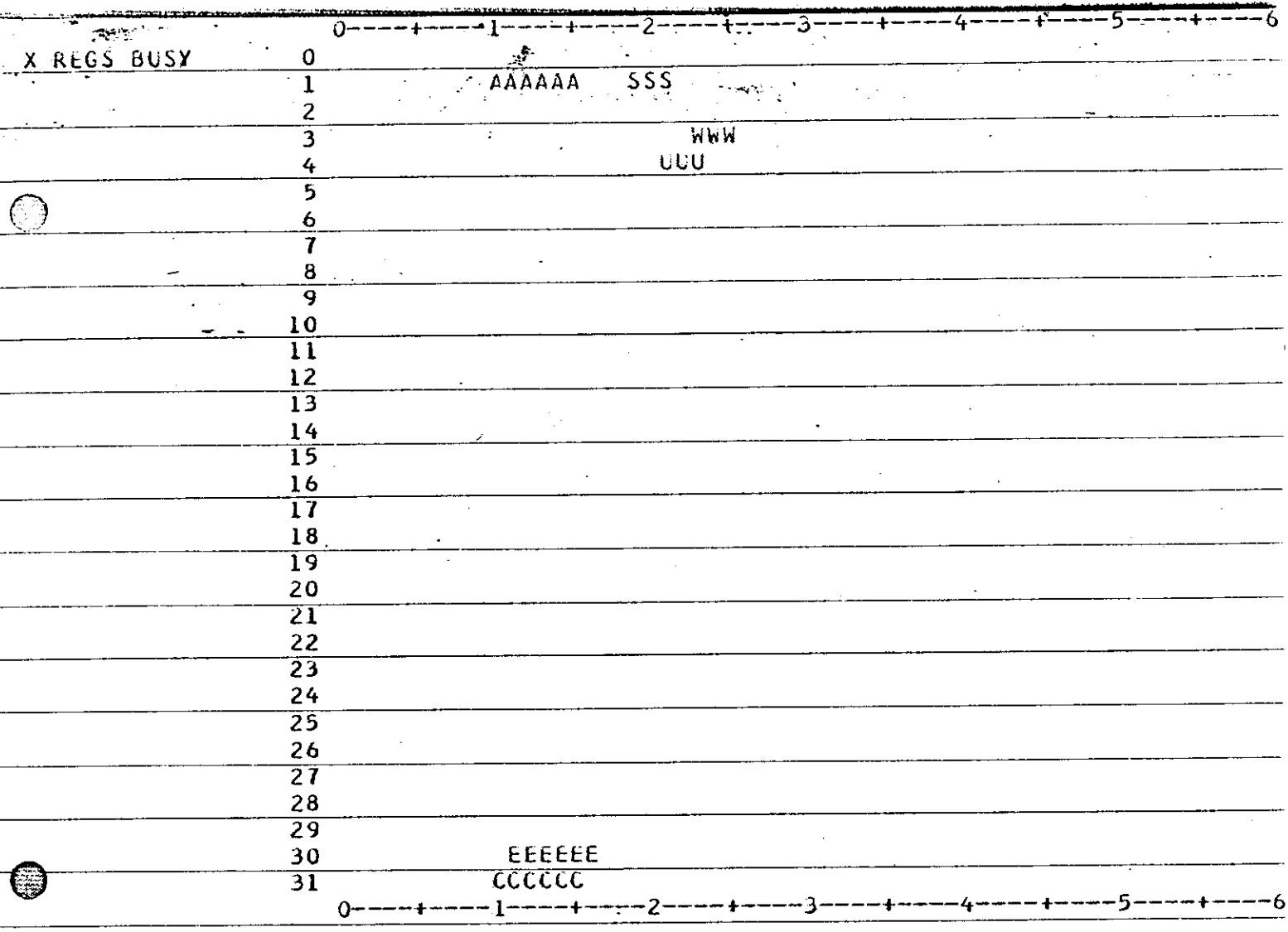



AEP
BOSC
BNOP
SKIP CONTROL- -SR 1
SR 2
SR 3
SR 4
SKXP
SKAP
A BUFFER
X BUFFER
A FACILITIES
FA 1
FA 2
FM 3
FD 4
IA 5
IM 6
ID 7
C 8
L 9
S 10
X FACILITIES
EA 1
EA 2
L 3
S 4
M 5
D 6
XA 7
C 8
MEMORY QUEUE (D)
MEMORY QUEUE (1)
MEMORY
085
L. Conway Archives



086

L. Conway Archives



087
L. Conway Archives

CURRENT JOB RUNNING PROCEDURES

This section describes the procedures to be followed in order to use the timing simulation program. These procedures are to be completely revised and expanded in the near future so that the programs may be stored on disk at the MOD 75 comp lab and users may submit runs directly at the comp lab (see Section 5).

To use the timing simulator at the present time:

(i) Write the assembly code input program for the Unroller (Section 1).

(ii) Prepare the machine parameter card required for the Timer input deck (Section 2).

(iii) Submit these items to L. Conway, Room 203, Extension 252.

088

L. Conway Archives

TABLE OF IMPLEMENTED INSTRUCTIONS

The table on the following pages lists the ACS-1 instruction set op-codes and indicates (with an X) if a given op is implemented in the Timing Simulator.

L. Conway Archives

OP		OP		OP		OP	
ACH	X	CEQXK	X	EQA	X	LD	
ACL	X	CGED		EQC	X	LDA	
ADN		CGEI	X	EQX	X	LDH	
ADR		CGEN	X	EXIT	X	LDHAA	
ADU		CGEX	X	EXITA	X	LDHBA	
AI	X	CGEXK	X	EXITL	X	LDHCA	
AN	X	CMEQD		EXITP		LDHDA	
ANDA	X	CMEQN	X			LL	X
ANDC	X	CMGED				LMA	
ANDX	X	CMGEN	X	FAFA	X	LMS	
AR	X	CNTAA	X	FAFC	X	LMX	
AU	X	CNTAX	X	FAFX	X	LR	X
AX	X	CNTDA	X	FOFA	X	LX	X
AXC	X	CNTDX	X	FOFC	X	LXA	
AXK	X	CNTT	X	FOFX	X	LXC	
		CUGEI	X			LXCA	
		CUGEX	X			LXH	X
BAND	X	CUGEXK	X	HIO			
BEQ	X	CVF	X				
BFAF	X	CVI	X			MAX	
BFOF	X	CVN	X	IC		MCX	X
BOR	X	CVS	X	IDA		MDN	
BTAF	X			IFA	X	MDR	
BTOF	X			IFX	X	MDU	
BU		DDN		IFZA	X	MI	X
BXOR	X	DDR		IFZX	X	MKL	X
		DI	X	IR		MKP	
		DMI		ITUMA		MKR	X
CBA	X	DMN		ITUMP		MLC	X
CBMA		DMR		IVIB		MLX	X
CBMX		DN	X			MMI	
CBX	X	DR	X	LA	X	MMN	
CEQD		DRX	X	LAA		MMU	
CEQI	X	DRXK	X	LAH	X	MN	X
CEQN	X	DX	X	LAT	X	MOT	
CEQX	X	DXK	X	LATH	X	MR	X

L. Conway Archives

OP		OP		OP		OP	
MRC	X	SCAN		SPF	X	TAFA	X
MSX		SCH	X	SPI	X	TAFC	X
MSXZ		SCL	X	SPX	X	TAFX	X
MTX		SDN		SR	X	TCH	
MU	X	SDR		STA	X	TOFA	X
MX	X	SDU		STAA		TOFC	X
MXA	X	SHA	X	STAH	X	TOFX	X
MXC	X	SHAC	X	STAT	X		
MXK	X	SHD		STATH	X		
MXP		SHDC		STD		XORA	X
MXS		SHDX		STDH		XORC	X
MXSO		SHDXC		STDHAA		XORX	X
MXT		SHX	X	STDHBA			
MZT		SHXC	X	STDHCA			
		SI	X	STDHDA			
		SIA	X	STL	X		
NOP	X	SIAC	X	STMA			
		SID		STMS			
		SIDC		STMX			
ORA	X	SIO		STMZ			
ORC	X	SIX	X	STMZA			
ORX	X	SIXC	X	STR	X		
		SKAND	X	STX	X		
		SKEQ	X	STXA			
PAUSE		SKFAF	X	STXH	X		
PI		SKFOF	X	SU	X		
		SKOR	X	SVC			
		SKTAF	X	SVR			
RND		SKTOF	X	SWA	X		
RX	X	SKXOR	X	SWX	X		
RXK	X	SN	X	SX	X		
		SNF	X				
		SNI	X				
		SNX	X				

L. Conway Archives

PLANNED MODIFICATIONS

Certain modifications to the simulation programs are now being made or are planned for the near future. These are briefly described below to assist users in their planning. Updates to this memo will be issued as these changes are included in the programs.

Unroller Changes

The control specification facilities will be extended.

Timing Simulator Changes

(i) Additional OPS will be implemented.
(ii) New output features and options will be added.

Job Running Procedure Changes

Currently jobs must be submitted to L. Conway who will handle the processing of the jobs. Two separate programs must be run consecutively to process one timing simulation. This results in a rather long overall turn-around time. To improve on this, the two programs will be merged, with the trace temporarily stored in core or on disk and automatically passed between them.

Also, the program will be placed on disk at the MOD 75 comp lab. The running of jobs will then be handled directly by the user, who will submit the assembly code input deck, parameter card, and appropriate JCL cards to call for the timing simulator.

These changes will greatly reduce over-all turn-around time and allow a much greater number of users to be served than is now possible.

L. Conway Archives

A UNIT INTLK SIMULATION:

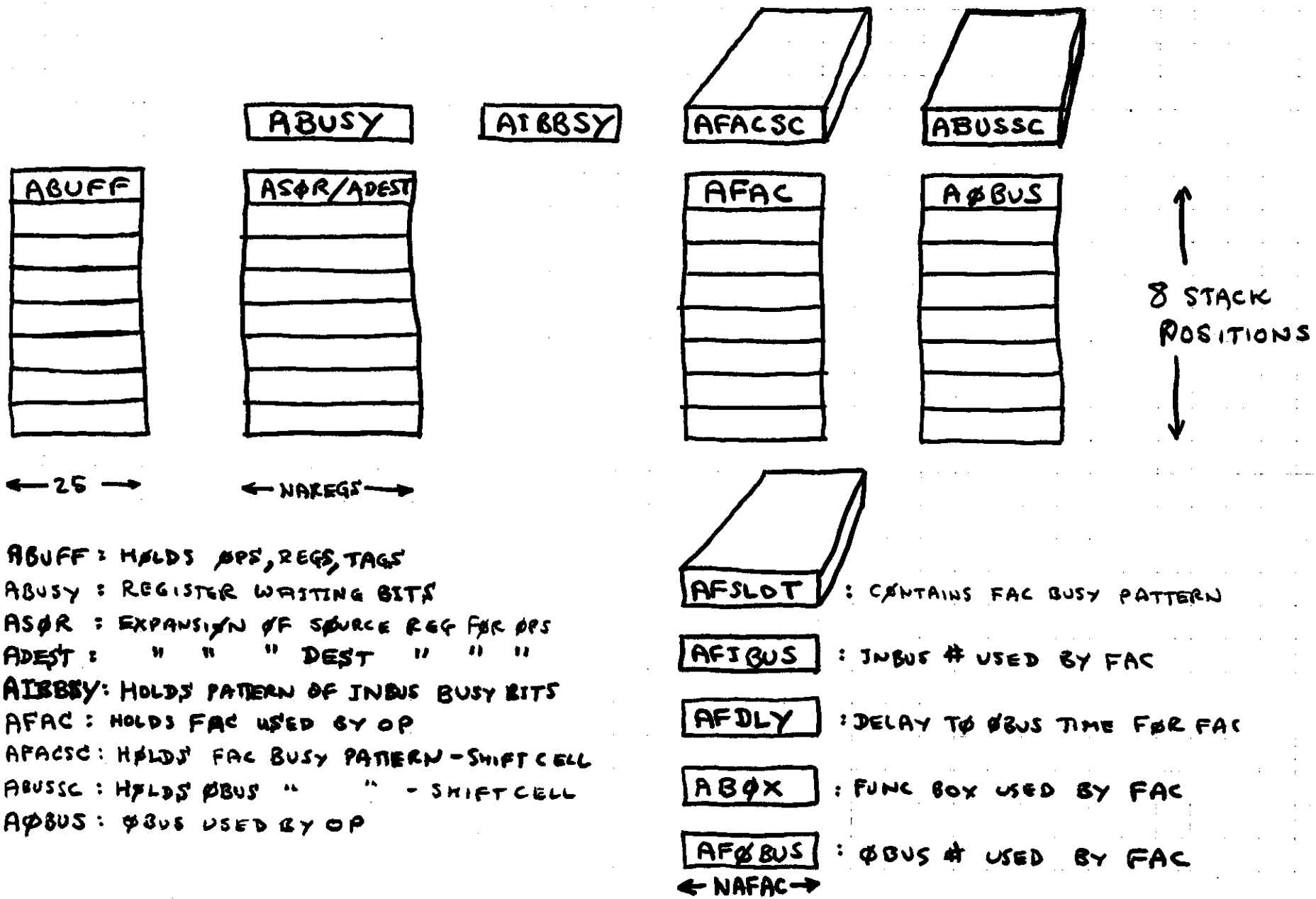
ENCLOSED IS A SAMPLE OF THE SORT OF CODE USED IN THE TIMING SIMULATOR. THE CODE IS CONDENSED FROM ACTUAL SIMULATOR CODE AND DESCRIBES THE BARE ESSENTIALS OF THE A UNIT INTLKS OF A POSSIBLE MACHINE DESIGN SIMILAR TO ACS. IT SHOULD SERVE AS A GUIDE TO HOW ONE MIGHT CODE A SIMULATION. IT DOES NOT DESCRIBE THE INTLKS COMPLETELY NOR IS THE MODEL USED MEANT TO REALLY MODEL ACS.

ENCLOSED ARE:

093

FEB 15 1968

L. Conway Archives

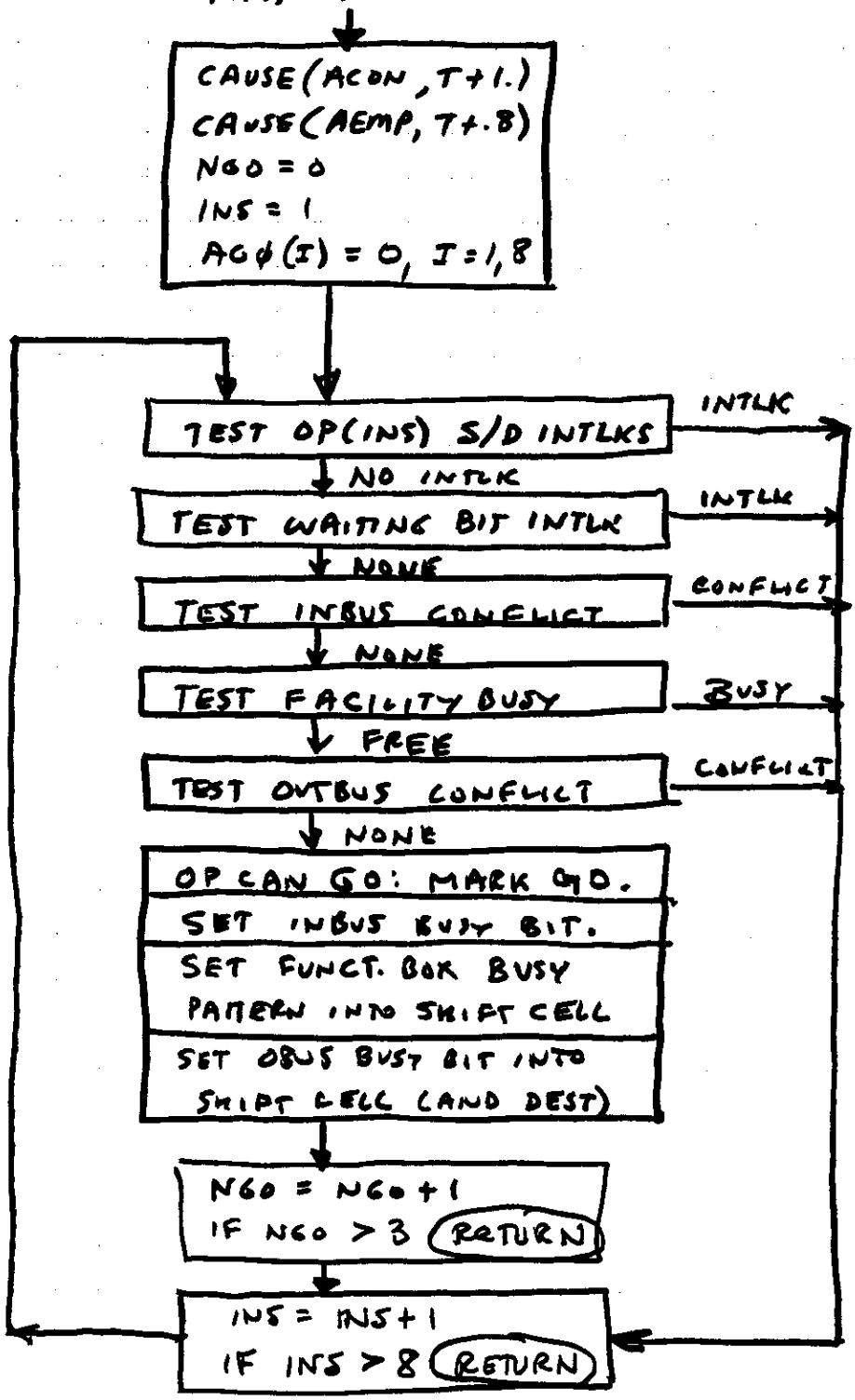


ABUFF : HOLDS OPS, REGS, TAGS
ABUSY : REGISTER WRITING BITS
AS$R : EXPANSION OF SOURCE REG FOR OPS
ADEST : " " " DEST " " "
AIBBSY: HOLDS PATTERN OF INBUS BUSY BITS
AFAC : HOLDS FAC USED BY OP
AFACSC : HOLDS FAC BUSY PATTERN - SHIFT CELL
ABUSSC : HOLDS $BUS " " - SHIFT CELL
A$BUS : $BUS USED BY OP

L. Conway Archives

SUBROUTINE XACON : SIMPLIFIED FLOWCHART

THIS ROUTINE SCANS THE STACK FOR OPS WHICH CAN GO - I.E. PASS ALL INTLK TEST. IT MARKS SUCH OP "GO" AND SETS CERTAIN BUSY AND SHIFT CELL PATTERNS.

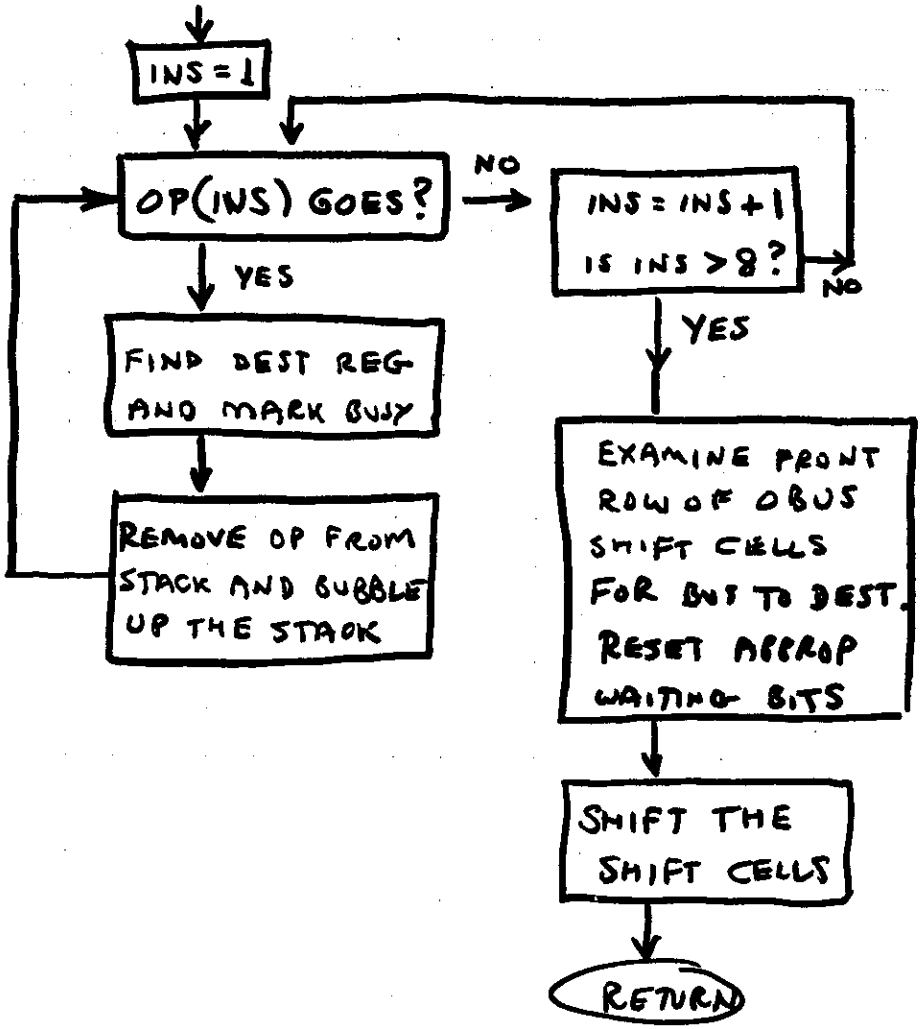


L. Conway Archives

SUBROUTINE XAEMP : SIMPLIFIED FLOWCHART:

THIS ROUTINE SCANS STACK FOR OPS WHICH CAN GO. IT ISSUES OPS BY SETTING APPROPRIATE WAITING BITS AND REMOVING OP FROM THE STACK

The routine also resets waiting bits when shift cells indicate outbussing to registers. Then shifts the shift cells.



L. Conway Archives

```
      SUBROUTINE XACØN
      CØMMØN ---
      CALL CAUSE (ACØN, TIME+1.0, 0, 0, 0)
      CALL CAUSE (AEMP, TIME+0.8, 0, 0, 0)
      DØ 1 I=1,8
    1 AGØ(I)=0
      NGØ=0
C  SCAN THE A CONTENDER STACK FOR INS=1 TO 8 TO FIND OPS
C  WHICH CAN GO - MARK THEM WITH AGØ(INS)=1.
      DØ 100 INS=1,8
      IF (AFULL(INS).EQ.0) GØ TØ 100
      IF (INS.EQ.1) GØ TØ 21
      INSM1 = INS-1
C  TEST THE SOURCE-DEST INTERLOCKS
      DØ 20 I=1, INSM1
      DØ 20 REG=1, NAREGS
      IF ((ASØR(INS,REG).EQ.1).AND.(ADEST(I,REG).EQ.1)) GØ TØ 100
      IF ((ADEST(INS,REG).EQ.1).AND.(ASØR(I,REG).EQ.1)) GØ TØ 100
      IF ((ADEST(INS,REG).EQ.1).AND.(ADEST(I,REG).EQ.1)) GØ TØ 100
   20 CØNTINUE
   21 CØNTINUE
C  TEST WAITING BIT INTERLOCK
      DØ 22 REG=1, NAREGS
      IF ((ASØR(INS,REG).EQ.1).AND.(ABUSY(REG).EQ.1)) GØ TØ 100
      IF ((ADEST(INS,REG).EQ.1).AND.(ABUSY(REG).EQ.1)) GØ TØ 100
   22 CØNTINUE
C  FIND FACILITY USED BY OP (INS)
      DØ 25 FAC=1, NAFAC
      IF(AFAC(INS,FAC).NE.0) GØ TØ 26
   25 CØNTINUE
   26 CONTINUE
C  SEE IF INBUS REQ'D BY FAC IS BUSY
      INBUS = AFIBUS(FAC)
      IF (AIBBSY(INBUS).EQ.1) GØ TØ 100
C  SEE IF FACILITY BOX REQ.D IS BUSY
      BØX = ABØX(FAC)
      IF (ABXBSY(BØX).EQ.1) GØ TØ 100
```


L. Conway Archives

(CONT.)

```
C     TEST FAC BUSY REQUIREMENTS AGAINST SHIFT CELL CONTENTS
C     WHICH INDICATE BUSY CONDITIONS SET BY PRIOR OPS.
          DØ 30 T=1,NSLØT
          IF((AFSLØT(FAC,T).EQ.1).AND.(AFACSC(FAC,T).EQ.1)) GØ TØ 100
       30 CONTINUE
C     TEST FOR OUTBUS CONFLICTS BY COMPARING BUS/TIME REQD
C     AGAINST CONTENTS OF SHIFT CELLS
          ØBUS = AFØBUS(FAC)
          DELAY = AFDLY(FAC)
          IF((AØBUS(INS,ØBUS).NE.0).AND.(ABUSSC(DBUS,DELAY).NE.0))
         X  GØ TØ 100
C     IF REACH THIS POINT, ALL TESTS PASSED, AND OP CAN GØ
C     SO MARK GO, AND SET APPROP. PATTERNS IN SHIFT CELLS.
          AIBBSY(INBUS)=1
          ABXBSY(INBUS)=1
          DØ 32 T=1,NSLØT
          IF(AFSLØT(FAC,T).EQ.0) AFACSC(FAC,T)=1
       32 CONTINUE
          ABUSSC(ØBUS,DELAY) = AØBUS(INS,ØBUS)
          AGØ(INS)=1
          NGØ=NGØ+1
          IF(NGØ.EQ.NAGØ) RETURN
      100 CONTINUE
          RETURN
          END
```


L. Conway Archives

SUBROUTINE XAEMP
COMMON ---

```
C   THIS ROUTINE FIRST SCANS THE A CONTENDER STACK FOR
C   OPS MARKED GO. IT ISSUES THE OPS BY SETTING THE
C   WAITING BITS AND REMOVING THE OP FROM THE STACK.
C
      DO 100 INS = 1,8
    5 IF(AGO(INS).EQ.0) GO TO 100
      ~~IF(AFULL(INS).EQ.0) GO TO 100~~
C   OP(INS) GOES - FIRST SET REG WAITING BITS
      DO 10 REG = 1,NAREGS
      IF(ADEST(INS,REG).NE.1) GO TO 10
      ABUSY(REG) = 1
   10 CONTINUE
C   NOW REMOVE THE OP FROM THE STACK AND BUBBLE UP STACK.
      AINPT = AINPT-1
      IF(INS.EQ.8) GO TO 31
      DO 30 I = INS,7
      AGO(I) = AGO(I+1)
      AFULL(I) = AFULL(I+1)
      DO 25 J = 1,25
   25 ABUFF(I,J) = ABUFF(I+1,J)
      DO 26 J = 1,NAREGS
      ASOR(I,J) = ASOR(I+1,J)
   26 ADEST(I,J) = ADEST(I+1,J)
      DO 27 FAC = 1,NAFAC
   27 AFAC(I,FAC) = AFAC(I+1,FAC)
      DO 28 BUS = 1,NABUS
   28 AOBUS(I,BUS) = AOBUS(I+1,BUS)
   30 CONTINUE
   31 AGO(8) = 0
      AFULL(8) = 0
      DO 125 J = 1,25
  125 ABUFF(8,J) = [illegible]
      DO 126 J = 1,NAREGS
      ADEST(8,J) = 0
  126 ASOR(8,J) = 0
```

099

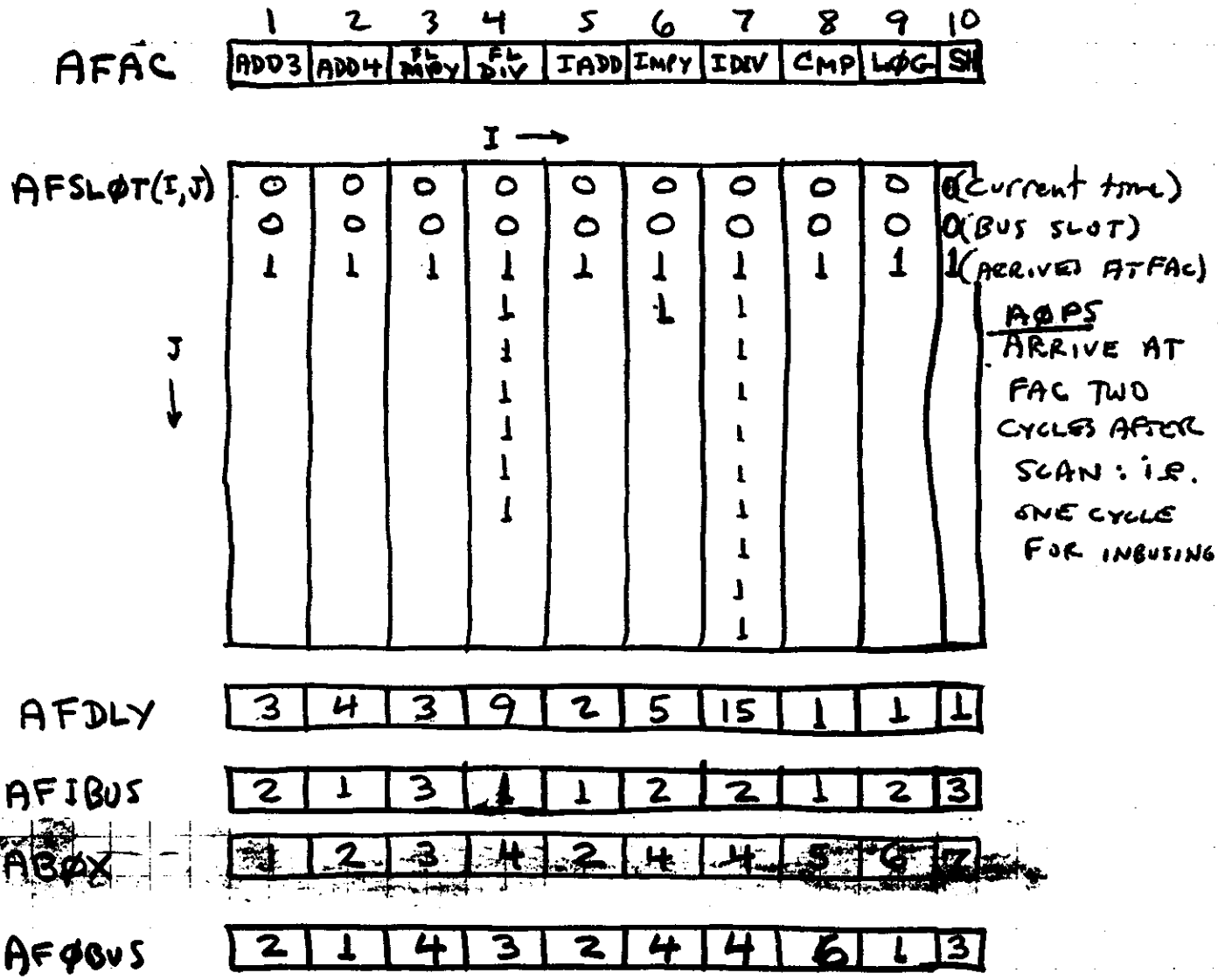
L. Conway Archives

(CONT.)

```
      DØ 127 FAC=1,NAFAC
  127 AFAC(8,FAC)=0
      DØ 128 BUS=1,NABUS
  128 AØBUS(8,BUS)=0
      GØ TØ 5
  100 CØNTINUE
C
C   THE NEXT FUNCTIØN ØF THE RØUTINE IS TØ CØNTRØL THE
C   WAITING BITS - RESET WHEN INDICATED BY SHIFT CELL
C   ENTRIES, THEN SHIFT THE SHIFT CELLS.
      DØ 210 BUS=1,NABUS
      DEST=ABUSSC(BUS,1)
      ABUSY(DEST)=0
  210 CØNTINUE
C   NOW SHIFT THE SHIFT CELLS
      DØ 299 I=1,10
      ABXBSY(I)=0
  299 AIBSSY(I)=0
      SLØTM1=NSLØT-1
      DØ 301 J=1,10
      DØ 300 SLØT=1,SLØTM1
  300 ABUSSC(J,SLØT)=ABUSSC(J,SLØT+1)
  301 ABUSSC(J,NSLØT)=0
      DØ 303 J=1,NAFAC
      DØ 302 SLØT=1,SLØTM1
  302 AFACSC(J,SLØT)=AFACSC(J,SLØT+1)
  303 AFASSC(J,NSLØT)=0
      RETURN
      END
```


L. Conway Archives

A FACILITIES, BUSES



L. Conway Archives

DOUBLE +,*,÷ CMP

CMP ~ INT +~ | LOGIC ~ INT *,÷ ~ | STORE ~ SHIFT ~

SP FL +4 | SP FL +3 | SP/MP FL *~

INT *,÷ SHIFT L SP FL * | SHIFT H SP FL ÷ | DP FL +4 INT +3 SP FL | LOGIC SP FL +4

No two ops using same reg return bus share same delay so no conditional interlock is involved

H L

B1 (1) B2 B3 (2) B4 B5 (3) B6

B7 B8 B9 B10

2K
(5) CMP (A) 1/1 SP 2/1 DP
(6) = To CBU

9K
B7A SP, DP (2) SP, DP INT + (A)
B8A I, DP

15K
SP, DP (4) DP FL +3/4 INT *,÷ SP/DP FL ÷ INT ÷ (B)
B9A
B10A I, DP

2K
(6) LOGIC (C)
B7B H

5K
(1) SP FL +3 (C)
B8B SP

3K
(7) SHIFT (C) & SWA
B9B H
B10B L

16K
(3) SP/MP FL *~ (D)
B10C SP, MP
B9C MP

(A) --- (D) ~ 5X5 MODULES

(MXD DIV USES SAME INTLKS AS DP DIV)

MHH 2/15/67

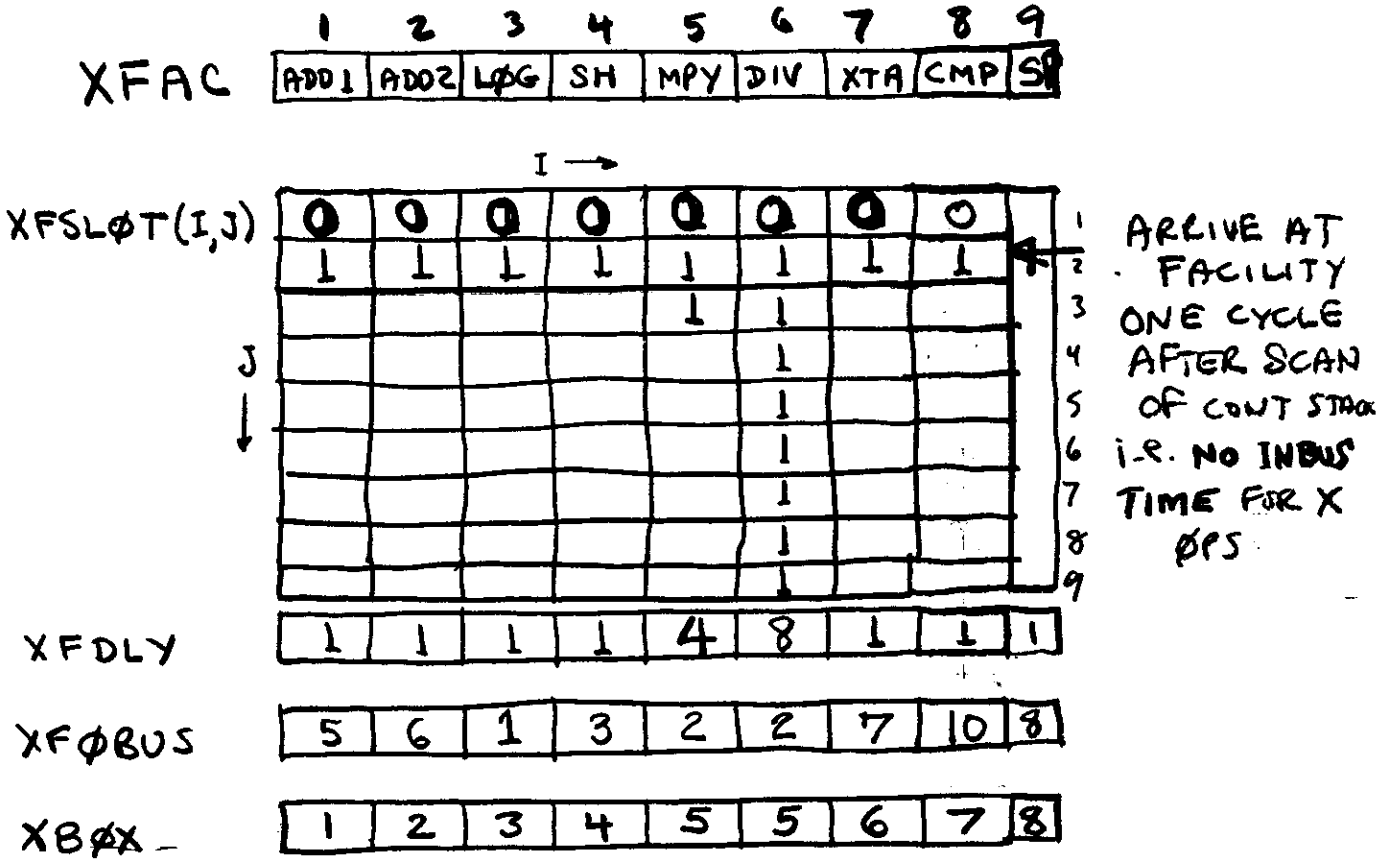
(3) STORE & A TO X

102

L. Conway Archives

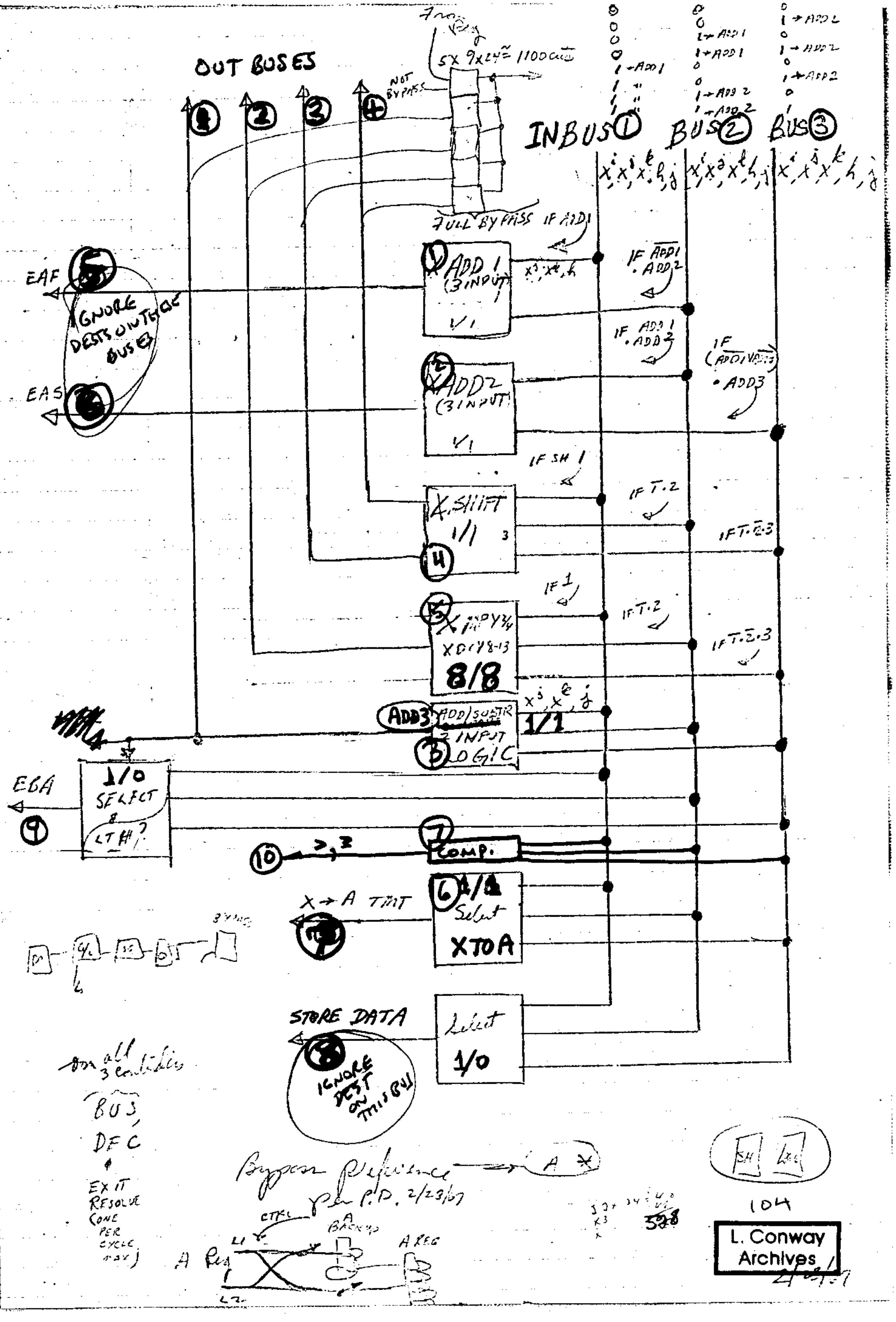
X FACILITIES, BUSES

Note: No Inbus Conflicts. Assume all inbuses to all functional units.



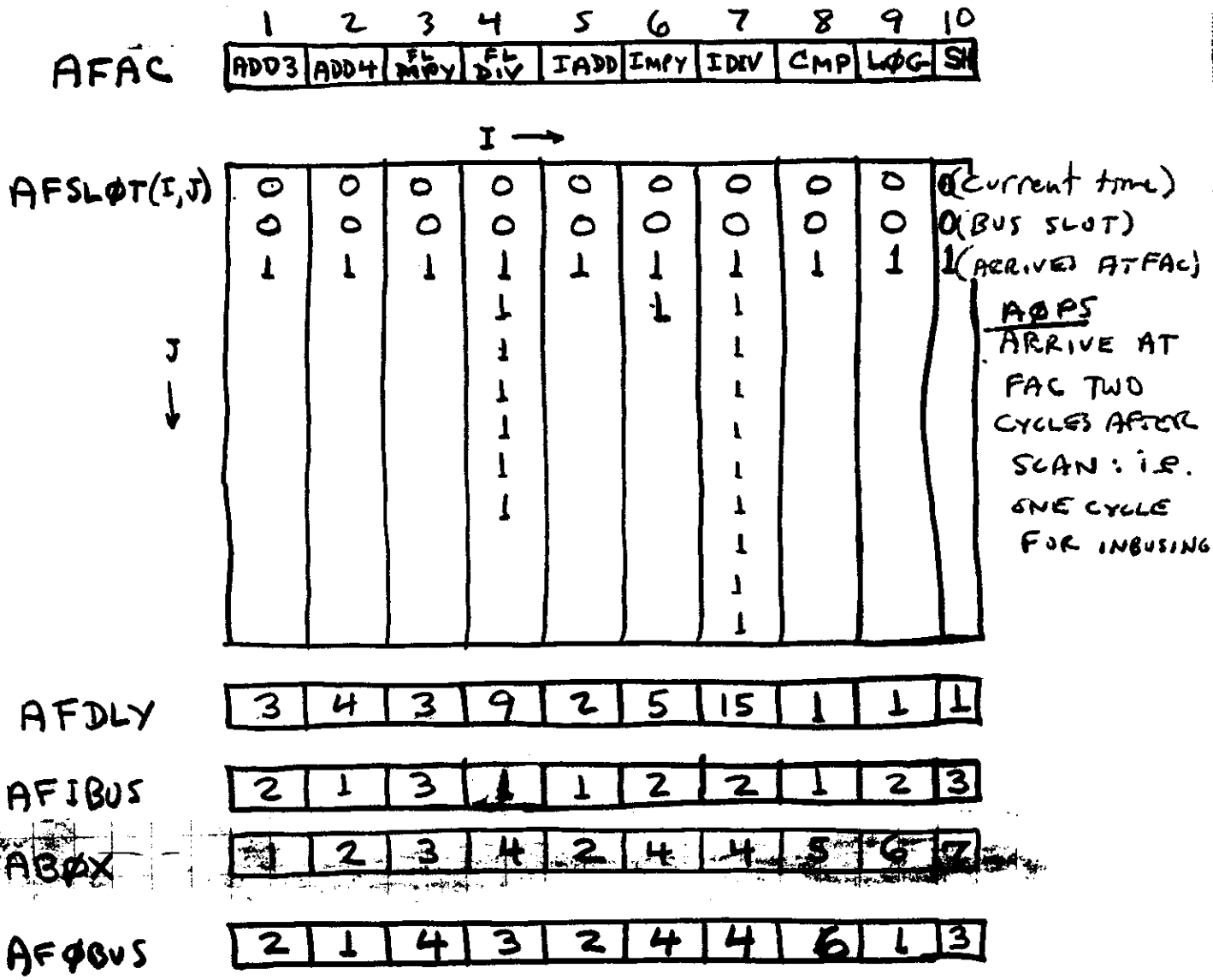
NØTE: LØG contains ADD, SUB, [redacted], LØG, etc.

L. Conway Archives

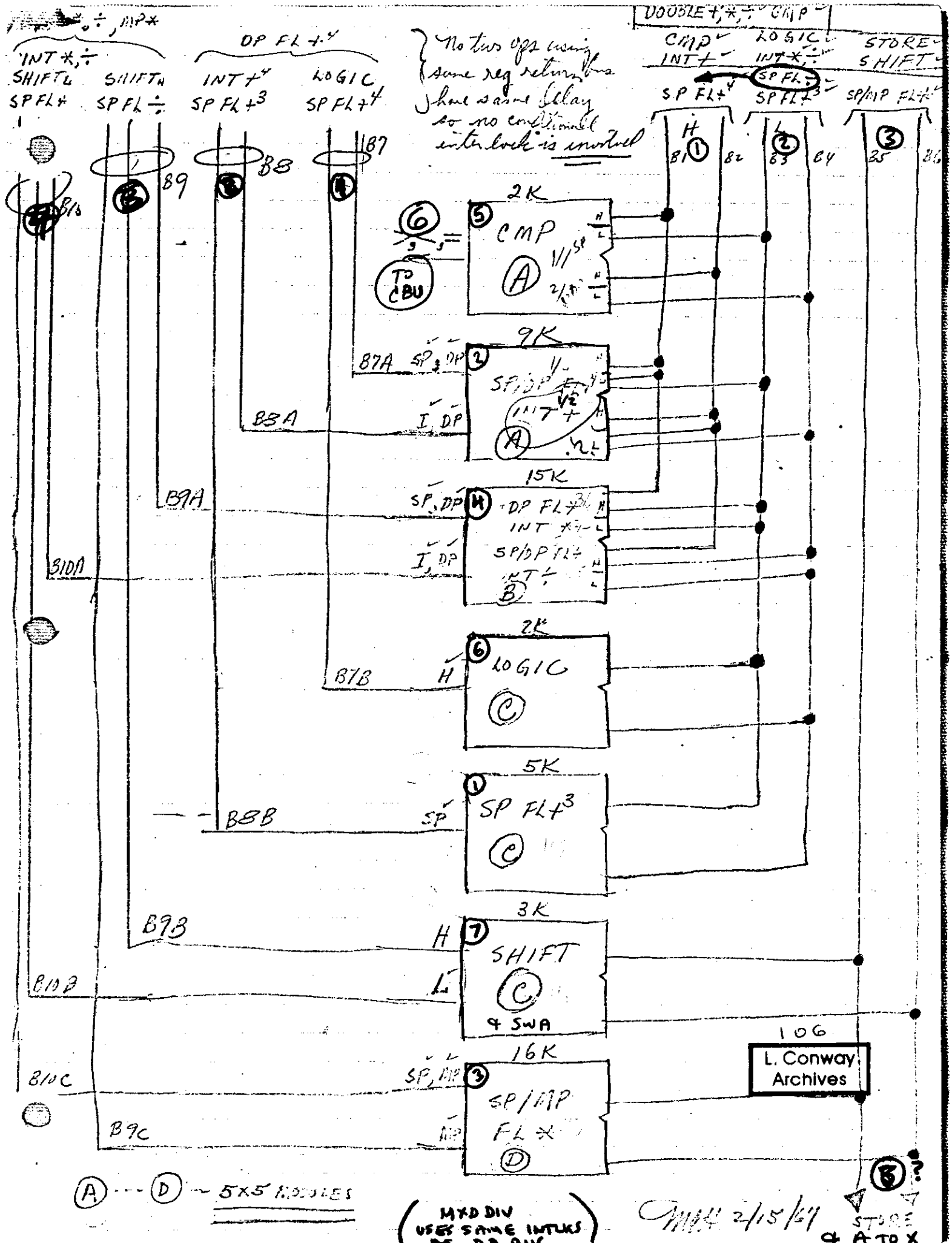


L. Conway Archives

A FACILITIES, BUSES



L. Conway Archives


DOUBLE +, *, ÷ CMP
CMP ← INT +, −
LOGIC ← INT *, ÷
STORE ← SHIFT
SP FL +4
SP FL +3
SP/MP FL +4
INT *, ÷
SHIFT L
SHIFT R
SP FL ÷
DP FL +4
INT +4
SP FL +3
LOGIC
SP FL +4
No two ops using same reg returns have same delay so no conditional interlock is involved
H
L
B1
B2
B3
B4
B5
B6
B7
B8
B9
B10
2K
CMP
A
1/1 SP
2/1 DP
9K
SP/DP +, −
INT +, −
A
15K
DP FL +, ÷
INT *, ÷
SP/DP FL ÷
INT ÷
B
2K
LOGIC
C
5K
SP FL +3
C
3K
SHIFT
C
4 SWA
16K
SP/MP
FL *
D
B7A
B8A
B7B
B8B
B9A
B10A
B10B
B10C
B9C
SP, DP
I, DP
H
SP
L
SP, MP
MP
TP CBU
A --- D ~ 5X5 MODULES
(MXD DIV USES SAME INTLKS AS DP DIV)
STORE 4 A TO X
106
L. Conway Archives

FYI Ed. Sussenguth

LIST OF FUNCTIONAL UNITS & THEIR PERFORMANCE:

ARITHMETIC UNIT: (Ref. A-UNIT DATA FLOW)

SP CMP	1/1 ← RATE/DELAY		SINGLE PRECISION COMPARE
INT +	1/2		INTEGER ADD
DP CMP	1/4		DOUBLE PRECISION COMPARE
SP FL +	1/4		SINGLE " FL. ADD
DP FL +	1/4		DOUBLE " " "
SP FL ÷	7/10 OR 8/10	(ENG. GOAL 7/9)	SINGLE " FL. DIVIDE
MP FL ÷	" "	" " " "	MIXED " " "
DP FL ÷	14/17	(ENG. GOAL 14/16)	DOUBLE " " "
SP INT ÷	9/14	(ENG. GOAL 9/13)	SINGLE " INTEGER "
MP INT ÷	"	" " "	MIXED " " "
SP FL *	1/3		SINGLE " FL. MULTIPLY
MP FL *	1/3		MIXED " " "
DP FL *	3/5		DOUBLE " " "
SP INT *	2/4		SINGLE " INTEGER "
MP INT *	2/4		MIXED " " "
LOGIC	1/1		
SP FL +	1/3		SINGLE PRECISION FL. ADD
SHIFT	1/1		
SWAP	1/1		

707

L. Conway Archives

7/21/67 CWS

LIST OF FUNCTIONAL UNITS, CONTINUED.

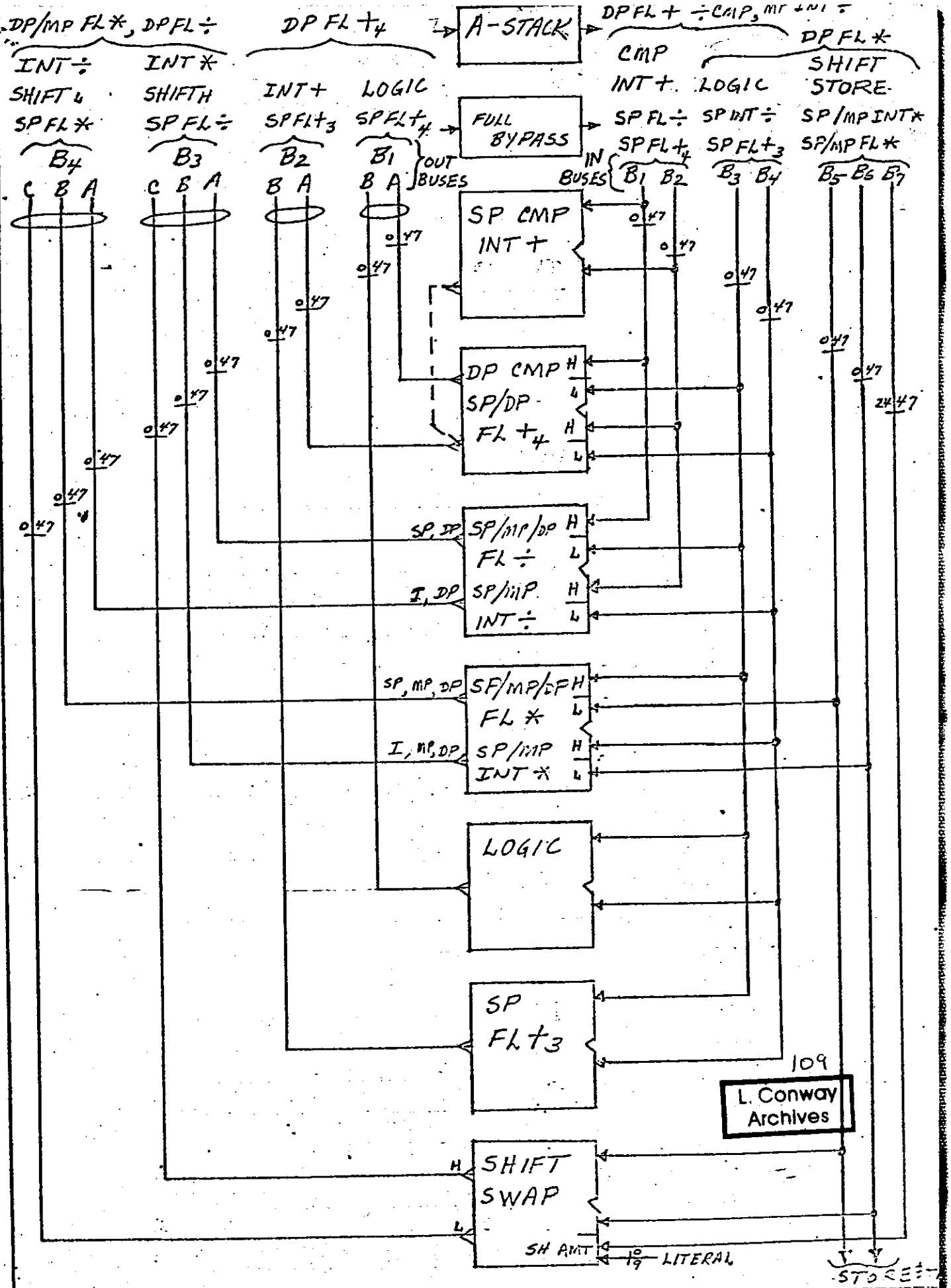
INDEX UNITS (Ref. X-UNIT DATA FLOW)

COMPARE	1/1	RATE/DELAY
SHIFT	1/1	
MULTIPLY	2/4.	
DIVIDE	8/8	AVERAGE TO 13/13 MAX. (DATA DEPENDENT)
ADD	1/1	(INCLUDING EBA ADD IF REQ)
SUBTRACT	1/1	
LOGIC	1/1	
EBA SELECT	1/1	(SELECT X^k+h or X^k).
ADD 1 (EA1)	1/1	
ADD 2 (EA2)	1/1	
MOVE SELECT	1/1	(TRANSMIT)
STORE SELECT	1/1	

L. Conway Archives

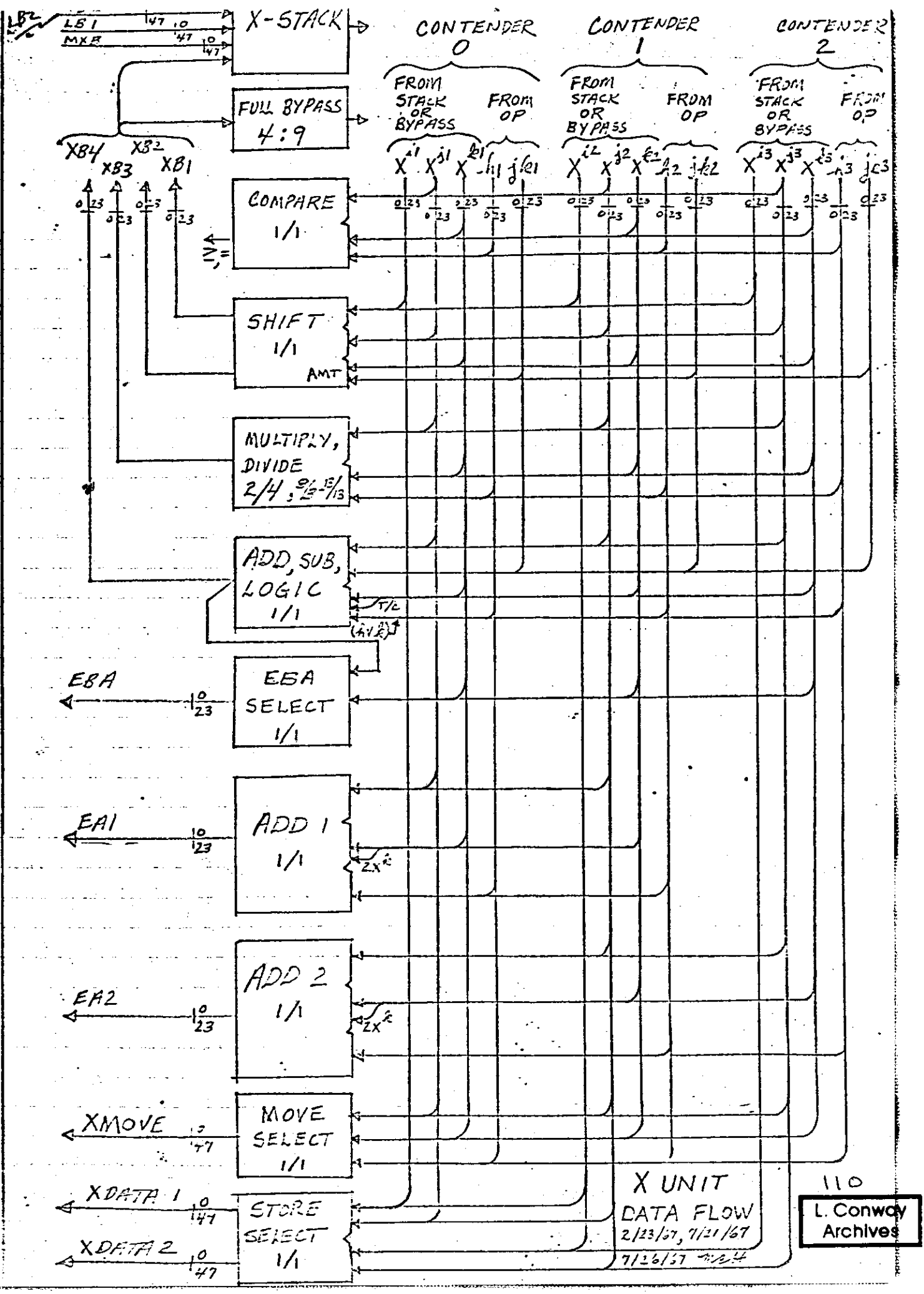
fig 2

7/21/67 MAC



A-UNIT DATA FLOW

MEH, GP 5/17/77
REV. 6/7/77



110

L. Conway Archives

COMMON/TAGS/D(256,70) (17 APRIL '67 UPSET)

		XΦP	AΦP	BΦP	A^J="S	A^J="A	A^{J+1}="S	A^{J+1}="A	A^J="S	A^J="A	A^{J+1}="S	A^K="S	A^{K+1}="S	XB^K="A	XB^{K+1}="A	CB^J="A	X^J="S	X^J="A	X^{J+1}="S	X^{J+1}="A	X^J="S
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	LXH	X																X			X
2	LX	X																X			X
3	LXA																				
4	STXH	X															X				X
5	STX	X															X				X
6	STXA																				
7	LXC																				
8	LXCA																				
9	LAH	X	X			X															X
10	LA	X	X			X															X
11	LAA																				
12	STAH	X	X		X																X
13	STA	X	X		X																X
14	STAA																				
15	LDH																				
16	LD																				
17	STDH																				
18	STD																				
19	LATH	X	X			X															X
20	LAT	X	X			X															X
21	STATH	X	X		X																X
22	STAT	X	X		X																X
23	LL	X	X			X															X
24	LR	X	X			X															X
25	STL	X	X		X																X
26	STR	X	X		X																X
27	LMX																				
28	STMX																				
29	LMA																				
30	STMA																				
31	LMS																				
32	STMS																				

L. Conway Archives

		X φ P	A φ P	B φ P	A^1 = S	A^1 = A	A^{i+1} = S	A^{i+1} = A	A^2 = S	A^2 = A	A^{i+1} = S	A^x = S	A^{x+1} = S	XB^2 = A	XB^2 = A	CB^{i+1} = A	X^{i+1} = S	X^1 = A	X^{i+1} = S	X^{i+1} = A	X^1 = S
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
33	STMZ																				
34	STMZA																				
35																					
36																					
37																					
38	MXA	X	X			X															X
39	MAX																				
40	MKL	X	X			X															
41	MKR	X	X			X															
42	MLX	X																X			
43	MXS																				
44	MSX																				
45	MSXZ																				
46	MXSφ																				
47	MXC	X																			X
48	MCX	X															X	X			
49	MLC	X	X						X							X					
50	MRC	X	X						X							X					
51	MXP																				
52	MKP																				
53																					
54																					
55																					
56	AN		X			X			X			X									
57	ADN																				
58	AR		X			X			X			X									
59	ADR																				
60	AU		X			X			X			X									
61	ADU																				
62	SN		X			X			X			X									
63	SDN																				
64	SR		X			X			X			X									

L. Conway Archives

		XΦP	AΦP	BΦP	A^I=S	A^I=D	A^{I+1}=S	A^{I+1}=D	A^J=S	A^J=D	A^{J+1}=S	A^K=S	A^{K+1}=S	XB^I=D	XB^J=D	CB^I=D	X^I=S	X^I=D	X^{I+1}=S	X^{I+1}=D	X^J=S
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
65	SDR																				
66	SU		X			X			X			X									
67	SDU																				
68	MN		X			X			X			X									
69	MDN																				
70	MR		X			X			X			X									
71	MDR																				
72	MU		X			X			X			X						—	—		
73	MDU																				
74	MMN																				
75	MMU																				
76	DN		X			X			X			X									
77	DDN																				
78	DR		X			X			X			X									
79	DDR																				
80	DMN																				
81	DMR																				
82	RND																				
83	SPF		X			X			X												
84	SNF		X			X			X												
85	CVS		X			X			X												
86	CVF		X			X			X												
87																					
88																					
89	AI		X			X			X			X									
90	SI		X			X			X			X									
91	MI		X			X			X			X									
92	MMI																				
93	DI		X			X			X			X									
94	DMI																				
95	ACL		X			X			X			X									
96	SCL		X			X			X			X									

L. Conway Archives

		XOP	AØP	BØP	A^I=S	A^I=D	A^{I+1}=S	A^{I+1}=D	A^J=S	A^J=D	AB^{I+1}=S	AK^{I+1}=S	AK^{I+1}=S	KB^I=D	KB^J=D	CB^I=D	X^I=S	X^I=A	X^{I+1}=S	X^{I+1}=A	X^J=S
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
97	ACH		X			X			X			X									
98	SCH		X			X			X			X									
99	SPI		X			X			X												
100	SNI		X			X			X												
101	CVN		X			X			X												
102	CVI		X			X			X												
103																					
104																					
105																					
106	AX	X																X			X
107	SX	X																X			X
108	MX	X																X			X
109	DRX	X																X		X	X
110	DX	X																X			X
111	RX	X																X			X
112	AXC	X																X			X
113	AXK	X																X			X
114	MXK	X																X			X
115	DRXK	X																X		X	X
116	DXK	X																X			X
117	RXK	X																X			X
118	SPX	X																X			X
119	SNX	X																X			X
120																					
121																					
122																					
123	CGEN	X	X						X			X				X					
124	CEQN	X	X						X			X				X					
125	CGED																				
126	CEQD																				
127	CMGEN	X	X						X			X				X					
128	CMEQN	X	X						X			X				X					

L. Conway Archives

		X≠P	A≠P	B≠P	A^I=S	A^I=D	A^{I+1}=S	A^{I+1}=D	A^J=S	A^J=D	A^{J+1}=S	A^K=S	A^{K+1}=S	XB^I=A	XB^J=A	CB^I=A	X^I=S	X^I=A	X^{I+1}=S	X^{I+1}=A	X^J=S
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
129	CMGED																				
130	CMEQD																				
131	CGEI	X	X						X			X				X					
132	CEQI	X	X						X			X				X					
133	CUGEI	X	X						X			X				X					
134	CGEX	X																			X
135	CEQX	X																			X
136	CUGEX	X																			X
137	CGEXK	X																			X
138	CEQXK	X																			X
139	CUGEXK	X																			X
140	CBA	X	X						X			X				X					
141	CBMA																				
142	CBX	X																			X
143	CBMX																				
144																					
145																					
146																					
147	SHA		X			X			X			X									
148	SHX	X																X			X
149	SHAC		X		X	X															
150	SHXC	X															X	X			
151	SHD																				
152	SHDX																				
153	SHDC																				
154	SHDXC																				
155	SWA		X		X	X			X	X											
156	SWX	X															X	X			X
157	IFA		X		X	X			X			X									
158	IFX	X															X	X			X
159	IFZA		X			X			X			X									
160	IFZX	X																X			X

L. Conway Archives

		XØP	AØP	BØP	$A^I=S$	$A^I=A$	$A^{I+1}=S$	$A^{I+1}=A$	$A^J=S$	$A^J=A$	$A^{J+1}=S$	$A^K=S$	$A^{K+1}=S$	$XB^I=A$	$XB^J=A$	$CB^I=A$	$X^I=S$	$X^I=A$	$X^{I+1}=S$	$X^{I+1}=A$	$X^J=S$
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
161	SIA		X			X			X			X									
162	SIX	X																X			X
163	SIAC		X		X	X															
164	SIXC	X															X	X			
165	SID																				
166	SIDC																				
167																					
168																					
169																					
170	ANDA		X			X			X			X									
171	TAFA		X			X			X			X									
172	FAFA		X			X			X			X									
173	ØRA		X			X			X			X									
174	TØFA		X			X			X			X									
175	FØFA		X			X			X			X									
176	EQA		X			X			X			X									
177	XØRA		X			X			X			X									
178	ANDX	X																X			X
179	TAFX	X																X			X
180	FAFX	X																X			X
181	ØRX	X																X			X
182	TØFX	X																X			X
183	FØFX	X																X			X
184	EQX	X																X			X
185	XØRX	X																X			X
186	ANDC	X																			
187	TAFC	X																			
188	FAFC	X																			
189	ØRC	X																			
190	TØFC	X																			
191	FØFC	X																			
192	EQC	X																			

L. Conway Archives

		XØP	AØP	BØP	$A^{I}=S$	$A^{I}=P$	$A^{I+1}=S$	$A^{I+1}=A$	$A^{J}=S$	$A^{J}=P$	$A^{J+1}=S$	$A^{K}=S$	$A^{K+1}=S$	$XB^{I}=A$	$XB^{I}=$	$CB^{I}=A$	$X^{I}=S$	$X^{I}=A$	$X^{I+1}=S$	$X^{I+1}=A$	$X^{I}=S$
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
193	XØRC	X																			
194	CNTT		X			X			X												
195	CNTAA		X			X			X			X									
196	CNTDA		X			X			X			X									
197	CNTAX	X																X			X
198	CNTDX	X																X			X
199																					
200																					
201																					
202	BAND	X		X																	
203	BTAF	X		X																	
204	BFAF	X		X																	
205	BØR	X		X																	
206	BTØF	X		X																	
207	BFØF	X		X																	
208	BEQ	X		X																	
209	BXØR	X		X																	
210	BU																				
211	EXIT	X	X																		
212	EXITL	X	X															X			
213	EXITA	X	X															X			X
214	EXITP																				
215	SKAND	X	X																		
216	SKTAF	X	X																		
217	SKFAF	X	X																		
218	SKØR	X	X																		
219	SKTØF	X	X																		
220	SKFØF	X	X																		
221	SKEQ	X	X																		
222	SKXØR	X	X																		
223	IVIB																				
224	NØP	X	X																		

L. Conway Archives

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

225 PAUSE
226 PI
227 SCAN
228 SVC
229 SVR
230 IC
231 IR
232
233
234
235 SIØ
236 HIØ
237 TCH
238 MTX
239 MXT
240 MZT
241 MØT
242 ITUMA
243 ITUMP
244 IDA
245 LDA
246 LDHAA
247 LDHBA
248 LDHCA
249 LDHDA
250 STDHAA
251 STDHBA
252 STDHCA
253 STDHDA
254
255
256

L. Conway Archives

		X^J=A	X^K=S	AB^3=D	C^2=S	C^1=A	C^J=S	ST=S	ST=D	SKIP	INV.ØP	EXIT	X REPL	A REPL	C^K=S
		21	22	23	24	25	26	27	28	29	30	31	32	33	34 35 36 37 38 39 40
1	LXH		X					X			X				
2	LX		X					X			X				
3	LXA														
4	STXH		X						X		X				
5	STX		X						X		X				
6	STXA														
7	LXC														
8	LXCA														
9	LAH		X	X				X			X			X	
10	LA		X	X				X			X			X	
11	LAA														
12	STAH		X						X		X				
13	STA		X						X		X				
14	STAA														
15	LDH														
16	LD														
17	STDH														
18	STD														
19	LATH		X	X				X			X			X	
20	LAT		X	X				X			X			X	
21	STATH		X						X		X				
22	STAT		X						X		X				
23	LL		X	X				X			X			X	
24	LR		X	X				X			X			X	
25	STL		X						X		X				
26	STR		X						X		X				
27	LMX														
28	STMX														
29	LMA														
30	STMA														
31	LMS														
32	STMS														

L. Conway Archives

		X^J=A	X^K=S	AB^I=B	C^I=A	C^I=S	ST=S	ST=A	SKIP	INV.ØP	EXIT	X REPL	A REPL	C^K=S					
		21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
33	STMZ																				
34	STMZA																				
35																					
36																					
37																					
38	MXA		X	X							X			X							
39	MAX																				
40	MKL			X							X			X					—	—	
41	MKR			X							X			X							
42	MLX										X										
43	MXS																				
44	MSX																				
45	MSXZ																				
46	MXSØ																				
47	MXC					X					X										
48	MCX						X				X										
49	MLC					X					X		X								
50	MRC					X					X		X								
51	MXP																				
52	MKP																				
53																					
54																					
55																					
56	AN										X										
57	ADN																				
58	AR										X										
59	ADR																				
60	AU										X										
61	ADU																				
62	SN										X										
63	SDN																				
64	SR										X										

L. Conway Archives

		XT=A	XK=S	AB=A	CI=S	CI=A	CJ=S	ST=S	ST=A	SKIP	INV. OP	EXIT	X REPL	A REPL	CK=S
		21	22	23	24	25	26	27	28	29	30	31	32	33	34
65	SDR														
66	SU										X				
67	SDU														
68	MN										X				
69	MDN														
70	MR										X				
71	MDR														
72	MU										X				
73	MDU														
74	MMN														
75	MMU														
76	DN										X				
77	DDN														
78	DR										X				
79	DDR														
80	DMN														
81	DMR														
82	RND														
83	SPF										X				
84	SNF										X				
85	CVS										X				
86	CVF										X				
87															
88															
89	AI										X				
90	SI										X				
91	MI										X				
92	MMI														
93	DI										X				
94	DMI														
95	ACL										X				
96	SCL										X				

L. Conway Archives

		XJ=D	XK=S	AB1=D	C1=S	C1=A	C2=S	ST=S	ST=D	SKIP / INV. OP	EXIT	X REPL	A REPL	CX=S
		21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
97	ACH										X										
98	SCH										X										
99	SPI										X										
100	SNI										X										
101	CVN										X										
102	CVI										X										
103																					
104																					
105																					
106	AX		X								X										
107	SX		X								X										
108	MX		X								X										
109	DRX		X								X										
110	DX		X								X										
111	RX		X								X										
112	AXC										X										
113	AXK										X										
114	MXK										X										
115	DRXK										X										
116	DXK										X										
117	RXK										X										
118	SPX										X										
119	SNX										X										
120																					
121																					
122																					
123	CGEN					X					X		X								
124	CEQN					X					X		X								
125	CGED																				
126	CEQD																				
127	CMGEN					X					X		X								
128	CMEQN					X					X		X								

L. Conway Archives

		X¹=A	X^K=S	AB¹=A	C¹=S	C¹=A	C^J=S	ST=S	ST=A	SKIP	INV.OP	EXIT	x REPL	A REPL	C^K=S						
		21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
129	CMGED																				
130	CMEQD																				
131	CGEI					X					X		X								
132	CEQI					X					X		X								
133	CUGEI					X					X		X								
134	CGEX		X			X					X										
135	CEQX		X			X					X										
136	CUGEX		X			X					X										
137	CGEXK					X					X										
138	CEQXK					X					X										
139	CUGEXK					X					X										
140	CBA					X					X		X								
141	CBMA																				
142	CBX		X			X					X										
143	CBMX																				
144																					
145																					
146																					
147	SHA										X										
148	SHX		X								X										
149	SHAC										X										
150	SHXC										X										
151	SHD																				
152	SHDX																				
153	SHDC																				
154	SHDXC																				
155	SWA										X										
156	SWX	X									X										
157	IFA										X										
158	IFX		X								X										
159	IFZA										X										
160	IFZX		X								X										

L. Conway Archives

		XF=A	XK=S	AR=A	CI=S	CI=A	CJ=S	ST=S	ST=D	Skip	INV. ØP	EXIT	X REPL	A REPL	CK=S
		21	22	23	24	25	26	27	28	29	30	31	32	33	34
161	SIA										X				
162	SIX		X								X				
163	SIAC										X				
164	SIXC										X				
165	SID														
166	SIDC														
167															
168															
169															
170	ANDA										X				
171	TAFA										X				
172	FAFA										X				
173	ØRA										X				
174	TØFA										X				
175	FØFA										X				
176	EQA										X				
177	XØRA										X				
178	ANDX		X								X				
179	TAFX		X								X				
180	FAFX		X								X				
181	ØRX		X								X				
182	TØFX		X								X				
183	FØFX		X								X				
184	EQX		X								X				
185	XØRX		X								X				
186	ANDC					X	X				X				X
187	TAFC					X	X				X				X
188	FAFC					X	X				X				X
189	ØRC					X	X				X				X
190	TØFC					X	X				X				X
191	FØFC					X	X				X				X
192	EQC					X	X				X				X

L. Conway Archives

		X1=A	X2=S	ABI=D	C1=S	C1=A	C2=S	ST=S	ST=D	SKIP	INV.ØP	EXIT	X REPL	A REPL	C2=S						
		21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
193	XØRC					X	X				X				X						
194	CNTT										X										
195	CNTAA										X										
196	CNTDA										X										
197	CNTAX		X								X										
198	CNTDX		X								X										
199																					
200																					
201																					
202	BAND		X		X		X				X										
203	BTAF		X		X		X				X										
204	BFAF		X		X		X				X										
205	BØR		X		X		X				X										
206	BTØF		X		X		X				X										
207	BFØF		X		X		X				X										
208	BEQ		X		X		X				X										
209	BXØR		X		X		X				X										
210	BU																				
211	EXIT										X	X									
212	EXITL										X	X									
213	EXITA										X	X									
214	EXITP																				
215	SKAND				X		X			X	X										
216	SKTAF				X		X			X	X										
217	SKFAF				X		X			X	X										
218	SKØR				X		X			X	X										
219	SKTØF				X		X			X	X										
220	SKFØF				X		X			X	X										
221	SKEQ				X		X			X	X										
222	SKXØR				X		X			X	X										
223	IVIB																				
224	NØP										X										

L. Conway Archives

21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

225 PAUSE
226 PI
227 SCAN
228 SVC
229 SVR
230 IC
231 IR
232
233
234
235 SIØ
236 HIØ
237 TCH
238 MTX
239 MXT
240 MZT
241 MØT
242 ITUMA
243 ITUMP
244 IDA
245 LDA
246 LDHAA
247 LDHBA
248 LDHCA
249 LDHDA
250 STDHAA
251 STDHBA
252 STDHCA
253 STDHDA
254
255
256

L. Conway Archives

(X UNIT FACILITY USAGE)

EA1 EA2 L S M D XA C

		41	42	43	44	45	46	47	48	49	50	51	52	53	54	55
1	LXH	2	2													
2	LX	2	2													
3	LXA															
4	STXH	2	2													
5	STX	2	2													
6	STXA															
7	LXC															
8	LXCA															
9	LAH	2	2													
10	LA	2	2													
11	LAA															
12	STAH	2	2													
13	STA	2	2													
14	STAA															
15	LDH															
16	LD															
17	STDH															
18	STD															
19	LATH	2	2													
20	LAT	2	2													
21	STATH	2	2													
22	STAT	2	2													
23	LL	2	2													
24	LR	2	2													
25	STL	2	2													
26	STR	2	2													
27	LMX															
28	STMX															
29	LMA															
30	STMA															
31	LMS															
32	STMS															

L. Conway Archives

EA1 EA2 L S M D XA C S

41 42 43 44 45 46 47 48 49 50 51 52 53 54 55

33	STMZ														
34	STMZA														
35															
36															
37															
38	MXA					1									
39	MAX														
40	MKL					1									
41	MKR					1									
42	MLX		1												
43	MXS														
44	MSX														
45	MSXZ														
46	MXSØ														
47	MXC					1									
48	MCX							1							
49	MLC														
50	MRC														
51	MXP														
52	MKP														
53															
54															
55															
56	AN														
57	ADN														
58	AR														
59	ADR														
60	AU														
61	ADU														
62	SN														
63	SDN														
64	SR														

L. Conway Archives

EA1 EA2 L S M D XA C

41 42 43 44 45 46 47 48 49 50 51 52 53 54 55

65 SDR
66 SU
67 SDU
68 MN
69 MDN
70 MR
71 MDR
72 MU
73 MDU
74 MMN
75 MMU
76 DN
77 DDN
78 DR
79 DDR
80 DMN
81 DMR
82 RND
83 SPF
84 SNF
85 CVS
86 CVF
87
88
89 AI
90 SI
91 MI
92 MMI
93 DI
94 DMI
95 ACL
96 SCL

L. Conway Archives

EA1 EA2 L S M D XA C

41 42 43 44 45 46 47 48 49 50 51 52 53 54 55

97	ACH														
98	SCH														
99	SPI														
100	SNI														
101	CVN														
102	CVI														
103															
104															
105															
106	AX			1											
107	SX			1											
108	MX					1									
109	DRX							1							
110	DX							1							
111	RX							1							
112	AXC			1											
113	AXK			1											
114	MXK					1									
115	DRXK							1							
116	DXK							1							
117	RXK							1							
118	SPX		1												
119	SNX		1												
120															
121															
122															
123	CGEN														
124	CEQN														
125	CGED														
126	CEQD														
127	CMGEN														
128	CMEQN														

L. Conway Archives

EA1 EA2 L S M D XA C

41 42 43 44 45 46 47 48 49 50 51 52 53 54 55

129	CMGED													
130	CMEQD													
131	CGEI													
132	CEQI													
133	CUGEI													
134	CGEX						1							
135	CEQX						1							
136	CUGEX						1							
137	CGEXK						1							
138	CEQXK						1							
139	CUGEXK						1							
140	CBA													
141	CBMA													
142	CBX						1							
143	CBMX													
144														
145														
146														
147	SHA													
148	SHX		1											
149	SHAC													
150	SHXC		1											
151	SHD													
152	SHDX													
153	SHDC													
154	SHDXC													
155	SWA													
156	SWX		1											
157	IFA													
158	IFX		1											
159	IF2A													
160	IF2X		1											

L. Conway Archives

		EA1	EA2	L	S	M	D	XA	C	S
		41	42	43	44	45	46	47	48	49	50	51	52	53	54	55
161	SIA															
162	SIX				1											
163	SIAC															
164	SIXC				1											
165	SID															
166	SIDC															
167																
168																
169																
170	ANDA															
171	TAFA															
172	FAFA															
173	ØRA															
174	TØFA															
175	FØFA															
176	EQA															
177	XØRA															
178	ANDX			1												
179	TAFX			1												
180	FAFX			1												
181	ØRX			1												
182	TØFX			1												
183	FØFX			1												
184	EQX			1												
185	XØRX			1												
186	ANDC									1						
187	TAFC									1						
188	FAFC									1						
189	ØRC									1						
190	TØFC									1						
191	FØFC									1						
192	EQC									1						

L. Conway Archives

EA1 EA2 L S M D XA C S

41 42 43 44 45 46 47 48 49 50 51 52 53 54 54

193	XØRC	1
194	CNTT	
195	CNTAA	
196	CNTDA	
197	CNTAX	1
198	CNTDX	1
199		
200		
201		
202	BAND	1
203	BTAF	1
204	BFAF	1
205	BØR	1
206	BTØF	1
207	BFØF	1
208	BEQ	1
209	BXØR	1
210	BU	
211	EXIT	
212	EXITL	1
213	EXITA	1
214	EXITP	
215	SKAND	
216	SKTAF	
217	SKFAF	
218	SKØR	
219	SKTØF	
220	SKFØF	
221	SKEQ	
222	SKXØR	
223	IVIB	
224	NØP	

L. Conway Archives

41 42 43 44 45 46 47 48 49 50 51 52 53 54 55

225 PAUSE
226 PI
227 SCAN
228 SVC
229 SVR
230 IC
231 IR
232
233
234
235 SIØ
236 HIØ
237 TCH
238 MTX
239 MXT
240 MZT
241 MØT
242 ITUMA
243 ITUMP
244 IDA
245 LDA
246 LDHAA
247 LDHBA
248 LDHCA
249 LDHDA
250 STDHAA
251 STDHBA
252 STDHCA
253 STDHDA
254
255
256

L. Conway Archives

(A UNIT FACILITY USEAGE)

FAI FAL FM FD IA IM ID C L S

56 57 58 59 60 61 62 63 64 65 66 67 68 69 70

1 LXH
2 LX
3 LXA
4 STXH
5 STX
6 STXA
7 LXC
8 LXCA
9 LAH
10 LA
11 LAA
12 STAH
13 STA
14 STAA
15 LDH
16 LD
17 STDH
18 STD
19 LATH
20 LAT
21 STATH
22 STAT
23 LL
24 LR
25 STL
26 STR
27 LMX
28 STMX
29 LMA
30 STMA
31 LMS
32 STMS

L. Conway Archives

FAI FA2 FM FD IA IM ID C L S

56 57 58 59 60 61 62 63 64 65 66 67 68 69 70

33	STMZ		
34	STMZA		
35			
36			
37			
38	MXA		
39	MAX		
40	MKL		
41	MKR		
42	MLX		
43	MXS		
44	MSX		
45	MSXZ		
46	MXSØ		
47	MXC		
48	MCX		
49	MLC		
50	MRC		
51	MXP		
52	MKP		
53			
54			
55			
56	AN	2 2	
57	ADN		
58	AR	2 2	
59	ADR		
60	AU	2 2	
61	ADU		
62	SN	2 2	
63	SDN		
64	SR	2 2	

L. Conway Archives

		FA1	FA2	FM	FD	IA	IM	ID	C	L	S
		56	57	58	59	60	61	62	63	64	65	66	67	68	69	70
65	SDR															
66	SU	2	2													
67	SDU															
68	MN			1												
69	MDN															
70	MR			1												
71	MDR															
72	MU			1												
73	MDU															
74	MMN															
75	MMU															
76	DN				1											
77	DDN															
78	DR				1											
79	DDR															
80	DMN															
81	DMR															
82	RND															
83	SPF									1						
84	SNF									1						
85	CVS										1					
86	CVF										1					
87																
88																
89	AI					1										
90	SI					1										
91	MI						1									
92	NMI															
93	DI							1								
94	DMI															
95	ACL					1										
96	SCL					1										

L. Conway Archives

		FA1	FA2	FM	FD	TA	JM	JD	C	L	S
		56	57	58	59	60	61	62	63	64	65	66	67	68	69	70
97	ACH					1										
98	SCH					1										
99	SPI					1										
100	SNI					1										
101	CVN		1													
102	CVI		1													
103																
104																
105																
106	AX															
107	SX															
108	MX															
109	DRX															
110	DX															
111	RX															
112	AXC															
113	AXK															
114	MXK															
115	DRXK															
116	DXK															
117	RXK															
118	SPX															
119	SNX															
120																
121																
122																
123	CGEN								1							
124	CEQN								1							
125	CGED															
126	CEQD															
127	CMGEN								1							
128	CMEQN								1							

L. Conway Archives

FAI FA2 FM FD IA IM ID C L S

56 57 58 59 60 61 62 63 64 65 66 67 68 69 70

129	CMGED														
130	CMEQD														
131	CGEI							1							
132	CEQI							1							
133	CUGEI							1							
134	CGEX														
135	CEQX														
136	CUGEX														
137	CGEXK														
138	CEQXK														
139	CUGEXK														
140	CBA							1							
141	CBMA														
142	CBX														
143	CBMX														
144															
145															
146															
147	SHA									1					
148	SHX														
149	SHAC									1					
150	SHXC														
151	SHD														
152	SHDX														
153	SHDC														
154	SHDXC														
155	SWA									1					
156	SWX														
157	IFA									1					
158	IFX														
159	IFZA									1					
160	IFZX														

L. Conway Archives

FAI FAL FM FD IA IN IM C L S

56 57 58 59 60 61 62 63 64 65 66 67 68 69 70

161	SIA	1
162	SIX	
163	SIAC	1
164	SIXC	
165	SID	
166	SIDC	
167		
168		
169		
170	ANDA	1
171	TAFA	1
172	FAFA	1
173	ØRA	1
174	TØFA	1
175	FØFA	1
176	EQA	1
177	XØRA	1
178	ANDX	
179	TAFX	
180	FAFX	
181	ØRX	
182	TØFX	
183	FØFX	
184	EQX	
185	XØRX	
186	ANDC	
187	TAFC	
188	FAFC	
189	ØRC	
190	TØFC	
191	FØFC	
192	EQC	

L. Conway Archives

FA1 FA2 FM FD JA IM ID C L S

56 57 58 59 60 61 62 63 64 65 66 67 68 69 70

193	XØRC	
194	CNTT	1
195	CNTAA	1
196	CNTDA	1
197	CNTAX	
198	CNTDX	
199		
200		
201		
202	BAND	
203	BTAF	
204	BFAF	
205	BØR	
206	BTØF	
207	BFØF	
208	BEQ	
209	BXØR	
210	BU	
211	EXIT	
212	EXITL	
213	EXITA	
214	EXITP	
215	SKAND	
216	SKTAF	
217	SKFAF	
218	SKØR	
219	SKTØF	
220	SKFØF	
221	SKEQ	
222	SKXØR	
223	IVIB	
224	NØP	

L. Conway Archives

56 57 58 59 60 61 62 63 64 65 66 67 68 69 70

225 PAUSE
226 PI
227 SCAN
228 SVC
229 SVR
230 IC
231 IR
232
233
234
235 SIØ
236 HIØ
237 TCH
238 MTX
239 MXT
240 MZT
241 MØT
242 ITUMA
243 ITUMP
244 IDA
245 LDA
246 LDHAA
247 LDHBA
248 LDHCA
249 LDHDA
250 STDHAA
251 STDHBA
252 STDHCA
253 STDHDA
254
255
256

L. Conway Archives

COMMON / RLS / VARIABLES

NAREGS	NXREGS	# A, X SOURCE-DEST REGS (etc.)
NABUS	NXBUS	# A, X I OR Ø BUSES (MAX)
AFULL(12)	XFULL(12)	FULL TRIGGER ON BUFF POSNS
AGØ(12)	XGØ(12)	GØ TRIGGER " " "
NAGØ	NXGØ	MAX # A, X ISSUE PER CYCLE
NATEST	NXTEST	# A, X POSNS TESTED " "
NABUF	NXBUF	# A, X BUFFER POSNS
ABUSYZ	XBUSYZ	DUMMY 0th REG BUSY TRIGGER
ABUSY(200)	XBUSY(200)	AREG, XREG WAITING TRIGGERS
ABUFF(12,100)	XBUFF(12,100)	A BUFFER, X BUFFER
ASØR(12,200)	XSOR(12,200)	SOURCE TAGS FOR AREGS, XREGS
ADEST(12,200)	XDEST(12,200)	DEST " " " "
AFAC(12,10)	XFAC(12,10)	FACILITIES USED BY A, X ØP IN BUF
AFACSC(4,10,20)	XFACSC(4,10,20)	A, X FACILITY SLOT BUSY SHIFT CELLS
ABUSSC(4,10,20)	XBUSSC(4,10,20)	A, X OUTBUS BUSY SHIFT CELLS
AIBBSY(10)	XIBBSY(10)	A, X INBUS BUSY VECTOR
AØBUS(12,10)	XØBUS(12,10)	A, X ØUTBUS DEST FOR ØP IN BUFF
AFSLØT(10,20)	XFSLØT(10,20)	SLØT BUSY PATTERN FOR A, X FACS
AFIBUS(10)	XFIBUS(10)	IN BUS FOR A, X FACS
AFØBUS(10)	XFØBUS(10)	OUTBUS " " " "
AFDLY(10)	XFDLY(10)	DELAY " " " "
NAFAC	NXFAC	# A, X FACS
NSLØT		MAX TIME SLOT USED IN SHIFT CELLS
Q(16,16)		The memory queue
SDBA(32,2)		A STORE DATA BUFFER
NQBUF		# of Q posns in model
NQTEST		# of Q posns tested for go
NQGØ		# max Q posns go / cycle
QINPT		Q INPUT POINTER
MEMDLY		Q TO REG MEMORY DELAY
MEMØRY(16)		MEMORY BOXES (BUSY)
NBØX		# MEMORY BØMS
MXTIME		MAX. RUN TIME. SEE MAIN
ØUTLVL		OUTPUT LEVEL CODE. SEE STATS
IQ(4,16)		INST MEM QUEUE
LØNGBR		ISSUED LONG BRANCH TRIGGER
SR(8), ST(8)		SKIP RES., SKIP TAKEN
SKAP, SKXP		SKIP X, A POINTER
NSBUF		# SKIP RING POSITIONS

L. Conway Archives

COMMON/RLS/VARIABLES (CONT)

APASS(200), XPASS(200)	THE REGISTER PASS BITS (BU → FRONT)
QUT(2)	TIME/DATE
JOB(6)	CHARACTERS IDENT. CURRENT JOB
JSTOP	STOP CONDITION TRIGGER
MEMCNT(16)	CNT. OUTST. MEM. REQS. FOR STOP COND.
ABOX(10), XBOX(10)	A, X FAC BOX NUMBER TABLES
ABXBSY(10), XBXBSY(10)	A, X FAC BOX BUSY TRIGGERS

L. Conway Archives

INDEXING ASØR(I,J), ADEST(I,J), ABUSY(I,J)

i.e. A-sources-destinations:

J = 1 : A(0)
J = 32 : A(31)
J = 33 : XB(0)
J = 64 : XB(31)
J = 65 : CB(0)
J = 88 : CB(23)

INDEXING XSØR(I,J), XDEST(I,J), XBUSY(I,J)

i.e. X - sources-destinations:

J = 1 : AB(0)
J = 32 : AB(31)
J = 33 : X(0)
J = 64 : X(31)
J = 65 : C(0)
J = 88 : C(31)
J = 89 : STØRAGE
J = 90 :

L. Conway Archives

The Decode Table D(256,50)

ØPS AND TAGS AS IN ØLD MPM SIM PRØGRAM DECØDE TABLE. 198 ØPS.

D(I,J)

J =	
1	X operation
2	A " "
3	BØS " "
4	A(I) = Source
5	A(I) = DEST
6	A(I+1) = source
7	A(I+1) = DEST
8	A(J) = Source
9	A(J) = DEST
10	A(J+1) = Source
11	A(K) = source
12	A(K+1) = source
13	XB(I) = DEST
14	XB(J) = DEST
15	CB(I) = DEST
16	X(I) = Source
17	X(I) = DEST
18	X(I+1) = Source
19	X(I+1) = DEST
20	X(J) = Source
21	X(J) = DEST
22	X(K) = Source
23	AB(I) = DEST
24	C(I) = Source
25	C(I) = DEST
26	C(J) = Source
27	STØRAGE = Source
28	STØRAGE = DEST
29	
30	ILLEGAL ØP TAG

L. Conway Archives

(D(256,50), cont.)

31	INDEX ADDER 1 (FOR EA'S)	X FACILITIES
32	INDEX ADDER 2 (FOR EA'r)	
33	LØG (ADD, SUB, COMP, LOG)	
34	SHIFTER	
35	MPY	
36	DIV	
37	X TØ A	
38	CMP	
39		
40		
41	ADD3	A FACILITIES
42	ADD4	
43	FMPY	
44	FDIV	
45	IADD	
46	IMPY	
47	IDIV	
48	CMP	
49	LØG	
50	SHFT	

L. Conway Archives

FORMAT OF XBUFF, ABUFF

XBUFF (12, 100), ABUFF (12, 100)

XBUFF (XINPT, J), ABUFF (AINPT, J)

J	: QUANTITY
1	LETTER IDENTIFYING INSTRUCTION
2	OP NUMBER (FOR DECODING)
3	I FIELD
4	J "
5	K "
6	EFF ADDRESS
7	A ØP
8	X ØP
9	EXIT FLAG (SKIP FLAG)
10	BRANCH SUCC.
11	SKIP SUCC.
12	BØP
13	SØP
14	BEØP
15	"EXIT GØES ALØNE THIS CYCLE" TAG
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

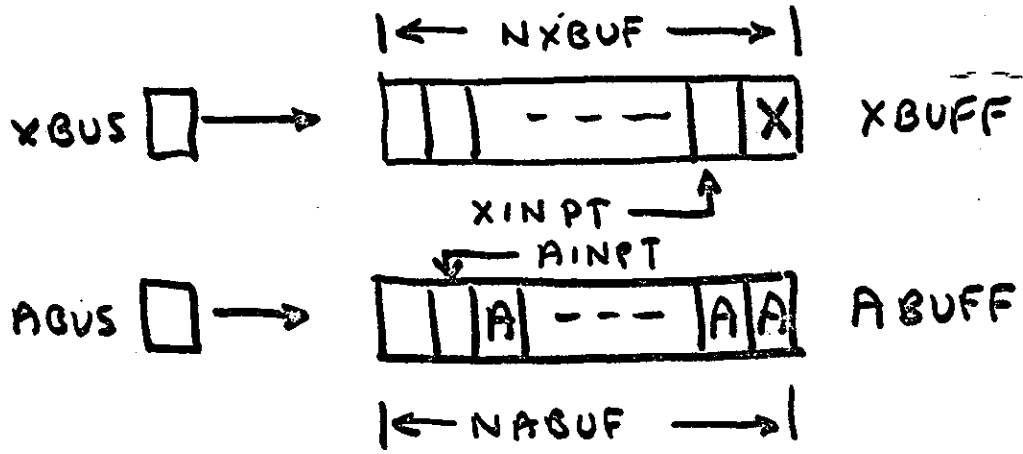
L. Conway Archives

THE $\phi 1$ - $\phi 3$ INTERFACE

I BUSING OF OPS TO A+X BUFFERS:

SUBJECT TO # OF A OR X OPS IN IB'S AVAILABLE FOR DISPATCHING TO A OR X BUFFERS, $\phi 1$ WANTS TO BUS UP TO NAOPS, NXOPS PER CYCLE TO $\phi 3$ (DEPENDING ON # BUSES IN MODEL).

WE SIMUL. THIS // ACTION SEQUENTIALLY:



$\phi 1$ EXAMINES INPUT POINTERS TO SEE IF AINPT > NABUF, XINPT > NXBUF. IF NOT, IT CAN PLACE A OR X OP IN ABUS OR XBUS AND CALL BUS TØ A, BUS TØ X SUBROUTINE TØ BUS OP TØ ABUFF, OR XBUFF. IT THEN CYCLES THRU THIS PROC. AGAIN TILL IT FILLS BUFFERS, RUNS OUT OF OPS, OR EXCEEDS ITS LIMITS · NAOPS, NXOPS.

INTERFACE:

AINPT	:	ABUFF INPUT POINTER
XINPT	:	X " " "
NABUF	:	# OF A BUFFERS
NXBUF	:	" " X "
ABUS (50)	:	THE A BUS
XBUS (50)	:	" X "
BUSTØA	:	SUBR. TO BUS ABUS TO ABUFF
BUSTØX	:	" " " X " " X "

(These update AINPT, XINPT)

L. Conway Archives

FORMAT OF EXEC-SIM OUTPUT CARDS

VARIABLE	STARTING COL.	FORMAT
INSADD	1	1XI5
MNEM(2)	7	1XA6
II	14	2XI2
IJ	18	1XI2
IK	21	1XI2
ILIT	24	2XI5
BRANCH	31	4XI1
SKIP	36	I1
IEXIT	37	I1
ACCADD	38	3XI5
NXADD	46	2XI5
ØPNUM	53	2XI3
LENGTH	58	2XI1
XØP	61	I1
AØP	62	I1
BØP	63	I1
SØP	64	I1
BEØP	65	I1

L. Conway Archives

```
*     OUTPUT TO BE USED AS INPUT TO TIMING PROGRAM
*
IC0000 AXKh   11,00,00,I00046,0  .000  I00046 IC0002  13C  2
IC0002 AXKh   12,00,00,I00047,0  000C  I00047 IC0004  13C  2
IC0004 AXKh   13,00,00,I00048,0  000   I00048 IC0006  13C  2
IC0006 AXKh   14,00,00,I00056,0  000   I00056 IC0008  13C  2
IC0008 LX     01,00,00,I99999,0  000   I99999 I00009   1   1
IC0009 LX     C0,11,00,I99999,0  000   I00046 IC001C   1   1
IC0010 LX     C1,11,00,I99999,0  000   I00046 I00011   1   1
IC0011 LX     C2,12,00,I99999,0  000   I00047 IC0012   1   1
IC0012 LX     C3,13,00,I99999,0  000   I00048 IC0013   1   1
IC0013 LXW    C4,01,00,I00048,0  000   I00049 IC0015   2   2
IC0015 LXW    C5,01,02,I00047,0  000   I00050 I00017   2   2
IC0017 LXW    00,01,01,I00047,0  000   I00049 I00019   2   2
IC0019 STX    01,C0,14,I99999,0  000   I00056 IC002C   3   1
IC002C STX    00,14,01,I99999,0  000   IC0057 IC0021   3   1
IC0021 STXh   C2,02,00,I00056,0  000   IC0058 IC0023   4   2
IC0023 STXh   C0,00,03,I00056,0  000   IC0059 IC0025   4   2
IC0025 LA     C0,00,11,I99999,0  000   I00046 IC0026   6   1
IC0026 LA     C1,00,13,I99999,0  000   I00048 IC0027   6   1
IC0027 LAW    C2,02,00,I00048,0  000   I00050 I00029   7   2
IC0029 STA    C0,C0,14,I99999,0  000   IC0056 IC0C30   8   1
IC003C STA    02,02,14,I99999,0  000   IC0058 IC0031   8   1
IC0031 STAh   C1,C2,02,IC0056,0  000   IC0060 IC0033   9   2
IC0033 LAD    C0,C0,11,I99999,0  000   I00046 I0034   1C   1
IC0034 LAD    C2,04,11,I99999,0  000   IC0050 IC0035   1C   1
I00035 LAD    C4,05,13,I99999,0  000   IC0053 IC0036   1C   1
IC0036 STAD   C0,C0,14,I99999,0  000   I00056 IC0037   12   1
IC0037 STAD   C2,04,14,I99999,0  000   I00060 I00038   12   1
IC0038 LADh   C6,C0,00,I24324,0  000   I24324 IC0040   11   2
IC004C LADh   C8,C0,04,I00056,0  000   IC0060 I00042   11   2
IC0042 LADh   10,03,05,I00056,0  00C   IC0064 IC0044   11   2
```

LONG/SHOR

OP CODE

ADDRESS OF NEXT INS

EFF ADDRESS TO BE ACCESSED

EXIT BIT

SKIP TO EXIT BIT (IF 1)

BRANCH SUCC ON EXIT BIT (IF 1)

EXIT BIT

H FIELD (DEC)

REGISTER K (DEC)

REGISTER J (DEC)

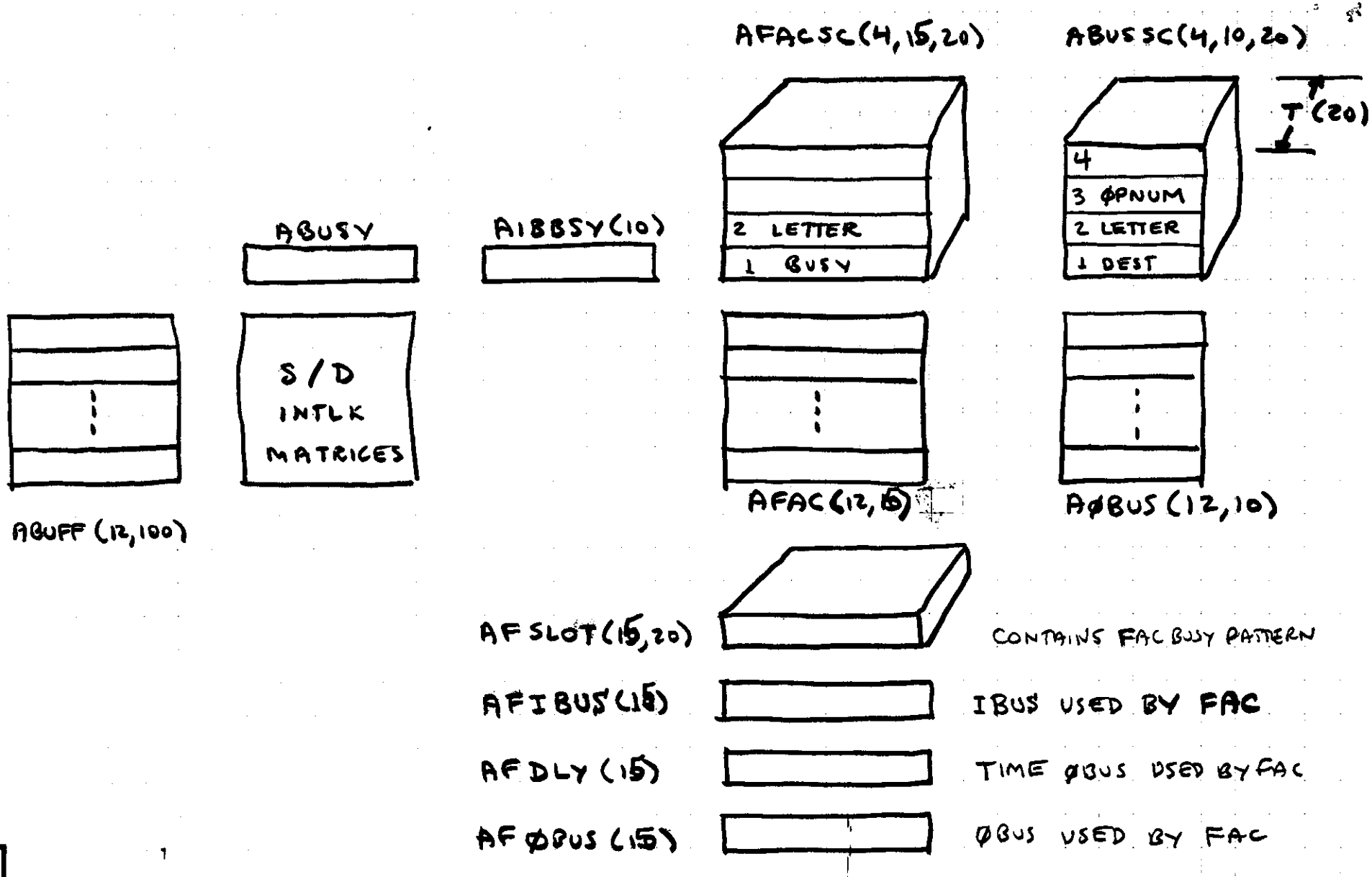
REGISTER I (DEC)

OP MNEMONIC

INST ADDRESS

151

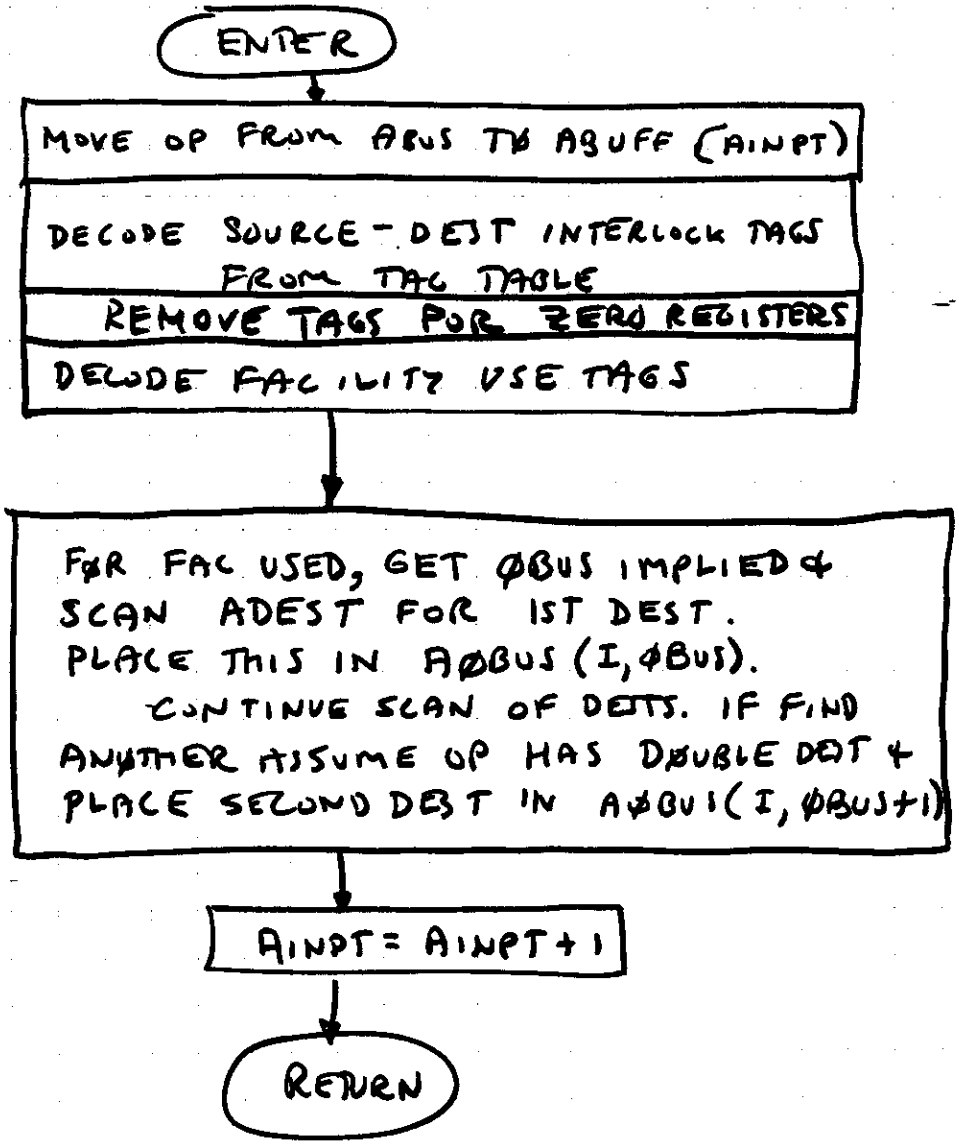
L. Conway Archives



L. Conway Archives

BLØCK DIAGRAM OF BUSTØA (BUSTØX)

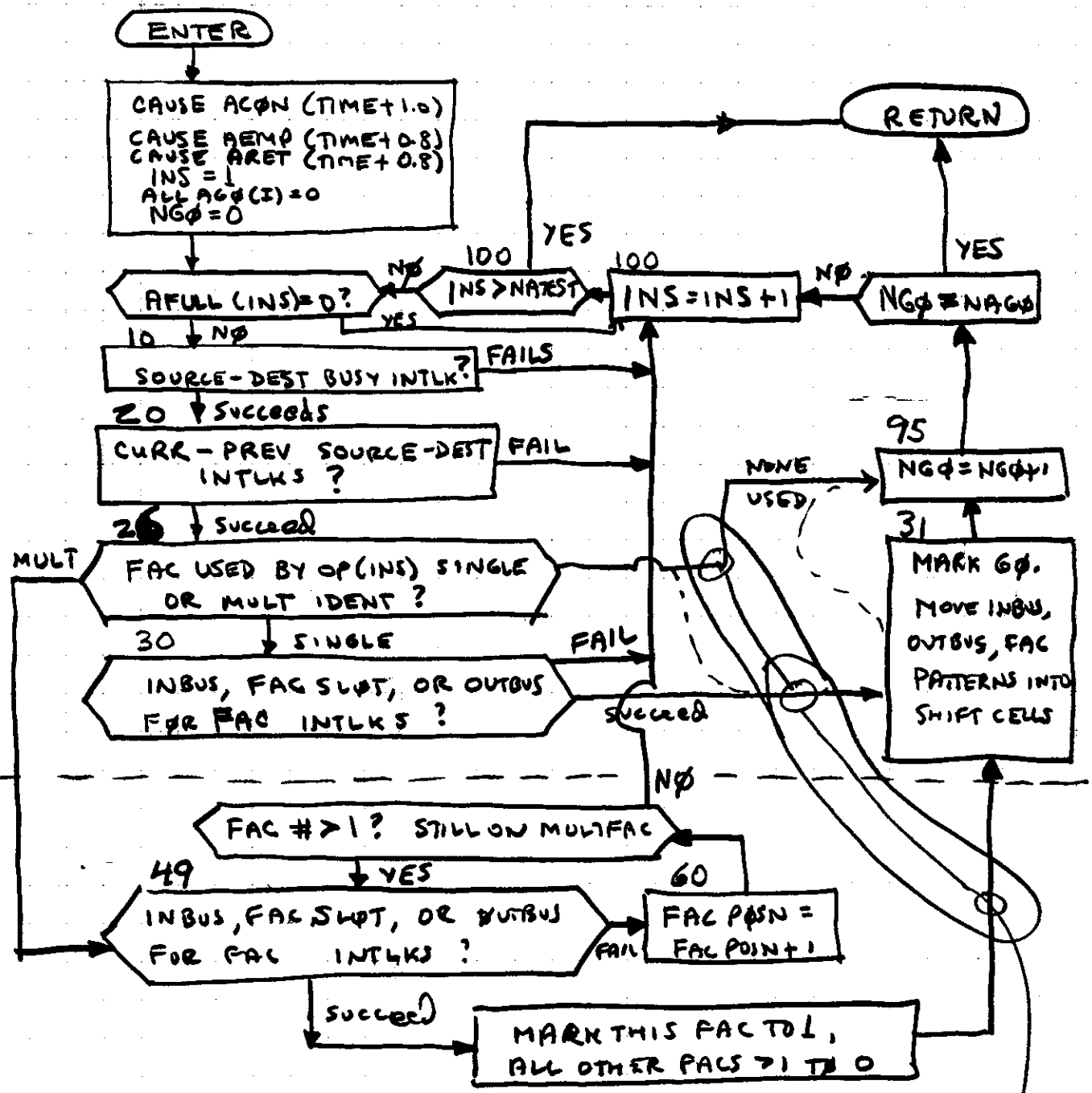
(BUS OP TO ABUFF. DECODE OP TAGS)



L. Conway Archives

BLØCK DIAGRAM ØF ACØN (XCØN)

(SCAN FØR 1ST NAGØ ØF NATEST WHICH CAN GØ)

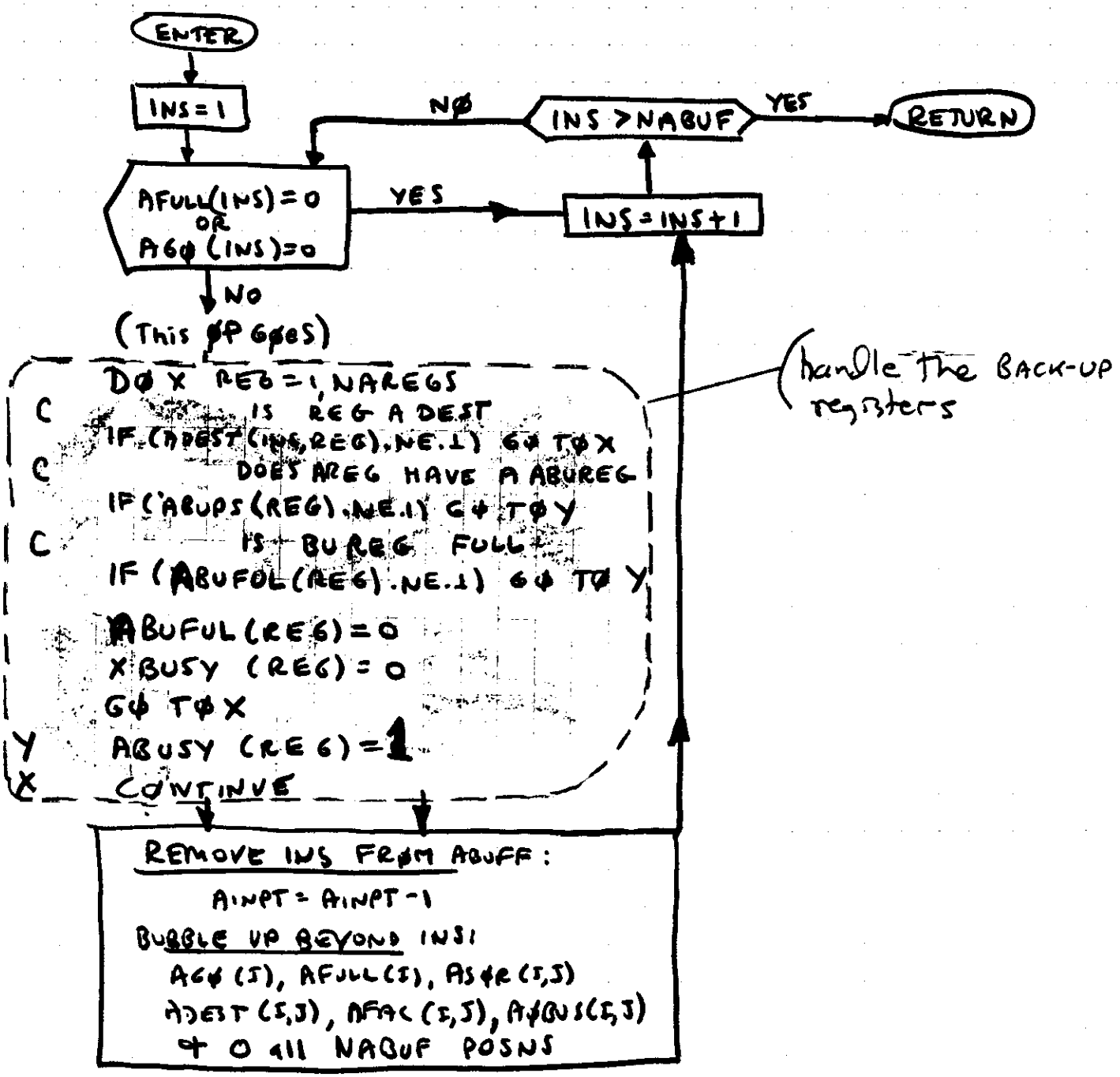


THIS SECTION HANDLES TEST FOR FACS FOR OPS USING ONE FAC OF A SET OF MULTIPE IDENT. FACS (I.E. ONE of two Ident adders)

154
L. Conway Archives

BLOCK DIAGRAM OF AEMP (XEMP) (AT .9 TIME)

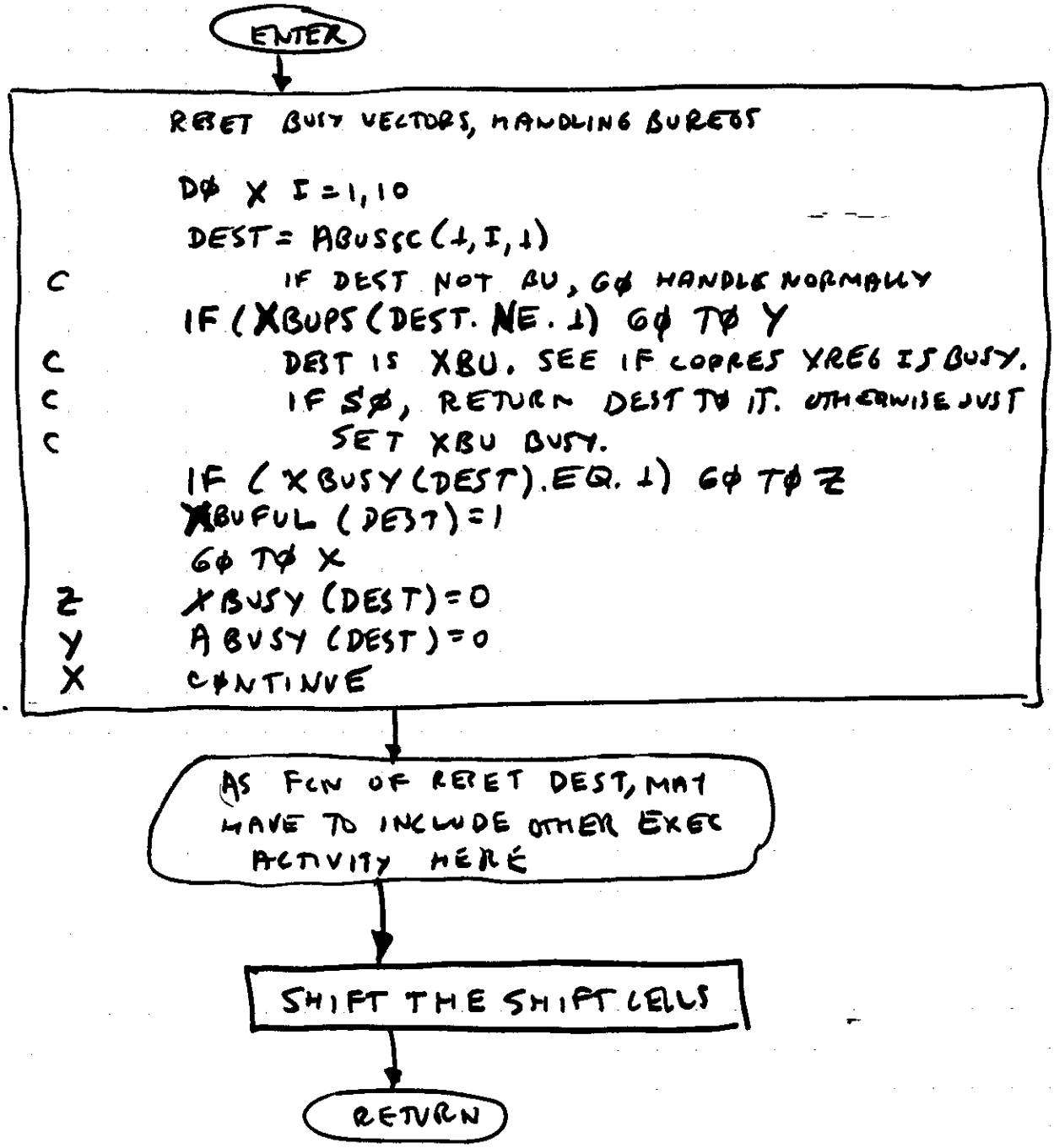
(Issue the GO BPS)



L. Conway Archives

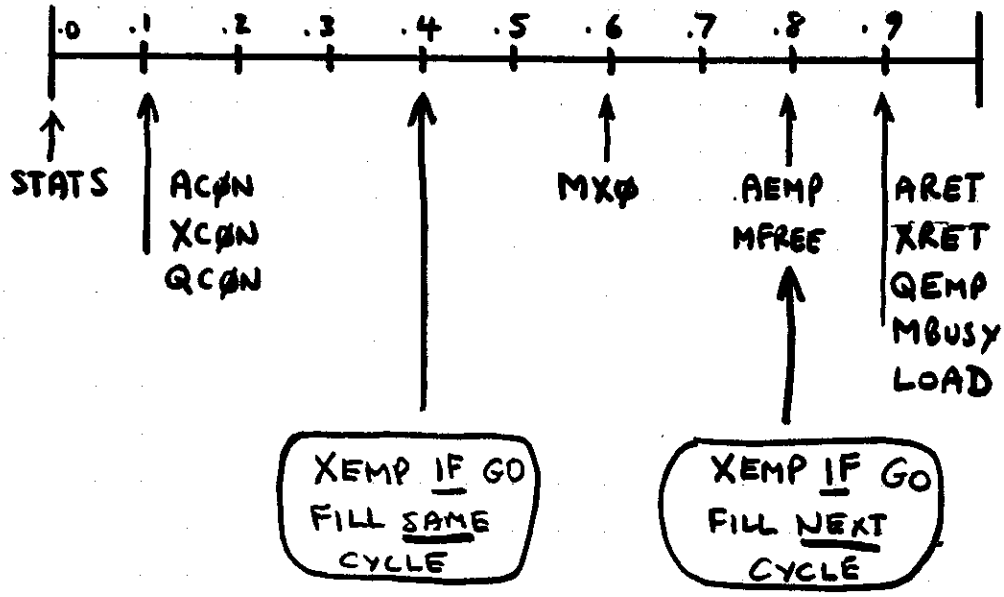
BLOCK DIAGRAM OF ARET (XRET)

USES ΦBUS SHIFT CELL TO RET BUS TO DESTS THOSE WHICH ARE DUE. THEN SHIFTS THE SHIFT CELLS.



L. Conway Archives

THE EVENT RUNNING TIMES WITHIN THE CYCLE:



L. Conway
Archives

STACK TØ REG TIMING

KEY DIFFERENCE BETWEEN

A and X Stack Algorithm and busing, Facilitres.

X overlaps intlk & inbusing to get results at reg at earliest time. Does not anticipate result arrival for ISvence of nxt op.

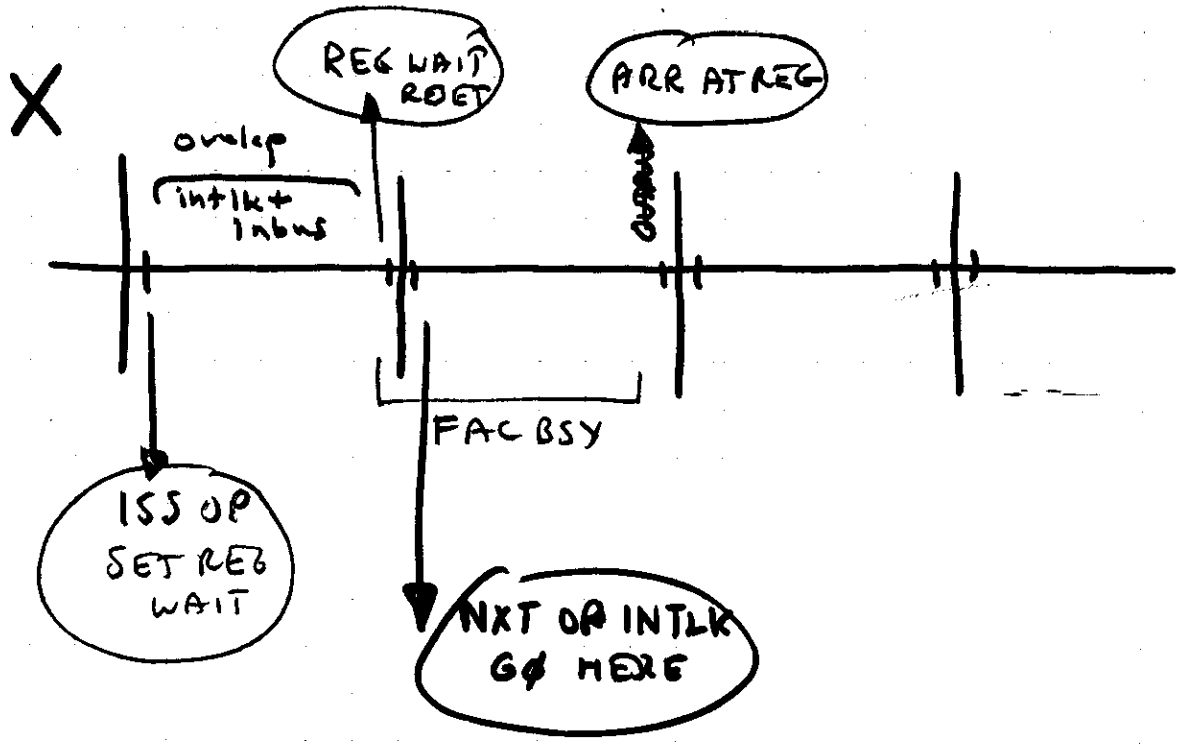
A has inbusing cycle. It does anticipate arrival at reg by 1 cycle so next op can (if dep) intlk and go during last exec cycle of prev. op.

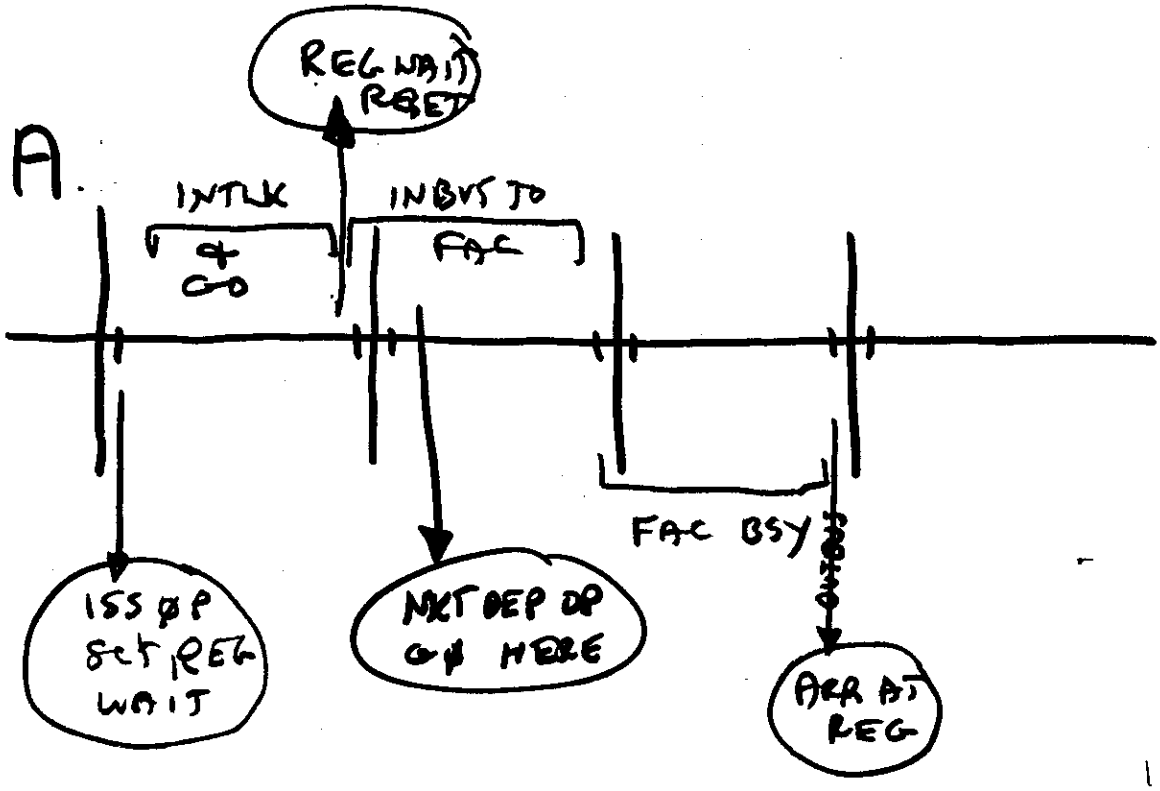
Effect: X gets results to registers one cycle earlier than A but they rep the same on dep ops with same timings.

L. Conway Archives

FULL BYPASSING

1/1

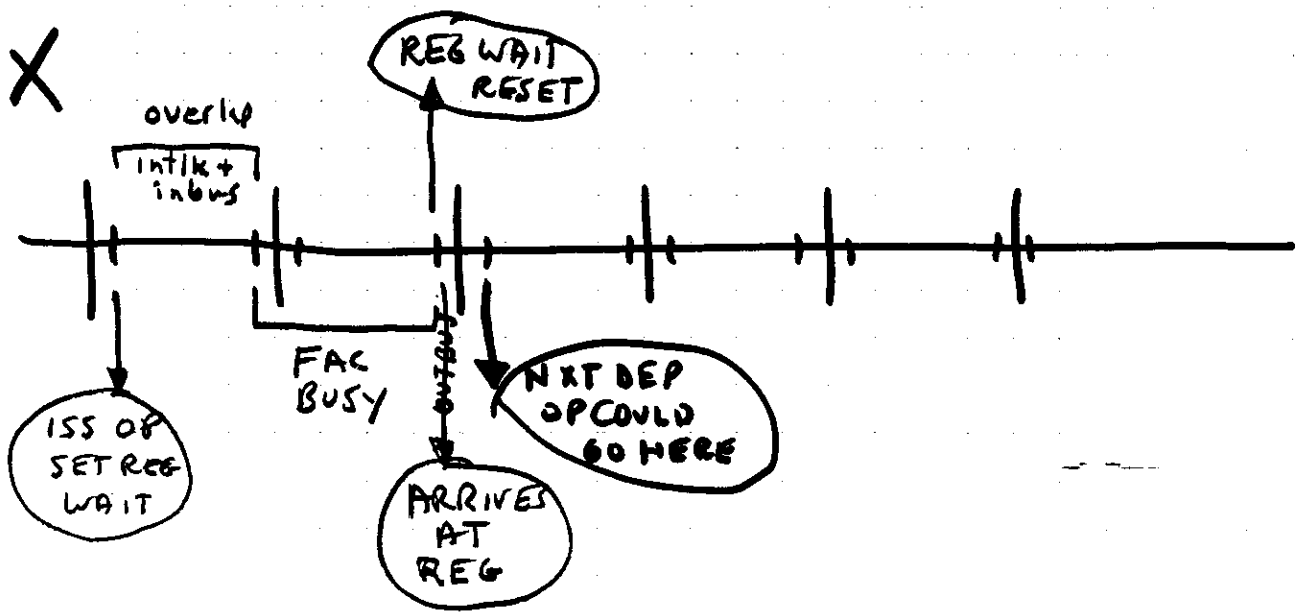


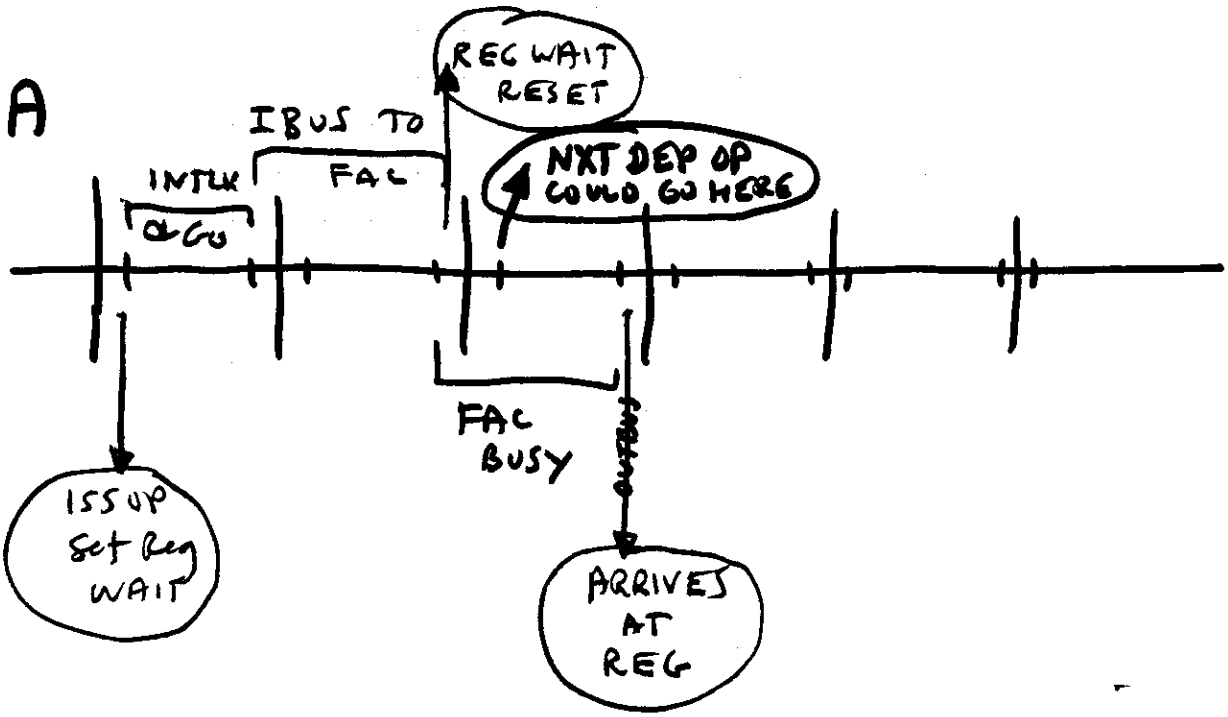


L. Conway Archives

NO BYPASSING

1/1





L. Conway Archives

COMMON / RLS / (continued)

18582 Q(16,16)	18838 SDBA(32,2)	18902 NQBUF	18903 NQTEST	18904 NQGØ
18905 QINPT	18906 QCØN	18907 QEMP	18908 MBUSY	18909 MFREE
18910 LØAD	18911 MEMDLY	18913 MEMØRY(16)	18928 NSØX	18930 EAV
18931 MXTIME	18933 ØUTLVL	18934 IQ(4,16)	18998 RTN	18999 LØNGBR
19000 SR(8)	19008 ST(8)	19016 SKXP	19017 SKAP	19018 NSBUF
19019 APASS(200)	19219 XPASS(200)	19419 ØUT(2)	19423 JØB(6)	19429 SSTØP	19430 MEMCNT(16)
19446 ABØX(15)	19461 ABXBSY(10)	19471 XBØX(15)	19486 XBXBSY(10)	19496

L. Conway Archives

CØMMØN /RLS/ Revised MAY 1 8 1967

1	2	3	4	5	6
FIRST	NAREGS	NXREGS	NABUS	NXBUS	STATS
7	8	9	10	11	12
ACØN	XCON	AEMP	XEMP	FILL	AFULL(12)
24	36	48	60	61	62
XFULL(12)	AGØ(12)	XGØ(12)	NAGØ	NXGØ	NATEST
63	64	65	66	67	267
NXTEST	NAFAC	NXFAC	ABUSYZ	ABUSY(200)	XBUSYZ
268	468	1668	2868		
XBUSY(200)	ABUFF(1200)	XBUFF(1200)	ASØR(2400)		
5268	7668	10068	12468		
XSØR(2400)	ADEST(2400)	XDEST(2400)	AFAC(180)		
12648	12828	14028	14029		
XFAC(180)	AFACSC(1200)	ARET	XFACSC(1200)		
15229	15230	16030	16040		
XRET	ABUSSC(800)	AIBBSY(10)	XBUSSC(800)		
16840	16850	16865	16985		
XIBBSY(10)	XFIBUS(15)	AØBUS(120)	XØBUS(120)		
17105	17405	17705			
AFSLØT(300)	XFSLOT(300)	AFIBUS(15)			
17720	17735	17750	17765		
AFDLY(15)	XFDLY(15)	AFØBUS(15)	XFØBUS(15)		
17780	17781	17782	17982		
NSLØT	ABUPSZ	ABUPS(200)	XBUPS(200)		
18182	18382				
ABUFUL(200)	XBUFUL(200)				

L. Conway Archives

CØMMØN /RLS/ Before Revision

MAY 16 1967

1	2	3	4	5	6	7
FIRST	NAREGS	NXREGS	NABUS	NXBUS	STATS	ACON
8	9	10	11	12		24
XCON	AEMP	XEMP	FILL	AFULL(12)		XFULL(12)
36	48	60	61	62		63
AGØ(12)	XGO(12)	NAGO	NXGØ	NATEST		NXTEST
64	65	66	67	267		
NAFAC	NXFAC	ABUSYZ	ABUSY(200)	XBUSYZ		
268	468	1668	2868			
XBUSY(200)	ABUFF(1200)	XBUFF(1200)	ASOR(2400)			
5268	7668	10068	12468			
XSOR(2400)	ADEST(2400)	XDEST(2400)	AFAC(120)			
12588	12708	13508	13509			
XFAC(120)	AFACSC(800)	ARET	XFACSC(800)			
14309	14310	15110	15120			
XRET	ABUSSC(800)	AIBBSY(10)	XBUSSC(800)			
15920	15930	15940	16060			
XIBBSY(10)	XFIBUS(10)	AØBUS(12)	XOBUS(120)			
16180	16380	16580				
AFSLOT(200)	XFSLOT(200)	AFIBUS(10)				
16590	16600	16610	16620			
AFDLY(10)	XFDLY(10)	AFØBUS(10)	XFØBUS(10)			
16630	16631	16632	16832			
NSLØT	ASUPSZ	ABUPS(200)	XBUPS(210),			
17032	17232					
ABUFUL(200)	XBUFUL(200)					

L. Conway Archives

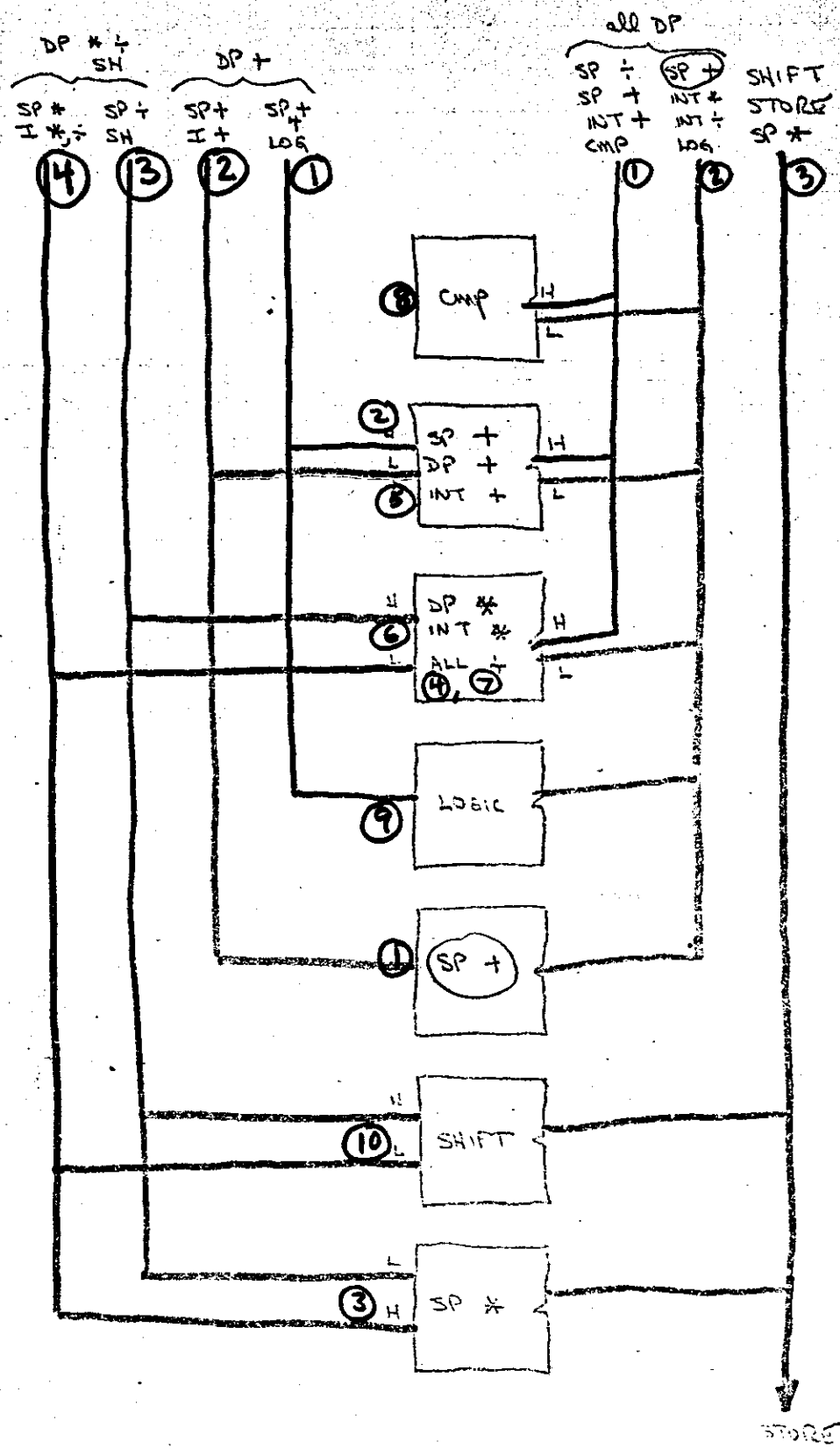
(cont.)

COMMON /RLS/ (continued)

17432 Q(16,16)	17688 SDBA(32,2)	17752 NQBUF	17753 NQTEST	17754 NQGØ
17755 QINPT	17756 QCØN	17757 QEMP	17758 MBUSY	17759 MFREE
17760 LØAD	17761 MEMDLY	17763 MEMORY(16)	17778 NBØX	17780 EAV
17781 MXTIME	17783 ØUTLVL	17784 IQ(4,16)	17848 RTN	17849 LØNGBR
17850 SR(8)	17858 ST(8)	17866 SKXP	17867 SKAP	17868 NSBUF
17869 APASS(200)	18069 XPASS(200)	18269 ØUT(2)	18273 SSTØP	18274 MEMCNT(16)
18290 ABØX(10)	18300 ABXBSY(10)	18310 XBØX(10),	18320 XBXBSY(10)	18330

L. Conway Archives

ZIS 5/8/67
Basic dal

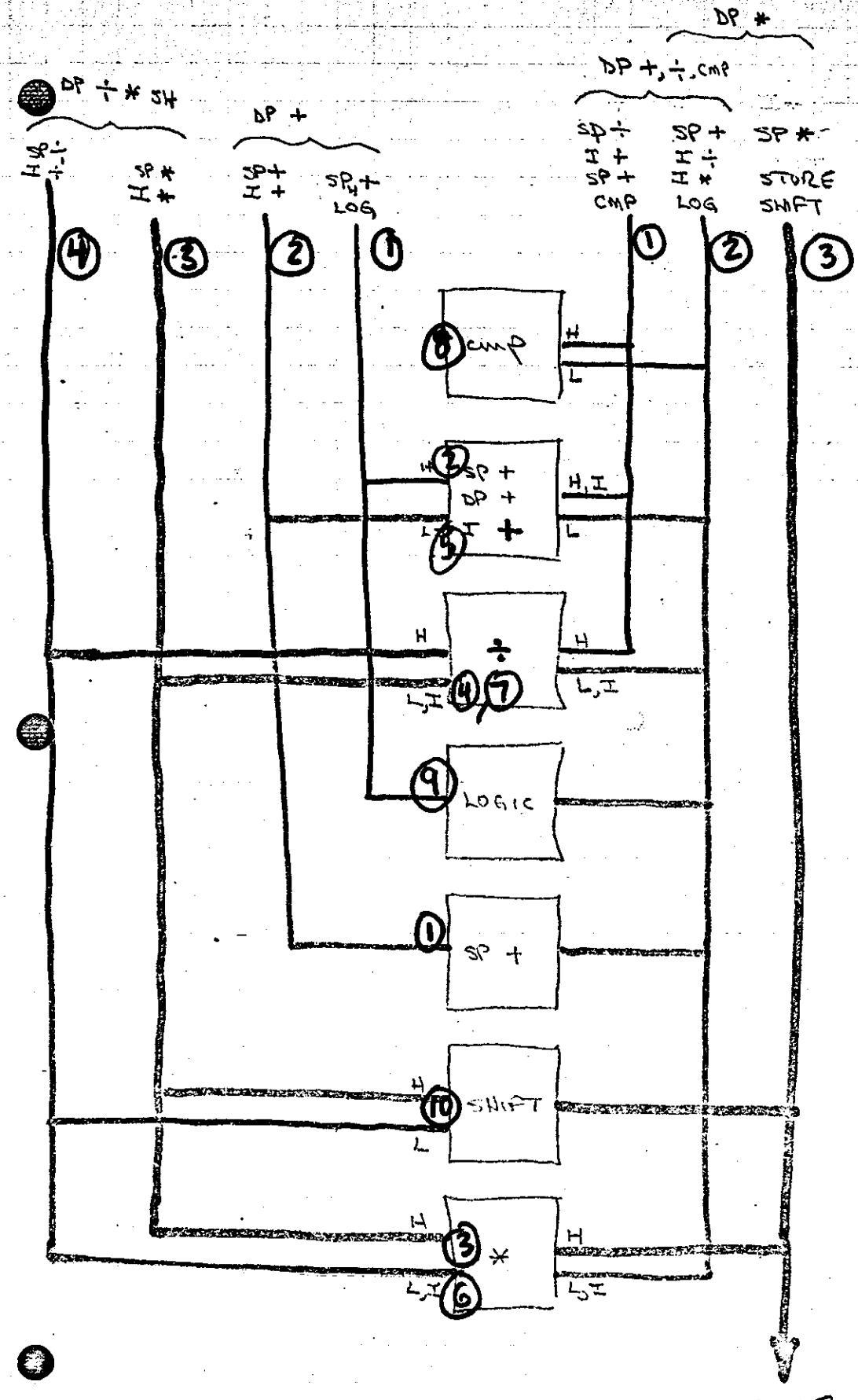


Bussing with

DO-ALL 3 BUSSES

L. Conway Archives

5/8/67



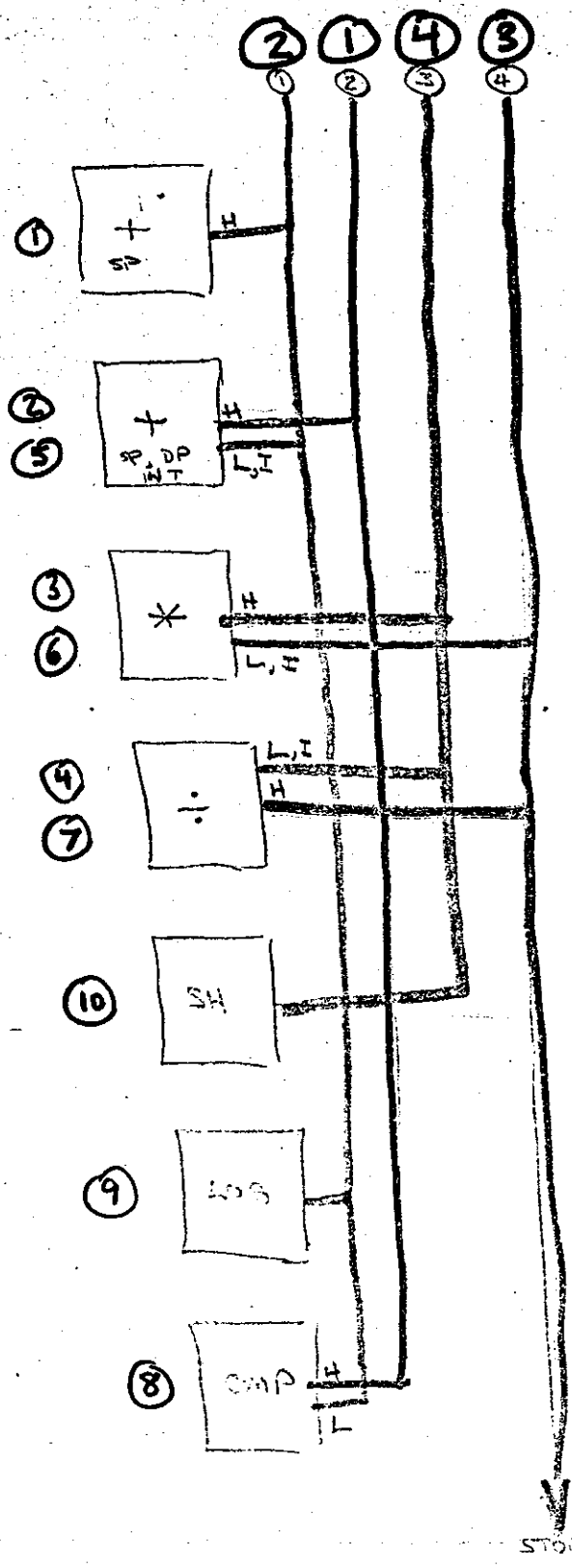
ONLY CHANGE:

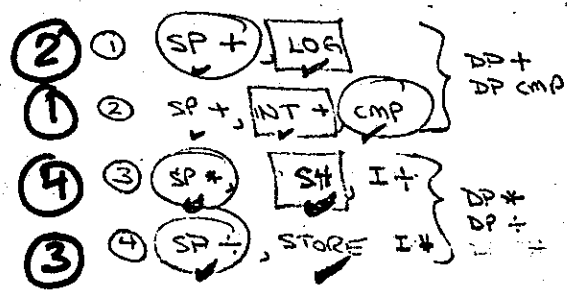
AFØBUS(6) = 3

L. Conway Archives

5/8/67

Bussing with SEP *, ÷ 4 BUSSES





CHANGES :

AFIBUS(3) = 4
AFIBUS(4) = 3
AFIBUS(6) = 3
AFIBUS(7) = 4
AFIBUS(10) = 4

AFIBUS(6) = 3

AFAC	1	2	3-4	5	6	7	8	9	10
IBUS	2	1	4	3	1	3	4	1	2	4

L. Conway Archives

Date: May 24, 1967
From (location or U.S. mail address): Advanced Computing Systems 988/031
Dept. & Bldg.:
Phone Ext.:

E N Sussenguth

IBM

Subject: MPM-BLCU Interface for Store Ops

Reference: Conversation with Mr. G. T. Paul, Mr. R. J. Robelen and Mr. J. R. Wierzbicki

To: File

It is desired to associate the Request Stack$_1$ with the ea buss$_1$ and the Request Stack$_2$ with the ea buss$_2$. Additionally, index type stores will ship 24 bit data words to Request Stack$_1$ on X DATA BUS$_1$ and to Request Stack$_2$ on X DATA BUS$_2$. These busses are associated with and energized at the same time as their corresponding ea busses. See diagram in Figure 1 for X unit, Figure 2 for the BLCU Request Stacks.

The situation of interest occurs with A unit type stores. The STO effective addresses are processed in the X unit initially. Store addresses are presented to the BLCU Data Request Stacks in strict order. Ea buss$_1$, ea buss$_2$ or both ea buss$_1$ and ea buss$_2$ may be selected. When both are selected, the first store will be on buss$_1$. A STO. BUS First-In-First-Out column is filled in order with the names (1 and/or 2) of the ea busses used to transmit the effective addresses to their corresponding Request Stacks. At the end of the X unit interlock cycle the A unit interlock cycle may respond to the particular store initiated by the X unit. When the A store address arrives at its corresponding Data Request Stack, it is assigned a slot in that particular stack and that slot name is placed in either the REQ1 or REQ2 First-In-First-Out column.

L. Conway Archives

RG2-0416-10

Meanwhile, the A unit instructions in contention examine the top two entries in the STO BUS column for validity. One or two stores are allowed to "go" if the entries are valid and the other standard interlocks are satisfied. The first store takes the buss designated by the name at the top of the STO. BUS buss column; the second store would take the next name in the column if it is different from the top name. Our current thinking is that two uninterlocked stores can "go" in the A unit if they are in adjacent contender positions. For this reason if three stores in a row are in contention and the store buss column is as follows:

V	Name
1	2
1	2
1	1
0	-
0	-

only the top A store will go this cycle. The column will then look like:

V	Name
1	2
1	1
0	-
0	-
0	-

The top two A uninterlocked stores will go now if they are adjacent in the contender stack. The first store will transmit on A DATA BUS_2 and the second on A DATA BUS_1 to the corresponding Request Stack. The name of the slot of the data buffer to be filled is taken from the top

L. Conway
Archives

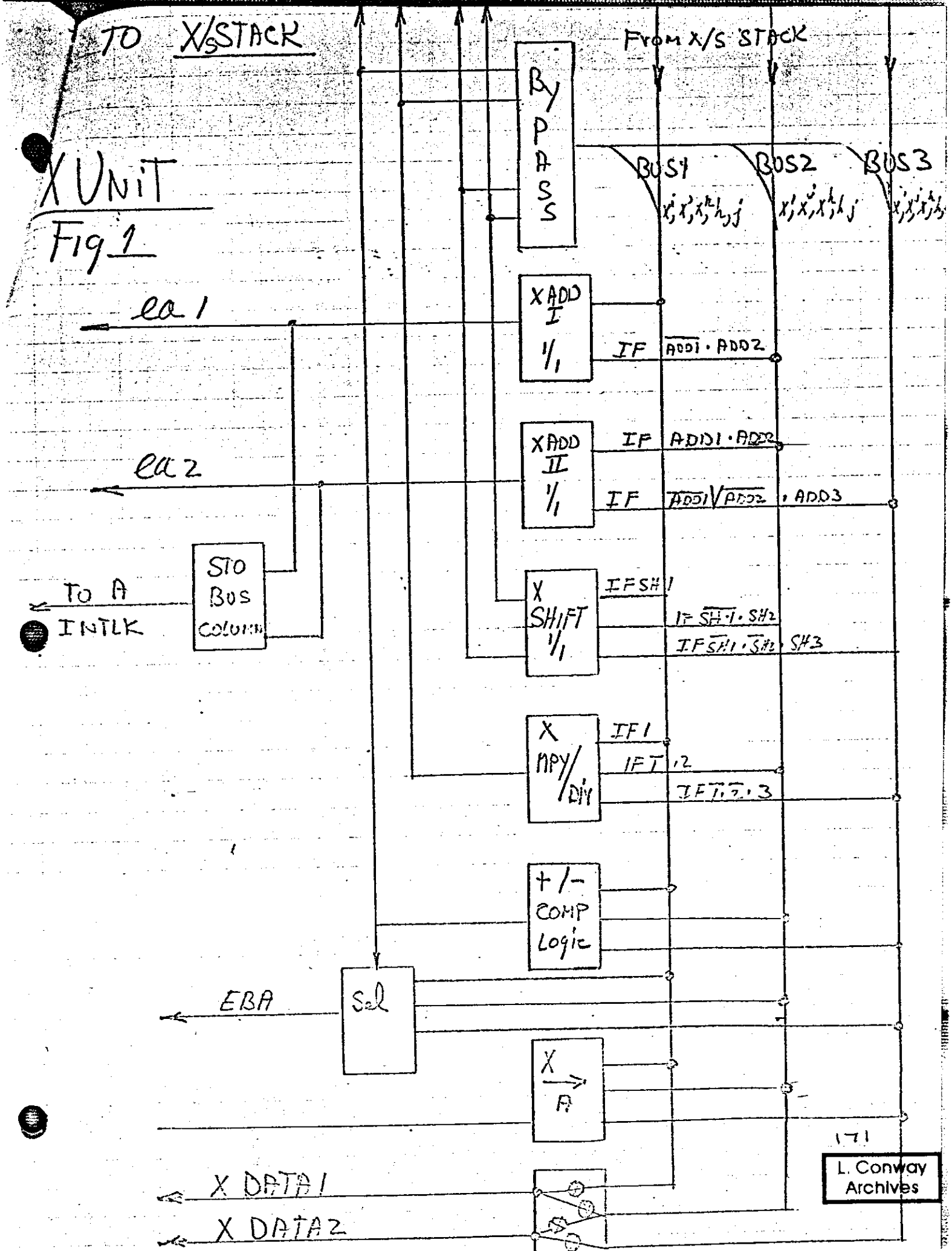
entries of the REQ1 or REQ2 column. Because two stores occupy both busses, only the top entry of each REQ column may be interrogated. The top entry is removed from the appropriate column when the data is entered into the named buffer.

G. T. Paul

L. Conway
Archives

X UNIT

Fig 1



L. Conway Archives

BLCU DATA REQUEST STACKS

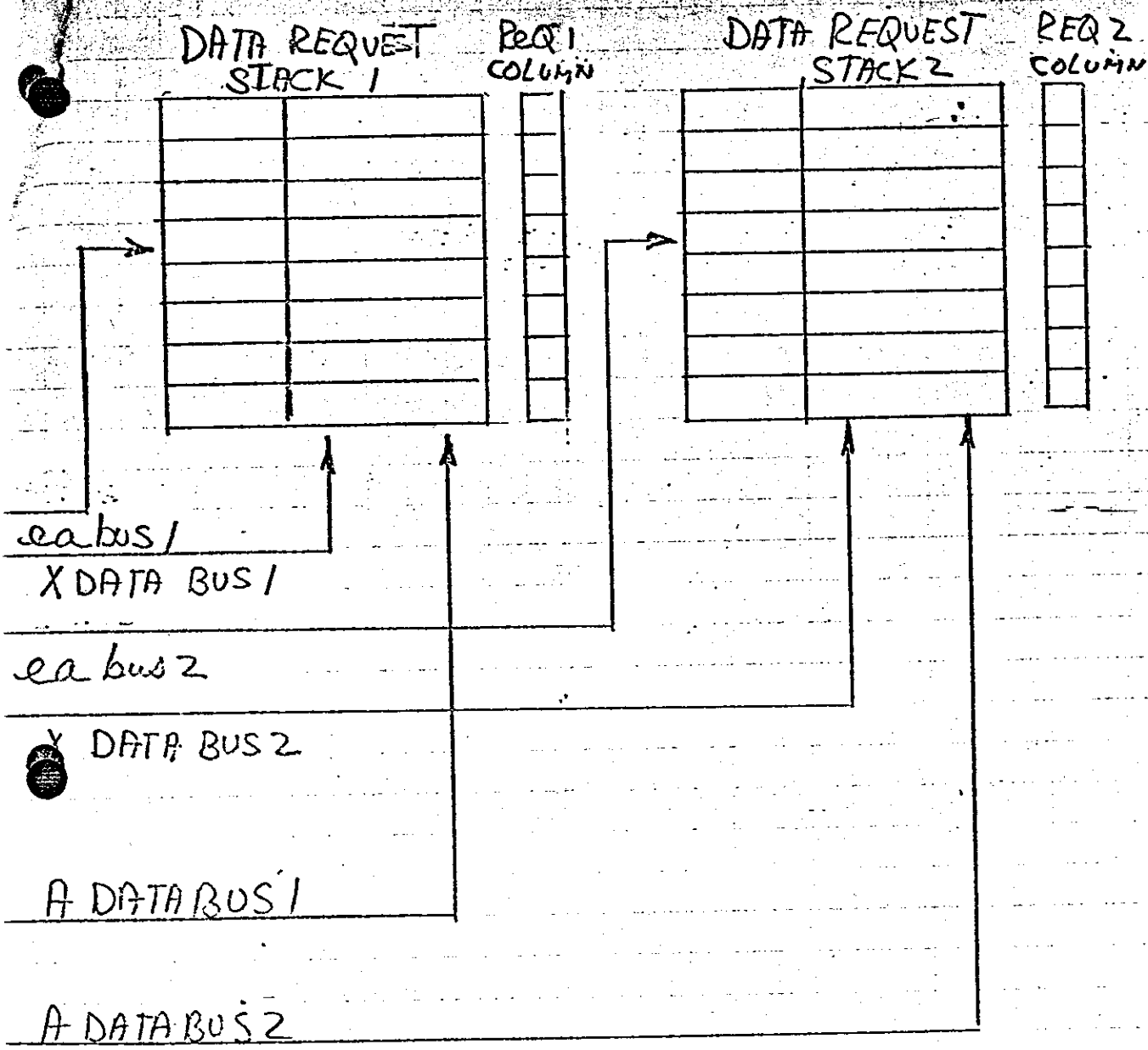
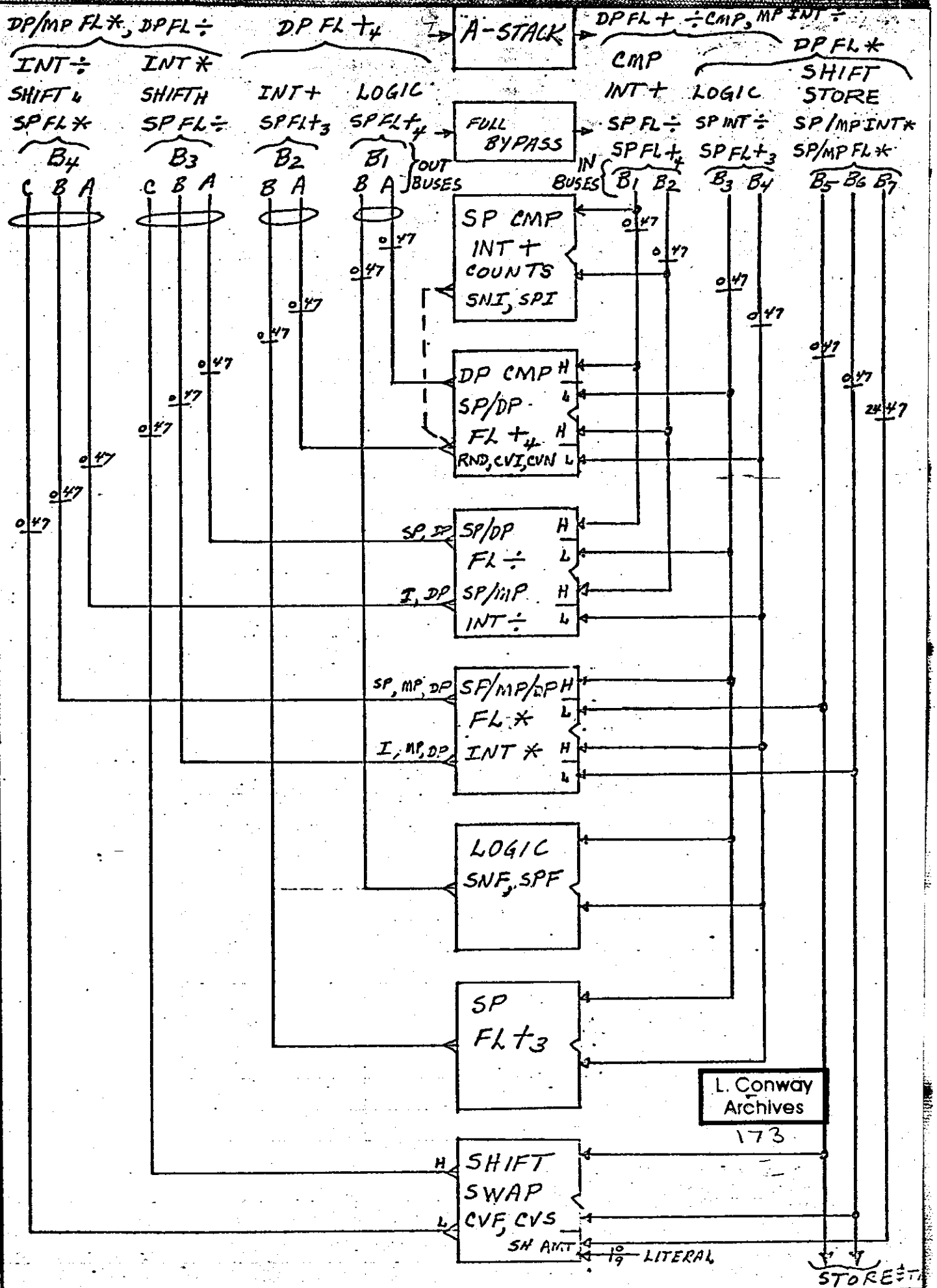


Fig 2

L. Conway Archives

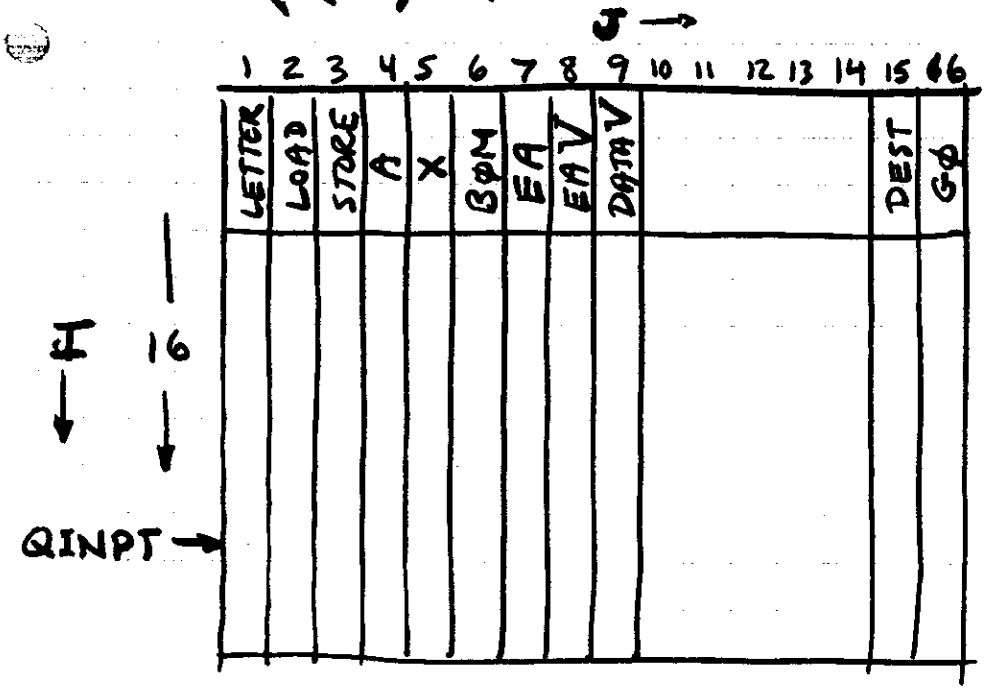


L. Conway Archives
173

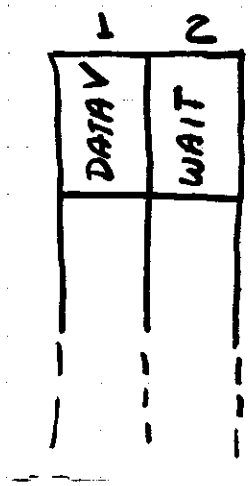
A-UNIT DATA FLOW

MEH. GP 5/17/67
REV. 6/7/67

Q(16,16)



SDBA(32,2)



Parameters: NQBUF total # fillable Q Posn's

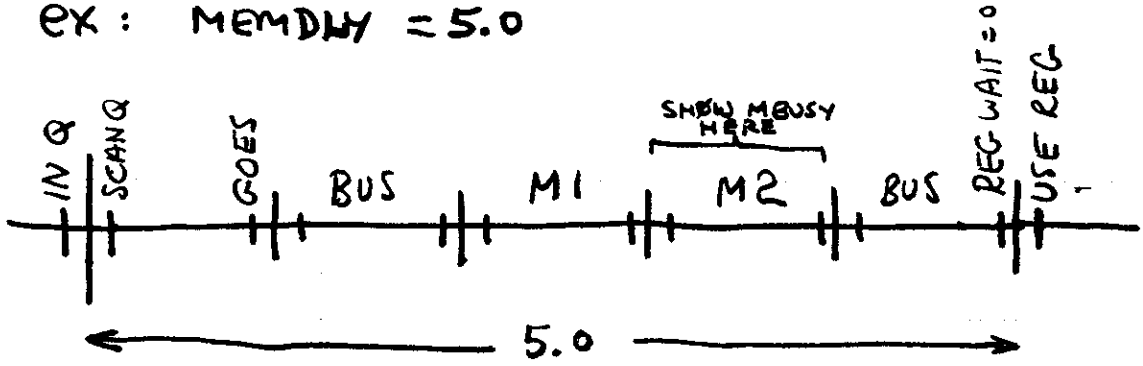
NQTEST # Posn's tested for GØ

NQGØ # ØPS MAX GØ / CYCLE

(Probable values: NQBUF = NQTEST = NQGØ = 8)

TIME DELAY Q TO REG = MEMDLY

ex: MEMDLY = 5.0



L. Conway Archives

```
      SUBROUTINE QRCON                    (RUNS AT .1)
C           ALGORITHM. LOADS OUT OF ORDER WITH BOM INTLK
C                      STORES IN ORDER
      CALL CAUSE (QCON, TIME+1.0, A, 0, 0)
      CALL CAUSE (QEMP, TIME+0.8, 0, 0, 0)
      NGO = 0
      DO 1 I=1, NQBUF
    1 Q(I,16) = 0
      DO 100 INS = 1, NQTEST
      IF(Q(I,8).EQ.0) GO TO 100
      IF (INS.EQ.1) GO TO 11
      INSM1 = INS-1
      DO 10 I = 1, INSM1
C           IF PREV INS TO SAME BOM, NOGO
      IF (Q(I,6).EQ.Q(INS,6)) GO TO 100
C           IF STORE AND PREV NOO STORE, NOGO
      IF((Q(I,3).EQ.1).AND.(Q(I,16).EQ.0)) GO TO 100
   10 CONTINUE
C           IF STORE AND DATA NOT AVAIL, NOGO
   11 IF((Q(INS,3).EQ.1).AND.(Q(INS,9).EQ.0)) GO TO 100
C           MARK GO
      Q(INS,16) = 1
      NGO = NGO+1
      IF(NGO.GT.NQGO) RETURN
  100 CONTINUE
      RETURN
      END
```


L. Conway Archives

SUBROUTINE XQEMP

```
      DØ 100 INS=1,NQBUF

    5 IF(Q(INS,16).EQ.0) GØ TØ 100
      IF(Q(INS,8).EQ.0) GØ TØ 100
C                ISSUE INS TØ MEMORY
      BØM=Q(INS,6)
      DEST=Q(INS,15)
      A = Q(INS,4)
      X = Q(INS,5)
      L = Q(INS,1)
      CALL CAUSE(MBUSY,TIME+MEMDLY-2.0,BØM,L,0)
      CALL CAUSE(MFREE,TIME+MEMDLY-1.0,BØM,0,0)
C              IF LØAD, CAUSE DEST LØAD IN MEMDLY CYCLES
      IF(Q(INS,2).EQ.1) CALL CAUSE(LØAD,TIME+MEMDLY,DEST,A,X)
C              REMØVE INS FRØM Q
      QINPT=QINPT-1
      M=NQBUF-1
      IF(INS.EQ.NQBUF) GØ TØ 31
      DØ 30 I=INS,M
      DØ 30 J=1,16
      Q(I,J)=Q(I+1,J)
   30 CØNTINVE
   31 CØNTINVE
      DØ 32 J=1,16
      Q(NQBUF,J)=0
   32 CØNTINVE
      GØ TØ 5
  100 CØNTINVE
      RETURN
      END
```


L. Conway Archives

SUBROUTINE SRQ

```
C
      ENTRY XMBUSY
      BOM = IPAR1
      L = IPAR2
      MEMORY(BOM) = L
      RETURN
C
      ENTRY XMFREE
      BOM = IPAR1
      MEMORY(BOM)=0
      RETURN
C
      ENTRY XLOAD
      DEST = IPAR1
      IF(ABUPS(DEST).NE.1) GO TO 9
      IF( ABUSY(DEST).EQ.1) GO TO 8
      ABUFUL(DEST) =1
      RETURN
    8 ABUSY (DEST)=0
    9 XBUSY (DEST)=0
      RETURN
C
      ENTRY XEAV
      DO 10 I = 1,NQBUF
      IF(Q(I,8).EQ.1) GO TO 10
      Q(I,8) =1
      RETURN
   10 CONTINUE
      A=1
      B=20000
      C=101
      CALL TROUBL( A,B,C)
      RETURN
C
```


L. Conway Archives

IN XACØN

```
C  - - - - TEST FOR SPEC ØPS HERE - - - - -
C        IF STØRE A TEST AVAIL ØF INBUS (STØREBUS)
C        IF AVAIL, SET BUSY. IF NØT, GØ TØ 100

      IF (ADEST(INS,89).NE.1) GØ TØ 27
      IF (AIBBSY(3).EQ.1) GØ TØ 100
      AIBBSY(3) = 1
      GØ TØ 95
C  27 CØNTINUE - - - - - - - - - - - - - - -
```

IN XAEMP

```
C  - - - - - TEST FØR SPEC ØPS HERE - - - - -
C     IF STØRE A SHIP DATA TØ BUFFER ØR Q
C     DEP. ØN STATE ØF BUFFER
      IF (ADEST(INS,89).NE.1) GØ TØ 7
      IF (SDBA(1,2).NE.1) GØ TØ 2
C        NØ STA WAITING. SET DATA IN SDBA
      DØ 3 I=1,32
      IF(SDBA(I,1).EQ.1) GØ TØ3
      SDBA(I,1)=1
    3 CØNTINUE
      GØ TØ 7
C           STA WAITING. DATA TØ Q, SHIFT SDBA
    2 CØNTINUE
      DØ 4 I=1,31
      SDBA(I,1)=SDBA(I+1,1)
    4 SDBA(I,2) = SDBA(I+1,2)
      SDBA(32,1)=0
      SDBA(32,2)=0
      DØ 50 INS = 1,NQBUF
      IF(Q(INS,3).NE.1) GØ TØ 50
      IF(Q(INS,4).NE.1) GØ TØ 50
      IF(Q(INS,9).EQ.1) GØ TØ 50
      Q(INS,9)=1
      GØ TØ 7
   50 CØNTINUE
C - - - 7 CØNTINUE - - - - - - - - - - - - -
```


L. Conway Archives

INXXCØN

```
C  --- TEST FOR SPEC OPS HERE ----------
C       IF L/S, TEST AVAILABILITY ØF QUEUE
      IF((XSØR(INS,89).NE.1).AND.(XDEST(INS,89).NE.1))GØTØ27
      IF(QINPT.GT.NQBUF) GØ TØ 100
   27 CØNTINUE
C  ----------
```

INXXEMP

```
C  --- TEST FOR SPEC OPS HERE ----------
C       IF L/S  SHIP TØ QUEUE
      IF((XSØR(INS,89).NE.1).AND.(XDEST(INS,89).NE.1))GØTØ 7
      CALL CAUSE(EAV,TIME+1.0,Q 0,0)
      IN = QINPT
      QINPT=QINPT+1
      Q(IN,1) = XBUFF(INS,1)
      Q(IN,4) = XBUFF(INS,7)
      IF(Q(IN,4).NE.1)  Q(IN,5)=1
      Q(IN,7) = XBUFF(INS,6)
      Q(IN,6) = MØD(Q(IN,7),[redacted]NBØX)+1
      IF(XSØR(INS,89).EQ.1)  Q(IN,2)=1
      IF(XDEST(INS,89).EQ.1)  Q(IN,3)=1
      IF((Q(IN,5).EQ.1).AND.(Q(IN,3).EQ.1))  Q(IN,9)=1
      DØ  8  REG=1,NXREGS
      IF(Q(IN,2).NE.1) GØ TØ 88
      IF(XDEST(INS,REG).EQ.1)  Q(IN,15)=REG
   88 CONTINUE
   8  CØNTINUE
C          IF STORE A, GET DATA OR SET WAIT
      IF((Q(IN,3).NE.1).ØR.(Q(IN,4).NE.1)) GØ TØ 7
C     IF(SDBA(1,1).NE.1) GØ TØ 6
                          DATA AVAIL
      Q(IN,9)=1
      DØ  50 I=1,31
      DØ  50 J=1,2
   50 SDBA(I,J) = SDBA(I+1,J)
      SDBA(32,1)=0
      SDBA(32,2)=0
      GØ TØ 7
```


L. Conway Archives

```
      6  CONTINUE
C            DATA NOT AVAIL. SET FIRST FREE WAIT BIT
         DO 4 I=1,31
         IF (SDBA(I,2).EQ. 1) GO TO 4
         SDBA(I,2) = 1
         GO TO 7
      4  CONTINUE
         A=1
         B=20000
         C=102
         CALL TROUBL(A,B,C)
      7  CONTINUE
C     ------------------------------------------
C        IGNORE SURFACE AS DEST
         IF (REG.EQ.89) GO TO 10
```

IN XXRET

```
C        IGNORE L/S BUSES
         IF((BUS.EQ.5).OR.(BUS.EQ.6)) GO TO 10
```


L. Conway Archives

STORE A

SDBA

I,1	I,2
# DATAV IN SDBA AVAIL TAG	# STA WAITING ON Q FOR DATA

XEMP X

AEMP A

If Q AVAIL ISSUE

If DATAREG AVAIL, ISSUE

(1) If waiting CNT ≥ 0
INCR waiting CNT

(2) If waiting CNT < 0
Set DATAV and
Set CNT = CNT + 1

(1) If waiting CNT ≥ 0
Search for 1st STA on Q
Place DATAV
Decr CNT

(2) If waiting CNT ≤ 0
Set = CNT − 1

equiv to

(1)' (If) if SDBA(1,2) = 1
✓ or SDBA(1,1) + (1,2) = 0
Set 1st AVAIL WAIT (I,2)

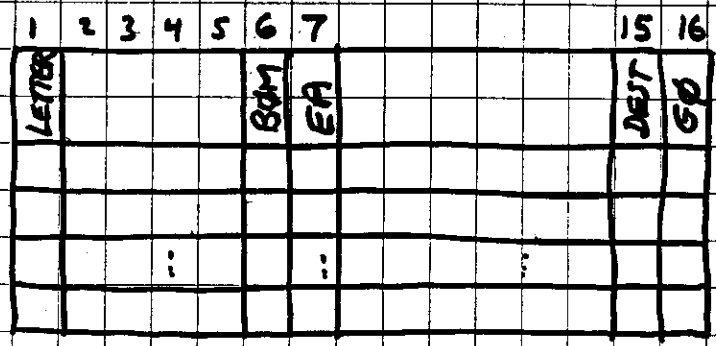
(2)' if SDBA(1,1) = 1
✓ Set DATAV on Q
& shift up SDBA

(1)' If SDBA(1,2) = 1
Search for 1st STA, Place DATAV
Shift up SDBA

(2) If SDBA (1,1) = 1
or SDBA (1,1) + (1,2) = 0
place 1 in 1st avail DATA value (I,1)

L. Conway Archives

IQ([redacted]16): INSTRUCTION FETCHING:



I. Interface with ØL: IFADD, IFDST, IFRTN
IFADD = INS FETCH ADDRESS. IFDST = IB DEST AND WILL BE NON ZERO IF VALID REQST PRESENT AT END OF CYCLE (IS Reset on taking reqst). IFRTN ⇒ PLACE DEST HERE ON COMPLETION OF FULL INST FETCH.

II. QCØN After DQ runs, run IQ. Compare 4 int1k against GØ DATA REQ BOMS. Issue up to 4 by marking GO.

III. QEMP After DQ issue, run IQ issue. CAUSE MBUSY, MFREE AT APPROP. Times with L=t. REMOVE OP issued. IF last op issued, cause RTN at approp time to place Dest into IFRTN. (see below-filling)

IV RTN Event to set DEST = IPAR1
IFRTN = DEST

V Filling of IQ: IN QEMP: After QEmp empties IQ, examine interface to see if reqst present. If so, and if IQ empty, place request in IQ expanding addresses and decoding BØMS setting letter, etc. Then zero incoming interface.

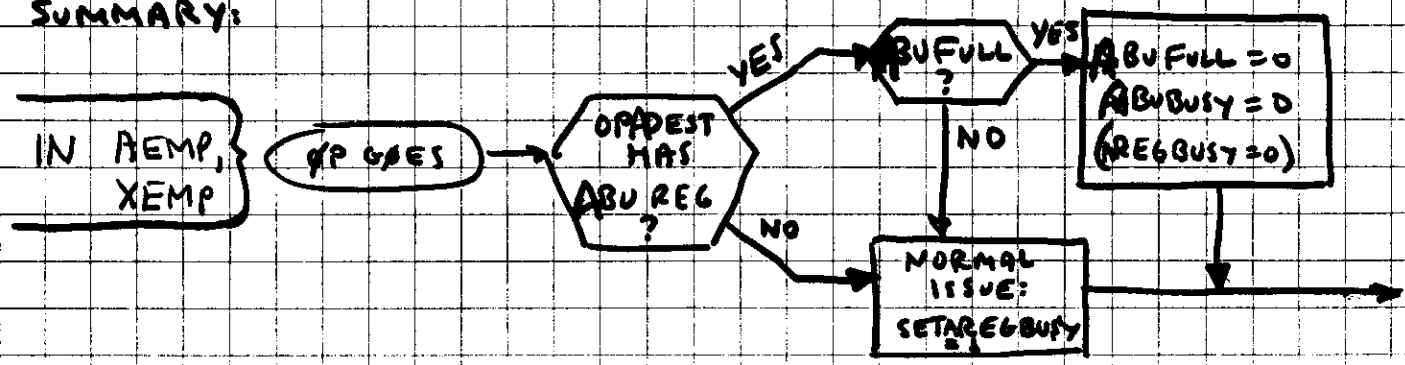
L. Conway Archives

HANDLING THE BACK-UP REGISTERS

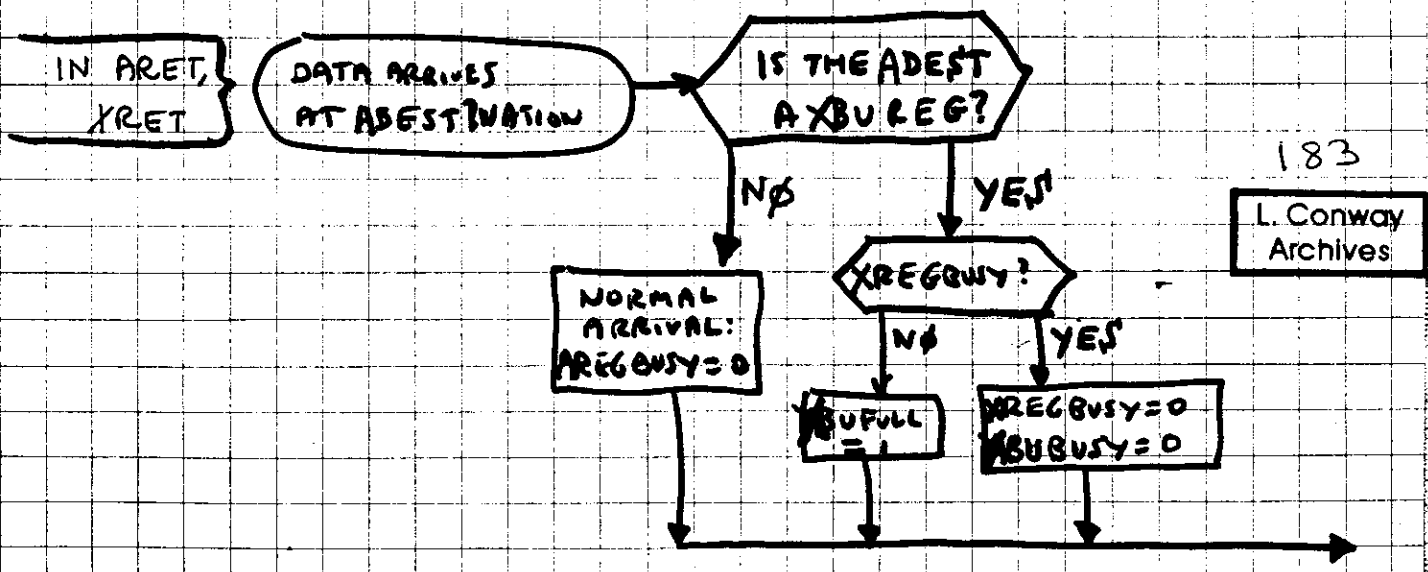
1. ØP ISSUANCE: ISSUANCE OF OP WHOSE DEST HAS NØ BACK-UP IS UNCHANGED. i.e. ISSUE LA in X has ABU AS DEST, Thus dest has no back-up and handling is normal.

   However if DEST has a back-up, such as for AREPL, check if back-up is full. If not, normal handling. If sØ, don't set REGBUSY, but set BUFULL=0 & BUBUSY=0 (corresp. to move of BACK-UP to FRONT.

SUMMARY:



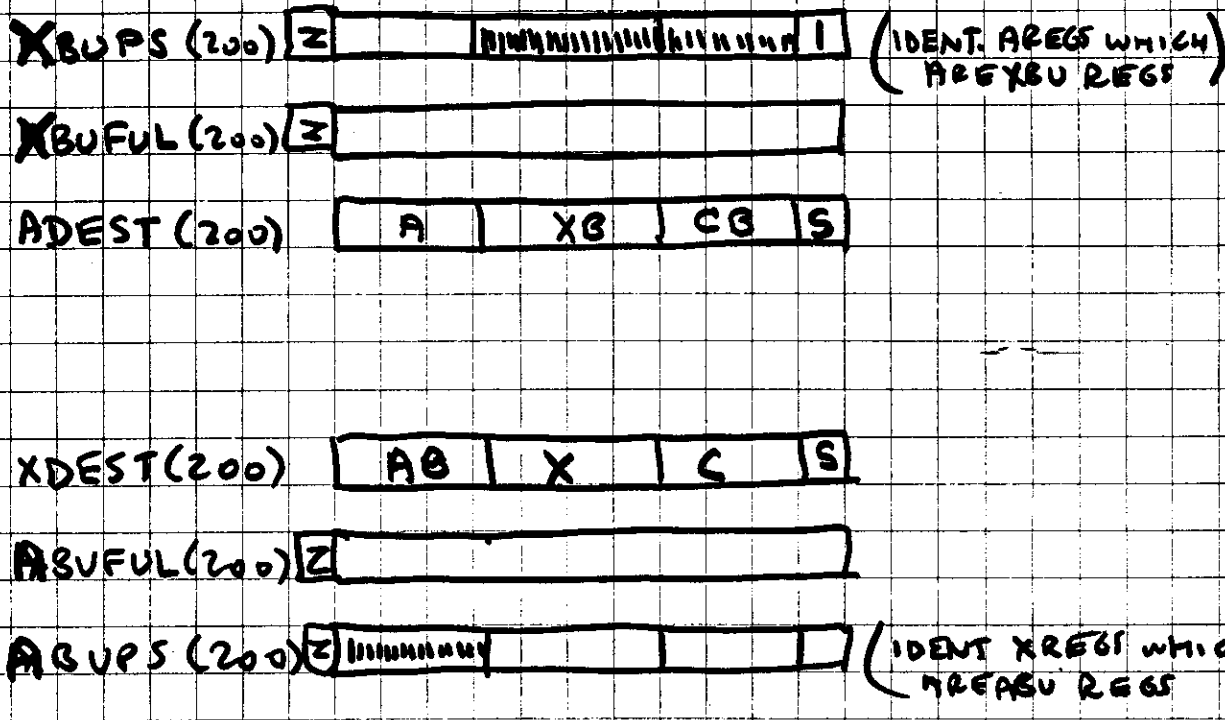
2. DATA ARRIVES AT A DEST:



L. Conway Archives

HANDLING THE BACK-UP REGS (CONT)

RELEVANT VARIABLES:



EXAMPLE:

DOES ADEST HAVE BUREG? ≡ IS ABUPS = 1?

IS ADEST AXBUREG? ≡ IS XBUPS = 1?

L. Conway Archives

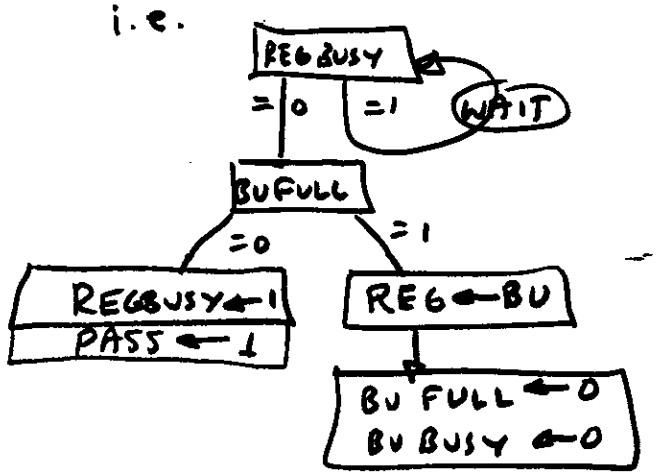
BU BUSY
BU FULL
REG BUSY

REPLACE

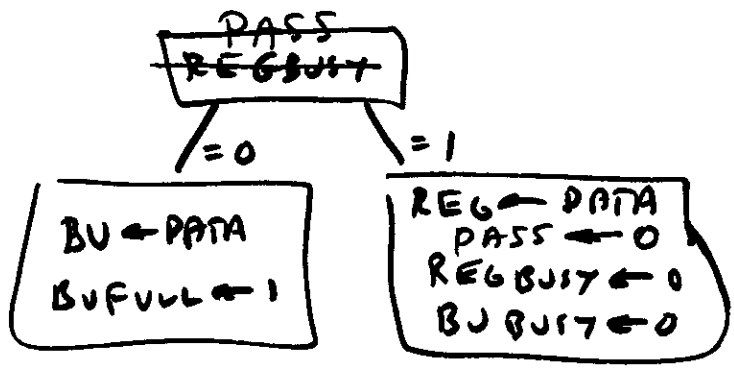
(TREAT AS ANY OP WITH REG AS DEST. IF BU FULL, THEN MOVE UP RIGHT AWAY)

WAIT FOR ALL USES TO FINISH

i.e.

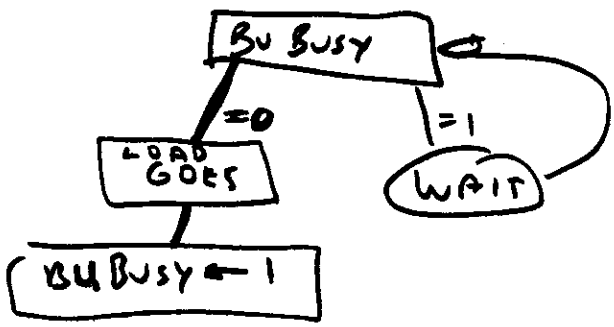


ARR. OF DATA



LOAD A INX

(TREAT AS ANY OP WITH BU AS DEST)



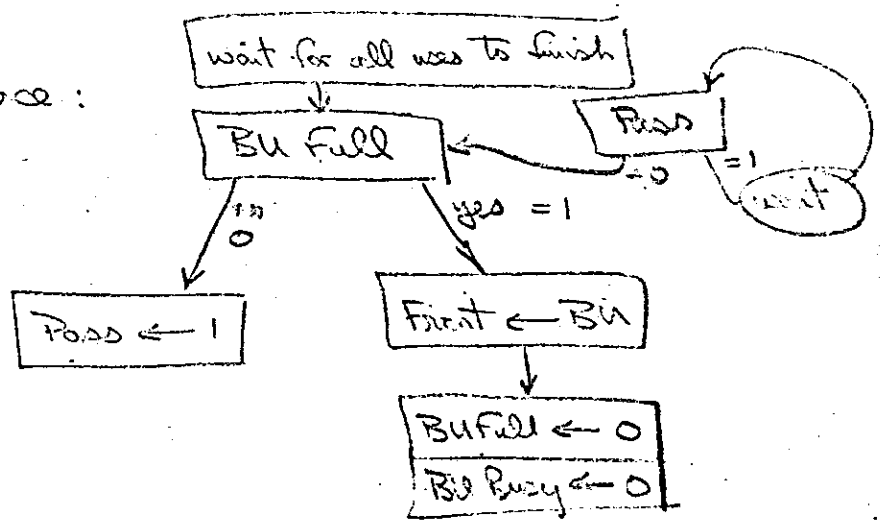
L. Conway Archives

Homan
7/25/66

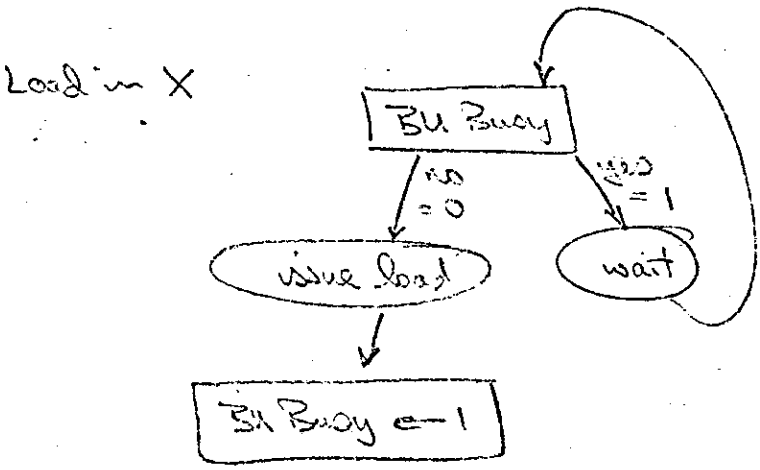
Back Up Registers

BU Busy
BU Full
Pass

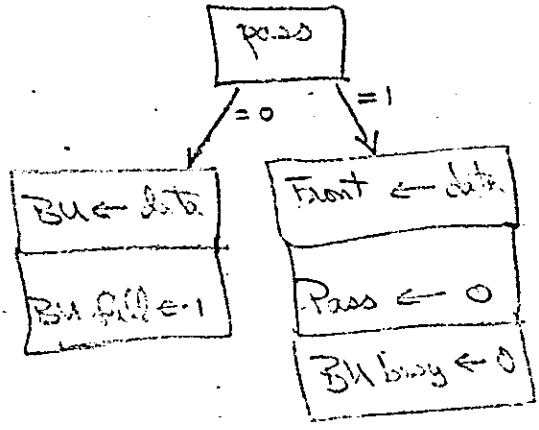
replace:



Load in X

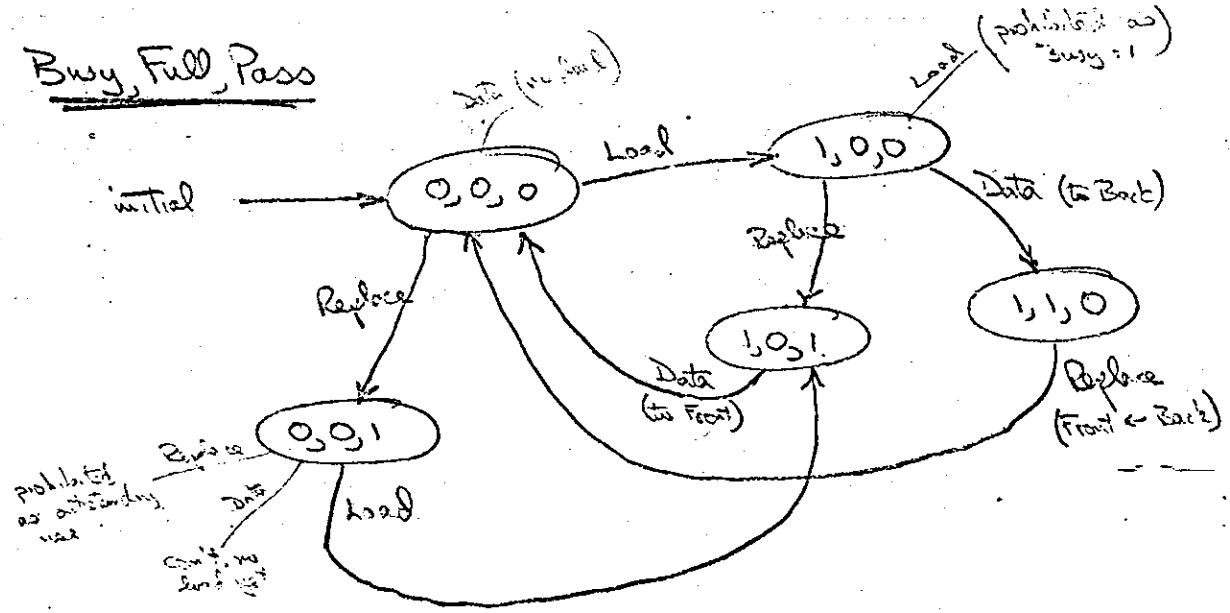


arrival of data



L. Conway Archives

Busy, Full, Pass



Notes

1) data arrival must follow load
2) load, replace in any order
3) replace, replace illegal as have outstanding use (ie data arrival)

L. Conway Archives

X unit

Load 1

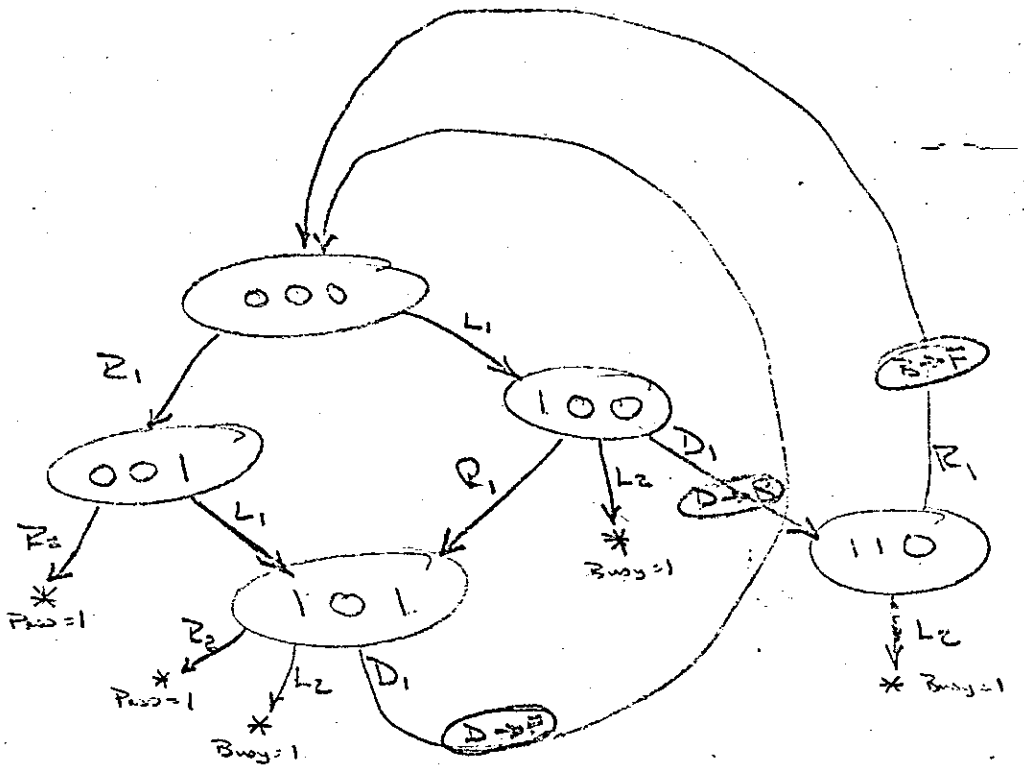
Load 2

A-unit

Repl 1

Repl 2

Busy, Full, Pass



at 1st interrupt: both L and R empty and data returned

hence only 000 possible

at 2nd interrupt: all 5 states possible:

data miss R-2 split possible

missing data miss L-D split possible

L. Conway Archives

SKIP EXECUTION

I INTLK : One skip at a time. Execute in order with respect to all starred ops, skips.

i.e. use SKIP OP TAG, SKIP FLAG TAG as mutual interlocks. NO SKIP CAN PASS FLAGGED OPS, NO FLAGGED OPS MAY PASS A SKIP, SKIPS IN ORDER.

INTERLOCK ON CONDITIONS AVAILABLE.

X : INTLK ON SHT AVAIL (NEXT SR = 0)
A : INTLK ON SHT RESOLVED (NEXT SR = 1)

II EXECUTION:

X : PLACE SR, ST or $\overline{ST}$; INCR X POINTER

A : CLEAR SR, ST; INCR A POINTER

ALL STARRED OPS: IF NO PRECEDING SKIP OPS ON SCAN:

(i) IF ST = 1, NOP AND MARK GO THE STARRED OPS (.1)

(ii) IF ST = 0, REMOVE SKIP FLAG AT END OF CYCLE (SCAN)

L. Conway Archives

EXECUTION OF EXIT INSTRUCTIONS

(A & X UNITS)

I INTLK +GØ: EXITS ONE AT A TIME AND IN ORDER. DO THIS WITH SPECIAL INTLK BIT IN X(A)BUFF FOR EXITS WHICH INTLKS ALL CODE BELOW. EXITS INTLK ON ANY BRANCHES ABOVE. INTLK ON ER. INTLK ON NORMAL S, BUS, FAC.

II EXECUTION: (i) AT END OF .1 SCAN CHECK FOR ANY NØGØ EXITS IN STACK. IF ANY, SET XHOLDT (AHOLDT) TO ONE. OTHERWISE ZERØ

(ii) ALSO AT END OF .1 SCAN, CHECK FOR GØ EXIT WITH ET. IF ANY, SET XFRCT (AFRCT) TØ ONE. OTHERWISE ZERØ.

(iii) IF $\overline{ET}$ EXECUTION PERF BY MERELY GOING AT END OF CYCLE. THUS INTLK ON ØPS BELØW IS GØNE AT NXT CYCLE

(iv) IF ET, NØP AND MARK GØ AT END OF .1 SCAN ALL ØPS BELØW EXIT.

L. Conway Archives

BRANCH & EXIT HANDLING

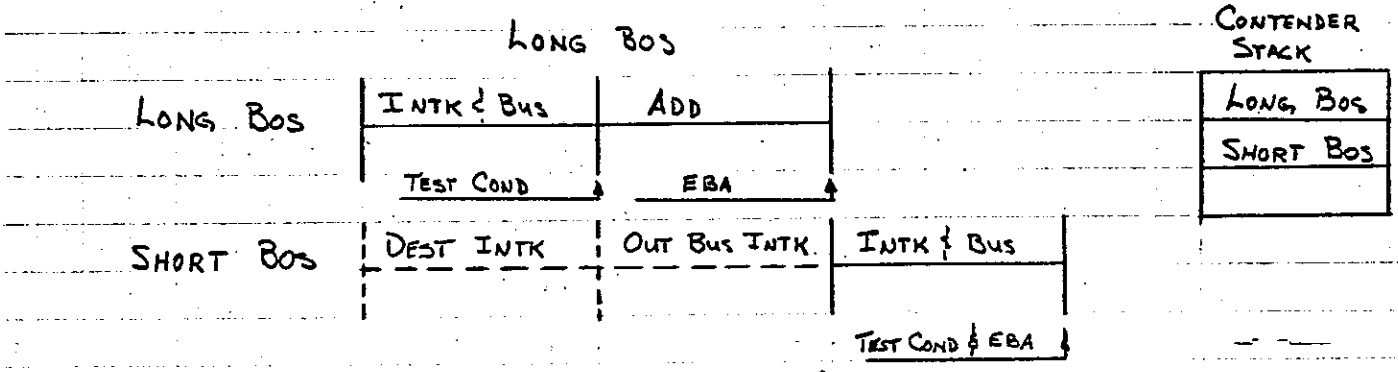
3/27/67

NEW BRANCH INFO

I. BRANCH INST INTK

A. ALL BOS INST'S HAVE COMMON DEST (EHT)

B. ALL BOS INST'S USE COMMON OUT (RETURN) BUS (TO EHT)



- IF LONG BOS IS UNSUCCESSFUL SHORT BOS COULD BE STARTED ONE CYCLE SOONER.
- WITH THE PROPER SELECTION OF BRANCH RESULTS, UP TO THREE BOS'S COULD BE EXECUTED PER CYCLE. (MUST MAINTAIN ORDER, CONDITIONAL INTK'S)

II. EXIT INST INTK

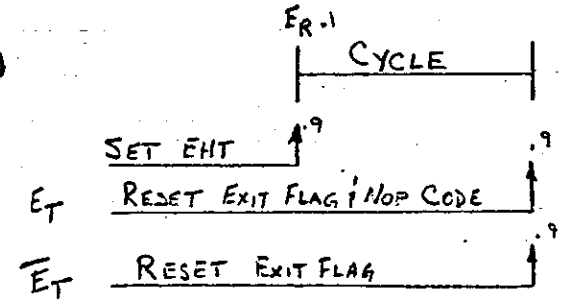
A. ALL EXITS INST'S HAVE COMMON SOURCE (EHT)

III SPECIAL INTK

A. ALL CODE BELOW AN EXIT FLAG IS INTK'D.

IV EXIT EXECUTION

A. CONTENDER



L. Conway Archives

V EXIT HISTORY TABLE

E_R	B_E	E_T	EBA

E_R = EXIT RESOLVED
B_E = BRANCH EXECUTED
E_T = EXIT TAKEN
EBA = LOW ORDER 3 BITS OF THE EFFECTIVE BRANCH ADDRESS.

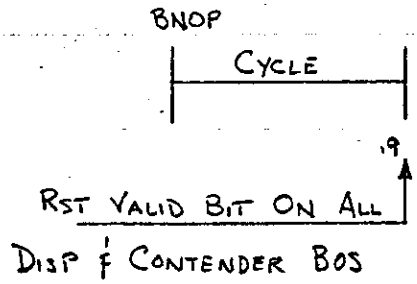
VI DEFINITIONS:

1) BOSC = [$\overline{\text{CONTENDER EXIT}}$ ∧ (DISP BOS AHEAD OF 1ST EXIT DISP EXIT ∨ ANY CONTENDER BOS)] ∨ [CONTENDER EXIT ∧ ANY CONTENDER BOS AHEAD OF 1ST CONTENDER EXIT]

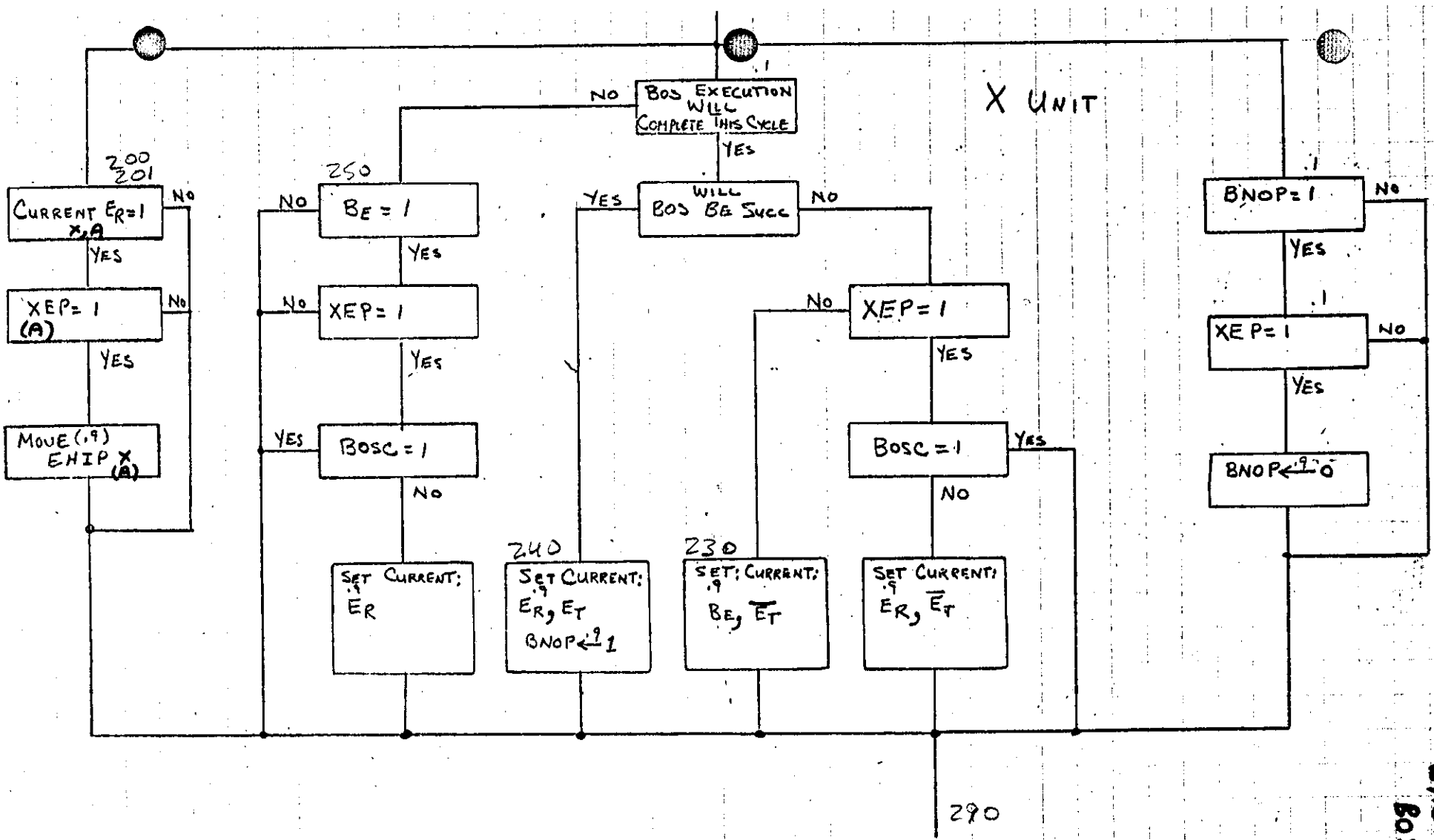
2) XEP - EXIT PASSED TO X UNIT DISPATCHERS =

SEE FLOW-CHART

3) BNOP -



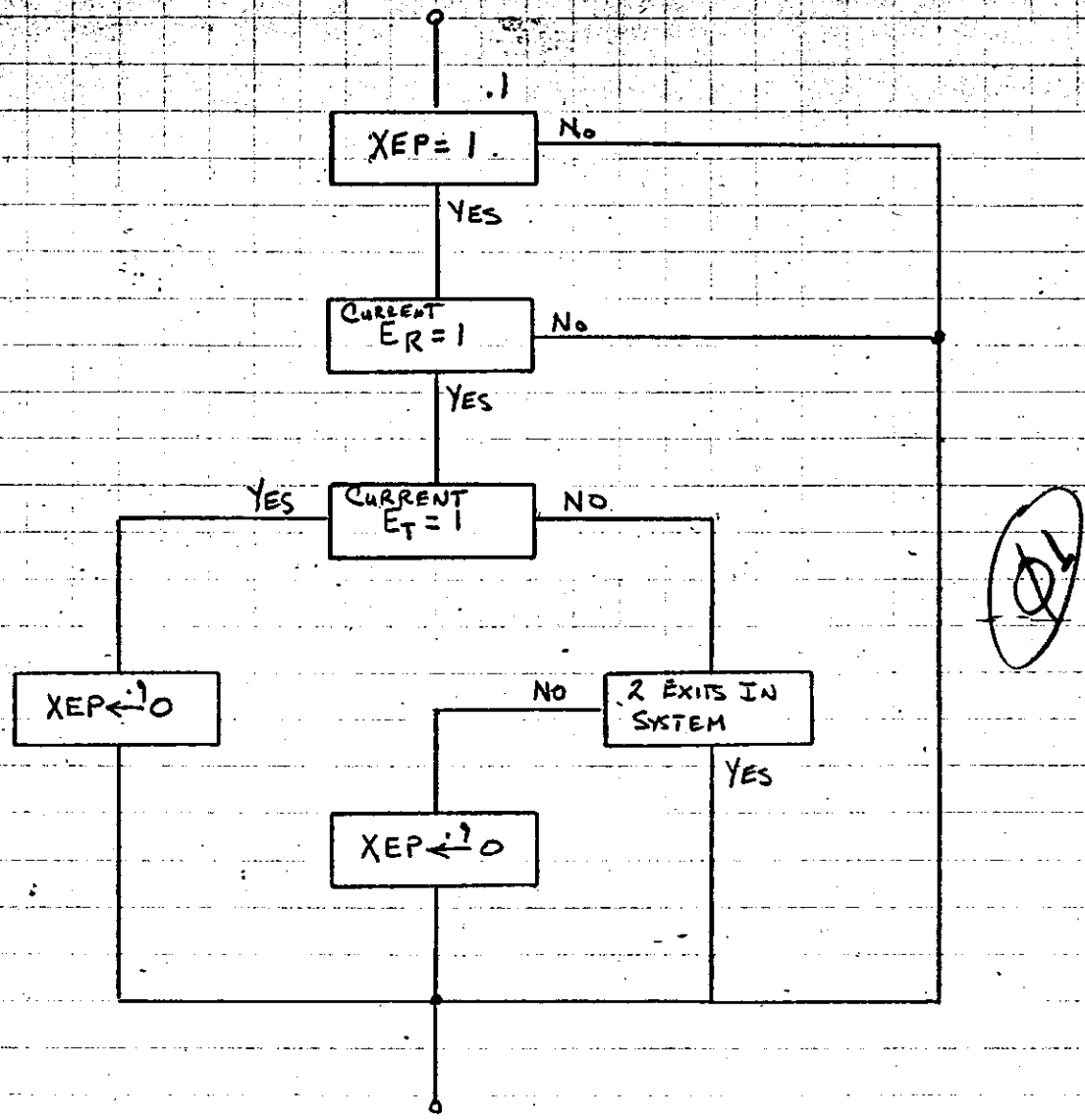
L. Conway Archives



EXECUTION OF BOS INSTRUCTIONS

3/27/67

L. Conway Archives



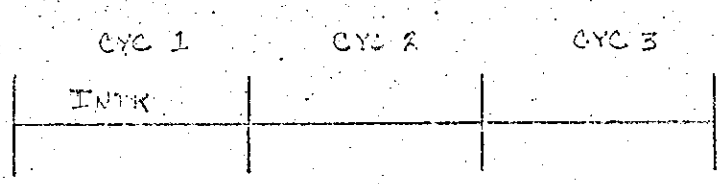
L. Conway Archives

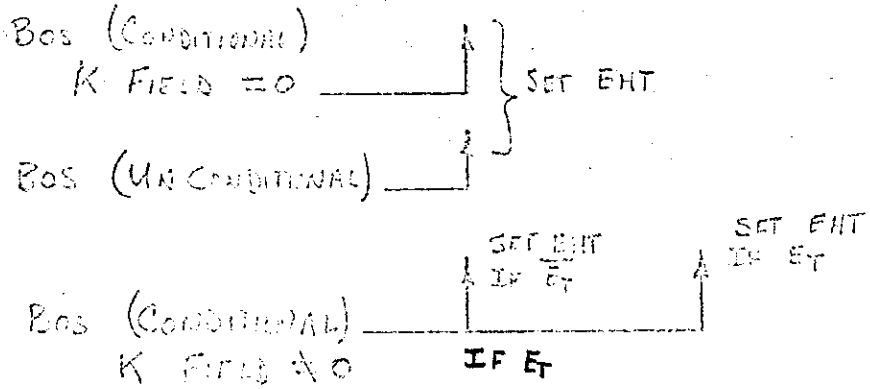
3/27/67

BRANCH IN X

OLD BRANCH INFO

- ONE BOS ~~EXECUTED~~ GO PER CYCLE





NOTE: SETTING OF EXIT HISTORY TABLE (E_R & E_T BITS) IMPLIES EBA VALID.

- THE SETTING OF THE EHT (E_T) IS DONE ONLY WHEN THE LAST BOS JUST PRIOR TO THE EXIT IS RESOLVED TO BE UNSUCCESSFUL.

IT EXIT INSTRUCTION

WHEN EXIT IS COMBINED WITH MLX, MXP, AXC

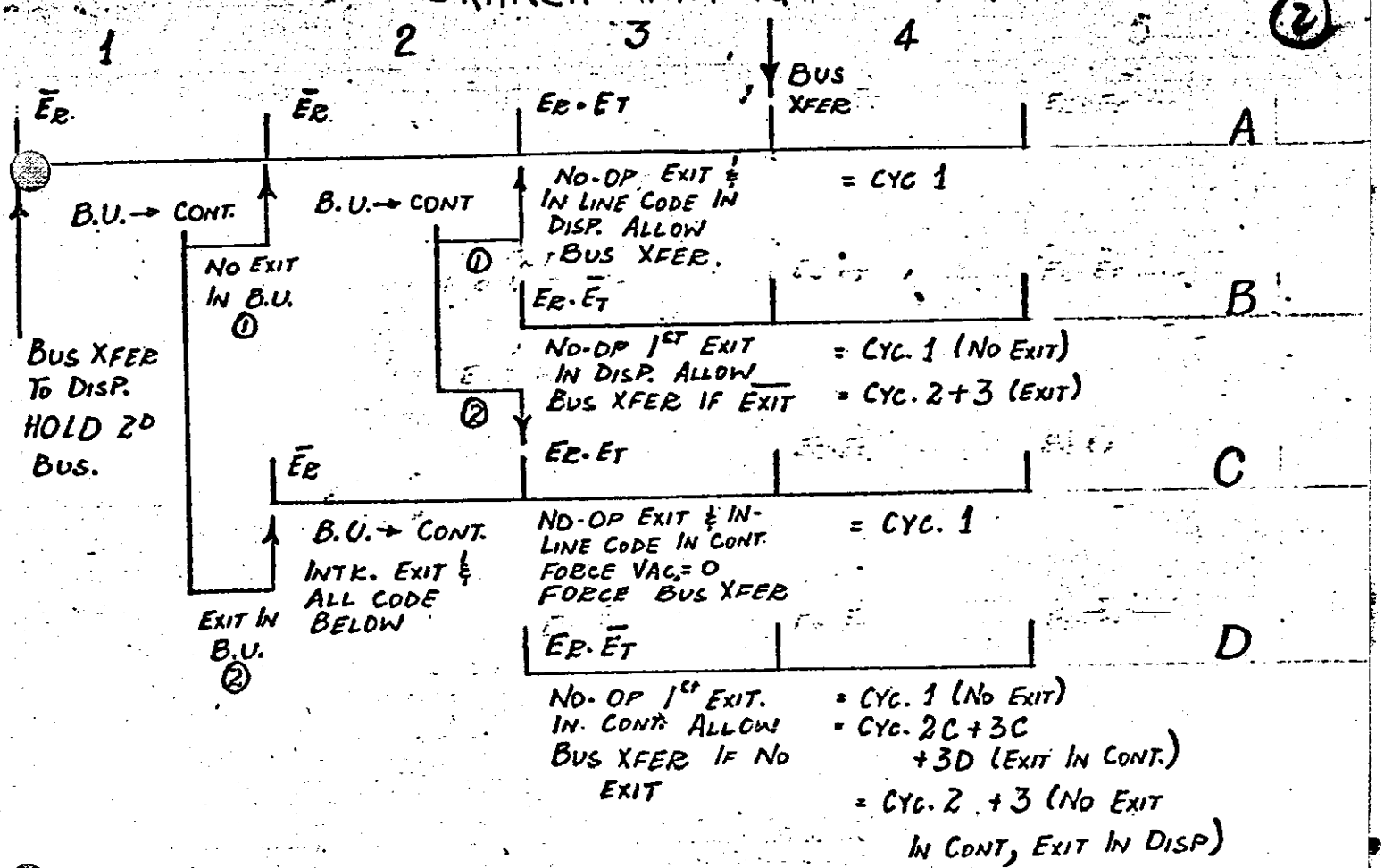
1) EXIT IS NEVER SKIPPED
2) MLX, MXP, AXC FUNCTION OF EXIT IS UNCHANGED
3) NOTE: COMPLETE EXIT JUST WHEN EXIT IS OMITTED

195

L. Conway Archives

3/17/67
BGG

BRANCH TIMING



IF E_R AT BUS-XFER TIME, ENTER AT CYC. 3A ($E_R \cdot E_T$) OR 3B ($E_R \cdot \bar{E}_T$

L. Conway Archives

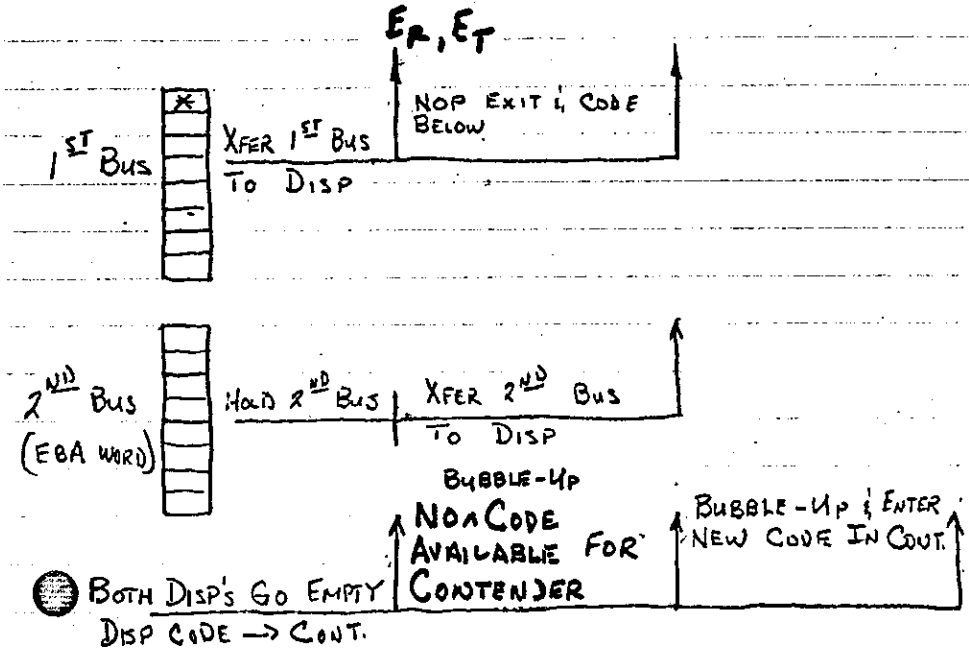
BRANCH TIMING

E_R E_T

CYC 1	CYC 2	CYC 3	CYC 4	CYC 5

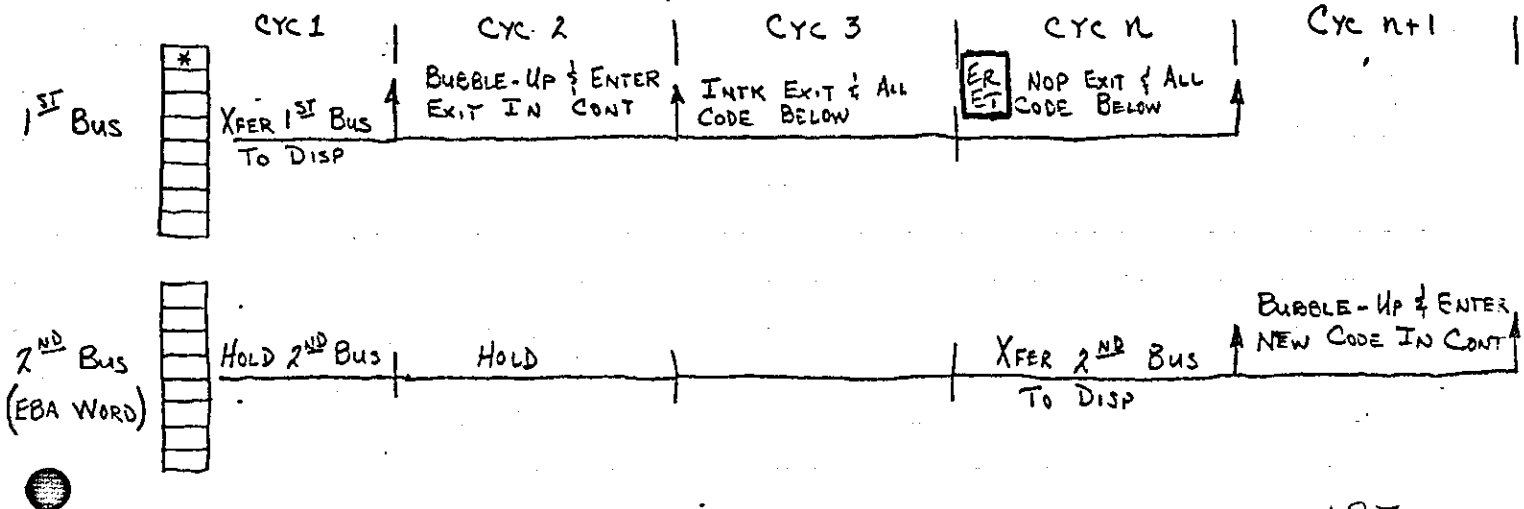
CASE I

BRANCH RESOLVED (TAKEN) BEFORE EXIT IS FOUND



CASE II

BRANCH RESOLVED (TAKEN) AFTER EXIT IS FOUND



L. Conway Archives

BRANCH

I. EXIT HANDLING

A. IB BUS

1. ANY EXIT ON 1ST BUS STOP 2ND BUS
2. ALLOW 1ST BUS XFER TO DISPATCHER (HOLD FURTHER BUS XFER

B. DISPATCHER (exit pres. + resolved)

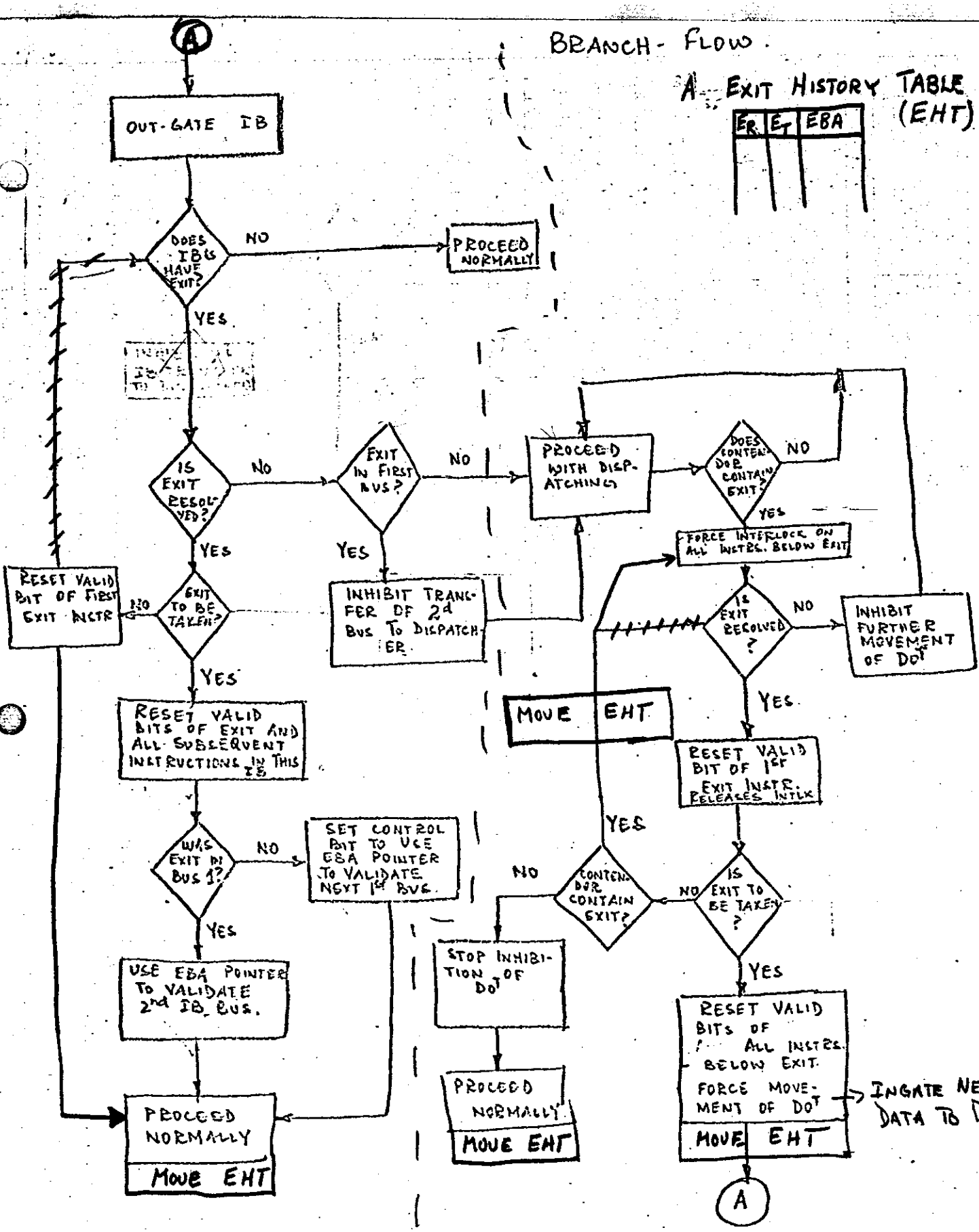
1. IF NO CONTENDER EXIT:
   - A. $E_R \overline{E}_T$ - RST 1ST EXIT IN EACH DISP (ONLY ONE DISP CAN HAVE EXIT)
   - B. IF NO 2ND DISP EXIT - RELEASE BUS XFER HOLD
2. $E_R E_T$
   - A. RESET 1ST EXIT & ALL CODE BELOW
   - B. MASK 1ST EXIT & ALL CODE BELOW FROM DISPATCHER BUBBLE-UP
   - C. RELEASE BUS XFER HOLD (CAN'T FORCE BUS XFER BECAUSE ALL CODE ABOVE EXIT MAY NOT HAVE REACHED CONTENDER STACK)

C. CONTENDER

1. INTK 1ST EXIT & ALL CODE BELOW IF $\overline{E}_R$ ALWAYS INTK 2ND EXIT & ALL CODE BELOW
2. $E_R E_T$
   - A. RESET 1ST EXIT & ALL CODE BELOW
   - B. MASK 1ST EXIT & ALL CODE BELOW FROM CONTENTION
   - C. FORCE BUS XFER (1ST & 2ND BUS)
   - D. PREVENT DISP BUBBLE-UP CODE FROM ENTERING CONTENDER STACK
3. $E_R \overline{E}_T$
   - A. RESET 1ST EXIT
   - B. RELEASE INTK ON ALL CODE BETWEEN 1ST & 2ND EXIT (2ND EXIT WILL BECOME 1ST ON NEXT CYCLE)
   - C. IF NO DISP EXIT OR 2ND CONTENDER EXIT RELEASE BUS XFER HOLD

L. Conway Archives

BRANCH - FLOW.



A - EXIT HISTORY TABLE (EHT)

ER	ET	EBA

L. Conway Archives

8/4 misc. new possibilities

REPLACES

X-UNIT

MAX OPS MUST MATCH DX FIELDS WITH X-~~BY~~PASS ~~BUSY~~ VECTOR ALONG WITH S/D INTERLOCK

IF A-X BUFFER "FULL" & BUFFER ID MATCHES DX FIELDS MOVE A-X BUFFER CONTENTS THROUGH ~~BACK-UP~~ now BUSS TO X STACK

IF A-X BUFFER ID DOESN'T MATCH DX FIELD SET X-~~BY~~PASS ~~BUSY~~ VECTOR & X REG. WAITING VECTOR

A-UNIT

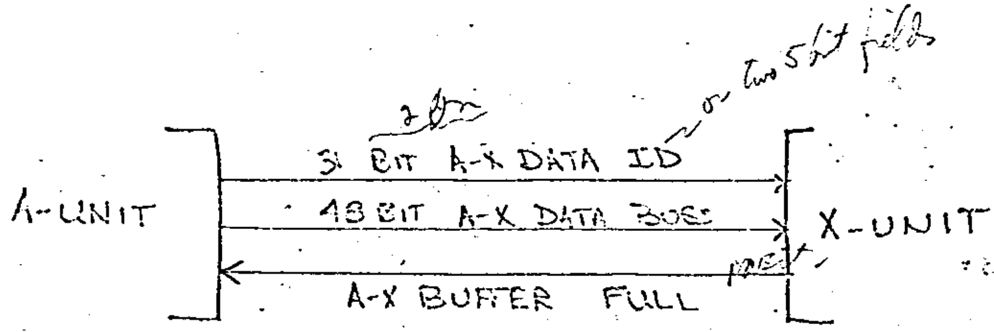
& its source A Reg & Sub etc.

MAX OPS INTERLOCK WITH A-X BUFFER "FULL"

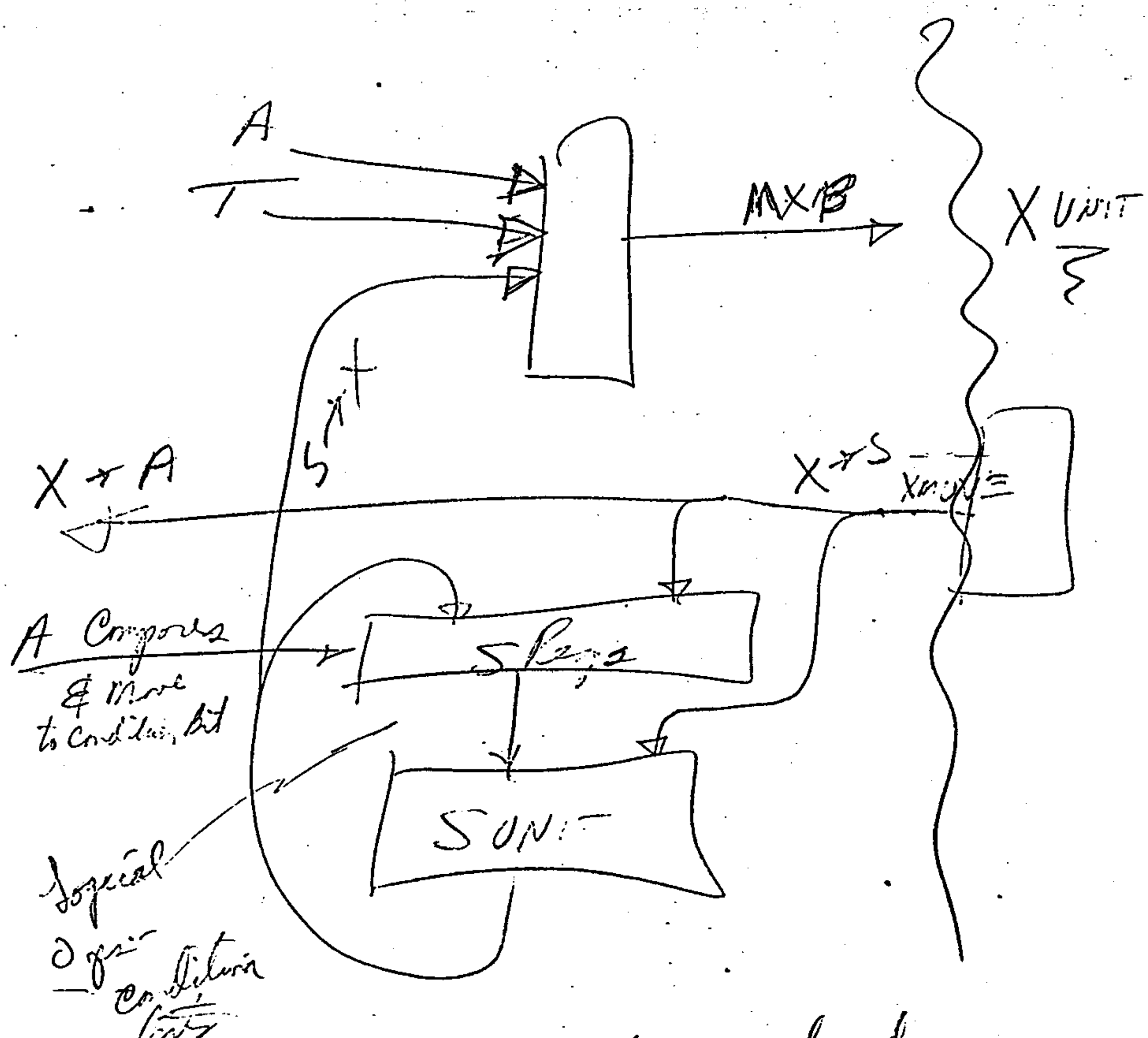
DATA SENT ON A-X DATA BUSS ALONG WITH DX ID's IF UNINTERLOCKED

IF DXID & X-~~BY~~PASS BUSY VECTOR [illegible] MATCH THEN A-X BUFFER BYPASSED & (X REG BUSY RESET) & DATA ADVANCES TO STACK VIA BACK-UP BUSS.

IF DXID & X-BYPASS BUSY VECTOR DON'T MATCH THEN A-X BUFFER LOADED WITH DATA & DXID FROM A-X DATA BUSS & A-X BUFFER "FULL" ACTIVATED.



L. Conway Archives



Decode & Bus Overlapped
with Sensing

X↔S Mixed Ops.

Conway & Homan
8/14/67 201

L. Conway
Archives

Load Index & Count (LXC) $i \rightarrow j$ field { Special case - multi step

Store Multiple Index $i+1 \rightarrow j$ field. { "

Shifts (Half word format) $j, k \rightarrow jk$ field & expand while buffering { This can be done for all ops

$i+1 \rightarrow j$ field.

Shifts (Full word format) $j \rightarrow i$ field

$j+1 \rightarrow j$ field

Index Arithmetic (Not Constant) $k \rightarrow h$ field & expand } This can be done for all Half word ops except MLX & EXITL

Set Positive Index
Set Negative Index } $j \rightarrow k$ field. { to use T/C gating into adder

Branches. & SIU $k \rightarrow j$ field { for $X^k + h$ & ignoring condition register problems.

Move Location to Index (MLX) also (EXITL) $j, k \rightarrow jk$ field { This can be done for all ops

X-UNIT
OUT-GATE FIELD SWAPPING

Move Condition Bit to Index Bit $i \rightarrow j$

7/26/57 mwk 202

L. Conway Archives

HEXADECIMAL ARITHMETIC

ADDITION TABLE

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
2	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
3	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12
4	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
5	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
6	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15
7	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16
8	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17
9	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19
B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A
C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

MULTIPLICATION TABLE

1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	1E	2B	3C	48	5A	69	78	87	96	A5	B4	C3	D2	E1

203
L. Conway Archives

Appendix D. Powers of Two Table

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25

204
L. Conway
Archives

PROG SIZE = 370,000 Bytes

(1) Reduce PROG TO 500x30:
will easily yield = 30000 Byte

(2) Make TAGS 1byte instead of
1/2 word:
will yield 256 x 70/2 = 8960 by
with some prog problems

(3) MAKE ØBUF 1byte instead of
1/2 word. This will take some
experimenting but yields:
35800/2 = 17900 bytes

TOTAL POSS REDUCTION BY COMMON
REDUCTION = 56,860 Bytes

L. Conway
Archives

Appendix E. Hexadecimal-Decimal Conversion Table

The table in this appendix provides for direct conversion of decimal and hexadecimal numbers in these ranges:

HEXADECIMAL	DECIMAL
000 to FFF	0000 to 4095

For numbers outside the range of the table, add the following values to the table figures:

HEXADECIMAL	DECIMAL
1000	4096
2000	8192
3000	12288
4000	16384
5000	20484
6000	24576
7000	28672
8000	32768
9000	36864
A000	40960
B000	45056
C000	49152
D000	53248
E000	57344
F000	61440

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
010	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
020	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
030	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
040	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
050	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
060	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
070	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
080	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
090	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A0	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B0	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C0	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D0	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E0	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F0	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
100	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
110	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
120	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
130	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
140	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
150	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
160	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
170	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
180	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
190	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A0	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B0	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C0	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D0	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E0	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F0	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511

206
L. Conway Archives

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
200	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
210	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
220	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
230	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
240	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
250	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
260	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
270	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
280	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
290	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A0	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B0	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C0	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D0	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E0	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F0	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
300	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
310	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
320	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
330	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
340	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
350	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
360	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
370	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
380	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
390	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A0	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B0	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C0	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D0	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E0	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F0	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
400	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
410	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
420	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
430	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
440	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
450	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
460	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
470	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
480	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
490	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A0	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B0	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C0	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D0	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E0	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F0	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
500	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
510	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
520	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
530	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
540	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
550	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
560	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
570	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
580	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
590	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A0	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B0	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C0	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D0	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E0	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F0	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

L. Conway Archives

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
600	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
610	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
620	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
630	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
640	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
650	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
660	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
670	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
680	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
690	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A0	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B0	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C0	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D0	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E0	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F0	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
700	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
710	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
720	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
730	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
740	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	~~1870~~	~~1871~~
750	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
760	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
770	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
780	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
790	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A0	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B0	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C0	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D0	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E0	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F0	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
800	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
810	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
820	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
830	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
840	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
850	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
860	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
870	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
880	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
890	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A0	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B0	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C0	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D0	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E0	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F0	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
900	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
910	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
920	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
930	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
940	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
950	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
960	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
970	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
980	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
990	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A0	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B0	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C0	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D0	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E0	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F0	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

208
L. Conway Archives

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A00	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A10	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A20	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A30	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A40	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A50	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A60	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A70	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A80	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A90	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA0	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB0	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B00	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B10	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B20	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B30	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B40	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B50	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B60	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B70	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B80	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B90	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA0	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB0	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC0	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD0	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE0	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF0	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C00	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C10	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C20	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C30	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C40	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C50	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C60	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C70	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C80	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C90	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA0	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB0	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC0	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD0	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE0	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF0	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D00	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D10	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D20	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D30	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D40	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D50	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D60	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D70	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D80	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D90	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA0	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB0	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC0	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD0	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE0	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF0	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583

L. Conway Archives

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E00	3584	3585	3586	3587	3583	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E10	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E20	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E30	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E40	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E50	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E60	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E70	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E80	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E90	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA0	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB0	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC0	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED0	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE0	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF0	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F00	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F10	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F20	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F30	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F40	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F50	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F60	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F70	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F80	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F90	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA0	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB0	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC0	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD0	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE0	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF0	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

210
L. Conway
Archives

```
LEVEL 2 FEB 67                    OS/360  FORTRAN H                    DATE  67.278/15.54.28

      COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

ISN 0002          IMPLICIT INTEGER*2(A-Z)
ISN 0003          COMMON          TIME,          IPAR1,         IPAR2,         IPAR3,
                 A AINPT,         NABUF,         ABUS(50),      XINPT,         NXBUF,
                 B XBUS(50),      IFADD,         IFDST,         IFRTN,         BRXP,
                 C BRAP,          ER(8),         BE(8),         ET(8),         NBBUF,
                 D AHOLDT,        XHOLDT,        AFRCT,         XFRCT,         BOSC,
                 E BNOP,          XFP,           AFP,           PH1(100),      PRINT,
                 F FSTADD,        NODOT,         NOPSC,         NDBUS,         NADSP,
                 G NXDSP
ISN 0004          COMMON/RLS/     FIRST,         NAREGS,        NXREGS,        NABUS,
                 A NXBUS,         STATS,         ACON,          XCON,          AEMP,
                 B XEMP,          MXO,           AFULL(12),     XFULL(12),     AGO(12),
                 C XGO(12),       NAGO,          NXGO,          NATEST,        NXTEST,
                 D NAFAC,         NXFAC,         ABUSYZ,        ABUSY(200),    XBUSYZ,
                 E XBUSY(200),    ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
                 F XSOR(12,200),ADEST(12,200),XDEST(12,200),                    AFAC(12,15),
                 G XFAC(12,15),   AFACSC(4,15,20),ARET,          XFACSC(4,15,20),XRET,
                 H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
                 I AOBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
                 J AFDLY(15),     XFDLY(15),     AFOBUS(15),    XFOBUS(15),    NSLOT,
                 K ABUPSZ,        ABUPS(200),    XBUPS(200),    ABUFUL(200),   XBUFUL(200),
                 L Q(16,16),      SDBA(32,2),    NQBUF,         NQTEST,        NQGO,
                 M QINPT,         QCON,          QEMP,          MBUSY,         MFREE,
                 N LOAD,          MEMDLY,        MEMORY(16),    NBOX,          EAV,
                 O MXTIME,        OUTLVL,        IQ(4,16),      RTN,           LONGBR,
                 P SR(8),         ST(8),         SKXP,          SKAP,          NSBUF,
                 Q APASS(200),    XPASS(200),    OUT(2),        JOB(6),        SSTOP,
                 R MEMCNT(16),    ABOX(15),      ABXBSY(10),    XBOX(15),      XBXBSY(10)
ISN 0005          COMMON/RLS/     LAST
ISN 0006          INTEGER OUT
ISN 0007          REAL MEMDLY,MXTIME
ISN 0008          REAL TIME
ISN 0009          EXTERNAL FINIS
ISN 0010          CALL ABNER(FINIS)
ISN 0011       1111 CONTINUE
ISN 0012          CALL INIT
ISN 0013          CALL JSTART(ENDRUN)
ISN 0014          CALL INTPHI
ISN 0015          IF(ENDRUN.EQ.1) STOP
                C                 STEP THRU CALENDAR
ISN 0017       1000 CALL TSTEP(EVENT)
ISN 0018          IF(SSTOP.EQ.1) GO TO 1111
ISN 0020          IF(OUTLVL.EQ.0) WRITE(6,900) EVENT
ISN 0022          IF(TIME.GT.MXTIME) GO TO 999
ISN 0024          GO TO (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,
                 X 23,24,25),EVENT
ISN 0025        1 CONTINUE
ISN 0026          CALL XSTATS
ISN 0027          GO TO 1000
```

211

L. Conway
Archives

```
ISN 0028          2 CONTINUE
ISN 0029            CALL XMXO
ISN 0030            GO TO 1000
ISN 0031          3 CONTINUE
ISN 0032            CALL XACON
ISN 0033            GO TO 1000
ISN 0034          4 CONTINUE
ISN 0035            CALL XXCON
ISN 0036            GO TO 1000
ISN 0037          5 CONTINUE
ISN 0038            CALL XAEMP
ISN 0039            GO TO 1000
ISN 0040          6 CONTINUE
ISN 0041            CALL XXEMP
ISN 0042            GO TO 1000
ISN 0043          7 CONTINUE
ISN 0044            CALL XARET
ISN 0045            GO TO 1000
ISN 0046          8 CONTINUE
ISN 0047            CALL XXRET
ISN 0048            GO TO 1000
ISN 0049          9 CONTINUE
ISN 0050            CALL XEAV
ISN 0051            GO TO 1000
ISN 0052         10 CONTINUE
ISN 0053            CALL XQCON
ISN 0054            GO TO 1000
ISN 0055         11 CONTINUE
ISN 0056            CALL XQEMP
ISN 0057            GO TO 1000
ISN 0058         12 CONTINUE
ISN 0059            CALL XMBUSY
ISN 0060            GO TO 1000
ISN 0061         13 CONTINUE
ISN 0062            CALL XMFREE
ISN 0063            GO TO 1000
ISN 0064         14 CONTINUE
ISN 0065            CALL XLOAD
ISN 0066            GO TO 1000
ISN 0067         15 CONTINUE
ISN 0068            CALL XRTN
ISN 0069            GO TO 1000
ISN 0070         16 CONTINUE
ISN 0071            GO TO 1000
ISN 0072         17 CONTINUE
ISN 0073            GO TO 1000
ISN 0074         18 CONTINUE
ISN 0075            GO TO 1000
ISN 0076         19 CONTINUE
ISN 0077            GO TO 1000
```

212

L. Conway
Archives

```
ISN 0078          20 CONTINUE
ISN 0079             GO TO 1000
ISN 0080          21 CONTINUE
ISN 0081             GO TO 1000
ISN 0082          22 CONTINUE
ISN 0083             GO TO 1000
ISN 0084          23 CONTINUE
ISN 0085             GO TO 1000
ISN 0086          24 CONTINUE
ISN 0087             GO TO 1000
ISN 0088          25 CONTINUE
ISN 0089             GO TO 1000
ISN 0090         999 CONTINUE
ISN 0091             A=1
ISN 0092             B=20000
ISN 0093             CALL TROUBL(A,B,0)
ISN 0094             GO TO 1111
ISN 0095         900 FORMAT(I8)
ISN 0096             END
```

213

L. Conway
Archives

COMMON PHAS1

DOTL - Pointer to first entry of DO table
DOSL - pointer to next available entry - if equal to DOTL then table is full
IBCL - variable used to point to entry in IB's
HISL - variable used to point to entry in HISTORY TABLE (not used anymore)
SKXP, SKXC, SKXS, SKAV, SKAC, SKAS, } variables for skip control - not needed in Phase I
CYCL - count of cycles at end of PHSMOP
KY - condition codes indicating path thru exception routines (see table for KY)
SY - variable indicating X entry (231) or A entry (193) for fetch or search routine
PTR - variable set for pointer to OP table area for dispatching OPs
XX - indicator showing storage request was made this cycle
XIC - X instruction counter - indicates exact address of op leaving disp.
AIC - A instruction counter - indicates exact address of op leaving disp.
ASA - IB address for search routine - set before entering SEARCH
NFA - IB address of requested IB as next in sequence - may be IB address from PSC or NSA+8
DPA - FETCH - variable that contains address for match with address on cards
DEN - fetch variable contains low value of position within IB being looked 0-7 for current op
DOT - variable indicating length of DO Table - usually set to 6
IBN -
XICR - holding area for XIC to keep it current for dispatcher
AICR - holding area for AIC to keep it current for dispatcher
PTJ - } pointers within OP area for moving IB's to dispatch positions
PTK -
XEXT - trigger set to properly process unresolved exits that have been dispatched from X
AEXT - trigger set to properly " " " " " " " " from A

L. Conway Archives

DO TABLE ENTRIES MAX 2Ø controlled by DOT - usually set to 6

DOTL = first entry DOSL = next available DOCL = current level

DOIB - 2 - pointer to IB table entry assigned to this level
DOST - 2 - beginning pointer of first instruction in IB (zero unless branched into)
DOAP - 2 - current pointer for A-ops (only used in DISP) - contains IB addr when in X-Disp
DOXP - 2 - current pointer for X-ops (only used in DISP) - contains IB address when in A-Disp
LDEV - 1 - when on - this level is valid
LDDV - 1 - " " the data for this IB is in OP area
LDCKD - 1 - " " the address is checked for proper sequence
LDSEQ - 1 - " " this IB is out of sequence - due to branch exit or PSC function
LDAW - 1 - " " A-Disp working on this level
LDAF - 1 - " " A-Disp finished with this level
LDXW - 1 - " " X-Disp working on this level
LDXF - 1 - " " X-Disp finished with this level

X-DISP = DO entries 17, 18 A-DISP = DO entries 19, 2Ø

IB entries - 12

IBA - address (multiple of eight) of data assigned this IB
IBV - this IB valid
IBW - storage fetch in progress - turned off when fetch completed

HIST Table is a pushup with oldest IB indicated by top entry & latest by bottom entry.

L. Conway Archives

```
LEVEL 2 FEB 67                          OS/360  FORTRAN H                       DATE  67.282/10.36.34

          COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

    ISN 0002            SUBROUTINE PHLIBS
                C
    ISN 0003            IMPLICIT INTEGER*2(A-K,M-S,U-Z)
    ISN 0004            IMPLICIT LOGICAL*1(L)
    ISN 0005            IMPLICIT REAL (T)
                C
    ISN 0006            COMMON       TIME,       IPAR1,      IPAR2,      IPAR3,
                       A AINPT,      NABUF,      ABUS(50),   XINPT,      NXBUF,
                       B XBUS(50),   IFADD,      IFDST,      IFRTN,      BRXP,
                       C BRAP,       ER(8),      BE(8),      ET(8),      NBBUF,
                       D AHOLDT,     XHOLDT,     AFRCT,      XFRCT,      BOSC,
                       E BNOP,       XEP,        AEP,        PH1(100),   PRINT,
                       F FSTADD,     NODOT,      NOPSC,      NDBUS,      NADSP,
                       G NXDSP
                C
                C              COMMON AREA FOR PHASE 1
                C
    ISN 0007            COMMON /PHAS1/ DOTL, DOSL, DOCL, IBCL, HISL,
                       A SKXV, SKXC, SKXS, SKAV, SKAC, SKAS, CYCL, KY, SY, PTR, XX,
                       C XIC, AIC, ASA, NFA, DFA, DEN, DOT, IBN,
                       D XICR, AICR, PTJ, PTK, XEXT, AEXT,
                       G DOIB(20), DOST(20), DOAP(20), DOXP(20),
                       H LDEV(20), LDDV(20), LDCKD(20), LDSEQ(20),
                       I LDAW(20), LDAF(20), LDXW(20), LDXF(20),
                       J HIST(12),  IBA(12), LIBV(12), LIBW(12),
                       K INOP(30), OP(4000), LBX(8), LBA(8),
                       L PBUF(8),
                       Y KNT(50), PHEND
                C
    ISN 0008            COMMON /PSCS/ LPSV(8), PSCA(8),PSCB(8)
                       *, XEX, XEXS, XEXA, XEXB, AEX, AEXS, AEXA, AEXB,
                       *  EBA, EBXS, EBXA, EBXB
                       *,EBX
                C
                C              UPDATE DO LEVELS
    ISN 0009          912 IF (LDXF(DOTL).AND.LDAF(DOTL)) GOTO 913
    ISN 0011            GOTO 915
    ISN 0012          913 CONTINUE
    ISN 0013            LDEV(DOTL)=0
    ISN 0014            IBCL=DOIB(DOTL)
    ISN 0015            DOIB(DOTL)=0
    ISN 0016            DOST(DOTL)=0
    ISN 0017            DOAP(DOTL)=0
    ISN 0018            DOXP(DOTL)=0
    ISN 0019            LDDV(DOTL)=0
    ISN 0020            LDAW(DOTL)=0
    ISN 0021            LDXW(DOTL)=0
    ISN 0022            LDAF(DOTL)=0
    ISN 0023            LDXF(DOTL)=0
```

216

L. Conway
Archives

```
ISN 0024            LDCKD(DOTL)=0
ISN 0025            LDSEQ(DOTL)=0
ISN 0026            DOTL=DOTL+1
ISN 0027            IF (DOTL.GT.DOT) DOTL=1
ISN 0029            IF (DOTL.NE.DOSL) GOTO 912
                C
                C               CHECK FOR RETURN OF REQUESTED IB
ISN 0031          915 CONTINUE
ISN 0032            IF (IFRTN.EQ.0) GOTO 925
ISN 0034            LIBW(IFRTN)=0
ISN 0035            PBUF(4)=IFRTN
ISN 0036            IFRTN=0
ISN 0037          925 CONTINUE
                C               CHECK FOR RESOLVED BRANCHES
ISN 0038            SEQ=0
ISN 0039            IF (XEX.NE.0) GOTO 941
ISN 0041            IF (AEX.NE.0) GOTO 942
ISN 0043            IF (XX.NE.0) GOTO 930
ISN 0045            IF (IFDST.NE.0) GOTO 930
ISN 0047            IF (LDEV(DOSL)) GOTO 930
ISN 0049            NNFA=NFA+8
ISN 0050            IF (NOPSC.EQ.0) GOTO 916
ISN 0052            DO 926 I=1,NOPSC
ISN 0053            IF (LPSV(I).EQ.0) GOTO 926
ISN 0055            IF (NFA.NE.PSCA(I)) GOTO 926
ISN 0057            NNFA=PSCB(I)
ISN 0058            SEQ=1
ISN 0059          926 CONTINUE
                C
ISN 0060          916 CONTINUE
ISN 0061            NFA=NNFA
                C               SCAN IB'S FOR REQUEST
ISN 0062            DO 917 I=1,I2
ISN 0063            IF (IBA(I).EQ.NFA) GOTO 920
ISN 0065          917 CONTINUE
ISN 0066            I=1
ISN 0067            IBCL=HIST(1)
                C
                C               REQUEST INDTRUCTION FETCH
                C
ISN 0068            IFADD=NFA/2
ISN 0069            IFDST=IBCL
ISN 0070            LIBW(IBCL)=5
                C               SET NEXT DO ENTRY
ISN 0071          918 CONTINUE
ISN 0072            XX=1
ISN 0073            PBUF(1)=NFA
ISN 0074            PBUF(2)=IBCL
ISN 0075            PBUF(3)=DOSL
ISN 0076            IBA(IBCL)=NFA
```

O.K. ✓

217

L. Conway Archives

IBM FORM NO. 419 0

```
ISN 0077            LIBV(IBCL)=1
ISN 0078            DO 919 J=1,11
ISN 0079          919 HIST(J)=HIST(J+1)
ISN 0080            HIST(12)=IBCL
ISN 0081            DOIB(DOSL)=IBCL
ISN 0082            LDEV(DOSL)=1
ISN 0083            LDCKD(DOSL)=0
ISN 0084            LDSEQ(DOSL)=SEQ
ISN 0085            LDDV(DOSL)=0
ISN 0086            LDAW(DOSL)=0
ISN 0087            LDAF(DOSL)=0
ISN 0088            LDXW(DOSL)=0
ISN 0089            LDXF(DOSL)=0
ISN 0090            DOSL=DOSL+1
ISN 0091            IF (DOSL.GT.DOT) DOSL=1
ISN 0093            GOTO 930
                C
                C               REQUEST IN IB'S - SET-UP
ISN 0094          920 CONTINUE
ISN 0095            IF (LIBV(I).EQ.0) GOTO 917
ISN 0097            IBCL=I
ISN 0098            DO 921 I=1,12
ISN 0099            IF (HIST(I).EQ.IBCL) GOTO 918
ISN 0101          921 CONTINUE
ISN 0102            GOTO 930
                C
ISN 0103          930 CONTINUE
ISN 0104            IF (NOPSC.LT.2) GOTO 935
ISN 0106            IF (LPSV(NOPSC).EQ.0) GOTO 935
                C               MOVE PROGRAM ENTRY INTO PSC
ISN 0108            K=1
ISN 0109            N=NOPSC-1
ISN 0110            DO 932 I=1,N
ISN 0111            IF (LPSV(I).EQ.0) K=I
ISN 0113          932 CONTINUE
ISN 0114            DO 933 I=K,N
ISN 0115            PSCA(I)=PSCA(I+1)
ISN 0116            PSCB(I)=PSCB(I+1)
ISN 0117            LPSV(I)=LPSV(I+1)
ISN 0118          933 CONTINUE
ISN 0119            LPSV(NOPSC)=0
ISN 0120            GOTO 935
                C
ISN 0121          941 CONTINUE
ISN 0122            XEX=0
ISN 0123            AEX=0
ISN 0124            EBXS=XEXS
ISN 0125            EBXA=XEXA
ISN 0126            EBXB=XEXB
ISN 0127            XAF=0
```

218

L. Conway
Archives

IBM FORM NO. 413 0

```
ISN 0128            GOTO 945
                C
ISN 0129          942 CONTINUE
ISN 0130            AEX=0
ISN 0131            EBXS=AEXS
ISN 0132            EBXA=AEXA
ISN 0133            EBXB=AEXB
ISN 0134            XAF=1
ISN 0135            GOTO 945
                C
                C               SCAN PSC'S & INVALIDATE MATCHES
ISN 0136          945 CONTINUE
ISN 0137            EBA=(EBXA+8)/8*8
ISN 0138            IF(EBXS.NE.0) EBA=EBXB/8*8
ISN 0140            IF (NOPSC.EQ.0) GOTO 950
ISN 0142            N=NOPSC-1
ISN 0143            N2=NOPSC-2
ISN 0144            IF (NOPSC.LT.2) N=1
ISN 0146            DO 946 I=1,NOPSC
ISN 0147            IF (LPSV(I).EQ.0) GOTO 946
ISN 0149            IF (PSCA(I).EQ.EBXA) GOTO 955
ISN 0151          946 CONTINUE
ISN 0152            IF (EBXS.EQ.0) GOTO 950
                C               INSERT BRANCH ENTRY INTO PSC
ISN 0154          943 CONTINUE
ISN 0155            IF (N.EQ.1) GOTO 949
ISN 0157            IF (LPSV(N).EQ.0) GOTO 949
ISN 0159            K=1
ISN 0160            DO 947 I=1,N2
ISN 0161            IF (LPSV(I).EQ.0) K=I
ISN 0163          947 CONTINUE
ISN 0164            DO 948 I=K,N2
ISN 0165            PSCA(I)=PSCA(I+1)
ISN 0166            PSCB(I)=PSCB(I+1)
ISN 0167            LPSV(I)=LPSV(I+1)
ISN 0168          948 CONTINUE
ISN 0169          949 CONTINUE
ISN 0170            LPSV(N)=1
ISN 0171            PSCA(N)=EBXA
ISN 0172            PSCB(N)=EBA
ISN 0173            GOTO 950
                C
ISN 0174          955 CONTINUE
ISN 0175            IF (PSCB(I).EQ.EBA) GOTO 935
ISN 0177            LPSV(I)=0
ISN 0178            GOTO 943
                C
ISN 0179          950 CONTINUE
ISN 0180            DOCL=DOTL
ISN 0181          951 CONTINUE
```

219

L. Conway
Archives

IBM FORM NO. 413 0

```
ISN 0182            IF (LDEV(DOCL).EQ.0) GOTO 952
ISN 0184            IF((LDXF(DOCL).EQ.0).AND.(XAF.EQ.0)) GO TO 953
ISN 0186            IF((LDAF(DOCL).EQ.0).AND.(XAF.EQ.1)) GO TO 953
ISN 0188          952 CONTINUE
ISN 0189            DOCL=DOCL+1
ISN 0190            IF (DOCL.GT.DOT) DOCL=1
ISN 0192            IF (DOCL.NE.DOSL) GOTO 951
                  C
                  C               NO AVAILABLE ENTRIES IN DO TABLE
ISN 0194            NFA=EBXA
ISN 0195            GOTO 935
                  C
ISN 0196          956 CONTINUE
ISN 0197            IF (LDEV(19).EQ.0) GOTO 954
ISN 0199            IF (DOXP(19).EQ.EBA) GOTO 935
ISN 0201            IF (LDEV(20).EQ.0) GOTO 954
ISN 0203            IF (DOXP(20).EQ.EBA) GOTO 935
ISN 0205            GOTO 954
                  C
ISN 0206          953 CONTINUE
ISN 0207            IF (IBA(DOIB(DOCL)).EQ.EBA) GOTO 935
ISN 0209            IF (XAF.EQ.1) GOTO 956
                  C
ISN 0211            IF(LDEV(17).EQ.0) GO TO 954
ISN 0213            IF(DOAP(17).EQ.EBA) GO TO 935
ISN 0215            IF(LDEV(18).EQ.0) GO TO 954
ISN 0217            IF(DOAP(18).EQ.EBA) GO TO 935
ISN 0219          954 CONTINUE
ISN 0220            LDSEQ(DOCL)=1
ISN 0221            ASA=EBA
ISN 0222            SY=0
ISN 0223            CALL SEARCH
                  C
ISN 0224          935 CONTINUE
ISN 0225            RETURN
                  C
ISN 0226            END
```

220
L. Conway
Archives

```
LEVEL 2 FEB 67                          OS/360  FORTRAN H                          DATE  67.265/14.00.59

          COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

ISN 0002            SUBROUTINE PHIMOP
ISN 0003            IMPLICIT INTEGER*2(A-K,M-S,U-Z)
ISN 0004            IMPLICIT LOGICAL*1(L)
ISN 0005            IMPLICIT REAL (T)
                    C
ISN 0006            COMMON       TIME,       IPAR1,      IPAR2,      IPAR3,
                   A AINPT,      NABUF,      ABUS(50),   XINPT,      NXBUF,
                   B XBUS(50),   IFADD,      IFDST,      IFRTN,      BRXP,
                   C BRAP,       ER(8),      BE(8),      ET(8),      NBBUF,
                   D AHOLDT,     XHOLDT,     AFRCT,      XFRCT,      BOSC,
                   E BNOP,       XEP,        AEP,        PHI(100),   PRINT,
                   F FSTADD,     NODOT,      NOPSC,      NDBUS,      NADSP,
                   G NXDSP
                    C
                    C                COMMON AREA FOR PHASE I
                    C
ISN 0007            COMMON /PHAS1/ DOTL, DOSL, DOCL, IBCL, HISL,
                   A SKXV, SKXC, SKXS, SKAV, SKAC, SKAS, CYCL, KY, SY, PTR, XX,
                   C XIC, AIC, ASA, NFA, DFA, DEN, DOT, IBN,
                   D XICR, AICR, PTJ, PTK, XEXT, AEXT,
                   G DOIB(20), DOST(20), DOAP(20), DOXP(20),
                   H LDEV(20), LDDV(20), LDCKD(20), LDSEQ(20),
                   I LDAW(20), LDAF(20), LDXW(20), LDXF(20),
                   J HIST(12),  IBA(12), LIBV(12), LIBW(12),
                   K INOP(30), OP(4000), LBX(8), LBA(8),
                   L PBUF(8),
                   Y KNT(50), PHEND
                    C
ISN 0008            COMMON /PSCS/ LPSV(8), PSCA(8),PSCB(8)
                   *, XEX, XEXS, XEXA, XEXB, AEX, AEXS, AEXA, AEXB,
                   *  EBA, EBXS, EBXA, EBXB
                    C
                    C       OP =  (1)INSADD, (2)MNEM1, (3)MNEM2, (4)MNEM3, (5)II, (6)IJ,
                    C             (7)IK, (8)ILIT, (9)SUCC, (10)SKP, (11)IEX, (12)ACCADR,
                    C             (13)NXADR, (14)OPNUM, (15)LNG, (16)XOP, (17)AOP,
                    C             (18)BROP, (19)SKOP, (20)SKEND, (21)SPAR1, (22)SPAR2,
                    C             (23)SPAR3, (24)SKIP, (25)VALID
                    C
                    C
ISN 0009            ENTRY XMXO
                    C
ISN 0010            MXO=2
ISN 0011            CALL CAUSE (MXO,TIME+1.0,IPAR1,IPAR2,IPAR3)
ISN 0012        711 CONTINUE
ISN 0013            XX=0
ISN 0014            NCTX=NXDSP
ISN 0015            NCTA=NADSP
ISN 0016            DO 3 I=1,8
ISN 0017            LBX(I)=0
```

221

L. Conway
Archives

```
ISN 0018            LBA(I)=0
ISN 0019            PBUF(I)=0
ISN 0020          3 CONTINUE
                C
                C               INDEX OP FLOW
                C
                C               MOVE X-OPS TO STACK FROM DISP
                C
ISN 0021            DOCL=17
ISN 0022            XIC=XICR
ISN 0023        111 CONTINUE
ISN 0024        112 IF (LDEV(DOCL).EQ.0) GOTO 130
ISN 0026            IF (LDXF(DOCL).NE.0) GOTO 116
ISN 0028            IF (LDXW(DOCL).NE.0) GOTO 113
ISN 0030            DOXP(DOCL)=MOD(XIC,8)
ISN 0031            LDXW(DOCL)=1
ISN 0032        113 IF (DOXP(DOCL).LT.7) GOTO 20
ISN 0034            IF (DOXP(DOCL).GT.7) GOTO 115
                C
                C               CHECK FOR I8 CROSSOVER
                C
ISN 0036            PTR=(DOXP(DOCL)*25)+((DOCL-1)*(25*8))
ISN 0037            IF (OP(PTR+25).EQ.0) GOTO 34
ISN 0039            IF (OP(PTR+16).EQ.0) GOTO 30
ISN 0041            IF (OP(PTR+15).EQ.1) GOTO 20
ISN 0043            IF (DOCL.GT.17) GOTO 130
ISN 0045            IF (LDEV(DOCL+1).NE.0) GOTO 20
ISN 0047            GOTO 130
ISN 0048        115 LDXF(DOCL)=1
ISN 0049        116 IF (DOCL.GT.17) GOTO 130
ISN 0051            DOCL=18
ISN 0052            GOTO 112
                C
                C               CHECK OP ENTRY
                C
ISN 0053         20 PTR=(DOXP(DOCL)*25)+((DOCL-1)*(25*8))
                C       BUS= 'X'
ISN 0054            BUS=231
ISN 0055            IF (OP(PTR+25).EQ.0) GOTO 34
ISN 0057            IF (OP(PTR+16).NE.0) GOTO 21
ISN 0059            GOTO 30
                C
                C               PROCESS INDEX OP
                C
ISN 0060         21 CONTINUE
ISN 0061            IF (XINPT.GT.NXBUF) GOTO 130
ISN 0063            IF (NCTX.LT.1) GOTO 130
ISN 0065            IF (OP(PTR+18).NE.0) GOTO 41
ISN 0067            IF (OP(PTR+20).NE.0) GOTO 45
                C
```

222

L. Conway
Archives

```
                C               MOVE X-OP TO XBUF AND GO TO NEXT
                C
ISN 0069            23 CONTINUE
ISN 0070               IF (XEXT.NE.0) GOTO 25
ISN 0072               LBX(NCTX)=OP(PTR+23)
ISN 0073               XBUS(1)=OP(PTR+23)*256
ISN 0074               XBUS(2)=OP(PTR+14)
ISN 0075               XBUS(3)=OP(PTR+5)
ISN 0076               XBUS(4)=OP(PTR+6)
ISN 0077               XBUS(5)=OP(PTR+7)
ISN 0078               XBUS(6)=OP(PTR+12)
ISN 0079               XBUS(7)=OP(PTR+17)
ISN 0080               XBUS(8)=OP(PTR+16)
ISN 0081               XBUS(9)=OP(PTR+11)
ISN 0082               XBUS(10)=OP(PTR+9)
ISN 0083               XBUS(11)=OP(PTR+10)
ISN 0084               XBUS(12)=OP(PTR+18)
ISN 0085               XBUS(13)=OP(PTR+19)
ISN 0086               XBUS(14)=OP(PTR+20)
ISN 0087               NCTX=NCTX-1
ISN 0088               CALL BUSTOX
ISN 0089               IF (OP(PTR+24).NE.0) GOTO 25
ISN 0091               OP(PTR+16)=0
ISN 0092               GOTO 30
                C
ISN 0093            25 CONTINUE
ISN 0094               XEXT=1
ISN 0095               IF (OP(PTR+20).NE.0) GOTO 130
ISN 0097               XEXT=0
ISN 0098               OP(PTR+16)=0
ISN 0099               BNOP=0
ISN 0100               IF (OP(PTR+9).EQ.0) GOTO 30
ISN 0102               XIC=OP(PTR+13)
ISN 0103               LDXF(DOCL)=1
ISN 0104               GOTO 111
                C
                C               STEP BY OP LENGTH (1,2) TO NEXT ENTRY
                C
ISN 0105            30 DOXP(DOCL)=DOXP(DOCL)+OP(PTR+15)
ISN 0106               IF (DOXP(DOCL).GT.7) LDXF(DOCL)=1
ISN 0108               XIC=XIC+OP(PTR+15)
ISN 0109               GOTO 111
                C
                C               STEP OVER SPACE (1) TO NEXT ENTRY
                C
ISN 0110            34 DOXP(DOCL)=DOXP(DOCL)+1
ISN 0111               IF (DOXP(DOCL).GT.7) LDXF(DOCL)=1
ISN 0113               XIC=XIC+1
ISN 0114               GOTO 111
                C
```

223

L. Conway
Archives

```
                C               CHECK X-BRANCH OPS
                C
ISN 0115           41 CONTINUE
ISN 0116              IF (BNOP.EQ.0) GOTO 23
ISN 0118              NCTX=NCTX-1
ISN 0119              OP(PTR+16)=0
ISN 0120              GOTO 30
                C
ISN 0121           45 CONTINUE
ISN 0122              IF (ER(BRXP).EQ.0) GOTO 23
ISN 0124              OP(PTR+20)=0
ISN 0125              IF (LDAF(DOCL).NE.0) GOTO 23
ISN 0127              XEX=1
ISN 0128              XEXS=OP(PTR+9)
ISN 0129              XEX8=OP(PTR+13)
ISN 0130              XEXA=DOAP(DOCL)
ISN 0131              GOTO 23
                C
                C
ISN 0132         130 CONTINUE
ISN 0133              XICR=XIC
                C
                C
ISN 0134         811 CONTINUE
                C
                C               ARITH OP FLOW
                C
                C               MOVE A-OPS TO STACK FROM DISP
                C
ISN 0135              DOCL=19
ISN 0136              AIC=AICR
ISN 0137         211 CONTINUE
ISN 0138         212 IF (LDEV(DOCL).EQ.0) GOTO 230
ISN 0140              IF (LDAF(DOCL).NE.0) GOTO 216
ISN 0142              IF (LDAW(DOCL).NE.0) GOTO 213
ISN 0144              DOAP(DOCL)=MOD(AIC,8)
ISN 0145              LDAW(DOCL)=1
ISN 0146         213 IF (DOAP(DOCL).LT.7) GOTO 70
ISN 0148              IF (DOAP(DOCL).GT.7) GOTO 215
                C
                C               CHECK FOR IB CROSSOVER
                C
ISN 0150              PTR=(DOAP(DOCL)*25)+((DOCL-1)*(25*8))
ISN 0151              IF (OP(PTR+25).EQ.0) GOTO 84
ISN 0153              IF (OP(PTR+17).EQ.0) GOTO 80
ISN 0155              IF (OP(PTR+15).EQ.1) GOTO 70
ISN 0157              IF (DOCL.GT.19) GOTO 230
ISN 0159              IF (LDEV(DOCL+1).NE.0) GOTO 70
ISN 0161              GO TO 230
ISN 0162         215 LDAF(DOCL)=1
```

no change

224

L. Conway Archives

```
ISN 0163       216 IF (DOCL.GT.19) GOTO 230
ISN 0165           DOCL=20
ISN 0166           GOTO 212
                C
                C              CHECK OP ENTRY
                C
ISN 0167        70 PTR=(DOAP(DOCL)*25)+((DOCL-1)*(25*8))  no change
                C     BUS= 'A'
ISN 0168           BUS=193
ISN 0169           IF (OP(PTR+25).EQ.0) GOTO 84
ISN 0171           IF (OP(PTR+17).NE.0) GOTO 71
ISN 0173           GOTO 80
                C
                C              PROCESS ARITH OP
                C
ISN 0174        71 CONTINUE
ISN 0175           IF (AINPT.GT.NABUF) GOTO 230
ISN 0177           IF (NCTA.LT.1) GOTO 230
ISN 0179           IF (OP(PTR+20).NE.0) GOTO 85
ISN 0181           IF(OP(PTR+24).NE.0) GO TO 75
                C
                C              MOVE A-OP TO ABUF AND GO TO NEXT
                C
ISN 0183        73 CONTINUE
ISN 0184           IF (AEXT.NE.0) GOTO 75
ISN 0186           LBA(NCTA)=OP(PTR+23)
ISN 0187           ABUS(1)=OP(PTR+23)*256
ISN 0188           ABUS(2)=OP(PTR+14)
ISN 0189           ABUS(3)=OP(PTR+5)
ISN 0190           ABUS(4)=OP(PTR+6)
ISN 0191           ABUS(5)=OP(PTR+7)
ISN 0192           ABUS(6)=OP(PTR+12)
ISN 0193           ABUS(7)=OP(PTR+17)
ISN 0194           ABUS(8)=OP(PTR+16)
ISN 0195           ABUS(9)=OP(PTR+11)
ISN 0196           ABUS(10)=OP(PTR+9)
ISN 0197           ABUS(11)=OP(PTR+10)
ISN 0198           ABUS(12)=OP(PTR+18)
ISN 0199           ABUS(13)=OP(PTR+19)
ISN 0200           ABUS(14)=OP(PTR+20)
ISN 0201           NCTA=NCTA-1
ISN 0202           CALL BUSTOA
ISN 0203           IF (OP(PTR+24).NE.0) GOTO 75
ISN 0205           OP(PTR+17)=0
ISN 0206           GOTO 80
                C
ISN 0207        75 CONTINUE
ISN 0208           AEXT=1
ISN 0209           IF (OP(PTR+20).NE.0) GOTO 230
ISN 0211           AEXT=0
ISN 0212           OP(PTR+17)=0
```

225

L. Conway
Archives

```
ISN 0213            IF (OP(PTR+9).EQ.0) GOTO 80
ISN 0215            AIC=OP(PTR+13)
ISN 0216            LDAF(DOCL)=1
ISN 0217            GOTO 211
                C.
                C
                C               STEP BY OP LENGTH (1,2) TO NEXT ENTRY
                C
ISN 0218         80 DOAP(DOCL)=DOAP(DOCL)+OP(PTR+15)
ISN 0219            IF (DOAP(DOCL).GT.7) LDAF(DOCL)=1
ISN 0221            AIC=AIC+OP(PTR+15)
ISN 0222            GOTO 211
                C
                C               STEP OVER SPACE (1) TO NEXT ENTRY
                C
ISN 0223         84 DOAP(DOCL)=DOAP(DOCL)+1
ISN 0224            IF (DOAP(DOCL).GT.7) LDAF(DOCL)=1
ISN 0226            AIC=AIC+1
ISN 0227            GOTO 211
                C
ISN 0228         85 CONTINUE
ISN 0229            IF (ER(BRAP).EQ.0) GOTO 73
ISN 0231            OP(PTR+20)=0
ISN 0232            IF (LDXF(DOCL).NE.0) GOTO 75
ISN 0234            AEX=1
ISN 0235            AEXS=OP(PTR+9)
ISN 0236            AEXB=OP(PTR+13)
ISN 0237            AEXA=DOXP(DOCL)
ISN 0238            GO TO 75
                C
                C
ISN 0239        230 CONTINUE
ISN 0240            AICR=AIC
ISN 0241            IF ((AEP.NE.0).AND.(ER(BRAP).NE.0)) AEP=0
                C
                C
                C               UPDATE DISP  A-FLOW
ISN 0243            IF (LDEV(19).EQ.0) GOTO 620
ISN 0245            IF (LDAF(19).EQ.0) GOTO 640
ISN 0247            IF (LDEV(20).EQ.0) GOTO 620
ISN 0249            IF (LDAF(20).EQ.0) GOTO 630
ISN 0251        620 CONTINUE
ISN 0252            AFIL=NDBUS
ISN 0253            ADSP=19
ISN 0254            LDEV(19)=0
ISN 0255            LDEV(20)=0
ISN 0256            GOTO 60
ISN 0257        630 CONTINUE
ISN 0258            J=19
ISN 0259            K=20
```

226

L. Conway
Archives

```
ISN 0260            DOIB(J)=DOIB(K)
ISN 0261            DOST(J)=DOST(K)
ISN 0262            DOAP(J)=DOAP(K)
ISN 0263            DOXP(J)=DOXP(K)
ISN 0264            LDEV(J)=LDEV(K)
ISN 0265            LDDV(J)=LDDV(K)
ISN 0266            LDAW(J)=LDAW(K)
ISN 0267            LDAF(J)=LDAF(K)
ISN 0268            LDXW(J)=LDXW(K)
ISN 0269            LDXF(J)=LDXF(K)
ISN 0270            LDCKD(J)=LDCKD(K)
ISN 0271            LDSEQ(J)=LDSEQ(K)
ISN 0272            PTJ=(J-1)*200
ISN 0273            PTK=(K-1)*200
ISN 0274            DO 633 I=1,200
ISN 0275        633 OP(PTJ+I)=OP(PTK+I)
ISN 0276        635 CONTINUE
ISN 0277            AIC=AIC/8*8+8
ISN 0278            AFIL=1
ISN 0279            ADSP=20
ISN 0280            LDEV(20)=0
ISN 0281            GOTO 60
ISN 0282        640 CONTINUE
ISN 0283            IF (LDEV(20).NE.0) GOTO 210
ISN 0285            GOTO 635
                C
                C               MOVE IB TO DISP PER DO ENTRY
                C
                C               CHECK THE DEC ORDER LEVEL
                C
ISN 0286         60 DOCL=DOTL
ISN 0287            IF ((AEP.NE.0).OR.(AHOLDT.NE.0)) GOTO 210
ISN 0289         61 IF (LDEV(DOCL).EQ.0) GOTO 62
ISN 0291            IF (LDAF(DOCL).EQ.0) GOTO 63
ISN 0293         62 DOCL=DOCL+1
ISN 0294            IF (DOCL.GT.DOT) DOCL=1
ISN 0296            IF (DOCL.NE.DOSL) GOTO 61
ISN 0298            GOTO 210
                C
                C               CHECK ADDRESS FOR LEVEL
                C
ISN 0299         63 CONTINUE
ISN 0300            IF (LDCKD(DOCL).NE.0) GOTO 67
ISN 0302            IBCL=DOIB(DOCL)
ISN 0303            IF ((AIC/8).EQ.(IBA(IBCL)/8)) GOTO 67
ISN 0305            IF (LDSEQ(DOCL).NE.0) GOTO 67
ISN 0307            GOTO 210
                C
                C               DATA ADDRESS CHECKED
                C
ISN 0308         67 LDCKD(DOCL)=1
```

227

L. Conway
Archives

IBM FORM NO. 413

```
ISN 0309            DOST(DOCL)=0
ISN 0310            DOAP(DOCL)=DOST(DOCL)
ISN 0311            IBCL=DOIB(DOCL)
                C
                C               CHECK IB WAIT & LIMIT
                C
ISN 0312            IF (LIBW(IBCL).NE.0) GOTO 210
                C
                C               CHECK IF DATA VALID
                C
ISN 0314            IF (LDDV(DOCL).NE.0) GOTO 68
ISN 0316            LDDV(DOCL)=1
ISN 0317            SY=193
ISN 0318            CALL FETCH
                C
ISN 0319            IF (KNT(9).NE.0) GOTO 210
ISN 0321            IF (LDDV(DOCL).EQ.0) GOTO 210
ISN 0323         68 LDAW(DOCL)=1
ISN 0324            LDAF(DOCL)=1
                C
ISN 0325            J=ADSP
ISN 0326            K=DOCL
ISN 0327            DOIB(J)=DOIB(K)
ISN 0328            DOAP(J)=DOST(K)
ISN 0329            DOST(J)=K
ISN 0330            DOXP(J)=IBA(DOIB(K))
ISN 0331            LDEV(J)=LDEV(K)
ISN 0332            LDDV(J)=LDDV(K)
ISN 0333            LDAW(J)=0
ISN 0334            LDAF(J)=0
ISN 0335            LDXW(J)=LDXW(K)
ISN 0336            LDXF(J)=LDXF(K)
ISN 0337            LDCKD(J)=LDCKD(K)
ISN 0338            LDSEQ(J)=LDSEQ(K)
ISN 0339            PTJ=(J-1)*200
ISN 0340            PTK=(K-1)*200
ISN 0341            DO 69 I=1,200
ISN 0342         69 OP(PTJ+I)=OP(PTK+I)
ISN 0343            AFIL=AFIL-1
ISN 0344            ADSP=ADSP+1
ISN 0345            AIC=AIC+8
ISN 0346            DO 66 I=20,200,25
ISN 0347         66 IF (OP(PTJ+I).NE.0) GOTO 210
ISN 0349            IF (AFIL.NE.0) GOTO 62
                C
ISN 0351        210 CONTINUE
ISN 0352            J=19
ISN 0353        703 CONTINUE
ISN 0354            IF (LDEV(J).EQ.0) GOTO 705
ISN 0356            IF (LDAF(J).NE.0) GOTO 705
```

228

L. Conway Archives

```
ISN 0358            K=(J-1)*200
ISN 0359            DO 702 I=1,8
ISN 0360            IF (OP(K+17).EQ.0) GOTO 701
ISN 0362            IF (OP(K+20).NE.0) GOTO 709
ISN 0364        701 K=K+25
ISN 0365        702 CONTINUE
ISN 0366        705 CONTINUE
ISN 0367            J=J+1
ISN 0368            IF (J.EQ.20) GOTO 703
ISN 0370            GOTO 712
ISN 0371        709 IF (OP(K+24).NE.0) GOTO 710
ISN 0373            AEP=1
ISN 0374            OP(K+24)=1
ISN 0375        710 IF (AEP.NE.0) GOTO 712
ISN 0377            OP(K+20)=0
ISN 0378            IF (LDXF(J).NE.0) GOTO 712
ISN 0380            AEX=1
ISN 0381            AEXS=OP(K+9)
ISN 0382            AEXB=OP(K+13)
ISN 0383            AEXA=DOXP(J)
ISN 0384        712 CONTINUE
ISN 0385            IF ((XEP.NE.0).AND.(ERIBRXP).NE.0)) XEP=0
                C
                C
                C               UPDATE DISP  X-FLOW
ISN 0387            IF (LDEV(17).EQ.0) GOTO 420
ISN 0389            IF (LDXF(17).EQ.0) GOTO 440
ISN 0391            IF (LDEV(18).EQ.0) GOTO 420
ISN 0393            IF (LDXF(18).EQ.0) GOTO 430
ISN 0395        420 CONTINUE
ISN 0396            XFIL=NDBUS
ISN 0397            XDSP=17
ISN 0398            LDEV(17)=0
ISN 0399            LDEV(18)=0
ISN 0400            GOTO 10
ISN 0401        430 CONTINUE
ISN 0402            J=17
ISN 0403            K=18
ISN 0404            DOIB(J)=DOIB(K)
ISN 0405            DOST(J)=DOST(K)
ISN 0406            DOAP(J)=DOAP(K)
ISN 0407            DOXP(J)=DOXP(K)
ISN 0408            LDEV(J)=LDEV(K)
ISN 0409            LDDV(J)=LDDV(K)
ISN 0410            LDAW(J)=LDAW(K)
ISN 0411            LDAF(J)=LDAF(K)
ISN 0412            LDXW(J)=LDXW(K)
ISN 0413            LDXF(J)=LDXF(K)
ISN 0414            LDCKD(J)=LDCKD(K)
ISN 0415            LDSEQ(J)=LDSEQ(K)
```

229

L. Conway
Archives

```
ISN 0416          PTJ=(J-1)*200
ISN 0417          PTK=(K-1)*200
ISN 0418          DO 433 I=1,200
ISN 0419      433 OP(PTJ+I)=OP(PTK+I)
ISN 0420      435 CONTINUE
ISN 0421          XIC=XIC/8*8+8
ISN 0422          XFIL=1
ISN 0423          XDSP=18
ISN 0424          LDEV(18)=0
ISN 0425          GOTO 10
ISN 0426      440 CONTINUE
ISN 0427          IF (LDEV(18).NE.0) GOTO 110
ISN 0429          GOTO 435
                C
                C              MOVE IB TO DISP PER DO ENTRY
                C
                C              CHECK THE DEC ORDER LEVEL
                C
ISN 0430       10 DOCL=DOTL
ISN 0431          IF ((XEP.NE.0).OR.(XHOLDT.NE.0)) GOTO 110
ISN 0433       11 IF (LDEV(DOCL).EQ.0) GOTO 12
ISN 0435          IF (LDXF(DOCL).EQ.0) GOTO 13
ISN 0437       12 DOCL=DOCL+1
ISN 0438          IF (DOCL.GT.DOT) DOCL=1
ISN 0440          IF (DOCL.NE.DOSL) GOTO 11
ISN 0442          GOTO 110
                C
                C              CHECK ADDRESS FOR LEVEL
                C
ISN 0443       13 CONTINUE
ISN 0444          IF (LDCKD(DOCL).NE.0) GOTO 17
ISN 0446          IBCL=DOIB(DOCL)
ISN 0447          IF ((XIC/8).EQ.(IBA(IBCL)/8)) GOTO 17
ISN 0449          IF (LDSEQ(DOCL).NE.0) GOTO 17
ISN 0451          LDSEQ(DOCL)=1
ISN 0452          ASA=XIC
ISN 0453          SY=231
ISN 0454          CALL SEARCH
ISN 0455          GOTO 110
                C
                C              DATA ADDRESS CHECKED
                C
ISN 0456       17 LDCKD(DOCL)=1
ISN 0457          DOST(DOCL)=0
ISN 0458          DOXP(DOCL)=DOST(DOCL)
ISN 0459          IBCL=DOIB(DOCL)
                C
                C              CHECK IB WAIT & LIMIT
                C
ISN 0460          IF (LIBW(IBCL).NE.0) GOTO 110
```

230

L. Conway Archives

IBM FORM NO. 413 0

```
                C
                C              CHECK IF DATA VALID
                C
ISN 0462              IF (LDDV(DOCL).NE.0) GOTO 18
ISN 0464              LDDV(DOCL)=1
ISN 0465              SY=231
ISN 0466              CALL FETCH
                C
ISN 0467              IF (KNT(8).NE.0) GOTO 110
ISN 0469              IF (LDDV(DOCL).EQ.0) GOTO 110
ISN 0471           18 LDXW(DOCL)=1
ISN 0472              LDXF(DOCL)=1
                C
ISN 0473              J=XDSP
ISN 0474              K=DOCL
ISN 0475              DOIB(J)=DOIB(K)
ISN 0476              DOXP(J)=DOST(K)
ISN 0477              DOST(J)=K
ISN 0478              DOAP(J)=IBA(DOIB(K))
ISN 0479              LDEV(J)=LDEV(K)
ISN 0480              LDDV(J)=LDDV(K)
ISN 0481              LDAW(J)=LDAW(K)
ISN 0482              LDAF(J)=LDAF(K)
ISN 0483              LDXW(J)=0
ISN 0484              LDXF(J)=0
ISN 0485              LDCKD(J)=LDCKD(K)
ISN 0486              LDSEQ(J)=LDSEQ(K)
ISN 0487              PTJ=(J-1)*200
ISN 0488              PTK=(K-1)*200
ISN 0489              DO 19 I=1,200
ISN 0490           19 OP(PTJ+I)=OP(PTK+I)
ISN 0491              XFIL=XFIL-1
ISN 0492              XDSP=XDSP+1
ISN 0493              XIC=XIC+8
ISN 0494              DO 16 I=20,200,25
ISN 0495           16 IF (OP(PTJ+I).NE.0) GOTO 110
ISN 0497              IF (XFIL.NE.0) GOTO 12
                C
                C
ISN 0499          110 CONTINUE
ISN 0500              BOSC=0
ISN 0501              J=17
ISN 0502          505 CONTINUE
ISN 0503              IF (LDEV(J).EQ.0) GOTO 510
ISN 0505              IF (LDXF(J).NE.0) GOTO 510
ISN 0507              K=(J-1)*200
ISN 0508              DO 504 I=1,8
ISN 0509              IF (OP(K+16).EQ.0) GOTO 503
ISN 0511              IF (OP(K+20).NE.0) GOTO 515
ISN 0513              IF (OP(K+18).EQ.0) GOTO 503
```

231

L. Conway
Archives

IBM FORM NO. 413 G

```
ISN 0515            IF (BNOP.EQ.0) GOTO 502
ISN 0517            OP(K+16)=0
ISN 0518            GOTO 503
ISN 0519        502 BOSC=1
ISN 0520        503 K=K+25
ISN 0521        504 CONTINUE
ISN 0522        510 CONTINUE
ISN 0523            J=J+1
ISN 0524            IF (J.EQ.18) GOTO 505
ISN 0526            GOTO 520
ISN 0527        515 IF (OP(K+24).NE.0) GOTO 516
ISN 0529            XEP=1
ISN 0530            OP(K+24)=1
ISN 0531        516 IF (XEP.NE.0) GOTO 520
ISN 0533            OP(K+20)=0
ISN 0534            IF (LDAF(J).NE.0) GOTO 520
ISN 0536            XEX=1
ISN 0537            XEXS=OP(K+9)
ISN 0538            XEXB=OP(K+13)
ISN 0539            XEXA=OOAP(J)
ISN 0540        520 CONTINUE
                C
ISN 0541        911 CONTINUE
                C               LOAD IB'S
                C
ISN 0542            IF (PHEND.GT.0) GOTO 930
ISN 0544            CALL PH1IBS
ISN 0545        930 CONTINUE
ISN 0546            CYCL=CYCL+1
ISN 0547            DO 999 I=1,100
ISN 0548        999 PH1(I)=0
ISN 0549            PH1(100)=PHEND
ISN 0550            IF (PHEND.GT.1) GOTO 931
ISN 0552            CALL PH1MAP
                C
ISN 0553        931 CONTINUE
ISN 0554            IF ((KNT(8).EQ.0).OR.(KNT(9).EQ.0)) GOTO 932
ISN 0556            IF (LDEV(17).OR.LDEV(18).OR.LDEV(19).OR.LDEV(20)) GOTO 932
ISN 0558            PHEND=2
ISN 0559        932 CONTINUE
ISN 0560            RETURN
ISN 0561            END
```

232

L. Conway
Archives

```
COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

ISN 0002            SUBROUTINE PH1SPT
                C
                C           SUPPORT SUBROUTINES - PHASE I
                C
ISN 0003            IMPLICIT INTEGER*2(A-K,M-S,U-Z)
ISN 0004            IMPLICIT LOGICAL*1(L)
ISN 0005            IMPLICIT REAL (T)
                C
ISN 0006            COMMON       TIME,      IPAR1,      IPAR2,      IPAR3,
                   A AINPT,      NABUF,     ABUS(50),   XINPT,      NXBUF,
                   B XBUS(50),   IFADD,     IFDST,      IFRTN,      BRXP,
                   C BRAP,       ER(8),     BE(8),      ET(8),      NBBUF,
                   D AHOLDT,     XHOLDT,    AERCT,      XERCT,      BOSC,
                   E BNOP,       XEP,       AEP,        PH1(100),   PRINT,
                   F FSTADD,     NODOT,     NOPSC,      NDBUS,      NADSP,
                   G NXDSP
                C
                C           COMMON AREA FOR PHASE I
                C
ISN 0007            COMMON /PHAS1/ DOTL, DOSL, DOCL, IBCL, HISL,
                   A SKXV, SKXC, SKXS, SKAV, SKAC, SKAS, CYCL, KY, SY, PTR, XX,
                   C XIC, AIC, ASA, NFA, DFA, DEN, DOT, IBN,
                   D XICR, AICR, PTJ, PTK, XEXT, AEXT,
                   G DOIB(20), DOST(20), DOAP(20), DOXP(20),
                   H LDFV(20), LDDV(20), LDCKD(20), LDSEQ(20),
                   I LDAW(20), LDAF(20), LDXW(20), LDXF(20),
                   J HIST(12),  IBA(12), LIBV(12), LIBW(12),
                   K INOP(30), OP(4000), LBX(8), LBA(8),
                   L PBUF(8),
                   Y KNT(50), PHEND
                C
                C
                C       OP =  (1)INSADD, (2)MNEM1, (3)MNEM2, (4)MNEM3, (5)II, (6)IJ,
                C             (7)IK, (8)ILIT, (9)SUCC, (10)SKP, (11)IEX, (12)ACCADR,
                C             (13)NXADR, (14)OPNUM, (15)LNG, (16)XOP, (17)AOP,
                C             (18)BROP, (19)SKOP, (20)SKEND, (21)SPAR1, (22)SPAR2,
                C             (23)SPAR3, (24)SKIP, (25)VALID
                C
                C
ISN 0008            COMMON /TAGS/ D(256,70)
                C
ISN 0009            COMMON /PSCS/ PSC(50)
                C
ISN 0010            COMMON/PROG/JTRACE(1000,30),INSLOC
ISN 0011            DIMENSION LETTER(36)
ISN 0012            DIMENSION INT(4300)
ISN 0013            DIMENSION LETR(38)
ISN 0014            DIMENSION LPSV(8)
ISN 0015            DATA LETR/38H0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ**/
ISN 0016            DATA LETTER/36HABCDEFGHIJKLMNOPQRSTUVWXYZ 123456789/
```

233

L. Conway
Archives

```
ISN 0017            DATA SCT,FCT/0,0/
ISN 0018            DIMENSION TABLE(199)
ISN 0019            DATA TABLE /1,2,4,5,0,9,10,12,13,4*0,19,20,21,22,0,23,0,24,0,25,
                   A 0,26,10*0,38,39,3*0,42,155,156,4*0,47,48,49,50,3*0,83,84,99,100,
                   B 118,119,89,90,91,0,93,0,95,96,97,98,106,107,108,109,112,113,
                   C 114,115,123,124,127,128,2*0,131,132,134,135,133,136,21*0,170,
                   D 171,172,173,174,175,176,177,178,179,180,181,182,183,184,185,195,
                   E 196,194,8*0,202,203,204,205,206,207,208,209,215,216,217,218,219,
                   F 220,221,222,11*0,56,0,58,0,60,0,62,0,64,0,66,0,68,0,0,70,0,72,0,
                   G 0,76,0,0,78,9*0,212/
ISN 0020            INTEGER*4 WRT(50)
ISN 0021            EQUIVALENCE (LPSV(1),PSC(1))
ISN 0022            EQUIVALENCE (INT(1),DOTL)
ISN 0023        801 FORMAT (1X15,1X3A2,1X,3(1XI2),2X15,4X3I1,3X15,2X15,2X13,2X,16I1)
ISN 0024        802 FORMAT(10XA2,1X15,1X6A1,3(1XI2),1X15,4X3I1,1X15,1X15,2X13,2X,
                   X   3I1,3XA2)
                C
                C
                C
                C               FETCH ROUTINE
                C                   TO FETCH OPS INTO IB BUFFER
                C
ISN 0025            ENTRY FETCH
                C
ISN 0026            FCT=FCT+1
ISN 0027            IBCL=DOIB(DOCL)
ISN 0028            DEN = DOST(DOCL)
ISN 0029            DFA=(IBA(IBCL)/8*8)+DEN
                C
                C               CLEAR IB OP AREA
                C
ISN 0030            PTS=((DOCL-1)*(25*8))
ISN 0031            PTR=PTS
ISN 0032            DO 115 I=1,8
ISN 0033            DO 114 J=1,25
ISN 0034            OP(PTR+J)=0
ISN 0035        114 CONTINUE
ISN 0036            PTR=PTR+25
ISN 0037        115 CONTINUE
ISN 0038            PAR=0
ISN 0039            PDL=LETR(DOIB(DOCL)+1)
                C
                C                   CHECK OP INPUT
                C
ISN 0040        120 IF (INOP(25).NE.0) GOTO 121
                C
                C                   READ OP CARD
                C
                C       READ(5,801,END=150)(INOP(I),I=1,30)
                C
```

234
L. Conway
Archives

```
                C
                C              TO USE UNROLLER OUTPUT, SET INOP FROM JTRACE
ISN 0042              DO 131 I=1,15
ISN 0043              INOP(I)=JTRACE(INSLOC,I)
ISN 0044          131 CONTINUE
ISN 0045              MN1=JTRACE(INSLOC,21)
ISN 0046              MN2=JTRACE(INSLOC,22)
ISN 0047              MN3=JTRACE(INSLOC,23)
ISN 0048              MN4=JTRACE(INSLOC,24)
ISN 0049              MN5=JTRACE(INSLOC,25)
ISN 0050              MN6=JTRACE(INSLOC,26)
                C
                C
ISN 0051              INSLOC=INSLOC+1
ISN 0052              INOP(25)=1
ISN 0053              IF (INOP(14).EQ.999) GOTO 123
ISN 0055              INOP(16)=0
ISN 0056              INOP(17)=0
ISN 0057              INOP(18)=0
ISN 0058              INOP(19)=0
ISN 0059              INOP(20)=0
ISN 0060              INOP(16)=D(INOP(14),1)
ISN 0061              INOP(17)=D(INOP(14),2)
ISN 0062              INOP(18)=D(INOP(14),3)
ISN 0063              INOP(19)=D(INOP(14),29)
ISN 0064              INOP(20)=D(INOP(14),31)
ISN 0065              IF (INOP(10).NE.1) INOP(10)=0
ISN 0067              PNT=MOD(INOP(1),26)+1
ISN 0068              INOP(23)=LETTER(PNT)
ISN 0069          123 CONTINUE
ISN 0070              IF(PRINT.GT.1) GO TO 121
ISN 0072              WRITE(6,802)INOP(23),INOP(1),MN1,MN2,MN3,MN4,MN5,MN6,
                     X  (INOP(I),I=5,17),PDL
ISN 0073          121 CONTINUE
ISN 0074              IF (INOP(14).EQ.999) GOTO 152
ISN 0076              IF (INOP(1).NE.DFA) GOTO 125
ISN 0078          126 CONTINUE
ISN 0079              PTR=PTS+(DEN*25)
ISN 0080              DO 122 I=1,25
ISN 0081              OP(PTR+I)=INOP(I)
ISN 0082              INOP(I)=0
ISN 0083          122 CONTINUE
ISN 0084              INOP(25)=0
ISN 0085              PAR=PAR+1
ISN 0086              DEN=DEN+OP(PTR+15)
ISN 0087              DFA=DFA+OP(PTR+15)
ISN 0088              IF (DEN.LE.7) GOTO 120
ISN 0090              RETURN
                C
                C
```

235

L. Conway
Archives

IBM FORM NO. 419 8

```
                C               ERROR - OP CARD ADDRESS IS NOT MATCH
                C               CHECK FOR SKIPS
                C
ISN 0091          125 CONTINUE
ISN 0092            DFA1=DFA/8*8
ISN 0093            DFA2=DFA1+8
ISN 0094            IF (INOP(1).LT.DFA1) GOTO 127
ISN 0096            IF (INOP(1).GE.DFA2) GOTO 127
ISN 0098            DFA=INOP(1)                    IF (PAR.NE.0) RETURN
ISN 0099            DEN=MOD(DFA,8)
ISN 0100            GOTO 126
                C
ISN 0101          127 CONTINUE
ISN 0102            KY=3
ISN 0103            KNT(3)=KNT(3)+1
ISN 0104          128 CONTINUE
ISN 0105            IF (PAR.NE.0) RETURN
ISN 0107            LOCKD(DOCL)=0
ISN 0108            LODV(DOCL)=0
ISN 0109            LOSEQ(DOCL)=0
ISN 0110            RETURN
                C
                C               END OF INPUT DATA
                C
ISN 0111          150 CONTINUE
ISN 0112            KY = 10
ISN 0113            GOTO 2000
ISN 0114          152 CONTINUE
ISN 0115            IF (SY.EQ.231) GOTO 153
ISN 0117            IF (SY.EQ.193) GOTO 154
ISN 0119            KY=6
ISN 0120            GOTO 128
ISN 0121          153 KNT(8)=1
ISN 0122            KY=4
ISN 0123            GOTO 128
ISN 0124          154 KNT(9)=1
ISN 0125            KY=5
ISN 0126            GOTO 128
                C
                C
                C               SEARCH ROUTINE
                C                   SEARCH IB'S FOR ADDRESS
                C
ISN 0127            ENTRY SEARCH
                C
ISN 0128            KY=9
ISN 0129            SCT=SCT+1
ISN 0130            ASH=ASA/8*8
ISN 0131            DO 201 I=1,12
ISN 0132            IF (ASH.EQ.IBA(I)) GOTO 210
```

236

I. Conway Archives

IBM FORM NO. 419 0

```
ISN 0134        201 CONTINUE
                C
                C               ADDRESS NOT IN IB'S - GET NEXT AVAILABLE IB
                C
ISN 0135            IBCL=HIST(1)
ISN 0136            DO 202 I=1,11
ISN 0137        202 HIST(I)=HIST(I+1)
ISN 0138            HIST(12)=IBCL
                C
                C               REQUEST INSTRUCTION FETCH
                C
ISN 0139            IFDST2=IFDST
ISN 0140            IF (IFDST2.NE.0) LIBV(IFDST2)=0
ISN 0142            LIBV(IBCL)=1
ISN 0143            LIBW(IBCL)=5
ISN 0144            IBA(IBCL)=ASA/8*8
ISN 0145            IFADD=ASA/2
ISN 0146            IFDST=IBCL
ISN 0147            PBUF(1)=ASA
ISN 0148            PBUF(2)=IBCL
ISN 0149            PBUF(3)=DOCL
                C
ISN 0150        206 CONTINUE
ISN 0151            XX=1
ISN 0152            NFA=IBA(IBCL)
ISN 0153            IF (LDXW(DOCL).OR.LDAW(DOCL)) GOTO 205
ISN 0155            DOIB(DOCL)=IBCL
ISN 0156            LDDV(DOCL)=0
ISN 0157            LDEV(DOCL)=1
ISN 0158            DOSL=DOCL
ISN 0159        203 CONTINUE
ISN 0160            DOSL=DOSL+1
ISN 0161            IF (DOSL.GT.DOT) DOSL=1
ISN 0163            IF (DOSL.EQ.DOTL) GOTO 204
ISN 0165            LDEV(DOSL)=0
ISN 0166            GOTO 203
ISN 0167        204 CONTINUE
ISN 0168            DOSL=DOCL+1
ISN 0169            IF (DOSL.GT.DOT) DOSL=1
ISN 0171            RETURN
                C
ISN 0172        205 CONTINUE
ISN 0173            KY = 15
ISN 0174            GOTO 2000
                C
                C               FOUR-WORD ADDRESS MATCH - SET POINTER TO ENTRY
                C
ISN 0175        210 CONTINUE
ISN 0176            IF (LIBV(I).EQ.0) GOTO 201
ISN 0178            IF (LDXW(DOCL).OR.LDAW(DOCL)) GOTO 215
```

237

L. Conway
Archives

```
ISN 0180            IBCL=I
ISN 0181            IF (IBCL.EQ.HIST(12)) GOTO 214
ISN 0183            DO 211 I=1,11
ISN 0184            IF (IBCL.EQ.HIST(I)) GOTO 212
ISN 0186        211 CONTINUE
ISN 0187        212 CONTINUE
ISN 0188            DO 213 J=I,11
ISN 0189        213 HIST(J)=HIST(J+1)
ISN 0190            HIST(12)=IBCL
ISN 0191        214 CONTINUE
ISN 0192            GOTO 206
ISN 0193
ISN 0193        215 CONTINUE
ISN 0194            KY = 14
ISN 0195            GOTO 2000
                C
                C
                C
ISN 0196            ENTRY INTPH1
                C
                C               INITIALIZE PHASI COMMON
ISN 0197            DO 10 I=1,4300
ISN 0198         10 INT(I)=0
ISN 0199            DO 12 I=1,50
ISN 0200            KNT(I)=0
ISN 0201            PSC(I)=0
ISN 0202         12 CONTINUE
ISN 0203            PHEND=0
                C
                C                   SET INITIAL VALUES
ISN 0204            DOTL=1
ISN 0205            DOSL=1
ISN 0206            DOT=6
ISN 0207            IF (NODOT.NE.0) DOT=NODOT
                C
                C                       AND SET XIC, AIC, ETC.
ISN 0209            PNT=1
ISN 0210            AIC=ESTADD
ISN 0211            XIC=AIC
ISN 0212            NEA=AIC
ISN 0213            AICR=AIC
ISN 0214            XICR=XIC
ISN 0215            IBA(1)=NEA
ISN 0216            LIBV(1)=1
ISN 0217            LIBW(1)=5
ISN 0218            LDEV(1)=1
ISN 0219            DOIB(1)=1
ISN 0220            DOSL=2
ISN 0221            IFADD=NEA/2
ISN 0222            IFDST=1
```


L. Conway
Archives

```
                C
ISN 0223            J=1
ISN 0224            DO 11 I=1,12
ISN 0225            J=J+1
ISN 0226            IF (J.GT.12) J=1
ISN 0228         11 HIST(I)=J
ISN 0229            RETURN
                C
                C
                C              ROUTINE TO DUMP VARIABLES AND BUFFERS
                C
ISN 0230            ENTRY BUFR
                C
ISN 0231            GOTO 2000
                C
ISN 0232         2011 FORMAT ('0KY='I2,' DOT='I2,' DOTL='I2,' DOSL='I2,' DOCL='I2,
                    * ' IBCL='I2,' HISL='I2,2X,'XIC='I6,4(1XI1),' AIC='I6,4(1XI1),
                    *' ASA='I6)
ISN 0233         2012 FORMAT (' DO=',3(2X,4(1XI2),8(1XI1)))
ISN 0234         2013 FORMAT ('0IBA=',12(1XI6,1XI1,I1))
ISN 0235         2014 FORMAT ('     ',12(6XI1))
ISN 0236         2015 FORMAT ('0BR=',8(3(1XI1),1XI6))
ISN 0237         2016 FORMAT (' **', 9(1XI6),/)
ISN 0238         2017 FORMAT ('0     CYCL  XINPT  NXBUF  AINPT  NABUF     SY    SCT',
                    * '    FCT    PTR')
ISN 0239         2019 FORMAT ('  IB,ST,AP,XP,  EV,CKD,SEQ,DV,AW,AF,XW,XF')
ISN 0240         2020 FORMAT ('0 KNT= ',10(1XI6))
                C
ISN 0241         2000 CONTINUE
ISN 0242            WRITE (6,2011) KY, DOT,DOTL,DOSL,DOCL,IBCL,HISL,XIC,BRXP,SKXV,
                    * SKXC,SKXS,AIC,BRAP,SKAV,SKAC,SKAS,ASA
ISN 0243            WRITE (6,2017)
ISN 0244            WRITE (6,2016) CYCL, XINPT, NXBUF, AINPT, NABUF,SY,SCT,FCT,PTR
ISN 0245            WRITE (6,2020)(KNT(I),I=1,10)
                C
ISN 0246            J=1
ISN 0247            DO 2003 I=1,12
ISN 0248            WRT(J)   = IBA(I)
ISN 0249            WRT(J+1)=LIBV(I)
ISN 0250            WRT(J+2)=LIBW(I)
ISN 0251            J=J+3
ISN 0252         2003 CONTINUE
ISN 0253            WRITE (6,2013)(WRT(J),J=1,36)
                C
ISN 0254            I=DOTL-1
ISN 0255            WRITE (6,2019)
ISN 0256            DO 2001 N=1,DOT,3
ISN 0257            DO 2002 K=1,36,12
ISN 0258            I=I+1
ISN 0259            IF (I.GT.DOT) I=1
```

239

L. Conway
Archives

```
ISN 0261            WRT(K)    = DOIB(I)
ISN 0262            WRT(K+1)  = DOST(I)
ISN 0263            WRT(K+2)  = DOAP(I)
ISN 0264            WRT(K+3)  = DOXP(I)
ISN 0265            WRT(K+4)  = LDEV(I)
ISN 0266            WRT(K+5)  = LDCKD(I)
ISN 0267            WRT(K+6)  = LDSEQ(I)
ISN 0268            WRT(K+7)  = LDDV(I)
ISN 0269            WRT(K+8)  = LDAW(I)
ISN 0270            WRT(K+9)  = LDAF(I)
ISN 0271            WRT(K+10)= LDXW(I)
ISN 0272            WRT(K+11)= LDXF(I)
ISN 0273       2002 CONTINUE
ISN 0274            WRITE (6,2012)(WRT(J),J=1,36)
ISN 0275       2001 CONTINUE
               C
ISN 0276       2098 CONTINUE
ISN 0277            CALL PDUMP (DDTL,INDP(1),0)
ISN 0278            IF (KY.LT.10) GOTO 2099
ISN 0280            PHEND=PHEND+2
ISN 0281       2099 CONTINUE
ISN 0282            GOTO 1000
               C
ISN 0283            ENTRY PH1MAP
               C
ISN 0284       1000 CONTINUE
ISN 0285            IF (LDEV(17).EQ.0) GOTO 1002
ISN 0287            IF (LDXF(17).NE.0) GOTO 1002
ISN 0289            PH1(1)=LETR(DOIB(17)+1)
ISN 0290            PH1(2)=LETR(DOST(17)+1)
ISN 0291            K=3200
ISN 0292            DO 1001 J=3,10
ISN 0293            IF (OP(K+16).EQ.0) GOTO 1001
ISN 0295            PH1(J)=OP(K+23)
ISN 0296            IF (OP(K+15).EQ.1) GOTO 1001
ISN 0298            PH1(J+1)=PH1(J)
ISN 0299       1001 K=K+25
ISN 0300       1002 CONTINUE
ISN 0301            PH1(13)=PH1(11)
ISN 0302            PH1(11)=0
ISN 0303            IF (LDEV(18).EQ.0) GOTO 1004
ISN 0305            IF (LDXF(18).NE.0) GOTO 1004
ISN 0307            PH1(11)=LETR(DOIB(18)+1)
ISN 0309            PH1(12)=LETR(DOST(18)+1)
ISN 0309            K=3400
ISN 0310            DO 1003 J=13,20
ISN 0311            IF (OP(K+16).EQ.0) GOTO 1003
ISN 0313            PH1(J)=OP(K+23)
ISN 0314            IF (OP(K+15).EQ.1) GOTO 1003
ISN 0316            PH1(J+1)=PH1(J)
```

240

L. Conway
Archives

IBM FORM NO. 413 0

```
ISN 0317      1003 K=K+25
ISN 0318      1004 CONTINUE
ISN 0319           PH1(21)=0
ISN 0320           IF (LDEV(19).EQ.0) GOTO 1006
ISN 0322           IF (LDAF(19).NE.0) GOTO 1006
ISN 0324           PH1(21)=LETR(DDIB(19)+1)
ISN 0325           PH1(22)=LETR(DDST(19)+1)
ISN 0326           K=3600
ISN 0327           DO 1005 J=23,30
ISN 0328           IF (OP(K+17).EQ.0) GOTO 1005
ISN 0330           PH1(J)=OP(K+23)
ISN 0331           IF (OP(K+15).EQ.1) GOTO 1005
ISN 0333           PH1(J+1)=PH1(J)
ISN 0334      1005 K=K+25
ISN 0335      1006 CONTINUE
ISN 0336           PH1(33)=PH1(31)
ISN 0337           PH1(31)=0
ISN 0338           IF (LDEV(20).EQ.0) GOTO 1008
ISN 0340           IF (LDAF(20).NE.0) GOTO 1008
ISN 0342           PH1(31)=LETR(DDIB(20)+1)
ISN 0343           PH1(32)=LETR(DDST(20)+1)
ISN 0344           K=3900
ISN 0345           DO 1007 J=33,40
ISN 0346           IF (OP(K+17).EQ.0) GOTO 1007
ISN 0348           PH1(J)=OP(K+23)
ISN 0349           IF (OP(K+15).EQ.1) GOTO 1007
ISN 0351           PH1(J+1)=PH1(J)
ISN 0352      1007 K=K+25
ISN 0353      1008 CONTINUE
ISN 0354           PH1(41)=0
ISN 0355           DO 1009 J=1,12
ISN 0356           PH1(J+40)=LETR(HIST(J)+1)
ISN 0357      1009 CONTINUE
ISN 0358           PH1(54)=LETR(DDTL+1)
ISN 0359           PH1(55)=LETR(DDSL+1)
ISN 0360           DO 1010 J=1,DDT
ISN 0361           IF (LDEV(J).NE.0) PH1(J+56)=LETR(DDIB(J)+1)
ISN 0363      1010 CONTINUE
ISN 0364           PBUF(8)=KY
ISN 0365           KY=0
ISN 0366           J=66
ISN 0367           PH1(J+1)=LETR(PBUF(2)+1)
ISN 0368           PH1(J+2)=LETR(PBUF(3)+1)
ISN 0369           PH1(J+4)=LETR(PBUF(4)+1)
ISN 0370           PH1(J+5)=LETR(PBUF(8)+1)
ISN 0371           J=71
ISN 0372           DO 1012 I=1,8
ISN 0373           IF (LPSV(I)) PH1(J+1)=LETR(38)
ISN 0375      1012 CONTINUE
ISN 0376           J=80
```

241

L. Conway
Archives

```
ISN 0377            PH1(J+1)=LBX(4)
ISN 0378            PH1(J+2)=LBX(3)
ISN 0379            PH1(J+3)=LBX(2)
ISN 0380            PH1(J+4)=LBX(1)
ISN 0381            PH1(J+6)=LBA(4)
ISN 0382            PH1(J+7)=LBA(3)
ISN 0383            PH1(J+8)=LBA(2)
ISN 0384            PH1(J+9)=LBA(1)
ISN 0385            PH1(J+11)=LETR(BOSC+1)
ISN 0386            PH1(J+12)=LETR(XEP+1)
ISN 0387            PH1(J+13)=LETR(AEP+1)
ISN 0388            DO 1011 K=1,100
ISN 0389       1011 PH1(K)=PH1(K)*256
ISN 0390            IF (PRINT.NE.0) RETURN
ISN 0392            WRITE (6,810) CYCL,(PH1(I),I=1,100)
ISN 0393        810 FORMAT ('0PH1 'I4,4(1X2A1,1X8A1),1X40A1,1X20A1)
                C
ISN 0394            RETURN
                C
                C
                C
ISN 0395            END
```

242

L. Conway
Archives

```
LEVEL 2 FEB 67                          OS/360  FORTRAN H                          DATE  67.248/17.38.16

          COMPILER OPTIONS - NAME=  MAIN,OPT=00,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

     ISN 0002            SUBROUTINE UNROLL
     ISN 0003            IMPLICIT  INTEGER*2(J)
     ISN 0004            IMPLICIT INTEGER*2(R)
     ISN 0005            DIMENSION         JSMTAB (200,8), KSYMAD (200)
     ISN 0006            DIMENSION  JABSOP (300), JCDTYP (300), JASTCL  (300)
     ISN 0007            DIMENSION  LCARDN (300), LACSLC (300), JROWNM (300)
     ISN 0008            DIMENSION JCOLP  (300)
     ISN 0009            DIMENSION  JSIDBB (300)
     ISN 0010            DIMENSION  JIN (300,80)
     ISN 0011            DIMENSION JTEMP(6),JEND(4)
     ISN 0012            DIMENSION JTEMP4 (7)
     ISN 0013            DIMENSION JIREG(2), JJREG(2), JKREG(2)
     ISN 0014            DIMENSION JHFLD (5)
     ISN 0015            DIMENSION JSTOP(6)
     ISN 0016            DATA JSTOP/'S','T','O','P',' ',' '/
     ISN 0017            DATA JZERO/'0'/
     ISN 0018            DATA  JEND/'E',  'N', 'D', ' '/
     ISN 0019            DATA   JBLANK,JSMCLN/' ', ':'/
     ISN 0020            DATA JASTER/'*'/
     ISN 0021            DATA JLPARN, JRPARN, JCOMMA/'(',')',','/
     ISN 0022            COMMON /AREA2/ JNB(36), JOPCDE (6,256), JSIDB (256),
                        *JITYPE  (256),  JEXITF (300)
     ISN 0023            DIMENSION ROWTOT(200),RCNUSF(200),RLPNTR(200),
                        X ROWCRT(200),RSFTYP(200),RNUMSF(200),JACDNO(200)
     ISN 0024            COMMON/AREA4/JN(80),I,IJ
     ISN 0025            COMMON/PROG/JTRACE(1000,30),INSLOC
     ISN 0026            INTEGER*2 INSLOC
     ISN 0027            DIMENSION JRNUM(10)
     ISN 0028            DATA JRNUM/'0','1','2','3','4','5','6','7','8','9'/
                    C                INITIALIZE PROG TRACE AREA TO 0
     ISN 0029            DO 2000 LLL=1,1000
     ISN 0030            DO 2000 LK=1,30
     ISN 0031       2000 JTRACE(LLL,LK)=0
     ISN 0032            INTR=1
     ISN 0033            INSLOC=1
     ISN 0034            WRITE(6,3000)
     ISN 0035            WRITE(6,3001)
     ISN 0036            JDECK=1
                    C
                    C
                    C     INITIALIZE ONE ENTRY IN JSMTAB
     ISN 0037            III = 1
     ISN 0038            DO  60  LJJ  = 1,8
     ISN 0039         60 JSMTAB (III,LJJ) = JBLANK
                    C     INITIALIZE
     ISN 0040            IJK=0
     ISN 0041            LOC=0
     ISN 0042            M  = 1
     ISN 0043            MM =  1
```

243

L. Conway
Archives

```
ISN 0044            KSKPST = 0
ISN 0045            KBRSTT = 0
ISN 0046            NN = 1
                C                INITIALIZE WORKING AREAS
ISN 0047            DO 2005 LLL=1,200
ISN 0048            JACDNO(LLL)=0
ISN 0049            ROWTOT(LLL)=0
ISN 0050            RCNUSF(LLL)=0
ISN 0051            RLPNTR(LLL)=0
ISN 0052            ROWCRT(LLL)=1
ISN 0053            RSFTYP(LLL)=1
ISN 0054            RNUMSF(LLL)=1
ISN 0055         2005 CONTINUE
ISN 0056            DO 2006 LLL=1,300
ISN 0057            JEXITF(LLL)=0
ISN 0058         2006 CONTINUE
                C
                C
ISN 0059        10     IJK=IJK+1
ISN 0060               IF(IJK.LT.300) GO TO 11
ISN 0062               WRITE(6,3)
ISN 0063               STOP
ISN 0064           11 LCARDN (IJK)  =  IJK
                C      CHECK FOR COMMENT CARD
ISN 0065           12 READ  (5,1)  JN
ISN 0066              IF  (JN(1) .EQ. JASTER )  GO TO 12
ISN 0068              I = 1
ISN 0069              CALL BLNKCK(I,INT)
ISN 0070              I= INT
ISN 0071           20 KCOUNT = 0
ISN 0072              LACSLC(IJK)  =  LOC
                C      SCAN TO PICK UP WORD
ISN 0073           28 KCOL = I
ISN 0074           30 I = I+1
ISN 0075              DO 35 L = 1,36
ISN 0076              IF (JN(I)  .EQ. JNB(L)) GO TO 30
ISN 0078           35 CONTINUE
                C      SAVE LENGTH OF WORD
ISN 0079              KSUM = I-KCOL
ISN 0080              KCOLL = I-1
                C      CHECK FOR BLANK OR SEMICOLON
ISN 0081              IF ( JN(I) .EQ.  JASTER)  GO TO  98
ISN 0083              IF (JN(I) .EQ. JBLANK) GO TO 100
ISN 0085              IF (JN(I) .EQ. JSMCLN) GO TO 40
                C      BRANCH   TO SEARCH SYMBOL TABLE
ISN 0087           40 ASSIGN  205  TO  IA
ISN 0088              GO TO  250
ISN 0089          205 GO TO (210),IB
                C      LABEL WAS IN SYMBOL TABLE
                C      STORE  CURRENT  LOCATION
```

244

L. Conway
Archives

```
ISN 0090          KSYMAD(MX-1) = LOC
                C STORE  CURRENT  CARD NUMBER
ISN 0091          JACDNO (MX-1) = LCARDN(IJK)
ISN 0092          GO TO  215
                C LABEL  NOT IN SYMBOL TABLE
ISN 0093        210 KSYMAD(III -1) = LOC
                C STORE CURRENT CARD NUMBER
ISN 0094          JACDNO(III -1) = LCARDN(IJK)
ISN 0095        215 I = I + 1
ISN 0096          CALL BLNKCK (I,INT)
ISN 0097          I = INT
ISN 0098          IF (I - 80)  28,990,990
ISN 0099         98 JEXITF(IJK) =  1
ISN 0100          I  =  I  +  1
                C CHECK FOR END CARD OR OP CODE
ISN 0101        100 L = 1
ISN 0102          DO 105 IJ = KCOL,KCOLL
ISN 0103          IF (JN (IJ) .EQ. JEND(L)) GO TO 105
ISN 0105          GO TO 130
ISN 0106        105 L = L + 1
                C END CARD
ISN 0107          JCDTYP(IJK)  = 12
ISN 0108          WRITE  (6,6)  JN
                C MOVE CURRENT CARD TO FILE
ISN 0109          DO 106  N = 1,80
ISN 0110        106 JIN (IJK,N) = JN(N)
ISN 0111          GO TO 699
ISN 0112        990 WRITE (6,7)
ISN 0113          GO TO   180
ISN 0114        993 WRITE  (6,4)
ISN 0115          GO TO 708
                C **
                C CHECK FOR OP-CODE
ISN 0116        130 JASTCL  (IJK)  = I
                C SAVE  STARTING COLUMN OF  OP-CODE
ISN 0117          JCOLP(IJK)  =  KCOL
                C CLEAR JTEMP
ISN 0118          DO 132 K=1,6
ISN 0119        132 JTEMP(K) = JBLANK
ISN 0120          L = 0
ISN 0121          DO 135 II = KCOL,KCOLL
ISN 0122          L = L+1
ISN 0123        135 JTEMP(L) = JN(II)
ISN 0124          KTOTAL = KCOLL - KCOL + 1
                C SEARCH OP CODE TABLE
ISN 0125          DO 140 II=1,256
ISN 0126          DO 139 L=1,6
ISN 0127          IF(JOPCDE(L,II).NE.JTEMP(L)) GO TO 140
ISN 0129        139 CONTINUE
ISN 0130          GO TO 160
```

245

L. Conway
Archives

IBM FORM NO. 413 8

```
ISN 0131      140   CONTINUE
              C     UNSUCCESSFUL SEARCH
ISN 0132            WRITE(6,8)
ISN 0133            GO TO 180
              C     SUCCESSFUL SEARCH
ISN 0134        160 CONTINUE
ISN 0135        161 JABSOP(IJK)=II
ISN 0136            IF(II.EQ.256) JABSOP(IJK)=999
ISN 0138            JCDTYP(IJK)=JITYPE(II)
ISN 0139            JSIDBB(IJK)=JSIDB(II)
ISN 0140            IF(JSIDB(II)-1) 165,165,170
ISN 0141        165 LOC = LOC + 1
ISN 0142            GO TO 175
ISN 0143        170 LOC = LOC + 2
ISN 0144        175 K = JCDTYP(IJK)
ISN 0145            GO TO (185,185), K
ISN 0146        180 WRITE  (6,2)   LACSLC (IJK),  JN
              C     MOVE CURRENT CARD TO FILE
ISN 0147            DO  107  N = 1,80
ISN 0148        107 JIN(IJK,N) = JN(N)
ISN 0149            GO TO 10
              C     SCAN TO A LEFT PAREN
ISN 0150        185 IK = JASTCL (IJK)
ISN 0151            DO 187 I = IK, 80
ISN 0152            IF (JN (I) .EQ. JLPARN) GO TO 300
ISN 0154        187 CONTINUE
ISN 0155            GO TO 180
              C     SCAN THE BRANCH AND/OR SKIP PARAMETERS
ISN 0156        300 JROWNM (IJK)  =  NN
ISN 0157            NNN  =  NN
ISN 0158        304 CONTINUE
ISN 0159        302 I = I + 1
ISN 0160        303 IF (JN(I) .EQ. JASTER) GO TO 320
ISN 0162            IF (JN(I) .EQ. JCOMMA) GO TO 340
ISN 0164            IF (JN(I) .EQ. JRPARN) GO TO 330
              C     CHECK FOR NUMERIC
ISN 0166            DO 305 J = 27,36
ISN 0167            IF (JN (I) .EQ. JNB(J)) GO TO 350
ISN 0169        305 CONTINUE
              C     IF CHARACTER IS NONE OF ABOVE-ASSUME TO BE LETTER
              C     COLLECT THE LABEL
ISN 0170        360 KCOL = I
ISN 0171        365 I = I + 1
ISN 0172            DO 370 L = 1,36
ISN 0173            IF (JN(I) .EQ. JNB (L)) GO TO 365
ISN 0175        370 CONTINUE
              C     CHECK FOR SKIP INSTRUCTION AND
              C     BYPASS SEARCH OF SYMBOL TABLE IF SKIP
ISN 0176        371 IF (JCDTYP(IJK) - 2)   372, 303, 372
ISN 0177        372 KSUM = I - KCOL
```


L. Conway
Archives

```
ISN 0178            KCOLL = I - 1
ISN 0179            KK = 0
                C   SEARCH  SYMBOL  TABLE
                C   BRANCH  INSTRUCTION
ISN 0180      400   ASSIGN  405 TO IA
ISN 0181            GO TO 250
ISN 0182      405   GO TO (410),IB
                C   LABEL WAS IN SYMBOL TABLE
                C   STORE INDEX  IN  POINTER
ISN 0183            RLPNTR(NN)  = MX - 1
ISN 0184            GO TO 303
                C   LABEL  NOT IN SYMBOL TABLE
ISN 0185      410   RLPNTR(NN) =  III - 1
ISN 0186            GO TO 303
                C   ASTERISK
ISN 0187      320   RSFTYP(NN) = 2
ISN 0188            GO TO 304
                C   RIGHT PAREN
ISN 0189      330   ROWTOT (NNN) =  ROWTOT (NNN) + 1
ISN 0190            NN = NN +1
ISN 0191            GO TO 180
                C   NUMBER
ISN 0192      350   RNUMSF(NN) = J - 27
ISN 0193            I = I + 1
ISN 0194            DO 307  J = 27,36
ISN 0195            IF (JN(I) .EQ. JNB(J))  GO TO 351
ISN 0197      307   CONTINUE
ISN 0198            GO TO 303
                C   TWO DIGIT PARAMETER
ISN 0199      351   RNUMSF(NN)  = 10*RNUMSF(NN)  + J - 27
ISN 0200            GO TO 302
                C   COMMA
ISN 0201      340   ROWTOT (NNN) =  ROWTOT (NNN) + 1
ISN 0202            NN = NN + 1
ISN 0203            GO TO 302
                C
                C
                C   SECOND PASS. UNROLL LOOPS AND PRODUCE FINAL OUTPUT
ISN 0204      699   K  = 1
ISN 0205            KN = 1
ISN 0206      700   K = KN
ISN 0207     7059   DO  7069  NI = 1,2
ISN 0208            JIREG(NI)     =   JZERO
ISN 0209            JJREG(NI)     =   JZERO
ISN 0210     7069   JKREG(NI)     =   JZERO
ISN 0211            DO 1719 L = 1,5
ISN 0212     1719   JHFLD(L) = JZERO
                C   CHECK  FOR   END  CARD
ISN 0213            IF (JCDTYP(K)  - 12 )  701,715,715
ISN 0214      701   KK = JCDTYP(K)
```

247

L. Conway
Archives

```
ISN 0215            GO TO ( 704,790,705) , KK
ISN 0216      725   KN  = KN + 1
ISN 0217            GO TO 730
              C     BRANCH INSTRUCTION
ISN 0218      704   IF (JEXITF(K) - 1 ) 703,7725,7725
ISN 0219     7725   IF (KSKPST - 1 ) 703,725,725
              C     SKIP  INSTRUCTION
ISN 0220      790   IF  (KSKPST - 1 ) 703, 7911,7911
ISN 0221     7911   IF  (JEXITF(K) - 1)791,725,725
ISN 0222      791   MM =2
              C     BRANCH OR SKIP INSTRUCTION
ISN 0223      703   JTYPE  = JCDTYP(K)
ISN 0224            N  =  JROWNM(K)
ISN 0225            GO TO 800
              C     ANALYZE I,J,K, AND H FIELDS
ISN 0226      730   I  =  JASTCL(K)
ISN 0227            DO 190 IM = 1,80
ISN 0228      190   JN(IM) = JIN (K,IM)
ISN 0229            CALL BLNKCK (I,INT)
ISN 0230            I = INT
ISN 0231            IF (I-80) 158,708,708
ISN 0232      158   IF (KK -7)  1589,708,708
ISN 0233     1589   IF (JN(I) .EQ. JCOMMA)  GO TO 157
ISN 0235            CALL ANIJK (JIREG)
ISN 0236            GO TO (993),IJ
ISN 0237      151   IF (KK - 6)  1519,155,155
ISN 0238     1519   IF (JN(I)  .NE. JCOMMA )  GO TO 993
ISN 0240            I = I+1
ISN 0241            IF (JN(I) .EQ. JCOMMA) GO TO 1539
ISN 0243            CALL ANIJK (JJREG)
ISN 0244            GO TO (993),IJ
ISN 0245            IF (KK - 5 )  153,155,155
ISN 0246      153   IF (JN(I)  .NE. JCOMMA) GO TO 159
ISN 0248     1539   I = I+1
ISN 0249            IF (JN(I)  .EQ. JCOMMA) GO TO 154
ISN 0251            CALL ANIJK (JKREG)
ISN 0252            GO TO (993),IJ
ISN 0253      155   IF  (JN(I)  .NE. JCOMMA )  GO TO 708
ISN 0255            I = I + 1
ISN 0256            IF (JN(I) .EQ. JCOMMA) GO TO 708
ISN 0258            DO 152 L = 1,26
ISN 0259      152   IF (JN(I) .EQ. JNB(L)) GO TO 708
ISN 0261            DO 172 L = 27,36
ISN 0262            IF (JN(I) .EQ. JNB(L)) GO TO 173
ISN 0264      172   CONTINUE
ISN 0265            GO TO 993
ISN 0266      173   KI = 0
ISN 0267            IA = I
ISN 0268      174   I = I + 1
ISN 0269            KI = KI + 1
```

248

L. Conway
Archives

```
ISN 0270            DO 176 L = 27,36
ISN 0271            IF (JN(I) .EQ. JN8(L))  GO TO 174
ISN 0273        176 CONTINUE
ISN 0274            J = 5
ISN 0275            KK =KI - 1
ISN 0276            DO 177 LL = 1,KI
ISN 0277            JHFLD(J) = JN(IA+KK)
ISN 0278            J = J - 1
ISN 0279        177 KK = KK-1
ISN 0280            GO TO 708
ISN 0281        157 I = I +1
ISN 0282            IF (JN(I)  .NE. JCOMMA)  GO TO 993
ISN 0284            GO TO 151
ISN 0285        159 IF (JN(I) .EQ. JBLANK)  GO TO 708
ISN 0287            GO TO 993
ISN 0288        154 I = I+1
ISN 0289            GO TO 155
                C   EXIT INSTRUCTION
ISN 0290        705 IF  (KBRSTT - 1 ) 707,720,720
ISN 0291        707 KN = KN + 1
ISN 0292            GO TO 708
                C   BRANCH STATE IS ACTIVE
ISN 0293        720 KN = JACDNO(KX)
ISN 0294            M = 2
ISN 0295            JNXTLC = KSYMAD(KX)
ISN 0296            GO TO  770
ISN 0297        708 CONTINUE
ISN 0298            IF(JSIDBB (K) - 1)  760, 760,765
ISN 0299        760 JNXTLC  = LACSLC (K)  + 1
ISN 0300            GO TO 770
ISN 0301        765 JNXTLC  = LACSLC (K) +  2
ISN 0302        770 LIN = JCOLP(K)
ISN 0303            LIN1 = JASTCL(K) -1
ISN 0304            DO 785 J = 1,7
ISN 0305        785 JTEMP4 (J) = JBLANK
ISN 0306            JJ  =  0
ISN 0307            DO 786  J =  LIN, LIN1
ISN 0308            JJ = JJ + 1
ISN 0309        786 JTEMP4 (JJ) = JIN (K,J)
                C
                C
                C               IF INPUT PARAM JDECK=1, PRINT-PUNCH CARD OF TRACE
ISN 0310            IF(JDECK.NE.1) GO TO 2900
                C   PRINT OUTPUT
ISN 0312        797 WRITE (6,950)  LACSLC(K),  (JTEMP4(J), J = 1,7),
                   *(JIREG(I), I =1,2), (JJREG(I),I=1,2),(JKREG(I), I = 1,2),
                   *(JHFLD(I), I = 1,5), KBRSTT,
                   *   KSKPST,JEXITF(K), (JHFLD(I), I = 1,5), JNXTLC,
                   *              JABSOP(K), JSIDBB(K)
                C
```

249
L. Conway
Archives

```
                C
                C               PLACE INST INTO TRACE
      ISN 0313   2900 CONTINUE
      ISN 0314        JTRACE(INTR,1)=LACSLC(K)
      ISN 0315        JIR=0
      ISN 0316        JJR=0
      ISN 0317        JKR=0
      ISN 0318        DO 2960 LK=1,10
      ISN 0319        IF(JIREG(1).EQ.JRNUM(LK)) JIR=JIR+10*(LK-1)
      ISN 0321        IF(JIREG(2).EQ.JRNUM(LK)) JIR=JIR+LK-1
      ISN 0323        IF(JJREG(1).EQ.JRNUM(LK)) JJR=JJR+10*(LK-1)
      ISN 0325        IF(JJREG(2).EQ.JRNUM(LK)) JJR=JJR+LK-1
      ISN 0327        IF(JKREG(1).EQ.JRNUM(LK)) JKR=JKR+10*(LK-1)
      ISN 0329        IF(JKREG(2).EQ.JRNUM(LK)) JKR=JKR+LK-1
      ISN 0331   2960 CONTINUE
      ISN 0332        JTRACE(INTR,5)=JIR
      ISN 0333        JTRACE(INTR,6)=JJR
      ISN 0334        JTRACE(INTR,7)=JKR
      ISN 0335        JHTEMP=0
      ISN 0336        DO 2961 LK=1,10
      ISN 0337        IF(JHFLD(5).EQ.JRNUM(LK)) JHTEMP=JHTEMP+LK-1
      ISN 0339        IF(JHFLD(4).EQ.JRNUM(LK)) JHTEMP=JHTEMP+10*(LK-1)
      ISN 0341        IF(JHFLD(3).EQ.JRNUM(LK)) JHTEMP=JHTEMP+100*(LK-1)
      ISN 0343        IF(JHFLD(2).EQ.JRNUM(LK)) JHTEMP=JHTEMP+1000*(LK-1)
      ISN 0345        IF(JHFLD(1).EQ.JRNUM(LK)) JHTEMP=JHTEMP+10000*(LK-1)
      ISN 0347   2961 CONTINUE
      ISN 0348        JTRACE(INTR,8)=JHTEMP
      ISN 0349        JTRACE(INTR,9)=KBRSTT
      ISN 0350        JTRACE(INTR,10)=KSKPST
      ISN 0351        JTRACE(INTR,11)=JEXITF(K)
      ISN 0352        JTRACE(INTR,12)=JHTEMP
      ISN 0353        JTRACE(INTR,13)=JNXTLC
      ISN 0354        JTRACE(INTR,14)=JABSOP(K)
      ISN 0355        JTRACE(INTR,15)=JSIDBB(K)
      ISN 0356        DO 2977 LK=1,6
      ISN 0357        JTRACE(INTR,LK+20)=JTEMP4(LK)
      ISN 0358   2977 CONTINUE
                C
                C
                C               INCR. TRACE INPUT POINTER
      ISN 0359        INTR=INTR+1
      ISN 0360        IF(INTR.LE.1000) GO TO 2950
      ISN 0362        WRITE(6,2910)
      ISN 0363        RETURN
      ISN 0364   2950 CONTINUE
      ISN 0365        GO TO (781,780), M
      ISN 0366    780 M = 1
      ISN 0367        KBRSTT  = 0
      ISN 0368    781 GO TO (796, 793, 795), MM
      ISN 0369    793 KSKPST  =  0
```

250
L. Conway
Archives

```
ISN 0370            MM = 1
ISN 0371            GO TO  796
ISN 0372        795 KSKPST  =  1
ISN 0373            MM = 1
ISN 0374        796 GO TO 700
ISN 0375        715 CONTINUE
ISN 0376            JTRACE(INTR,1)=LACSLC(K)
ISN 0377            JTRACE(INTR,14)=JABSOP(K)
ISN 0378            JTRACE(INTR,15)=JSIDBB(K)
ISN 0379            DO 2978 LK=1,6
ISN 0380            JTRACE(INTR,LK+20)=JSTOP(LK)
ISN 0381       2978 CONTINUE
ISN 0382            WRITE(6,3002)
ISN 0383            RETURN
                C
                C
                C     ANALYZE BRANCH OR SKIP ENTRY IN TABLE
ISN 0384          800 GO TO (802,803),JTYPE
                C     EXIT IF BRANCH STATE ACTIVE
ISN 0385          802 IF (KBRSTT - 1) 803,725, 725
ISN 0386          803 IF (ROWCRT (N)  - ROWTOT(N)) 801, 801, 725
                C     PROCESS NEXT ROW
ISN 0387          801 KL = ROWCRT (N)
ISN 0388              KL = KL + N - 1
ISN 0389              L  = RSFTYP (KL)
ISN 0390              GO TO (819,830), JTYPE
ISN 0391          819 GO TO  (820,830), L
                C     SUCCESS
ISN 0392          820 KX = RLPNTR(KL)
                C     FAILURE
ISN 0393          830 RCNUSF (KL) = RCNUSF (KL)  + 1
ISN 0394              IF (RCNUSF (KL)- RNUMSF (KL)) 850,840, 840
ISN 0395          840 ROWCRT(N) = ROWCRT (N) + 1
ISN 0396          850 GO TO (860,880),L
ISN 0397          860 GO TO (870,890),JTYPE
ISN 0398          870 KBRSTT = 1
ISN 0399          880 GO TO 725
ISN 0400          890 MM  = 3
ISN 0401              GO TO 725
                C     SEARCH  SYMBOL TABLE  ROUTINE
ISN 0402          250 IB = 1
ISN 0403              KKCNT =  0
ISN 0404              DO  260  MX =  1,III
ISN 0405              IF (KKCNT - KSUM) 255,225,255
ISN 0406          255 KKCNT = 0
ISN 0407              J = 0
ISN 0408              DO 260  L = KCOL,KCOLL
ISN 0409              J = J + 1
ISN 0410              IF (JSMTAB(MX,J) .EQ.  JN(L))
                     *KKCNT = KKCNT + 1
```

251

L. Conway
Archives

```
ISN 0412        260 CONTINUE
                C     LABEL  NOT  IN  SYMBOL TABLE
                C     STORE  LABEL
ISN 0413              L = 1
ISN 0414              DO  265  II  = KCOL,KCOLL
ISN 0415              JSMTAB(III,L) = JN(II)
ISN 0416        265   L = L + 1
                C     INITIALIZE NEXT SYMBOLIC  LOCATION TO  BLANKS
ISN 0417              III  = III + 1
ISN 0418              DO 268  LJJ = 1,8
ISN 0419        268   JSMTAB  (III,LJJ) = JBLANK
ISN 0420              GO TO  1A,(205,405)
                C     LABEL WAS IN SYMBOL TABLE
                C     DOUBLE CHECK FOR CORRECT MATCH
ISN 0421        225   IF (KSUM - 8) 226,230,990
ISN 0422        226   IF (JSMTAB(MX-1,J+1) .NE. JBLANK)  GO TO 255
ISN 0424        230   IB = 2
ISN 0425              GO TO  1A,(205,405)
ISN 0426          1   FORMAT (80A1)
ISN 0427          2   FORMAT  (' ',  I10,  80A1)
ISN 0428          3   FORMAT   (' ','TOO MANY INPUT  CARDS')
ISN 0429          4   FORMAT  (' ', 'OPERAND FIELD ERROR')
ISN 0430          6   FORMAT(' ',10X,80A1)
ISN 0431          7   FORMAT  (' ERROR ON FOLLOWING CARD')
ISN 0432          8   FORMAT  (' ','OP CODE  ON NEXT CARD  NOT  IMPLEMENTED')
ISN 0433        950   FORMAT  (' ',I6,1X,7A1,1X,2(2A1,1X),2A1,2X,5A1,4X,3I1,
                     *3X,5A1,2X,I5,2X,I3,2X,I1)
ISN 0434       2910   FORMAT(' TRACE EXCEEDS 1000 INSTRUCTIONS - - - TERM.UNROLL')
ISN 0435       3000   FORMAT(' - - - - - UNROLLER INPUT PROGRAM - - - - - -')
ISN 0436       3001   FORMAT(1H0)
ISN 0437       3002   FORMAT(1H1)
ISN 0438              END
```

IBM FORM NO. 419 8

L. Conway
Archives

COMPILER OPTIONS - NAME= MAIN,OPT=00,LINECNT=56,SOURCE,EBCDIC,NOLIST,NODECK,LOAD,MAP,NOEDIT,ID,XREF

```
ISN 0002            SUBROUTINE BLNKCK (N,INN)
ISN 0003            IMPLICIT  INTEGER*2(J)
ISN 0004            INTEGER*2 N,INN
ISN 0005            DATA  JBLANK/' '/
ISN 0006            COMMON/AREA4/JN(80),II,IJ
ISN 0007            DO 10 I = N,80
ISN 0008            IF (JN(I) .EQ. JBLANK) GO TO 10
ISN 0010            INN=I
ISN 0011            RETURN
ISN 0012         10 CONTINUE
ISN 0013            INN= 80
ISN 0014            RETURN
ISN 0015            END
```


L. Conway Archives

```
LEVEL 14 ( 1 JUN 67)                    OS/360  FORTRAN H                    DATE  68.074/14.59.34

          COMPILER OPTIONS - NAME=  MAIN,OPT=00,LINECNT=56,SOURCE,EBCDIC,NOLIST,NODECK,LOAD,MAP,NOEDIT,ID,XREF

     ISN 0002            SUBROUTINE ANIJK (JANS)
     ISN 0003            IMPLICIT INTEGER*2(J)
     ISN 0004            INTEGER*2 JANS
     ISN 0005            DATA JZERO/'0'/
     ISN 0006            COMMON /AREA2/ JNB(36), JOPCDE (6,256), JSIDB (256),
                        *JITYPE  (256),  JEXITF (300)
     ISN 0007            COMMON/AREA4/JN(80),I,IJ
     ISN 0008            DIMENSION JANS(2)
     ISN 0009            IJ = 2
     ISN 0010            DO 10 L = 1,26
     ISN 0011            IF (JN(I)  .EQ. JNB(L))  GO TO 40
     ISN 0013         10 CONTINUE
     ISN 0014         15 DO 20 L= 27,36
     ISN 0015            IF (JN(I) .EQ. JNB(L)) GO TO 25
     ISN 0017         20 CONTINUE
     ISN 0018            IJ = 1
     ISN 0019            RETURN
     ISN 0020         25 I = I + 1
     ISN 0021            DO 30 L = 27,36
     ISN 0022            IF (JN(I) .EQ. JNB(L)) GO TO 35
     ISN 0024         30 CONTINUE
     ISN 0025            JANS(1) = JZERO
     ISN 0026            JANS(2) = JN(I-1)
     ISN 0027            RETURN
     ISN 0028         35 JANS(1) = JN (I-1)
     ISN 0029            JANS(2) = JN (I)
     ISN 0030            I = I + 1
     ISN 0031            RETURN
     ISN 0032         40 I = I + 1
     ISN 0033            GO TO 15
     ISN 0034            END
```


L. Conway
Archives

```
LEVEL 5 DEC 66                          DATE  YS/360  FORTRAN H                          DATE  67.236/21.58.05

        COMPILER OPTIONS - NAME=  MAIN,OPT=00,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

ISN 0002            BLOCK DATA
ISN 0003            IMPLICIT INTEGER*2(A-Z)
                C
                C           FORM THE DECODE TABLE HERE WITH DATA STATEMENTS
ISN 0004            COMMON/TAGS/
                   X        T01(256),T02(256),T03(256),T04(256),T05(256),T06(256),
                   X T07(256),T08(256),T09(256),T10(256),T11(256),T12(256),T13(256),
                   X T14(256),T15(256),T16(256),T17(256),T18(256),T19(256),T20(256),
                   X T21(256),T22(256),T23(256),T24(256),T25(256),T26(256),T27(256),
                   X T28(256),T29(256),T30(256),T31(256),T32(256),T33(256),T34(256),
                   X T35(256),T36(256),T37(256),T38(256),T39(256),T40(256),T41(256),
                   X T42(256),T43(256),T44(256),T45(256),T46(256),T47(256),T48(256),
                   X T49(256),T50(256),T51(256),T52(256),T53(256),T54(256),T55(256),
                   X T56(256),T57(256),T58(256),T59(256),T60(256),T61(256),T62(256),
                   X T63(256),T64(256),T65(256),T66(256),T67(256),T68(256),T69(256),
                   X T70(256)
ISN 0005            DATA T01/1,1,0,1,1,3*0,1,1,0,1,1,5*0,8*1,11*0,1,0,3*1,4*0,4*1,
                   X 55*0,14*1,3*0,1,1,0,0,1,1,0,0,10*1,0,1,5*0,1,0,1,5*0,1,0,1,0,1,
                   Y 0,1,0,1,13*0,16*1,3*0,1,1,3*0,8*1,0,3*1,0,8*1,0,1,32*0/
ISN 0006            DATA T02/8*0,1,1,0,1,1,5*0,8*1,11*0,1,0,1,1,7*0,1,1,5*0,1,0,1,0,
                   X 1,0,1,0,1,0,1,0,1,0,1,0,1,0,0,0,1,0,1,4*0,4*1,0,0,3*1,0,1,0,8*1,
                   Y 20*0,1,1,0,0,1,1,0,0,3*1,6*0,1,6*0,1,0,1,5*0,1,0,1,0,1,0,1,0,1,
                   Z 6*0,8*1,16*0,3*1,14*0,3*1,0,8*1,0,1,32*0/
ISN 0007            DATA T03/201*0,8*1,47*0/
ISN 0008            DATA T04/11*0,1,1,7*0,1,1,0,0,1,1,122*0,1,5*0,1,0,1,5*0,1,93*0/
ISN 0009            DATA T05/8*0,1,1,8*0,1,1,0,0,1,1,13*0,1,0,1,1,14*0,1,0,1,0,1,0,
                   X 1,0,1,0,1,0,1,0,1,0,1,3*0,1,0,1,4*0,4*1,0,0,3*1,0,1,0,8*1,44*0,
                   Y 1,0,1,5*0,1,0,1,0,1,0,1,0,1,6*0,8*1,16*0,3*1,60*0/
ISN 0010            DATA T06/256*0/
ISN 0011            DATA T07/256*0/
ISN 0012            DATA T08/48*0,1,1,5*0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,3*0,1,0,
                   X 1,4*0,4*1,0,0,3*1,0,1,0,8*1,20*0,1,1,0,0,1,1,0,0,3*1,6*0,1,6*0,
                   Y 1,7*0,1,0,1,0,1,0,1,8*0,8*1,16*0,3*1,60*0/
ISN 0013            DATA T09/154*0,1,101*0/
ISN 0014            DATA T10/256*0/
ISN 0015            DATA T11/55*0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,3*0,1,0,1,10*0,
                   X 3*1,0,1,0,4*1,24*0,1,1,0,0,1,1,0,0,3*1,6*0,1,6*0,1,9*0,1,0,1,0,
                   Y 1,8*0,8*1,17*0,2*1,60*0/
ISN 0016            DATA T12/256*0/
ISN 0017            DATA T13/256*0/
ISN 0018            DATA T14/256*0/
ISN 0019            DATA T15/48*0,1,1,72*0,1,1,0,0,1,1,0,0,3*1,6*0,1,116*0/
ISN 0020            DATA T16/3*0,1,1,42*0,1,101*0,1,5*0,1,0,1,5*0,1,92*0/
ISN 0021            DATA T17/1,1,39*0,1,5*0,1,57*0,14*1,28*0,1,0,1,5*0,1,0,1,0,1,0,
                   X 1,0,1,13*0,8*1,11*0,1,1,13*0,1,1,43*0/
ISN 0022            DATA T18/256*0/
ISN 0023            DATA T19/108*0,1,5*0,1,141*0/
ISN 0024            DATA T20/1,1,0,1,1,3*0,1,1,0,1,1,5*0,8*1,11*0,1,8*0,1,58*0,14*1,
                   X 14*0,6*1,0,0,1,5*0,1,7*0,1,0,1,0,1,0,1,15*0,8*1,11*0,1,1,14*0,
```

255

L. Conway
Archives

```
     Y 1,43*0/
ISN 0025      DATA T21/155*0,1,100*0/
ISN 0026      DATA T22/1,1,0,1,1,3*0,1,1,0,1,1,5*0,8*1,11*0,1,67*0,6*1,22*0,
     X 3*1,5*0,1,5*0,1,9*0,1,0,1,0,1,15*0,8*1,11*0,1,1,3*0,8*1,47*0/
ISN 0027      DATA T23/8*0,1,1,8*0,1,1,0,0,1,1,13*0,1,0,1,1,215*0/
ISN 0028      DATA T24/201*0,8*1,5*0,8*1,34*0/
ISN 0029      DATA T25/46*0,1,0,1,1,72*0,1,1,0,0,1,1,0,0,10*1,0,1,43*0,8*1,63*0/
ISN 0030      DATA T26/47*0,1,137*0,8*1,8*0,8*1,5*0,8*1,34*0/
ISN 0031      DATA T27/1,1,6*0,1,1,8*0,1,1,0,0,1,1,232*0/
ISN 0032      DATA T28/3*0,1,1,6*0,1,1,7*0,1,1,0,0,1,1,230*0/
ISN 0033      DATA T29/214*0,8*1,34*0/
ISN 0034      DATA T30/0,0,1,0,0,3*1,0,0,1,0,0,5*1,8*0,11*1,0,1,3*0,4*1,4*0,
     X 5*1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,3*1,0,1,0,4*1,4*0,1,1,
     Y 3*0,1,0,1,8*0,3*1,14*0,3*1,0,0,1,1,0,0,1,1,10*0,1,0,4*1,4*0,
     Z 4*1,10*0,5*1,29*0,3*1,8*0,1,3*0,1,8*0,1,0,32*1/
ISN 0035      DATA T31/210*0,3*1,43*0/
ISN 0036      DATA T32/48*0,1,1,72*0,1,1,0,0,1,1,0,0,3*1,6*0,1,116*0/
ISN 0037      DATA T33/8*0,1,1,8*0,1,1,0,0,1,1,13*0,1,0,1,1,215*0/
ISN 0038      DATA T34/185*0,8*1,63*0/
ISN 0039      DATA T35/256*0/
ISN 0040      DATA T36/256*0/
ISN 0041      DATA T37/256*0/
ISN 0042      DATA T38/256*0/
ISN 0043      DATA T39/256*0/
ISN 0044      DATA T40/256*0/
ISN 0045      DATA T41/2,2,0,2,2,3*0,2,2,0,2,2,5*0,8*2,230*0/
ISN 0046      DATA T42/2,2,0,2,2,3*0,2,2,0,2,2,5*0,8*2,230*0/
ISN 0047      DATA T43/41*0,1,63*0,1,1,4*0,1,1,4*0,1,1,58*0,8*1,11*0,1,1,3*0,
     X 8*1,0,0,1,1,43*0/
ISN 0048      DATA T44/147*0,1,0,1,5*0,1,0,1,0,1,0,1,0,1,92*0/
ISN 0049      DATA T45/107*0,1,5*0,1,142*0/
ISN 0050      DATA T46/108*0,3*1,3*0,3*1,139*0/
ISN 0051      DATA T47/37*0,1,0,1,1,5*0,1,209*0/
ISN 0052      DATA T48/133*0,6*1,0,0,1,114*0/
ISN 0053      DATA T49/47*0,1,137*0,8*1,63*0/
ISN 0054      DATA T50/256*0/
ISN 0055      DATA T51/256*0/
ISN 0056      DATA T52/256*0/
ISN 0057      DATA T53/256*0/
ISN 0058      DATA T54/256*0/
ISN 0059      DATA T55/256*0/
ISN 0060      DATA T56/55*0,2,0,2,0,2,0,2,0,2,0,2,0,2,190*0/
ISN 0061      DATA T57/55*0,2,0,2,0,2,0,2,0,2,0,2,0,2,34*0,1,1,154*0/
ISN 0062      DATA T58/67*0,1,0,1,0,1,184*0/
ISN 0063      DATA T59/75*0,1,0,1,178*0/
ISN 0064      DATA T60/88*0,1,1,4*0,1,1,1,1,1,1,93*0,3*1,60*0/
ISN 0065      DATA T61/90*0,1,165*0/
ISN 0066      DATA T62/92*0,1,163*0/
ISN 0067      DATA T63/48*0,1,1,72*0,1,1,0,0,1,1,0,0,3*1,6*0,1,116*0/
ISN 0068      DATA T64/82*0,1,1,85*0,8*1,79*0/
```

256

L. Conway
Archives

```
ISN 0069          DATA T65/84*0,1,1,60*0,1,0,1,5*0,1,0,1,0,1,0,1,0,1,93*0/
ISN 0070          DATA T66/256*0/
ISN 0071          DATA T67/256*0/
ISN 0072          DATA T68/256*0/
ISN 0073          DATA T69/256*0/
ISN 0074          DATA T70/256*0/
                C
                C
                C
                C               UNROLLER DATA
                C
ISN 0075          DIMENSION JNB(36),  JITYPE  (256)
ISN 0076          DIMENSION  JOPCDE  (6,256), JSIDB  (256)
ISN 0077          DIMENSION J1(96),J2(96),J3(96),J4(96),J5(96),J6(96),J7(96),J8(96),
                 XJ9(96),J10(96),J11(96),J12(96),J13(96),J14(96),J15(96),J16(96)
ISN 0078          EQUIVALENCE (J1,JOPCDE(1,1)   ),
                 X            (J2,JOPCDE(1,17)  ),
                 X            (J3,JOPCDE(1,33)  ),
                 X            (J4,JOPCDE(1,49)  ),
                 X            (J5,JOPCDE(1,65)  ),
                 X            (J6,JOPCDE(1,81)  ),
                 X            (J7,JOPCDE(1,97)  ),
                 X            (J8,JOPCDE(1,113) ),
                 X            (J9,JOPCDE(1,129) ),
                 X            (J10,JOPCDE(1,145)),
                 X            (J11,JOPCDE(1,161)),
                 X            (J12,JOPCDE(1,177)),
                 X            (J13,JOPCDE(1,193)),
                 X            (J14,JOPCDE(1,209)),
                 X            (J15,JOPCDE(1,225)),
                 X            (J16,JOPCDE(1,241))
ISN 0079          DATA  J1/192HL X H      L X       L X A      S T H      S T X
                 A      S T X A    L X C      L X C A    L A H      L A
                 B L A A      S T A H    S T A      S T A A    L D H      L D
                 C     /
ISN 0080          DATA  J2/192HS T D H    S T D      L A T H    L A T      S T A
                 A T H   S T A T    L L       L R       S T L      S T R
                 B L M X      S T M X    L M A      S T M A    L M S      S T M
                 C S   /
ISN 0081          DATA  J3/192HS T M Z    S T M Z A
                 A      M X A      M A X      M K L      M K R      M L X
                 B M X S      M S X      M S X Z    M X S O    M X C      M C X
                 C     /
ISN 0082          DATA  J4/192HM L C      M R C      M X P      M K P
                 A                        A N        A D N      A R
                 B A D R      A U        A D U      S N        S D N      S R
                 C     /
ISN 0083          DATA  J5/192HS D R      S U        S D U      M N        M D N
                 A      M R        M D R      M U        M D U      M M N
                 B M M U      D N        D D N      D R        D D R      D M N
```

257

L. Conway
Archives

```
                C       /
ISN 0084         DATA  J6/192HD M R       R N D       S P F       S N F       C V S
                A       C V F                                 A I         S I
                B M I         M M I      D I         D M I       A C L       S C L
                C       /
ISN 0085         DATA  J7/192HA C H       S C H       S P I       S N I       C V N
                A       C V I                                             A X
                B S X         M X        D R X       D X         R X         A X C
                C       /
ISN 0086         DATA  J8/192HA X K       M X K       D R X K     D X K       R X K
                A       S P X       S N X
                B C G E N     C E Q N    C G E D     C E Q D     C M G E N   C M E
                C Q N   /
ISN 0087         DATA  J9/192HC M G E D   C M E Q D   C G E I     C E Q I     C U G
                A E I   C G E X     C E Q X     C U G E X   C G E X K   C E Q X K
                B C U G E X K C B A      C B M A     C B X       C B M X
                C       /
ISN 0088         DATA J10/192H                        S H A       S H X       S H A
                A C     S H X C     S H D       S H D X     S H D C     S H D X C
                B S W A       S W X      I F A       I F X       I F Z A     I F Z
                C X     /
ISN 0089         DATA J11/192HS I A       S I X       S I A C     S I X C     S I D
                A       S I D C                                           A N D A
                B T A F A     F A F A    O R A       T O F A     F O F A     E Q A
                C       /
ISN 0090         DATA J12/192HX O R A     A N D X     T A F X     F A F X     O R X
                A       T O F X     F O F X     E Q X       X O R X     A N D C
                B T A F C     F A F C    O R C       T O F C     F O F C     E Q C
                C       /
ISN 0091         DATA J13/192HX O R C     C N T T     C N T A A   C N T D A   C N T
                A A X   C N T D X                                         B A N D
                B B T A F     B F A F    B O R       B T O F     B F O F     B E Q
                C       /
ISN 0092         DATA J14/192HB X O R     B U         E X I T     E X I T L   E X I
                A T A   E X I T P   S K A N D   S K T A F   S K F A F   S K O R
                B S K T O F   S K F O F   S K E Q     S K X O R   I V I B     N O P
                C       /
ISN 0093         DATA J15/192HP A U S E   P I         S C A N     S V C       S V R
                A       I C         I R
                B S I O       H I O      T C H       M T X       M X T       M Z T
                C       /
ISN 0094         DATA J16/192HM O T       I T U M A   I T U M P   I D A       L D A
                A       L D H A A   L D H B A   L D H C A   L D H D A   S T D H A A
                B S T D H B A S T D H C A S T D H D A                     S T O
                C P     /
ISN 0095         DATA   JNB/  'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
                *   'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V',
                *   'W', 'X', 'Y', 'Z', '0', '1', '2', '3', '4', '5', '6', '7',
                *   '8', '9'/
12  ISN 0096     DATA JSIDB/1,2,2,1,4*2,1,2,2,1,2,2,1,2,1,2,1,2,1,11*2,
```

258

L. Conway
Archives

```
                     2          2,2,0,0,0,1,1,2,2,10*1,2,0,0,0,9*1,
                     3          22*1,0,0,8*1,
                     4          6*1,0,0,0,7*1,5*2,1,1,0,0,0,6*1,
                     5          8*1,3*2,4*1,0,0,0,14*1,
                     6          6*1,0,0,0,23*1,
                     7          6*1,0,0,0,8*2,4*1,2,8*1,2,1,
                     8          1,1,2,1,2,1,1,0,0,0,2,18*1,0,0,1/
ISN 0097              DATA JITYPE/201*4,9*1,4*3,8*2,33*4,12/
ISN 0098              COMMON /AREA2/  JNB,JOPCDE,JSIDB,JITYPE,JEXITF
ISN 0099              END
```


L. Conway
Archives

```
LEVEL 2 FEB 67                          OS/360  FORTRAN H                          DATE  67.255/11.33.51

          COMPILER OPTIONS - NAME=  MAIN,OPT=00,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

      ISN 0002            SUBROUTINE INIT
      ISN 0003            IMPLICIT INTEGER*2(A-Z)
      ISN 0004            DIMENSION COM(300)
      ISN 0005            DIMENSION SAV(20000)
      ISN 0006            COMMON       TIME,       IPAR1,      IPAR2,      IPAR3,
                         A AINPT,      NABUF,      ABUS(50),   XINPT,      NXBUF,
                         B XBUS(50),   IFADD,      IFDST,      IFRTN,      BRXP,
                         C BRAP,       ER(8),      BE(8),      ET(8),      NBBUF,
                         D AHOLDT,     XHOLDT,     AERCT,      XERCT,      BOSC,
                         E BNOP,       XEP,        AEP,        PH1(100),   PRINT,
                         F ESTADD,     NODOT,      NOPSC,      NDBUS,      NADSP,
                         G NXDSP
      ISN 0007            COMMON/RLS/  FIRST,      NAREGS,     NXREGS,     NABUS,
                         A NXBUS,      STATS,      ACON,       XCON,       AEMP,
                         B XEMP,       MXO,        AFULL(12),  XFULL(12),  AGO(12),
                         C XGO(12),    NAGO,       NXGO,       NATEST,     NXTEST,
                         D NAFAC,      NXFAC,      ABUSYZ,     ABUSY(200), XBUSYZ,
                         E XBUSY(200), ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
                         F XSOR(12,200),ADEST(12,200),XDEST(12,200),       AFAC(12,15),
                         G XFAC(12,15), AFACSC(4,15,20),ARET,      XFACSC(4,15,20),XRET,
                         H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
                         I AOBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
                         J AFDLY(15),  XFDLY(15),  AFOBUS(15), XFOBUS(15), NSLOT,
                         K ABUPSZ,     ABUPS(200), XBUPS(200), ABUFUL(200),XBUFUL(200),
                         L Q(16,16),   SDBA(32,2), NQBUF,      NQTEST,     NQGO,
                         M QINPT,      QCON,       QEMP,       MBUSY,      MFREE,
                         N LOAD,       MEMDLY,     MEMORY(16), NBOX,       EAV,
                         O MXTIME,     OUTLVL,     IQ(4,16),   RTN,        LONGBR,
                         P SR(8),      ST(8),      SKXP,       SKAP,       NSBUF,
                         Q APASS(200), XPASS(200), OUT(2),     JOB(6),     SSTOP,
                         R MEMCNT(16), ABOX(15),   ABXBSY(10), XBOX(15),   XBXBSY(10)
      ISN 0008            COMMON/RLS/  LAST
      ISN 0009            INTEGER OUT
      ISN 0010            REAL MEMDLY,MXTIME
      ISN 0011            REAL TIME
      ISN 0012            COMMON /CALNDR/             ISL,        ITL,        LINK(200),
                         A CTIME(200),  NEVENT(200), KOL1(200),  KOL2(200),  KOL3(200)
      ISN 0013            REAL CTIME
      ISN 0014            REAL X
      ISN 0015            INTEGER I
      ISN 0016            COMMON/TAGS/D(256,70)
      ISN 0017            EQUIVALENCE(COM(1),TIME),(X,CTIME(1))
      ISN 0018            EQUIVALENCE(SAV(1),FIRST)
                     C
                     C
                     C                ZERO ALL COMMON
12    ISN 0019             DO 520 I=1,300
11    ISN 0020         520 COM(I)=0
10    ISN 0021             DO 525 I=1,20000
9
8                260
7
6     L. Conway
5     Archives
4
3
2
```

```
ISN 0022        525 SAV(I)=0
ISN 0023        526 CONTINUE
                C
                C
                C               INITIALIZE THE CALENDAR
ISN 0024            DO 92 ITL=2,199
ISN 0025         92 LINK(ITL)=ITL+1
ISN 0026            ISL=2
ISN 0027            ITL=1
ISN 0028            X=1.0E30
ISN 0029            TIME=0.0
                C
                C
                C               INITIALIZE THE EVENT NUMBERS
ISN 0030            STATS=1
ISN 0031            MXO=2
ISN 0032            ACON=3
ISN 0033            XCON=4
ISN 0034            AEMP=5
ISN 0035            XEMP=6
ISN 0036            ARET=7
ISN 0037            XRET=8
ISN 0038            EAV=9
ISN 0039            QCON=10
ISN 0040            QEMP=11
ISN 0041            MBUSY=12
ISN 0042            MFREE=13
ISN 0043            LOAD=14
ISN 0044            RTN=15
                C
                C
                C               SET UP STARTING EVENTS
ISN 0045            CALL CAUSE(STATS,TIME+0.0,0,0,0)
ISN 0046            CALL CAUSE( ACON,TIME+0.1,0,0,0)
ISN 0047            CALL CAUSE( XCON,TIME+0.1,0,0,0)
ISN 0048            CALL CAUSE( QCON,TIME+0.1,0,0,0)
ISN 0049            CALL CAUSE( MXO ,TIME+0.6,0,0,0)
                C
                C
                C               INITIALIZE THE MACHINE PARAMETERS
ISN 0050            BRXP=1
ISN 0051            BRAP=1
ISN 0052            SKXP=1
ISN 0053            SKAP=1
ISN 0054            NAREGS=90
ISN 0055            NXREGS=90
ISN 0056            AINPT=1
ISN 0057            QINPT=1
ISN 0058            XINPT=1
ISN 0059            DO 50 I=1,32
```


L. Conway
Archives

```
ISN 0060          ABUPS(I)=1
ISN 0061       50 XBUPS(I)=0
ISN 0062          DO 51 I=33,89
ISN 0063          ABUPS(I)=0
ISN 0064       51 XBUPS(I)=1
ISN 0065          NSLOT=15
                C          INITIALIZE AFAC TABLES
ISN 0066          NABUS=6
ISN 0067          NAFAC=10
ISN 0068          DO 10 I=1,10
ISN 0069       10 AFSLOT(I,3)=1
ISN 0070          DO 9 J=4,9
ISN 0071        9 AFSLOT(4,J)=1
ISN 0072          AFSLOT(6,4)=1
ISN 0073          DO 8 J=4,12
ISN 0074        8 AFSLOT(7,J)=1
ISN 0075          AFDLY(1)=3
ISN 0076          AFDLY(2)=4
ISN 0077          AFDLY(3)=3
ISN 0078          AFDLY(4)=9
ISN 0079          AFDLY(5)=2
ISN 0080          AFDLY(6)=5
ISN 0081          AFDLY(7)=15
ISN 0082          AFDLY(8)=1
ISN 0083          AFDLY(9)=1
ISN 0084           AFDLY(10)=1
ISN 0085          AFIBUS(1)=2
ISN 0086          AFIBUS(2)=1
ISN 0087          AFIBUS(3)=3
ISN 0088          AFIBUS(4)=1
ISN 0089          AFIBUS(5)=1
ISN 0090          AFIBUS(6)=2
ISN 0091          AFIBUS(7)=2
ISN 0092          AFIBUS(8)=1
ISN 0093          AFIBUS(9)=2
ISN 0094          AFIBUS(10)=3
ISN 0095          AFOBUS(1)=2
ISN 0096          AFOBUS(2)=1
ISN 0097          AFOBUS(3)=4
ISN 0098          AFOBUS(4)=3
ISN 0099          AFOBUS(5)=2
ISN 0100          AFOBUS(6)=4
ISN 0101          AFOBUS(7)=4
ISN 0102          AFOBUS(8)=6
ISN 0103          AFOBUS(9)=1
ISN 0104          AFOBUS(10)=3
ISN 0105          ABOX(1)=1
ISN 0106          ABOX(2)=2
ISN 0107          ABOX(3)=3
ISN 0108          ABOX(4)=4
```


L. Conway
Archives

```
ISN 0109          ABOX(5)=2
ISN 0110          ABOX(6)=4
ISN 0111          ABOX(7)=4
ISN 0112          ABOX(8)=5
ISN 0113          ABOX(9)=6
ISN 0114          ABOX(10)=7
                C          INITIALIZE XFAC TABLES
ISN 0115          NXBUS=10
ISN 0116          NXFAC=9
ISN 0117          DO 11 I=1,9
ISN 0118       11 XFSLOT(I,2)=1
ISN 0119          XFSLOT(5,3)=1
ISN 0120          DO 12 I=3,9
ISN 0121       12 XFSLOT(6,I)=1
ISN 0122          XFDLY(1)=1
ISN 0123          XFDLY(2)=1
ISN 0124          XFDLY(3)=1
ISN 0125          XFDLY(4)=1
ISN 0126          XFDLY(5)=4
ISN 0127          XFDLY(6)=8
ISN 0128          XFDLY(7)=1
ISN 0129          XFDLY(8)=1
ISN 0130          XFDLY(9)=1
ISN 0131          XFOBUS(1)=5
ISN 0132          XFOBUS(2)=6
ISN 0133          XFOBUS(3)=1
ISN 0134          XFOBUS(4)=3
ISN 0135          XFOBUS(5)=2
ISN 0136          XFOBUS(6)=2
ISN 0137          XFOBUS(7)=7
ISN 0138          XFOBUS(8)=10
ISN 0139          XFOBUS(9)=8
ISN 0140          XBOX(1)=1
ISN 0141          XBOX(2)=2
ISN 0142          XBOX(3)=3
ISN 0143          XBOX(4)=4
ISN 0144          XBOX(5)=5
ISN 0145          XBOX(6)=5
ISN 0146          XBOX(7)=6
ISN 0147          XBOX(8)=7
ISN 0148          XBOX(9)=8
ISN 0149          NAFAC=11
ISN 0150          NABUS=7
ISN 0151          AFDLY(11)=1
ISN 0152          AFIBUS(11)=1
ISN 0153          AFOBUS(11)=7
ISN 0154          ABOX(11)=8
ISN 0155          AFSLOT(11,3)=1
ISN 0156          D(39,1)=1
ISN 0157          D(39,2)=1
```

Temp. fix to allow MAX of

L. Conway Archives

```
ISN 0158        D(39,11)=1
ISN 0159        D(39,13)=1
ISN 0160        D(39,17)=1
ISN 0161        D(39,30)=0
ISN 0162        D(39,32)=1
ISN 0163        D(39,66)=1
ISN 0164        RETURN
ISN 0165        END
```


L. Conway Archives

```
LEVEL 5 DEC 66                    DATE  YS/360  FORTRAN H                    DATE  67.191/03.47.48

          COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

ISN 0002          SUBROUTINE XACON
ISN 0003           IMPLICIT INTEGER*2(A-Z)
ISN 0004           COMMON       TIME,       IPAR1,      IPAR2,      IPAR3,
                  A AINPT,      NABUF,      ABUS(50),   XINPT,      NXBUF,
                  B XBUS(50),   IFADD,      IFDST,      IFRTN,      BRXP,
                  C BRAP,       ER(8),      BE(8),      ET(8),      NBBUF,
                  D AHOLDT,     XHOLDT,     AERCT,      XERCT,      BOSC,
                  E BNOP,       XEP,        AEP,        PH1(100),   PRINT,
                  F ESTADD,     NOOUT,      NOPSC,      NOBUS,      NADSP,
                  G NXOSP
ISN 0005           COMMON/RLS/  FIRST,      NAREGS,     NXREGS,     NABUS,
                  A NXBUS,      STATS,      ACON,       XCON,       AEMP,
                  B XEMP,       MXC,        AFULL(12),  XFULL(12),  AGO(12),
                  C XGO(12),    NAGO,       NXGO,       NATEST,     NXTEST,
                  D NAFAC,      NXFAC,      ABUSYZ,     ABUSY(200), XBUSYZ,
                  E XBUSY(200), ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
                  F XSOR(12,200),ADEST(12,200),XDEST(12,200),       AFAC(12,15),
                  G XFAC(12,15), AFACSC(4,15,20),ARET,      XFACSC(4,15,20),XRET,
                  H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
                  I AOBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
                  J AFDLY(15),  XFDLY(15),  AFOBUS(15),  XFOBUS(15),  NSLOT,
                  K ABUPSZ,     AFUPS(200), XBUPS(200), ABUFUL(200), XBUFUL(200),
                  L Q(16,16),   SDBA(32,2),  NQBUF,      NQTEST,     NQGO,
                  M QINPT,      QCON,       QEMP,       MBUSY,      MFREE,
                  N LOAD,       MEMDLY,     MEMORY(16), NBOX,       EAV,
                  O SXTIME,     OUTLVL,     IQ(4,16),   RTN,        LONGBR,
                  P SR(8),      ST(8),      SKXP,       SKAP,       NSBUF,
                  Q APASS(200), XPASS(200), OUT(2),     JOB(6),     SSTOP,
                  R BEMCNT(16), ABOX(15),   AoXBSY(10), XBOX(15),   XBXBSY(10)
ISN 0006           COMMON/RLS/  LAST
ISN 0007           INTEGER OUT
ISN 0008           DIMENSION SOBSY(200),DESBSY(200)
ISN 0009           REAL MEMDLY,MXTIME
ISN 0010           REAL TIME
ISN 0011           CALL CAUSE(ACON,TIME+1.0,0,0,0)
ISN 0012           CALL CAUSE(AEMP,TIME+0.7,0,0,0)
ISN 0013           CALL CAUSE(ARET,TIME+0.8,0,0,0)
ISN 0014           NOUT=0
ISN 0015           STOPT=0
ISN 0016           DO  1 I=1,NABUF
ISN 0017         1 AGO(I)=0
                 C- - - - - - - - IF SKIP TAKEN,NOP ALL STARRED UPS UP TO 1ST EXEC SKIP - -
ISN 0018           DO 80 I=1,NABUF
ISN 0019           IF(ABUFF(I,13).EQ.0) GO TO 79
ISN 0021           IF(ABUFF(I, 9).EQ.0) GO TO 84
ISN 0023           IF(ABUFF(I,11).EQ.0) GO TO 84
ISN 0025        79 CONTINUE
ISN 0026           IF(ABUFF(I,11).EQ.0) GO TO 80
ISN 0028           IF(ABUFF(I,9).EQ.0) GO TO 80
```

265

L. Conway
Archives

```
ISN 0030            DO 81 K=1,NAREGS
ISN 0031            ASOR(I,K)=0
ISN 0032         81 ADEST(I,K)=0
ISN 0033            DO 82 K=1,NAFAC
ISN 0034            AFAC(I,K)=0
ISN 0035         82 ADBUS(I,K)=0
ISN 0036            ABUFF(I,2)=0
ISN 0037            DO 83 K=9,15
ISN 0038         83 ABUFF(I,K)=0
ISN 0039         80 CONTINUE
ISN 0040         84 CONTINUE
                 C- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                 C                  THIS EVENT SCANS ABUFF FOR INST WHICH CAN GO
                 C                  SCAN FOR NAGO OUT OF NATEST
ISN 0041            DO 10 REG=1,NAREGS
ISN 0042            SORBSY(REG)=0
ISN 0043         10 DESBSY(REG)=ABUSY(REG)
ISN 0044            DO 100 INS=1,NATEST
ISN 0045            IF(AFULL(INS).EQ.0) GO TO 100
ISN 0047            IF(INS.EQ.1) GO TO 21
ISN 0049            DO 11 REG=1,NAREGS
ISN 0050            SORBSY(REG)=SORBSY(REG)+ASOR(INS-1,REG)
ISN 0051         11 DESBSY(REG)=DESBSY(REG)+ADEST(INS-1,REG)
ISN 0052            INSM1=INS-1
ISN 0053            DO 20 I=1,INSM1
                 C                  PREV EXIT INTLKS ALL CODE BELOW
ISN 0054            IF(ABUFF(I,14).EQ.1) GO TO 100
                 C                  PREV SKIP INTLKS ALL STARRED CODE BELOW
                 C                  AND ALL SKIPS BELOW
ISN 0056            IF(ABUFF(I,13).EQ.0) GO TO 20
ISN 0058            IF((ABUFF(INS,13).EQ.1).OR.(ABUFF(INS,9).EQ.1)) GO TO 100
ISN 0060         20 CONTINUE
ISN 0061         21 CONTINUE
                 C                  IF EXIT,INTLK AGAINST ER
ISN 0062            IF(ABUFF(INS,14).NE.1) GO TO 28
ISN 0064            IF(ER(BRAP).NE.1) GO TO 100
                 C                  EXIT PART OF OP GOES, MARK GO EXIT.
ISN 0066            ABUFF(INS,15)=1
ISN 0067         28 CONTINUE
ISN 0068            DO 22 REG=1,NAREGS
ISN 0069            IF((ASOR(INS,REG).EQ.1).AND.(DESBSY(REG).NE.0)) GO TO 100
ISN 0071            IF((ADEST(INS,REG).EQ.1).AND.(SORBSY(REG).NE.0)) GO TO 100
ISN 0073            IF(REG.EQ.89) GO TO 22
ISN 0075            IF((ADEST(INS,REG).EQ.1).AND.(DESBSY(REG).NE.0)) GO TO 100
ISN 0077         22 CONTINUE
                 C                  FIND FAC USED
ISN 0078            DO 25 FAC=1,NAFAC
ISN 0079            IF(AFAC(INS,FAC).NE.0) GO TO 26
ISN 0081         25 CONTINUE
                 C                  NO FAC USED. ISSUE OP
```


L. Conway
Archives

```
ISN 0082            FAC=0
ISN 0083         26 CONTINUE
                 C - - - - - - - - TEST FOR SPECIAL OPS HERE - - - - - - - - - - - - - - -
ISN 0084            SPEC=0
                 C                  IF STORE A, TEST AVAIL OF INBUS (STORE BUS)
                 C                  IF AVAIL, SET BUSY. IF NOT, GO TO 100
ISN 0085            IF(ADEST(INS,89).NE.1) GO TO 27
ISN 0087            SPEC=1
                 C                  PREV NOGO STORE INTLKS
ISN 0088            IF(INS.EQ.1) GO TO 18
ISN 0090            DO 16 I=1,INSM1
ISN 0091            IF((ADEST(I,89).EQ.1).AND.(AGO(I).EQ.0)) GO TO 100
ISN 0093         16 CONTINUE
ISN 0094         18 CONTINUE
ISN 0095            IF(AIBBSY(3).EQ.1) GO TO 17
ISN 0097            AIBBSY(3)=1
ISN 0098            STORE=1
ISN 0099            GO TO 27
ISN 0100         17 IF(STORE.NE.1) GO TO 100
ISN 0102            STORE=2
ISN 0103         27 CONTINUE
                 C                  IF SKIP,INTLK AGAINST PREV NOGO STARRED OPS,SHT RESOLVED.
ISN 0104            IF(ABUFF(INS,13).NE.1) GO TO 132
ISN 0106            IF(SK(SKAP).NE.1) GO TO 100
ISN 0108            IF(INS.EQ.1) GO TO 131
ISN 0110            DO 130 I=1,INSM1
ISN 0111            IF((ABUFF(I,9).EQ.1).AND.(AGO(I).NE.1)) GO TO 100
ISN 0113        130 CONTINUE
ISN 0114        131 SPEC=1
ISN 0115        132 CONTINUE
                 C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                 C                  IF NORMAL OR SPEC OP AND NGO =NAGO, DO NOT ISSUE
                 C                  IF REPLACE OR NOP, CAN ISSUE ANYWAY.
ISN 0116            IF(((FAC.NE.0).OR.(SPEC.NE.0)).AND.(NGO.EQ.NAGO)) GO TO 100
                 C                  IF NO FACS USED GO DIRECTLY TO 95
ISN 0118            IF(FAC.EQ.0) GO TO 95
                 C                  IF MULT IDENT FAC, GO TO SPEC HANDLING
ISN 0120            IF(AFAC(INS,FAC).GT.1) GO TO 49
                 C                  CHECK INBUS,FAC SLOT,OUTBUS INTLKS
ISN 0122            INBUS=AFIBUS(FAC)
ISN 0123            IF(AIBBSY(INBUS).EQ.1) GO TO 100
ISN 0125            BOX=ABOX(FAC)
ISN 0126            IF(ABXBSY(BOX).EQ.1) GO TO 100
ISN 0128            DO 30 T=1,NSLOT
ISN 0129            IF((AFSLOT(FAC,T).EQ.1).AND.(AFACSC(1,FAC,T).EQ.1)) GO TO 100
ISN 0131         30 CONTINUE
ISN 0132            OBUS=AFOBUS(FAC)
ISN 0133            DELAY=AFDLY(FAC)
ISN 0134            IF((AOBUS(INS,OBUS).NE.0).AND.(ABUSSC(1,OBUS,DELAY).NE.0))
                   X GO TO 100
```


L. Conway
Archives

```
ISN 0136            IF((AOBUS(INS,OBUS+1).NE.0).AND.(ABUSSC(1,OBUS+1,DELAY).NE.0).AND.
                   X ((OBUS+1).LE.NABUS)) GO TO 100
                C                SUCCESS.  MARK GO AND SET SHIFT CELLS
ISN 0138         31 CONTINUE
ISN 0139            AIBBSY(INBUS)=1
ISN 0140            ABXBSY(BOX)=1
ISN 0141            ABUFF1=ABUFF(INS,1)
ISN 0142            DO 32 T=1,NSLOT
ISN 0143            IF(AFSLOT(FAC,T).EQ.0) GO TO 32
ISN 0145            AFACSC(1,FAC,T)=1
ISN 0146            AFACSC(2,FAC,T)=ABUFF1
ISN 0147         32 CONTINUE
ISN 0148            ABUSSC(1,OBUS,DELAY)=AOBUS(INS,OBUS)
ISN 0149            ABUSSC(2,OBUS,DELAY)=ABUFF(INS,1)
ISN 0150            ABUSSC(3,OBUS,DELAY)=ABUFF(INS,2)
ISN 0151            IF(AOBUS(INS,OBUS+1).EQ.0) GO TO 95
ISN 0153            IF((OBUS+1).GT.NABUS) GO TO 95
ISN 0155            ABUSSC(1,OBUS+1,DELAY)=AOBUS(INS,OBUS+1)
ISN 0156            ABUSSC(2,OBUS+1,DELAY)=ABUFF(INS,1)
ISN 0157            ABUSSC(3,OBUS+1,DELAY)=ABUFF(INS,2)
ISN 0158            GO TO 95
                C                SPEC ROUTINE TO HANDLE MULT IDENT FAC INTLK
ISN 0159         49 CONTINUE
ISN 0160            INBUS=AFIBUS(FAC)
ISN 0161            IF(AIBBSY(INBUS).EQ.1) GO TO 60
ISN 0163            BOX=ABOX(FAC)
ISN 0164            IF(ABXBSY(BOX).EQ.1) GO TO 60
ISN 0166            DO 50 T=1,NSLOT
ISN 0167            IF((AFSLOT(FAC,T).EQ.1).AND.(AFACSC(1,FAC,T).EQ.1)) GO TO 60
ISN 0169         50 CONTINUE
ISN 0170            OBUS=AFOBUS(FAC)
ISN 0171            DELAY=AFDLY(FAC)
ISN 0172            IF((AOBUS(INS,OBUS).NE.0).AND.(ABUSSC(1,OBUS,DELAY).NE.0))
                   X GO TO 60
                C                SUCCESS
ISN 0174            DO 51 BUS=1,NABUS
ISN 0175         51 IF(BUS.NE.OBUS) AOBUS(INS,BUS)=0
ISN 0177            GO TO 31
ISN 0178         60 CONTINUE
ISN 0179            FAC=FAC+1
ISN 0180            IF((FAC.GT.NAFAC).OR.(AFAC(INS,FAC).LE.1)) GO TO 100
ISN 0182            GO TO 49
ISN 0183         95 CONTINUE
ISN 0184            AGO(INS)=1
                C                IF OP USES NO FACILITIES, AND IS NOT SPECIAL OP THEN IT
                C                IS A REPLACE OP, AND GOES WITHOUT INCREMENTING NGO.
ISN 0185            IF((FAC.NE.0).OR.(SPEC.NE.0)) NGO=NGO+1
ISN 0187        100 CONTINUE
                C - - - - - - - - - - - - - EXIT EXECUTION - - - - - - - - - - - - - - - - - -
                C                CHECK FOR NOGO EXITS TO SET AHOLDT
```


L. Conway
Archives

```
ISN 0188            AHOLDT=0
ISN 0189            DO 200 I=1,NABUF
ISN 0190            IF((ABUFF(I,14).EQ.1).AND.(ABUFF(I,15).NE.1)) AHOLDT=1
ISN 0192          200 CONTINUE
                  C                 CHECK FOR GO EXIT,ET
ISN 0193            AFRCT=0
ISN 0194            DO 201 I=1,NABUF
ISN 0195            IF(ABUFF(I,15).NE.1) GO TO 201
ISN 0197            IF(ABUFF(I,14).NE.1) GO TO 201
ISN 0199            ABUFF(I,14)=0
ISN 0200            ABUFF(I,15)=0
ISN 0201            IF(ABUFF(I,10).EQ.1) GO TO 202
ISN 0203          201 CONTINUE
ISN 0204            GO TO 300
                  C                 FOUND GO EXIT,ET. NOP AND MARK GO ALL CODE BENEATH IT.
                  C                 ALSO SET AFRCT.
ISN 0205          202 AFRCT=1
ISN 0206            IF(I.EQ.NABUF) GO TO 300
ISN 0208            I=I+1
ISN 0209            DO 203 J=I,NABUF
ISN 0210            AGO(J)=1
ISN 0211            DO 204 K=1,NAREGS
ISN 0212            ASGR(J,K)=0
ISN 0213          204 ADEST(J,K)=0
ISN 0214            DO 205 K=1,10
ISN 0215          205 ACBUS(J,K)=0
ISN 0216            ABUFF(I,2)=0
ISN 0217            DO 206 K=9,15
ISN 0218          206 ABUFF(J,K)=0
ISN 0219            DO 207 K=1,NAFAC
ISN 0220            AFAC(J,K)=0
ISN 0221          207 CONTINUE
ISN 0222          203 CONTINUE
ISN 0223          300 CONTINUE
                  C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                  C- - - - - - - - IF SKIP NOT TAKEN,REMOVE FLAGS FROM ALL OPS THRU 1ST SKIP
ISN 0224            DO 85 I=1,NABUF
ISN 0225            IF(ABUFF(I,11).EQ.0) ABUFF(I,9)=0
ISN 0227            IF(ABUFF(I,13).EQ.1) GO TO 86
ISN 0229          85 CONTINUE
ISN 0230          86 CONTINUE
                  C- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
ISN 0231            RETURN
ISN 0232            END
```

269

L. Conway
Archives

```
LEVEL 5 DEC 66                          DATE  YS/360  FORTRAN H                          DATE  67.191/03.49.45

          COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

ISN 0002            SUBROUTINE XAEMP
ISN 0003            IMPLICIT INTEGER*2(A-Z)
ISN 0004            COMMON       TIME,       IPAR1,       IPAR2,       IPAR3,
                   A AINPT,      NABUF,      ABUS(50),    XINPT,       NXBUF,
                   B XBUS(50),   IFADD,      IFDST,       IFRTN,       BRXP,
                   C BRAP,       ER(8),      BE(8),       ET(8),       NBBUF,
                   D AHOLDT,     XHOLDT,     AFRCT,       XFRCT,       BOSC,
                   E BNOP,       XEP,        AEP,         PH1(100),    PRINT,
                   F FSTADD,     NODOT,      NOPSC,       NDBUS,       NADSP,
                   G NXDSP
ISN 0005            COMMON/RLS/  FIRST,      NAREGS,      NXREGS,      NABUS,
                   A NXBUS,      STATS,      ACON,        XCON,        AEMP,
                   B XEMP,       MXQ,        AFULL(12),   XFULL(12),   AGO(12),
                   C XGO(12),    NAGO,       NXGO,        NATEST,      NXTEST,
                   D NAFAC,      NXFAC,      ABUSYZ,      ABUSY(200),  XBUSYZ,
                   E XBUSY(200), ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
                   F XSOR(12,200),ADEST(12,200),XDEST(12,200),          AFAC(12,15),
                   G XFAC(12,15), AFACSC(4,15,20),ARET,      XFACSC(4,15,20),XRET,
                   H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
                   I AOBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
                   J AFDLY(15),  XFDLY(15),  AFOBUS(15),  XFOBUS(15),  NSLOT,
                   K ABUPSZ,     ABUPS(200), XBUPS(200),  ABUFUL(200), XBUFUL(200),
                   L Q(16,16),   SDBA(32,2), NQBUF,       NQTEST,      NQGO,
                   M QINPT,      QCON,       QEMP,        MBUSY,       MFREE,
                   N LOAD,       MEMDLY,     MEMORY(16),  NBOX,        EAV,
                   O MXTIME,     OUTLVL,     IQ(4,16),    RTN,         LONGBR,
                   P SR(8),      ST(8),      SKXP,        SKAP,        NSBUF,
                   Q APASS(200), XPASS(200), OUT(2),      JOB(6),      SSTOP,
                   R MEMCNT(16), ABOX(15),   ABXBSY(10),  XBOX(15),    XBXBSY(10)
ISN 0006            COMMON/RLS/  LAST
ISN 0007            INTEGER OUT
ISN 0008            COMMON/TAGS/D(256,70)
ISN 0009            REAL MEMDLY,MXTIME
ISN 0010            REAL TIME
ISN 0011            DO 100 INS=1,NABUF
ISN 0012          5 IF(AGO(INS).EQ.0) GO TO 100
ISN 0014            IF(AFULL(INS).EQ.0) GO TO 100
                   C                ISSUE INS
                   C - - - - - - - - TEST FOR SPECIAL OPS HERE - - - - - - - - - - - - - - -
                   C                 IF STORE A, SHIP DATA TO BUFFER OR Q DEP ON BUFFER STATE
ISN 0016            IF(ADEST(INS,89).NE.1) GO TO 7
ISN 0018            IF(SDBA(1,2).EQ.1) GO TO 2
                   C                NO STA WAITING. SET DATA IN SDBA
ISN 0020            DO 3 I=1,32
ISN 0021            IF(SDBA(I,1).EQ.1) GO TO 3
ISN 0023            SDBA(I,1)=1
ISN 0024            GO TO 7
ISN 0025          3 CONTINUE
ISN 0026            A=1
```

270

L. Conway
Archives

```
ISN 0027          B=20000
ISN 0028          C=104
ISN 0029          CALL TROUBL(A,B,C)
ISN 0030          GO TO 7
                C          STA WAITING. DATA TO Q, SHIFT SDBA
ISN 0031        2 CONTINUE
ISN 0032          DO 4 I=1,31
ISN 0033          SDBA(I,1)=SDBA(I+1,1)
ISN 0034        4 SDBA(I,2)=SDBA(I+1,2)
ISN 0035          SDBA(32,1)=0
ISN 0036          SDBA(32,2)=0
ISN 0037          DO 50 II=1,NOBUF
ISN 0038          IF(Q(II,3).NE.1) GO TO 50
ISN 0040          IF(Q(II,4).NE.1) GO TO 50
ISN 0042          IF(Q(II,9).EQ.1) GO TO 50
ISN 0044          Q(II,9)=1
ISN 0045          GO TO 7
ISN 0046       50 CONTINUE
ISN 0047          A=1
ISN 0048          B=20000
ISN 0049          C=103
ISN 0050          CALL TROUBL(A,B,C)
ISN 0051        7 CONTINUE
                C          ISSUE SKIP-RESET SR, INCR SKIP POINTER
ISN 0052          IF(ABUFF(INS,13).NE.1) GO TO 60
ISN 0054          ST(SKAP)=0
ISN 0055          SR(SKAP)=0
ISN 0056          SKAP=SKAP+1
ISN 0057          IF(SKAP.GT.NSBUF) SKAP=1
ISN 0059       60 CONTINUE
                C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
ISN 0060          OP = ABUFF(INS,2)
ISN 0061          REPL=D(OP,33)
                C          FIRST SET BUSY VECTOR
ISN 0062          DO 10 REG=1,NAREGS
                C          IS REG A DEST
ISN 0063          IF(ADEST(INS,REG).NE.1) GO TO 10
                C          IGNORE STORAGE AS DEST
ISN 0065          IF(REG.EQ.89) GO TO 10
                C          DOES AREG HAVE AN ABUREG
ISN 0067          IF(ABUPS(REG).NE.1) GO TO 9
                C          CAN THIS OP DO BACK-UP TO FRONT MOVE.
                C          I.E.,IS THIS A REPLACE OR A TO X MOVE OP.
ISN 0069          IF(REPL.EQ.0) GO TO 9
                C          IS ABUREG FULL
ISN 0071          IF(ABUFUL(REG).NE.1) GO TO 99
ISN 0073          ABUFUL(REG)=0
ISN 0074          XBUSY(REG)=0
ISN 0075          GO TO 10
ISN 0076       99 APASS(REG)=ABUFF(INS,1)
```


L. Conway
Archives

```
ISN 0077          9 ABUSY(REG)=ABUFF(INS,1)
ISN 0078         10 CONTINUE
                 C                REMOVE INS FROM BUFF
ISN 0079            AINPT=AINPT-1
ISN 0080            M=NABUF-1
ISN 0081            IF(INS.EQ.NABUF) GO TO 31
ISN 0083            DO 30 I=INS,M
ISN 0084            AGO(I)=AGO(I+1)
ISN 0085            AFULL(I)=AFULL(I+1)
ISN 0086            DO 25 J=1,25
ISN 0087         25 ABUFF(I,J)=ABUFF(I+1,J)
ISN 0088            DO 26 J=1,NAREGS
ISN 0089            ASOR(I,J)=ASOR(I+1,J)
ISN 0090         26 ADEST(I,J)=ADEST(I+1,J)
ISN 0091            DO 27 FAC=1,NAFAC
ISN 0092         27 AFAC(I,FAC)=AFAC(I+1,FAC)
ISN 0093            DO 28 BUS=1,NABUS
ISN 0094         28 AOBUS(I,BUS)=AOBUS(I+1,BUS)
ISN 0095         30 CONTINUE
ISN 0096         31 CONTINUE
ISN 0097            AGO(NABUF)=0
ISN 0098            AFULL(NABUF)=0
ISN 0099            DO 125 J=1,25
ISN 0100        125 ABUFF(NABUF,J)=0
ISN 0101            DO 126 J=1,NAREGS
ISN 0102            ASOR(NABUF,J)=0
ISN 0103        126 ADEST(NABUF,J)=0
ISN 0104            DO 127 FAC=1,NAFAC
ISN 0105        127 AFAC(NABUF,FAC)=0
ISN 0106            DO 128 BUS=1,NABUS
ISN 0107        128 AOBUS(NABUF,BUS)=0
ISN 0108            GO TO 5
ISN 0109        100 CONTINUE
ISN 0110            RETURN
ISN 0111            END
```


L. Conway
Archives

```
LEVEL 5 DEC 69                    DATE = YS/360  FORTRAN H                    DATE = 07/[illegible]/[illegible]

        COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

ISN 0002            SUBROUTINE XARET
ISN 0003            IMPLICIT INTEGER*2(A-Z)
ISN 0004            COMMON       TIME,       IPAR1,       IPAR2,       IPAR3,
                   A AINPT,      NABUF,      ABUS(50),    XINPT,       NXBUF,
                   B XBUS(50),   IFADD,      IFDST,       IFRTN,       BRXP,
                   C BRAP,       ER(8),      BE(8),       ET(8),       NBBUF,
                   D AHOLDT,     XHOLDT,     AFRCT,       XFRCT,       BOSC,
                   E BNOP,       XEP,        AEP,         PH1(100),    PRINT,
                   F FSTADD,     NODOT,      NOPSC,       NDBUS,       NADSP,
                   G NXDSP
ISN 0005            COMMON/RLS/  FIRST,      NAREGS,      NXREGS,      NABUS,
                   A NXBUS,      STATS,      ACON,        XCON,        AEMP,
                   B XEMP,       MXO,        AFULL(12),   XFULL(12),   AGO(12),
                   C XGO(12),    NAGO,       NXGO,        NATEST,      NXTEST,
                   D NAFAC,      NXFAC,      ABUSYZ,      ABUSY(200),  XBUSYZ,
                   E XBUSY(200), ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
                   F XSOR(12,200),ADEST(12,200),XDEST(12,200),          AFAC(12,15),
                   G XFAC(12,15), AFACSC(4,15,20),ARET,      XFACSC(4,15,20),XRET,
                   H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
                   I AOBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
                   J AFDLY(15),  XFDLY(15),  AFOBUS(15),  XFOBUS(15),  NSLOT,
                   K ABUPSZ,     ABUPS(200), XBUPS(200),  ABUFUL(200), XBUFUL(200),
                   L Q(16,16),   SDBA(32,2), NQBUF,       NQTEST,      NQGO,
                   M QINPT,      QCON,       QEMP,        MBUSY,       MFREE,
                   N LOAD,       MEMDLY,     MEMORY(16),  NBOX,        EAV,
                   O MXTIME,     OUTLVL,     IQ(4,16),    RTN,         LONGBR,
                   P SR(8),      ST(8),      SKXP,        SKAP,        NSBUF,
                   Q APASS(200), XPASS(200), OUT(2),      JOB(6),      SSTOP,
                   R MEMCNT(16), ABOX(15),   ABXBSY(10),  XBOX(15),    XBXBSY(10)
ISN 0006            COMMON/RLS/  LAST
ISN 0007            INTEGER OUT
ISN 0008            REAL MEMDLY,MXTIME
ISN 0009            REAL TIME
                C               USES OBUS SHIFT CELLS TO BUS TO DESTINATIONS
                C               PERFORMS ANY EXEC ACTIVITY ASSOC WITH RETURN
                C               FIRST RESET THE WAITING VECTOR
ISN 0010            DO 10 BUS=1,NABUS
ISN 0011            DEST=ABUSSC(1,BUS,1)
                C               IF DEST NOT XBU GO HANDLE NORMALLY
ISN 0012            IF(XBUPS(DEST).NE.1) GO TO 9
                C               DEST IS XBU. SEE IF CORRESP XREG IS BUSY
                C               IF SO, RETURN DEST TO IT. ELSE SET XBU BUSY.
ISN 0014            IF(XPASS(DEST).NE.0) GO TO 8
ISN 0016            XBUFUL(DEST)=1
ISN 0017            GO TO 10
ISN 0018          8 XBUSY(DEST)=0
ISN 0019            XPASS(DEST)=0
ISN 0020          9 ABUSY(DEST)=0
ISN 0021         10 CONTINUE
```

273

L. Conway
Archives

```
C                PLACE ANY EXEC ACTIVITY HERE - - - - -
C                SHIFTS THE SHIFT CELLS
ISN 0022            DO 99 I=1,10
ISN 0023            ABXBSY(I)=0
ISN 0024         99 AIBBSY(I)=0
ISN 0025            SLOTM1=NSLOT-1
ISN 0026            DO 101 J=1,10
ISN 0027            DO 100 SLOT=1,SLOTM1
ISN 0028            ABUSSC(1,J,SLOT)=ABUSSC(1,J,SLOT+1)
ISN 0029            ABUSSC(2,J,SLOT)=ABUSSC(2,J,SLOT+1)
ISN 0030            ABUSSC(3,J,SLOT)=ABUSSC(3,J,SLOT+1)
ISN 0031        100 CONTINUE
ISN 0032            ABUSSC(1,J,NSLOT)=0
ISN 0033            ABUSSC(2,J,NSLOT)=0
ISN 0034            ABUSSC(3,J,NSLOT)=0
ISN 0035        101 CONTINUE
ISN 0036            DO 103 J=1,NAFAC
ISN 0037            DO 102 SLOT=1,SLOTM1
ISN 0038            AFACSC(1,J,SLOT)=AFACSC(1,J,SLOT+1)
ISN 0039            AFACSC(2,J,SLOT)=AFACSC(2,J,SLOT+1)
ISN 0040        102 CONTINUE
ISN 0041            AFACSC(1,J,NSLOT)=0
ISN 0042            AFACSC(2,J,NSLOT)=0
ISN 0043        103 CONTINUE
ISN 0044            RETURN
ISN 0045            END
```


L. Conway
Archives

```
        COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

ISN 0002            SUBROUTINE DECBUS
ISN 0003            IMPLICIT INTEGER*2(A-Z)
ISN 0004            COMMON          TIME,       IPAR1,      IPAR2,      IPAR3,
                   A AINPT,         NABUF,      ABUS(50),   XINPT,      NXBUF,
                   B XBUS(50),      IFADD,      IFDST,      IFRTN,      BRXP,
                   C BRAP,          ER(8),      BE(8),      ET(8),      NBBUF,
                   D AHOLDT,        XHOLDT,     AFRCT,      XFRCT,      BOSC,
                   E BNOP,          XEP,        AEP,        PH1(100),   PRINT,
                   F FSTADD,        NODOT,      NOPSC,      NDBUS,      NADSP,
                   G NXDSP
ISN 0005            COMMON/RLS/     FIRST,      NAREGS,     NXREGS,     NABUS,
                   A NXBUS,         STATS,      ACON,       XCON,       AEMP,
                   B XEMP,          MXO,        AFULL(12),  XFULL(12),  AGO(12),
                   C XGO(12),       NAGO,       NXGO,       NATEST,     NXTEST,
                   D NAFAC,         NXFAC,      ABUSYZ,     ABUSY(200), XBUSYZ,
                   E XBUSY(200),    ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
                   F XSOR(12,200),ADEST(12,200),XDEST(12,200),          AFAC(12,15),
                   G XFAC(12,15),   AFACSC(4,15,20),ARET,       XFACSC(4,15,20),XRET,
                   H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
                   I AOBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
                   J AFDLY(15),     XFDLY(15),  AFOBUS(15),  XFOBUS(15),  NSLOT,
                   K ABUPSZ,        ABUPS(200), XBUPS(200),  ABUFUL(200), XBUFUL(200),
                   L Q(16,16),      SDBA(32,2), NQBUF,       NQTEST,     NQGO,
                   M QINPT,         QCON,       QEMP,        MBUSY,      MFREE,
                   N LOAD,          MEMDLY,     MEMORY(16),  NBOX,       EAV,
                   O MXTIME,        OUTLVL,     IQ(4,16),    RTN,        LONGBR,
                   P SR(8),         ST(8),      SKXP,        SKAP,       NSBUF,
                   Q APASS(200),    XPASS(200), OUT(2),      JOB(6),     SSTOP,
                   R MEMCNT(16),    ABOX(15),   ABXBSY(10),  XBOX(15),   XBXBSY(10)
ISN 0006            COMMON/RLS/     LAST
ISN 0007            INTEGER OUT
ISN 0008            COMMON/TAGS/D(256,70)
ISN 0009            REAL MEMDLY,MXTIME
ISN 0010            REAL TIME
                C
                C
                C
ISN 0011            ENTRY BUSTOA
                C               MOVE OP FROM ABUS TO ABUFF(AINPT)
ISN 0012            DO 10 I=1,25
ISN 0013         10 ABUFF(AINPT,I)=ABUS(I)
ISN 0014            AFULL(AINPT)=1
                C               PERFORM COMPLETE OP DECODE HERE
                C               FIRST DECODE SOURCE-DEST INTERLOCK TAGS
ISN 0015            I=AINPT
ISN 0016            OP=ABUFF(I,2)
ISN 0017            II=ABUFF(I,3)
ISN 0018            IJ=ABUFF(I,4)
ISN 0019            IK=ABUFF(I,5)
```


L Conway
Archives

```
                C               TEST VALID OP TAG TO SEE IF OP VALID
     ISN 0020         IF(D(OP,30).EQ.0) GO TO 11
                C               INVALID OP. ISSUE ERROR MESSAGE, INCR AINPT,
                C               AND RETURN THUS MAKING OP INTO NOP.
     ISN 0022         WRITE(6,998)
     ISN 0023         WRITE(6,999) OP,ABUS(1)
     ISN 0024         WRITE(6,998)
     ISN 0025         AINPT=AINPT+1
     ISN 0026         RETURN
     ISN 0027      11 CONTINUE
                C
                C               SET A(I) SOURCE
     ISN 0028         IF(D(OP, 4).EQ.1) ASOR(I,II+1)=1
                C
                C               SET A(I) DEST
     ISN 0030         IF(D(OP, 5).EQ.1)ADEST(I,II+1)=1
                C
                C               SET A(I+1) SOURCE
     ISN 0032         IF(D(OP, 6).EQ.1) ASOR(I,MOD(II+1,32)+1)=1
                C
                C               SET A(I+1) DEST
     ISN 0034         IF(D(OP, 7).EQ.1)ADEST(I,MOD(II+1,32)+1)=1
                C
                C               SET A(J) SOURCE
     ISN 0036         IF(D(OP, 8).EQ.1) ASOR(I,IJ+1)=1
                C
                C               SET A(J) DEST
     ISN 0038         IF(D(OP, 9).EQ.1)ADEST(I,IJ+1)=1
                C
                C               SET A(J+1) SOURCE
     ISN 0040         IF(D(OP,10).EQ.1) ASOR(I,MOD(IJ+1,32)+1)=1
                C
                C               SET A(K) SOURCE
     ISN 0042         IF(D(OP,11).EQ.1) ASOR(I,IK+1)=1
                C
                C               SET A(K+1) SOURCE
     ISN 0044         IF(D(OP,12).EQ.1) ASOR(I,MOD(IK+1,32)+1)=1
                C
                C               SET XB(I) DEST
     ISN 0046         IF(D(OP,13).EQ.1) ADEST(I,II+33)=1
                C
                C               SET XB(J) DEST
     ISN 0048         IF(D(OP,14).EQ.1) ADEST(I,IJ+33)=1
                C
                C               SET CB(I) DEST
     ISN 0050         IF(D(OP,15).EQ.1) ADEST(I,II+65)=1
                C               SET STORAGE DEST
     ISN 0052         IF(D(OP,28).EQ.1) ADEST(I,89)=1
                C
                C               REMOVE ANY SOURCE-DEST TAGS ON A(0),XBU(0)
```


L. Conway
Archives

```
ISN 0054            ASOR(I,1)=0
ISN 0055            ADEST(I,1)=0
ISN 0056            ASOR(I,33)=0
ISN 0057            ADEST(I,33)=0
                C               SET FACILITY USE TAGS, BUS DEST TAGS
ISN 0058            DO 20 FAC=1,NAFAC
ISN 0059            AFAC(I,FAC)=D(OP,FAC+55)
ISN 0060            IF(AFAC(I,FAC).EQ.0) GO TO 20
ISN 0062            OBUS=AFOBUS(FAC)
ISN 0063            DO 25 DEST=1,NAREGS
ISN 0064            IF(ADEST(I,DEST).NE.0) GO TO 26
ISN 0066         25 CONTINUE
ISN 0067            GO TO 20
ISN 0068         26 AOBUS(I,OBUS)=DEST
                C               CHECK FOR DOUBLE DEST. IF SO, PLACE ON ADJ. BUS
ISN 0069            DEST1=DEST+1
ISN 0070            IF(DEST1.GT.NAREGS) GO TO 20
ISN 0072            DO 27 DEST2=DEST1,NAREGS
ISN 0073            IF(ADEST(I,DEST2).NE.0) GO TO 28
ISN 0075         27 CONTINUE
ISN 0076            GO TO 20
ISN 0077         28 AOBUS(I,OBUS+1)=DEST2
ISN 0078         20 CONTINUE
                C               INCREMENT AINPT
ISN 0079            AINPT=AINPT+1
ISN 0080            RETURN
                C
                C
                C
ISN 0081            ENTRY BUSTOX
                C               MOVE OP FROM XBUS TO XBUFF(XINPT)
ISN 0082            DO 110 I=1,25
ISN 0083        110 XBUFF(XINPT,I)=XBUS(I)
ISN 0084            XFULL(XINPT)=1
                C               PERFORM COMPLETE OP DECODE HERE
                C               FIRST DECODE SOURCE-DEST INTERLOCK TAGS
ISN 0085            I=XINPT
ISN 0086            OP=XBUFF(I,2)
ISN 0087            II=XBUFF(I,3)
ISN 0088            IJ=XBUFF(I,4)
ISN 0089            IK=XBUFF(I,5)
                C               TEST VALID OP TAG TO SEE IF OP VALID
ISN 0090            IF(D(OP,30).EQ.0) GO TO 111
                C               INVALID OP. ISSUE ERROR MESSAGE, INCR XINPT,
                C               AND RETURN THUS MAKING OP INTO NOP.
ISN 0092            WRITE(6,998)
ISN 0093            WRITE(6,999) OP,XBUS(1)
ISN 0094            WRITE(6,998)
ISN 0095            XINPT=XINPT+1
ISN 0096            RETURN
```


L. Conway
Archives

```
ISN 0097        111 CONTINUE
                C
                C               SET X(I) SOURCE
ISN 0098            IF(D(OP,16).EQ.1) XSOR(I,II+33)=1
                C
                C               SET X(I) DEST
ISN 0100            IF(D(OP,17).EQ.1)XDEST(I,II+33)=1
                C
                C               SET X(I+1) SOURCE
ISN 0102            IF(D(OP,18).EQ.1) XSOR(I,MOD(II+1,32)+33)=1
                C
                C               SET X(I+1) DEST
ISN 0104            IF(D(OP,19).EQ.1)XDEST(I,MOD(II+1,32)+33)=1
                C
                C               SET X(J) SOURCE
ISN 0106            IF(D(OP,20).EQ.1) XSOR(I,IJ+33)=1
                C
                C               SET X(J) DEST
ISN 0108            IF(D(OP,21).EQ.1)XDEST(I,IJ+33)=1
                C
                C               SET X(K) SOURCE
ISN 0110            IF(D(OP,22).EQ.1) XSOR(I,IK+33)=1
                C
                C               SET AB(I) DEST
ISN 0112            IF(D(OP,23).EQ.1)XDEST(I,II+1)=1
                C
                C               SET C(I) SOURCE
ISN 0114            IF(D(OP,24).EQ.1) XSOR(I,II+65)=1
                C
                C               SET C(I) DEST
ISN 0116            IF(D(OP,25).EQ.1)XDEST(I,II+65)=1
                C
                C               SET C(J) SOURCE
ISN 0118            IF(D(OP,26).EQ.1) XSOR(I,IJ+65)=1
                C
                C               SET C(K) SOURCE
ISN 0120            IF(D(OP,34).EQ.1) XSOR(I,IK+65)=1
                C
                C               SET STORAGE SOURCE
ISN 0122            IF(D(OP,27).EQ.1) XSOR(I,89)=1
                C
                C               SET STORAGE DEST
ISN 0124            IF(D(OP,28).EQ.1)XDEST(I,89)=1
                C               REMOVE ANY SOURCE-DEST TAGS ON X(0),ABU(0)
ISN 0126            XSOR(I,1)=0
ISN 0127            XDEST(I,1)=0
ISN 0128            XSOR(I,33)=0
ISN 0129            XDEST(I,33)=0
                C - - - - - - - -SPECIAL DECODE FOR BRANCH OPS - - - - - - - - - - - - - -
                C               PLACE NO FACILITY IF K FIELD = 0 FOR BRANCH OP
```


L. Conway
Archives

```
ISN 0130            IF((XBUFF(I,12).EQ.1).AND.(IK.EQ.0)) GO TO 121
                C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                C                SET FACILITY USE TAGS, BUS DEST TAGS
ISN 0132            DO 120 FAC=1,NXFAC
ISN 0133            XFAC(I,FAC)=D(OP,FAC+40)
ISN 0134            IF(XFAC(I,FAC).EQ.0) GO TO 120
ISN 0136            OBUS=XFOBUS(FAC)
ISN 0137            DO 125 DEST=1,NXREGS
ISN 0138            IF(XDEST(I,DEST).NE.0) GO TO 126
ISN 0140          125 CONTINUE
ISN 0141            GO TO 120
ISN 0142          126 XOBUS(I,OBUS)=DEST
                C                CHECK FOR DOUBLE DEST. IF SO, PLACE ON ADJ BUS
ISN 0143            DESTP1=DEST+1
ISN 0144            IF(DESTP1.GT.NXREGS) GO TO 120
ISN 0146            DO 127 DEST2=DESTP1,NXREGS
ISN 0147            IF(XDEST(I,DEST2).NE.0) GO TO 128
ISN 0149          127 CONTINUE
ISN 0150            GO TO 120
ISN 0151          128 XOBUS(I,OBUS+1)=DEST2
ISN 0152          120 CONTINUE
ISN 0153          121 CONTINUE
                C                INCREMENT XINPT
ISN 0154            XINPT=XINPT+1
ISN 0155            RETURN
ISN 0156          998 FORMAT(1H )
ISN 0157          999 FORMAT(21H ERROR - - - OP TYPE ,I3,14H, INSTRUCTION ,A1,
                   X  53H, IS NOT HANDLED BY THE SIMULATOR - - - - - - - - - - -)
ISN 0158            END
```


L. Conway
Archives

```
LEVEL 2 FEB 67                    OS/360  FORTRAN H                    DATE  67.265/19.31.21

          COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

ISN 0002            SUBROUTINE JSTART(ENDRUN)
ISN 0003            IMPLICIT INTEGER*2(A-Z)
ISN 0004            COMMON          TIME,       IPAR1,      IPAR2,      IPAR3,
                   A AINPT,         NABUF,      ABUS(50),   XINPT,      NXBUF,
                   B XBUS(50),      IFADD,      IFOST,      IFRTN,      BRXP,
                   C BRAP,          ER(8),      BE(8),      ET(8),      NBBUF,
                   D AHOLDT,        XHOLDT,     AFRCT,      XFRCT,      BOSC,
                   E BNOP,          XEP,        AEP,        PH1(100),   PRINT,
                   F FSTADD,        NODUT,      NOPSC,      NOBUS,      NADSP,
                   G NXDSP
ISN 0005            COMMON/RLS/     FIRST,      NAREGS,     NXREGS,     NABUS,
                   A NXBUS,         STATS,      ACON,       XCON,       AEMP,
                   B XEMP,          MXQ,        AFULL(12),  XFULL(12),  AGO(12),
                   C XGO(12),       NAGO,       NXGO,       NATEST,     NXTEST,
                   D NAFAC,         NXFAC,      ABUSYZ,     ABUSY(200), XBUSYZ,
                   E XBUSY(200),    ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
                   F XSOR(12,200),ADEST(12,200),XDEST(12,200),          AFAC(12,15),
                   G XFAC(12,15), AFACSC(4,15,20),ARET,     XFACSC(4,15,20),XRET,
                   H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
                   I AOBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
                   J AFDLY(15),     XFDLY(15),  AFOBUS(15), XFOBUS(15), NSLOT,
                   K ABUPSZ,        ABUPS(200), XBUPS(200), ABUFUL(200), XBUFUL(200),
                   L Q(16,16),      SDBA(32,2), NQBUF,      NQTEST,     NQGO,
                   M QINPT,         QCON,       QEMP,       MBUSY,      MFREE,
                   N LOAD,          MEMDLY,     MEMORY(16), NBOX,       EAV,
                   O MXTIME,        OUTLVL,     IQ(4,16),   RTN,        LONGBR,
                   P SR(8),         ST(8),      SKXP,       SKAP,       NSBUF,
                   Q APASS(200),    XPASS(200), OUT(2),     JOB(6),     SSTOP,
                   R MEMCNT(16),    ABOX(15),   ABXBSY(10), XBOX(15),   XBXBSY(10)
ISN 0006            COMMON/RLS/     LAST
ISN 0007            INTEGER OUT
ISN 0008            REAL MEMDLY,MXTIME
ISN 0009            REAL TIME
ISN 0010            DIMENSION AREPT(10),XREPT(10)
ISN 0011            INTEGER*2 ENDRUN
                C                READ PARAM CARD FOR JOB
ISN 0012            READ(5,100,END=10)(JOB(I),I=1,6),
                   W                          NABUF,NATEST,NAGO,NXBUF,NXTEST,NXGO,
                   X NQBUF,NQTEST,NQGO,NBOX,NBBUF,NSBUF,NODUT,NOPSC,NOBUS,NADSP,NXDSP,
                   Y MXTIME,MEMDLY,OUTLVL,FSTADD
ISN 0013            PRINT=OUTLVL
ISN 0014            ENDRUN=0
                C                WRITE NEW JOB HEADER
ISN 0015            WRITE(6,202)
ISN 0016            WRITE(6,200)
ISN 0017            WRITE(6,300)
ISN 0018            WRITE(6,200)
ISN 0019            WRITE(6,200)
ISN 0020            WRITE(6,400) (JOB(I),I=1,6)
```

280

L. Conway
Archives

```
ISN 0021            WRITE(6,201)
ISN 0022            CALL TMTU(OUT(1))
ISN 0023            WRITE(6,3333) OUT(1),OUT(2)
ISN 0024            WRITE(6,200)
ISN 0025            WRITE(6,200)
ISN 0026            WRITE(6,500)
ISN 0027            WRITE(6,200)
ISN 0028            WRITE(6,600) NABUF,NXBUF,NQBUF
ISN 0029            WRITE(6,201)
ISN 0030            WRITE(6,700) NATEST,NXTEST,NQTEST
ISN 0031            WRITE(6,201)
ISN 0032            WRITE(6,800) NAGO,NXGO,NQGO
ISN 0033            WRITE(6,201)
ISN 0034            WRITE(6,900) MEMDLY
ISN 0035            WRITE(6,201)
ISN 0036            WRITE(6,901) NBOX
ISN 0037            WRITE(6,201)
ISN 0038            WRITE(6,910) NBBUF,NSBUF,NODOT
ISN 0039            WRITE(6,201)
ISN 0040            WRITE(6,920) NOPSC
ISN 0041            WRITE(6,201)
ISN 0042            WRITE(6,930) NDBUS
ISN 0043            WRITE(6,201)
ISN 0044            WRITE(6,940) NADSP,NXDSP
ISN 0045            WRITE(6,200)
ISN 0046            WRITE(6,200)
ISN 0047            WRITE(6,1000)
ISN 0048            WRITE(6,201)
                C               CALC REP TIMES
ISN 0049            DO 20 I=1,NAFAC
ISN 0050            AREPT(I)=0
ISN 0051            DO 20 J=1,NSLOT
ISN 0052         20 AREPT(I)=AREPT(I)+AFSLOT(I,J)
ISN 0053            WRITE(6,1001)(AREPT(I),I=1,NAFAC)
ISN 0054            WRITE(6,1002)(AFDLY(I),I=1,NAFAC)
ISN 0055            WRITE(6,1003)(AFIBUS(I),I=1,NAFAC)
ISN 0056            WRITE(6,1005)(ABOX(I),I=1,NAFAC)
ISN 0057            WRITE(6,1004)(AFOBUS(I),I=1,NAFAC)
ISN 0058            WRITE(6,200)
ISN 0059            WRITE(6,200)
ISN 0060            WRITE(6,2000)
                C               CALC REP TIMES
ISN 0061            DO 30 I=1,NXFAC
ISN 0062            XREPT(I)=0
ISN 0063            DO 30 J=1,NSLOT
ISN 0064         30 XREPT(I)=XREPT(I)+XFSLOT(I,J)
ISN 0065            WRITE(6,1001)(XREPT(I),I=1,NXFAC)
ISN 0066            WRITE(6,1002)(XFDLY(I),I=1,NXFAC)
ISN 0067            WRITE(6,1005)(XBOX(I),I=1,NXFAC)
ISN 0068            WRITE(6,1004)(XFOBUS(I),I=1,NXFAC)
```


L. Conway
Archives

```
ISN 0069          WRITE(6,202)
ISN 0070          CALL UNROLL
ISN 0071          RETURN
ISN 0072       10 CONTINUE
ISN 0073          ENDRUN=1
ISN 0074          RETURN
ISN 0075      100 FORMAT(6A1,2X,17I2,17X,F7.1,1X,F4.1,1XI2,1XI5)
ISN 0076      101 FORMAT(1H1,6A1)
ISN 0077      200 FORMAT(1H0)
ISN 0078      201 FORMAT(1H )
ISN 0079      202 FORMAT(1H1)
ISN 0080      300 FORMAT(120H - - - - - - - - - - - - - - - - - - - - - -  ACS-1  MPM  S
                 XIMULATION  PROGRAM  - - - - - - - - - - - - - - - - - - - - - - - - -)
ISN 0081      400 FORMAT(30H INPUT PROGRAM FOR THIS RUN = ,6A1)
ISN 0082      500 FORMAT(38H MACHINE PARAMETERS FOR THIS RUN - - -)
ISN 0083      600 FORMAT(22H NUMBER OF A BUFFERS =,I2, 5X,21HNUMBER OF X BUFFERS =,
                 X I2, 5X,21HNUMBER OF Q BUFFERS =,I2)
ISN 0084      700 FORMAT(22H NUMBER A OPS TESTED =,I2, 5X,21HNUMBER X OPS TESTED =,
                 X I2, 5X,21HNUMBER Q OPS TESTED =,I2)
ISN 0085      800 FORMAT(22H MAX A OPS ISS/CYCLE =,I2, 5X,21HMAX X OPS ISS/CYCLE =,
                 X I2, 5X,21HMAX Q OPS ISS/CYCLE =,I2)
ISN 0086      900 FORMAT(22H MINIMUM Q-MEM DELAY =,F4.1)
ISN 0087      901 FORMAT(22H NUMBER OF BOMS      =,I2)
ISN 0088      910 FORMAT(22H NUMBER BRANCH REGS  =,I2, 5X,21HNUMBER OF SKIP REGS =,
                 X I2, 5X,21HSIZE OF DO TABLE    =,I2)
ISN 0089      920 FORMAT(22H NUMBER OF PSC REGS  =,I2)
ISN 0090      930 FORMAT(22H NUMBER DISP BUSES   =,I2)
ISN 0091      940 FORMAT(22H MAX A OPS DSP/CYCLE =,I2, 5X,21HMAX X OPS DSP/CYCLE =,
                 X I2)
ISN 0092     1000 FORMAT(66H A FACILITIES - - FA1  FA2   FM   FD   IA   IM   ID   C
                 X    L    S)
ISN 0093     1001 FORMAT(16H REP TIME    =   ,15(3XI2))
ISN 0094     1002 FORMAT(16H DELAY TIME =    ,15(3XI2))
ISN 0095     1003 FORMAT(16H INBUS       =   ,15(3XI2))
ISN 0096     1004 FORMAT(16H OUTBUS      =   ,15(3XI2))
ISN 0097     1005 FORMAT(16H BOX         =   ,15(3XI2))
ISN 0098     2000 FORMAT(66H X FACILITIES - - EA1  EA2    L    S    M    D   XA    C
                 X   SP       )
ISN 0099     3333 FORMAT(19H TIME/DATE OF RUN =,2(1XZ8))
ISN 0100          END
```

282

L. Conway
Archives

```
LEVEL 5 DEC 66                          DATE  YS/360  FORTRAN H                          DATE  67.144/09.18.12

      COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

ISN 0002              SUBROUTINE XQCON
ISN 0003              IMPLICIT INTEGER*2(A-Z)
ISN 0004              COMMON        TIME,       IPAR1,      IPAR2,      IPAR3,
                     A AINPT,       NABUF,      ABUS(50),   XINPT,      NXBUF,
                     B XBUS(50),    IFADD,      IFDST,      IFRTN,      BRXP,
                     C BRAP,        ER(8),      BE(8),      ET(8),      NBBUF,
                     D AHOLDT,      XHOLDT,     AFRCT,      XFRCT,      BOSC,
                     E BNUP,        XEP,        AEP,        PHI(100),   PRINT,
                     F FSTADD,      NODOT,      NOPSC,      NDBUS,      NADSP,
                     G NXDSP
ISN 0005              COMMON/RLS/   FIRST,      NAREGS,     NXREGS,     NABUS,
                     A NXBUS,       STATS,      ACON,       XCON,       AEMP,
                     B XEMP,        MXG,        AFULL(12),  XFULL(12),  AGO(12),
                     C XGO(12),     NAGO,       NXGO,       NATEST,     NXTEST,
                     D NAFAC,       NXFAC,      ABUSYZ,     ABUSY(200), XBUSYZ,
                     E XBUSY(200),  ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
                     F XSOR(12,200),ADEST(12,200),XDEST(12,200),AFAC(12,15),
                     G XFAC(12,15), AFACSC(4,15,20),ARET,    XFACSC(4,15,20),XRET,
                     H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
                     I AOBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
                     J AFDLY(15),   XFDLY(15),  AFOBUS(15), XFOBUS(15), NSLOT,
                     K ABUPSZ,      ABUPS(200), XBUPS(200), ABUFUL(200), XBUFUL(200),
                     L Q(16,16),    SDBA(32,2), NQBUF,      NQTEST,     NQGO,
                     M QINPT,       QCON,       QEMP,       MBUSY,      MFREE,
                     N LOAD,        MEMDLY,     MEMORY(16), NBOX,       EAV,
                     O MXTIME,      OUTLVL,     IQ(4,16),   RTN,        LONGBR,
                     P SR(8),       ST(8),      SKXP,       SKAP,       NSBUF,
                     Q APASS(200),  XPASS(200), OUT(2),     JOB(6),     SSTOP,
                     R MEMCNT(16),  ABOX(15),   ABXBSY(10), XBOX(15),   XBXBSY(10)
ISN 0006              COMMON/RLS/   LAST
ISN 0007              INTEGER OUT
ISN 0008              REAL MEMDLY,MXTIME
ISN 0009              REAL TIME
                    C               ALGORITHM...LOADS OUT OF ORDER WITH BOM INTLK.
                    C               STORES IN ORDER.
ISN 0010              CALL CAUSE(QCON,TIME+1.0,0,0,0)
ISN 0011              CALL CAUSE(QEMP,TIME+0.8,0,0,0)
ISN 0012              NGO=0
ISN 0013              DO 1 I=1,NQBUF
ISN 0014            1 Q(I,16)=0
ISN 0015              DO 100 INS=1,NQTEST
ISN 0016              IF(Q(INS,8).EQ.0) GO TO 100
ISN 0018              IF(INS.EQ.1) GO TO 11
ISN 0020              INSM1=INS-1
ISN 0021              DO 10 I=1,INSM1
                    C               IF PREV GO INS TO SAME BOM, NOGO
ISN 0022              IF((Q(I,6).EQ.Q(INS,6)).AND.(Q(I,16).EQ.1)) GO TO 100
                    C               IF PREV INS TO SAME WORD, NOGO
ISN 0024              IF(Q(I,7).EQ.Q(INS,7)) GO TO 100
```


L. Conway
Archives

```
                C               IF STORE AND PREV NOGO STORE, NOGO
ISN 0026            IF(Q(INS,3).NE.1) GO TO 10
ISN 0028            IF((Q(I,3).EQ.1).AND.(Q(I,16).EQ.0)) GO TO 100
ISN 0030         10 CONTINUE
                C               IF STORE AND DATA NOT AVAIL, NOGO
ISN 0031         11 IF((Q(INS,3).EQ.1).AND.(Q(INS,9).EQ.0)) GO TO 100
                C               MARK GO
ISN 0033            Q(INS,16)=1
ISN 0034            NGO=NGO+1
ISN 0035            IF(NGO.GT.NQGO) GO TO 101
ISN 0037        100 CONTINUE
ISN 0038        101 CONTINUE
                C               TEST INS FETCH REQ FOR ISSUANCE
ISN 0039            DO 200 II=1,4
ISN 0040            IF(IQ(II,1).EQ.0)GO TO 200
                C               COMPARE BOM REQD AGAINST GO DATA REQSTS
ISN 0042            DO 150 ID=1,NQBUF
ISN 0043            IF((IQ(II,6).EQ. Q(ID,6)).AND.( Q(ID,16).EQ.1)) GO TO 200
ISN 0045        150 CONTINUE
                C               MARK INS FETCH REQ GO
ISN 0046            IQ(II,16)=1
ISN 0047        200 CONTINUE
ISN 0048            RETURN
ISN 0049            END
```


L. Conway
Archives

```
      COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

ISN 0002            SUBROUTINE XQEMP
ISN 0003            IMPLICIT INTEGER(A-Z)
ISN 0004            COMMON /[illegible]/          IPAR1,       IPAR2,       IPAR3,
                   A AINPT,      NABUF,        ABUS(50),    XINPT,       NXBUF,
                   B XBUS(50),   IFADD,        IFDST,       IFRTN,       BRXP,
                   C BRAP,       ER(8),        BE(8),       ET(8),       NBBUF,
                   D AHOLDT,     XHOLDT,       AFRCT,       XFRCT,       BOSC,
                   E BNOP,       XEP,          AEP,         PH1(100),    PRINT,
                   F FSTADD,     NODOT,        NOPSC,       NDBUS,       NADSP,
                   G NXDSP
ISN 0005            COMMON/RLS/  FIRST,        NAREGS,      NXREGS,      NABUS,
                   A NXBUS,      STATS,        ACON,        XCON,        AEMP,
                   B XEMP,       MXO,          AFULL(12),   XFULL(12),   AGO(12),
                   C XGO(12),    NAGO,         NXGO,        NATEST,      NXTEST,
                   D NAFAC,      NXFAC,        ABUSYZ,      ABUSY(200),  XBUSYZ,
                   E XBUSY(200), ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
                   F XSOR(12,200),ADEST(12,200),XDEST(12,200),           AFAC(12,15),
                   G XFAC(12,15), AFACSC(4,15,20),ARET,       XFACSC(4,15,20),XRET,
                   H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
                   I AOBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
                   J AFDLY(15),  XFDLY(15),    AFOBUS(15),  XFOBUS(15),  NSLOT,
                   K ABUPSZ,     ABUPS(200),   XBUPS(200),  ABUFUL(200), XBUFUL(200),
                   L Q(16,16),   SDBA(32,2),   NQBUF,       NQTEST,      NQGO,
                   M QINPT,      QCON,         QEMP,        MBUSY,       MFREE,
                   N LOAD,       MEMDLY,       MEMORY(16),  NBOX,        EAV,
                   O MXTIME,     OUTLVL,       IQ(4,16),    RTN,         LONGBR,
                   P SR(8),      ST(8),        SKXP,        SKAP,        NSBUF,
                   Q APASS(200), XPASS(200),   OUT(2),      JOB(6),      SSTOP,
                   R MEMCNT(16), ABOX(15),     ABXBSY(10),  XBOX(15),    XBXBSY(10)
ISN 0006            COMMON/RLS/  LAST
ISN 0007            INTEGER OUT
ISN 0008            REAL MEMDLY,MXTIME
ISN 0009            REAL TIME
ISN 0010            DIMENSION CODE(12)
ISN 0011            DATA CODE/1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9,1HA,1HB,1HC/
              C
              C
              C                ISSUE GO DATA REQ
ISN 0012            DO 100 INS=1,NQBUF
ISN 0013          5 IF(Q(INS,16).EQ.0) GO TO 100
ISN 0015            IF(Q(INS,8 ).EQ.0) GO TO 100
              C                ISSUE INS TO MEMORY
ISN 0017            BOM=Q(INS,6)
ISN 0018            DEST=Q(INS,15)
ISN 0019            A=Q(INS,4)
ISN 0020            X=Q(INS,5)
ISN 0021            L=Q(INS,1)
ISN 0022            MEMCNT(BOM)=MEMCNT(BOM)+1
ISN 0023            CALL CAUSE(MBUSY,TIME+MEMDLY-3.0,BOM,L,0)
```

285

L. Conway
Archives

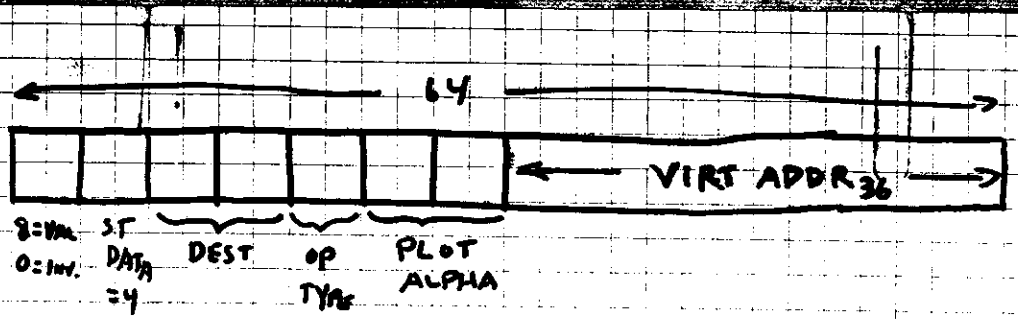
```
C                IF LOAD, CAUSE DEST LOAD IN MEMDLY CYCLES
ISN 0025         IF(Q(INS,2).EQ.1)CALL CAUSE(LOAD,TIME+MEMDLY-1.0,DEST,A,X)
C                REMOVE INS FROM QUEUE
ISN 0027         DO 30 I=INS,M
ISN 0028         DO 30 J=1,16
ISN 0029         Q(I,J)=Q(I+1,J)
ISN 0031      30 CONTINUE
ISN 0032         DO 32 J=1,16
ISN 0033         Q(NQBUF,J)=0
ISN 0034      32 CONTINUE
ISN 0035         GO TO 5
ISN 0036     100 CONTINUE
C
C
C                ISSUE GO INS FETCH REQ TEST
ISN 0037         DO 200 II=1,4
ISN 0038         IF(IQ(II,1).EQ.0) GO TO 200
ISN 0040         IF(IQ(II,16).EQ.0)GO TO 200
C                ISSUE REQ
ISN 0042         BOM=IQ(II,6)
ISN 0043         DEST=IQ(II,15)
ISN 0044         L=IQ(II,1)
ISN 0045         MEMCNT(BOM)=MEMCNT(BOM)+1
ISN 0046         CALL CAUSE(MBUSY,TIME+MEMDLY-3.0,BOM,L,0)
ISN 0047         CALL CAUSE(MFREE,TIME+MEMDLY-2.1,BOM,0,0)
C                ZERO POSN IN IQ
ISN 0048         DO 150 I=1,16
ISN 0049    150  IQ(II,I)=0
C                IF LAST OF 4 INS FETCH REQSTS, CAUSE RTN
ISN 0050         DO 160 I=1,4
ISN 0051         IF(IQ(I,1).NE.0) GO TO 200
ISN 0053    160  CONTINUE
ISN 0054         CALL CAUSE(RTN,TIME+MEMDLY-1.0,DEST,0,0)
ISN 0055    200  CONTINUE
C
C
C                FILL IQ IF EMPTY AND INS REQ PRESENT ON INTERFACE
C                FIRST TEST IF IQ EMPTY
ISN 0056         DO 250 II=1,4
ISN 0057         IF(IQ(II,1).NE.0) GO TO 400
ISN 0059    250  CONTINUE
C                TEST IF REQ PRESENT
ISN 0060         IF(IFDST.EQ.0) GO TO 400
C                FILL IQ
ISN 0062         BOM=MOD(IFADD,NBOX)+1
```

286

L. Conway
Archives

```
ISN 0067            DO 300 II=1,6
ISN 0068            IQ(II,1)=CODE(IFDST)
ISN 0069            IQ(II,6)=BOM+II-1
ISN 0070            IQ(II,7)=IFADD
ISN 0071            IQ(II,15)=IFDST
ISN 0072        300 CONTINUE
                C                   ZERO INTERFACE
ISN 0073            IFADD=0
ISN 0074            IFDST=0
ISN 0075        400 CONTINUE
ISN 0076            RETURN
ISN 0077            END
```


L. Conway
Archives



'BLCU' ——— call every cycle

'BLCNTL' ——— to get BLM control card
Placed after MPM control card.

DAL
DQI
INS

3 × 64 bits requests

L. Conway Archives

Current MPM — MQ interface:

C Variables:

Q(16,16)
SDBA(32,2)
NQBUF
NQTEST
NQGO
QINPT
MEMOLY
MEMORY(16)
NBOX
IQ(4,16)
MEMCNT(16) (STOR CONFN)

EVENTS - CAUSATION

.1	.8	.9
QCON	MFREE	QEMP
		RTN
		MBUSY
		LOAD
		EAV

XBON — test for availability of Q space, as one of the interlocks.

XEMP — If GO HIS, Place on the interface

ACQN — STORE A — KEPT IN ORDER, INTLK ON BUS

AEMP — PLACE DATA ON INTERFACE

L. Conway Archives

INS Q:

INTERFACE VARIABLES: IFDST, IFADD, IFRTN

IF (IFDST N.E. 0) FILL IQ (or PLACE REQST)

CODE(IFDST) is destination literal

ZERO QL interface IFDST, IFADD

RTN:

IFRTN = DEST

L. Conway Archives

```
      COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

ISN 0002            SUBROUTINE SRQ
ISN 0003            IMPLICIT INTEGER*2(A-Z)
ISN 0004            COMMON      TIME,       IPAR1,      IPAR2,      IPAR3,
                   A AINPT,     NABUF,      ABUS(50),   XINPT,      NXBUF,
                   B XBUS(50),  IFADD,      IFDST,      IFRTN,      BRXP,
                   C BRAP,      ER(8),      BE(8),      ET(8),      NBBUF,
                   D AHOLDT,    XHOLDT,     AFRCT,      XFRCT,      BOSC,
                   E BNOP,      XEP,        AEP,        PH1(100),   PRINT,
                   F FSTADD,    NODOT,      NOPSC,      NDBUS,      NADSP,
                   G NXDSP
ISN 0005             COMMON/RLS/ FIRST,      NAREGS,     NXREGS,     NABUS,
                   A NXBUS,     STATS,      ACON,       XCON,       AEMP,
                   B XEMP,      MXO,        AFULL(12),  XFULL(12),  AGO(12),
                   C XGO(12),   NAGO,       NXGO,       NATEST,     NXTEST,
                   D NAFAC,     NXFAC,      ABUSYZ,     ABUSY(200), XBUSYZ,
                   E XBUSY(200), ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
                   F XSOR(12,200),ADEST(12,200),XDEST(12,200),          AFAC(12,15),
                   G XFAC(12,15), AFACSC(4,15,20),ARET,     XFACSC(4,15,20),XRET,
                   H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
                   I AOBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
                   J AFDLY(15),  XFDLY(15),  AFOBUS(15), XFOBUS(15), NSLOT,
                   K ABUPSZ,     ABUPS(200), XBUPS(200), ABUFUL(200),XBUFUL(200),
                   L Q(16,16),   SDBA(32,2), NQBUF,      NQTEST,     NQGO,
                   M QINPT,      QCON,       QEMP,       MBUSY,      MFREE,
                   N LOAD,       MEMDLY,     MEMORY(16), NBOX,       EAV,
                   O MXTIME,     OUTLVL,     IQ(4,16),   RTN,        LONGBR,
                   P SR(8),      ST(8),      SKXP,       SKAP,       NSBUF,
                   Q APASS(200), XPASS(200), OUT(2),     JOB(6),     SSTOP,
                   R MEMCNT(16), ABOX(15),   ABXBSY(10), XBOX(15),   XBXBSY(10)
ISN 0006             COMMON/RLS/ LAST
ISN 0007            INTEGER OUT
ISN 0008            REAL MEMDLY,MXTIME
ISN 0009            REAL TIME
                  C
ISN 0010            ENTRY XMBUSY
ISN 0011            BOM=IPAR1
ISN 0012            L=IPAR2
ISN 0013            MEMORY(BOM)=L
ISN 0014            RETURN
                  C
ISN 0015            ENTRY XMFREE
ISN 0016            BOM=IPAR1
ISN 0017            MEMCNT(BOM)=MEMCNT(BOM)-1
ISN 0018            MEMORY(BOM)=0
ISN 0019            RETURN
                  C
ISN 0020            ENTRY XLOAD
ISN 0021            DEST=IPAR1
ISN 0022            IF(ABUPS(DEST).NE.1) GO TO 9
```

291

L. Conway Archives

```
ISN 0024            IF(APASS(DEST).NE.0) GO TO 8
ISN 0026            ABUFUL(DEST)=1
ISN 0027            RETURN
ISN 0028          8 ABUSY(DEST)=0
ISN 0029            APASS(DEST)=0
ISN 0030          9 XBUSY(DEST)=0
ISN 0031            RETURN
                  C
ISN 0032            ENTRY XRTN
ISN 0033            DEST=IPAR1
ISN 0034            IFRTN=DEST
ISN 0035            RETURN
                  C
ISN 0036            ENTRY XEAV
ISN 0037            DO 10 I=1,NQBUF
ISN 0038            IF(Q(I,8).EQ.1) GO TO 10
ISN 0040            Q(I,8)=1
ISN 0041            RETURN
ISN 0042         10 CONTINUE
ISN 0043            A=1
ISN 0044            B=20000
ISN 0045            C=101
ISN 0046            CALL TROUBL(A,B,C)
ISN 0047            RETURN
                  C
ISN 0048            END
```


L. Conway
Archives

```
      COMPILER OPTIONS - NAME=  MAIN,OPT=00,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

ISN 0002        SUBROUTINE XSTATS
ISN 0003        IMPLICIT INTEGER*2(A-Z)
ISN 0004        COMMON          TIME,  *        IPAR1,          IPAR2,          IPAR3,
               A AINPT,         NABUF,          ABUS(50),       XINPT,          NXBUF,
               B XBUS(50),      IFADD,          IFDST,          IFRTN,          BRXP,
               C BRAP,          ER(8),          BE(8),          ET(8),          NBBUF,
               D AHOLDT,        XHOLDT,         AFRCT,          XFRCT,          BOSC,
               E BNOP,          XEP,            AEP,            PH1(100),       PRINT,
               F FSTADD,        NODOT,          NOPSC,          NDBUS,          NADSP,
               G NXDSP
ISN 0005        COMMON/RLS/     FIRST,          NAREGS,         NXREGS,         NABUS,
               A NXBUS,         STATS,          ACON,           XCON,           AEMP,
               B XEMP,          MXO,            AFULL(12),      XFULL(12),      AGO(12),
               C XGO(12),       NAGO,           NXGO,           NATEST,         NXTEST,
               D NAFAC,         NXFAC,          ABUSYZ,         ABUSY(200),     XBUSYZ,
               E XBUSY(200),    ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
               F XSOR(12,200),ADEST(12,200),XDEST(12,200),              AFAC(12,15),
               G XFAC(12,15),   AFACSC(4,15,20),ARET,           XFACSC(4,15,20),XRET,
               H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
               I AOBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
               J AFDLY(15),     XFDLY(15),      AFOBUS(15),     XFOBUS(15),     NSLOT,
               K ABUPSZ,        ABUPS(200),     XBUPS(200),     ABUFUL(200),    XBUFUL(200),
               L Q(16,16),      SDBA(32,2),     NQBUF,          NQTEST,         NQGO,
               M QINPT,         QCON,           QEMP,           MBUSY,          MFREE,
               N LOAD,          MEMDLY,         MEMORY(16),     NBOX,           EAV,
               O MXTIME,        OUTLVL,         IQ(4,16),       RTN,            LONGBR,
               P SR(8),         ST(8),          SKXP,           SKAP,           NSBUF,
               Q APASS(200),    XPASS(200),     OUT(2),         JOB(6),         SSTOP,
               R MEMCNT(16),    ABOX(15),       ABXBSY(10),     XBOX(15),       X8XBSY(10)
ISN 0006        COMMON/RLS/     LAST
ISN 0007        INTEGER OUT
ISN 0008        REAL MEMDLY,MXTIME
ISN 0009        REAL TIME
ISN 0010        INTEGER INDEX
ISN 0011        COMMON/OBUF/SPH1(80,100),SABUFF(12,100),SXBUFF(12,100),
               A SAREG(32,100),SXREG(32,100),SABREG(32,100),SCBIT(24,100),
               B SCBBIT(24,100),SAFAC(15,100),SXFAC(15,100),SQ(16,100),
               C SMEM(16,100),SIQ(4,100),SB(34,100),SS(10,100)
ISN 0012        DIMENSION BB(2)
ISN 0013        EQUIVALENCE(BB(1),SPH1(1,1))
ISN 0014        DIMENSION XMN(10),AMN(10)
ISN 0015        DIMENSION BSYM(68),SSYM(20)
ISN 0016        DATA SSYM/40HSR 1SR 2SR 3SR 4SR 5SR 6SR 7SR 8SKXPSKAP/
ISN 0017        DATA BSYM/136HER 1ER 2ER 3ER 4ER 5ER 6ER 7ER 8BE 1BE 2BE 3BE 4BE 5
               ABE 6BE 7BE 8ET 1ET 2ET 3ET 4ET 5ET 6ET 7ET 8BRXPBRAPXHLTAHLTXFCTAF
               BCTXEP AEP BOSCBNOP/
ISN 0018        DATA XMN/20HEAEAL S M D XAC SP  /
ISN 0019        DATA AMN/20HFAFAFMFDIAIMIDC L S /
ISN 0020        DATA BLNK/2H  /
```

293

L. Conway
Archives

```
                C               OUTPUT LEVELS AS FOLLOWS
                C               OUTLVL=0    FULL DEBUG INCLUDED
                C               OUTLVL=1    CYC/CYC INCLUDED
                C               OUTLVL=2    FULL 100 CYC INCLUDED
                C               OUTLVL=3    MIN  100 CYC INCLUDED
                C               FIRST TIME THRU, BLANK THE OUTPUT ARRAYS
ISN 0021            ABNORM=0
ISN 0022            IF(TIME.GT.0.0) GO TO 60
ISN 0024            DO   50 INDEX=1,35800
ISN 0025         50 BB(INDEX)=BLNK
ISN 0026         60 CONTINUE
ISN 0027            CALL CAUSE(STATS,TIME+1.0,0,0,0)
                C
                C - - - - - - - - - - -  PLACE PER CYCLE OUTPUT HERE - - - - - - - - - - -
                C
                C               OUTPUT PER CYCLE IF OUTLVL LE 1
ISN 0028            IF(OUTLVL.GT.1) GO TO 100
ISN 0030            WRITE(6,1000)TIME
ISN 0031        100 CONTINUE
ISN 0032            ITIME=TIME
ISN 0033            JT=MOD(ITIME,100)+1
ISN 0034            IF(JT.NE.1) GO TO 2050
ISN 0036            IF(TIME.EQ.0.0) GO TO 2050
                C
                C - - - - - - - - - OUTPUT 100 CYCLE OUTPUT - - - - - - - - - - - - - - - -
                C
ISN 0038        2500 CONTINUE
ISN 0039            ITIME=TIME
ISN 0040            BTIME=ITIME-MOD(ITIME,100)
ISN 0041            IF(MOD(ITIME,100).EQ.0) BTIME=ITIME-100
ISN 0043            FTIME=ITIME-1
ISN 0044            CALL TMTU(OUT(1))
ISN 0045            WRITE(6,2610) BTIME,FTIME,(JOB(I),I=1,6),OUT(1),OUT(2)
ISN 0046            WRITE(6,2611)
ISN 0047            IF(OUTLVL.EQ.3) GO TO 888
                C               OUTPUT DISP REG AND PH1
ISN 0049            WRITE(6,2630)
ISN 0050            WRITE(6,2640)(SPH1(1,T),T=1,100)
ISN 0051            WRITE(6,2641)(SPH1(2,T),T=1,100)
ISN 0052            DO 90 I=1,8
ISN 0053            J=I+2
ISN 0054         90 WRITE(6,2622)I,(SPH1(J,T),T=1,100),I
ISN 0055            WRITE(6,2630)
ISN 0056            WRITE(6,2643)(SPH1(11,T),T=1,100)
ISN 0057            WRITE(6,2641)(SPH1(12,T),T=1,100)
ISN 0058            DO 91 I=1,8
ISN 0059            J=I+12
ISN 0060         91 WRITE(6,2622)I,(SPH1(J,T),T=1,100),I
ISN 0061            WRITE(6,2630)
ISN 0062            WRITE(6,2644)(SPH1(21,T),T=1,100)
```


L. Conway
Archives

```
ISN 0063            WRITE(6,2641)(SPH1(22,T),T=1,100)
ISN 0064            DO 92 I=1,8
ISN 0065            J=I+22
ISN 0066         92 WRITE(6,2622)I,(SPH1(J,T),T=1,100),I
ISN 0067            WRITE(6,2630)
ISN 0068            WRITE(6,2645)(SPH1(31,T),T=1,100)
ISN 0069            WRITE(6,2641)(SPH1(32,T),T=1,100)
ISN 0070            DO 93 I=1,8
ISN 0071            J=I+32
ISN 0072         93 WRITE(6,2622)I,(SPH1(J,T),T=1,100),I
ISN 0073            WRITE(6,2630)
ISN 0074            WRITE(6,2646)(SPH1(41,T),T=1,100)
ISN 0075            DO 94 I=42,80
ISN 0076         94 WRITE(6,2642)(SPH1(I,T),T=1,100)
ISN 0077        888 CONTINUE
                C                OUTPUT BRANCH CONTROLS
ISN 0078            WRITE(6,2630)
ISN 0079            I=1
ISN 0080            WRITE(6,2637)BSYM(1),BSYM(2),(SB(1,T),T=1,100)
ISN 0081            DO 104 J=1,17,8
ISN 0082            DO 104 K=1,NBBUF
ISN 0083            I=J+K-1
ISN 0084            IF(I.EQ.1) GO TO 104
ISN 0086            WRITE(6,2638)BSYM(2*I-1),BSYM(2*I),(SB(I,T),T=1,100)
ISN 0087        104 CONTINUE
ISN 0088            DO 103 I=25,34
ISN 0089        103 WRITE(6,2638)BSYM(2*I-1),BSYM(2*I),(SB(I,T),T=1,100)
                C                OUTPUT SKIP CONTROLS
ISN 0090            WRITE(6,2630)
ISN 0091            WRITE(6,2647)SSYM(1),SSYM(2),(SS(1,T),T=1,100)
ISN 0092            DO 102 K=2,NSBUF
ISN 0093        102 WRITE(6,2648)SSYM(2*K-1),SSYM(2*K),(SS(K,T),T=1,100)
ISN 0094            DO 101 K=9,10
ISN 0095        101 WRITE(6,2648)SSYM(2*K-1),SSYM(2*K),(SS(K,T),T=1,100)
                C                OUTPUT ABUFF
ISN 0096            WRITE(6,2630)
ISN 0097            I=1
ISN 0098            WRITE(6,2623)I,(SABUFF(1,T),T=1,100),I
ISN 0099            DO 110 I=2,NABUF
ISN 0100        110 WRITE(6,2622)I,(SABUFF(I,T),T=1,100),I
                C                OUTPUT XBUFF
ISN 0101            WRITE(6,2630)
ISN 0102            I=1
ISN 0103            WRITE(6,2624)I,(SXBUFF(I,T),T=1,100),I
ISN 0104            DO 111 I=2,NXBUF
ISN 0105        111 WRITE(6,2622)I,(SXBUFF(I,T),T=1,100),I
                C                OUTPUT A FACILITIES
ISN 0106            WRITE(6,2630)
ISN 0107            I=1
ISN 0108            WRITE(6,2631)AMN(I),I,(SAFAC(I,T),T=1,100),I
```


L. Conway
Archives

```
ISN 0109            DO 108 I=2,NAFAC
ISN 0110          108 WRITE(6,2635)AMN(I),I,(SAFAC(I,T),T=1,100),I
                  C                 OUTPUT X FACILITIES
ISN 0111            WRITE(6,2630)
ISN 0112            I=1
ISN 0113            WRITE(6,2632)XMN(I),I,(SXFAC(I,T),T=1,100),I
ISN 0114            DO 109 I=2,NXFAC
ISN 0115          109 WRITE(6,2635)XMN(I),I,(SXFAC(I,T),T=1,100),I
                  C                 OUTPUT Q BUSY
ISN 0116            WRITE(6,2630)
ISN 0117            IF(OUTLVL.EQ.3) GO TO 889
ISN 0119            I=1
ISN 0120            WRITE(6,2633)I,(  SQ(I,T),T=1,100),I
ISN 0121            DO 106 I=2,NQBUF
ISN 0122          106 WRITE(6,2622) I,(SQ(I,T),T=1,100),I
                  C                 OUTPUT IQ BUSY
ISN 0123            WRITE(6,2630)
ISN 0124            I=1
ISN 0125            WRITE(6,2636)I,(SIQ(I,T),T=1,100),I
ISN 0126            DO 105 I=2,4
ISN 0127          105 WRITE(6,2622)I,(SIQ(I,T),T=1,100),I
                  C                 OUTPUT MEM BUSY
ISN 0128            WRITE(6,2630)
ISN 0129            I=1
ISN 0130            WRITE(6,2634)I,(SMEM(I,T),T=1,100),I
ISN 0131            DO 107 I=2,NBOX
ISN 0132          107 WRITE(6,2622) I,(SMEM(I,T),T=1,100),I
                  C                 OUTPUT AREGS    BUSY
ISN 0133            WRITE(6,2630)
ISN 0134            J=0
ISN 0135            I=1
ISN 0136            WRITE(6,2625)J,( SAREG(I,T),T=1,100),J
ISN 0137            DO 112 I=2,32
ISN 0138            J=I-1
ISN 0139          112 WRITE(6,2622)J,( SAREG(I,T),T=1,100),J
                  C                 OUTPUT ABU REGS BUSY
ISN 0140            WRITE(6,2630)
ISN 0141            J=0
ISN 0142            I=1
ISN 0143            WRITE(6,2626)J,(SABREG(I,T),T=1,100),J
ISN 0144            DO 113 I=2,32
ISN 0145            J=I-1
ISN 0146          113 WRITE(6,2622)J,(SABREG(I,T),T=1,100),J
                  C                 OUTPUT XREGS    BUSY
ISN 0147            WRITE(6,2630)
ISN 0148            J=0
ISN 0149            I=1
ISN 0150            WRITE(6,2627)J,( SXREG(I,T),T=1,100),J
ISN 0151            DO 114 I=2,32
ISN 0152            J=I-1
```


L. Conway
Archives

```
ISN 0153       114 WRITE(6,2622)J,( SXREG(I,T),T=1,100),J
                C                  OUTPUT C BITS    BUSY
ISN 0154           WRITE(6,2630)
ISN 0155           J=0
ISN 0156           I=1
ISN 0157           WRITE(6,2628)J,( SCBIT(I,T),T=1,100),J
ISN 0158           DO 115 I=2,24
ISN 0159           J=I-1
ISN 0160       115 WRITE(6,2622)J,( SCBIT(I,T),T=1,100),J
                C                  OUTPUT CBU BITS  BUSY
ISN 0161           WRITE(6,2630)
ISN 0162           J=0
ISN 0163           I=1
ISN 0164           WRITE(6,2629)J,(SCBBIT(I,T),T=1,100),J
ISN 0165           DO 116 I=2,24
ISN 0166           J=I-1
ISN 0167       116 WRITE(6,2622)J,(SCBBIT(I,T),T=1,100),J
                C
ISN 0168           WRITE(6,2630)
ISN 0169       889 CONTINUE
ISN 0170           DO 2550 INDEX=1,35800
ISN 0171      2550 BB(INDEX)=BLNK
                C
                C- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                C                  IF ABNORMAL TERMINATION,I.E. ENTERED AT FINIS,
                C                  WRITE OUT OBUF, CALL TROUBL, THEN STOP.
ISN 0172           IF(ABNORM.EQ.0) GO TO 7777
ISN 0174           A=1
ISN 0175           B=20000
ISN 0176           C=7777
ISN 0177           CALL TROUBL(A,B,C)
ISN 0178           STOP
                C
ISN 0179           ENTRY FINIS
ISN 0180           ABNORM=1
ISN 0181           GO TO 2500
ISN 0182      7777 CONTINUE
                C - - - - - - - - FILL CYCLE POSITION IN 100 CYCLE BUFFER - - - - - - - -
                C
ISN 0183           IF(SSTOP.EQ.1) RETURN
ISN 0185      2050 CONTINUE
ISN 0186           CYCLE=JT
ISN 0187           CALL STOBUF(CYCLE)
                C                  TEST FOR STOP CONDITION
ISN 0188           IF(PH1(100).EQ.0) RETURN
ISN 0190           DO 200 I=1,NABUF
ISN 0191           IF(ABUFF(I,1).NE.0) RETURN
ISN 0193       200 CONTINUE
ISN 0194           DO 201 I=1,NXBUF
ISN 0195           IF(XBUFF(I,1).NE.0) RETURN
```

297

L. Conway
Archives

```
ISN 0197        201 CONTINUE
ISN 0198            DO 206 I=1,NQBUF
ISN 0199            IF(Q(I,1).NE.0) RETURN
ISN 0201        206 CONTINUE
ISN 0202            DO 207 I=1,NBOX
ISN 0203            IF(MEMCNT(I).NE.0) RETURN
ISN 0205            IF(MEMORY(I).NE.0) RETURN
ISN 0207        207 CONTINUE
ISN 0208            DO 208 I=1,NAREGS
ISN 0209            IF(ABUSY(I).NE.0) RETURN
ISN 0211        208 CONTINUE
ISN 0212            DO 209 N=1,NXREGS
ISN 0213            IF(XBUSY(I).NE.0) RETURN
ISN 0215        209 CONTINUE
ISN 0216            DO 202 I=1,NAFAC
ISN 0217            DO 202 J=1,NSLOT
ISN 0218            IF(AFACSC(2,I,J).NE.0) RETURN
ISN 0220        202 CONTINUE
ISN 0221            DO 203 I=1,NXFAC
ISN 0222            DO 203 J=1,NSLOT
ISN 0223            IF(XFACSC(2,I,J).NE.0) RETURN
ISN 0225        203 CONTINUE
ISN 0226            DO 204 I=1,NABUS
ISN 0227            DO 204 J=1,NSLOT
ISN 0228            IF(ABUSSC(2,I,J).NE.0) RETURN
ISN 0230        204 CONTINUE
ISN 0231            DO 205 I=1,NXBUS
ISN 0232            DO 205 J=1,NSLOT
ISN 0233            IF(XBUSSC(2,I,J).NE.0) RETURN
ISN 0235        205 CONTINUE
                C               STOP CONDITION TRUE. TERMINATE RUN.
ISN 0236            SSTOP=1
ISN 0237            GO TO 2500
ISN 0238       1000 FORMAT(6H TIME=,F6.2)
ISN 0239       2610 FORMAT(1H1,17X,16HSIMULATED TIME =,I5,3H TO,I5,10X,
                   A 16HINPUT PROGRAM = ,6A1,10X,
                   B 16HREAL TIME/DATE =,2(1XZ8))
ISN 0240       2611 FORMAT(1H )
ISN 0241       2622 FORMAT(18H                    ,I2,1X,100A1,I2)
ISN 0242       2623 FORMAT(18H A BUFFER           ,I2,1X,100A1,I2)
ISN 0243       2624 FORMAT(18H X BUFFER           ,I2,1X,100A1,I2)
ISN 0244       2625 FORMAT(18H A REGS BUSY        ,I2,1X,100A1,I2)
ISN 0245       2626 FORMAT(18H ABU REGS BUSY      ,I2,1X,100A1,I2)
ISN 0246       2627 FORMAT(18H X REGS BUSY        ,I2,1X,100A1,I2)
ISN 0247       2628 FORMAT(18H C BITS BUSY        ,I2,1X,100A1,I2)
ISN 0248       2629 FORMAT(18H CBU BITS BUSY      ,I2,1X,100A1,I2)
ISN 0249       2630 FORMAT(21X,101HO----+----1----+----2----+----3----+----4----+----5
                   X----+----6----+----7----+----8----+----9----+----0)
ISN 0250       2631 FORMAT(15H A FACILITIES  ,A2,1XI2,1X,100A1,I2)
ISN 0251       2632 FORMAT(15H X FACILITIES  ,A2,1XI2,1X,100A1,I2)
```


L. Conway
Archives

```
ISN 0252     2633 FORMAT(18H MEMORY QUEUE (D) ,I2,1X,100A1,I2)
ISN 0253     2634 FORMAT(18H MEMORY           ,I2,1X,100A1,I2)
ISN 0254     2635 FORMAT(15H                ,A2,1XI2,1X,100A1,I2)
ISN 0255     2636 FORMAT(18H MEMORY QUEUE (I) ,I2,1X,100A1,I2)
ISN 0256     2637 FORMAT(16H BRANCH CONTROL-,2A2,1X,100A1,2H +)
ISN 0257     2638 FORMAT(16H                 ,2A2,1X,100A1,2H +)
ISN 0258     2640 FORMAT(20H DSPX1             IB,1X,100A1,2H +)
ISN 0259     2641 FORMAT(20H                   DO,1X,100A1,2H +)
ISN 0260     2642 FORMAT(20H                    +,1X,100A1,2H +)
ISN 0261     2643 FORMAT(20H DSPX2             IB,1X,100A1,2H +)
ISN 0262     2644 FORMAT(20H DSPA1             IB,1X,100A1,2H +)
ISN 0263     2645 FORMAT(20H DSPA2             IB,1X,100A1,2H +)
ISN 0264     2646 FORMAT(20H PH1                +,1X,100A1,2H +)
ISN 0265     2647 FORMAT(16H SKIP CONTROL- -,2A2,1X,100A1,2H +)
ISN 0266     2648 FORMAT(16H                 ,2A2,1X,100A1,2H +)
ISN 0267          END
```


L. Conway
Archives

```
LEVEL 5 DEC 66                          DATE  YS/360  FORTRAN H                                    DATE  67.140/00.03.55

        COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

ISN 0002          SUBROUTINE STOBUF(CYCLE)
ISN 0003          IMPLICIT INTEGER*2(A-Z)
ISN 0004          COMMON          TIME,          IPAR1,          IPAR2,          IPAR3,
                 A AINPT,         NABUF,         ABUS(50),       XINPT,          NXBUF,
                 B XBUS(50),      IFADD,         IFDST,          IFRTN,          BRXP,
                 C BRAP,          ER(8),         BE(8),          ET(8),          NBBUF,
                 D AHOLDT,        XHOLDT,        AFRCT,          XFRCT,          BOSC,
                 E BNOP,          XEP,           AEP,            PH1(100),       PRINT,
                 F FSTADD,        NODOT,         NOPSC,          NDBUS,          NADSP,
                 G NXDSP
ISN 0005          COMMON/RLS/     FIRST,         NAREGS,         NXREGS,         NABUS,
                 A NXBUS,         STATS,         ACON,           XCON,           AEMP,
                 B XEMP,          MXO,           AFULL(12),      XFULL(12),      AGO(12),
                 C XGO(12),       NAGO,          NXGO,           NATEST,         NXTEST,
                 D NAFAC,         NXFAC,         ABUSYZ,         ABUSY(200),     XBUSYZ,
                 E XBUSY(200),    ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
                 F XSOR(12,200),ADEST(12,200),XDEST(12,200),                     AFAC(12,15),
                 G XFAC(12,15),   AFACSC(4,15,20),ARET,          XFACSC(4,15,20),XRET,
                 H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
                 I ADBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
                 J AFDLY(15),     XFDLY(15),     AFOBUS(15),     XFOBUS(15),     NSLOT,
                 K ABUPSZ,        ABUPS(200),    XBUPS(200),     ABUFUL(200),    XBUFUL(200),
                 L Q(16,16),      SDBA(32,2),    NQBUF,          NQTEST,         NQGO,
                 M QINPT,         QCON,          QEMP,           MBUSY,          MFREE,
                 N LOAD,          MEMDLY,        MEMORY(16),     NBOX,           EAV,
                 O MXTIME,        OUTLVL,        IQ(4,16),       RTN,            LONGBR,
                 P SR(8),         ST(8),         SKXP,           SKAP,           NSBUF,
                 Q APASS(200),    XPASS(200),    OUT(2),         JDB(6),         SSTOP,
                 R MEMCNT(16),    ABOX(15),      ABXBSY(10),     XBOX(15),       XBXBSY(10)
ISN 0006          COMMON/RLS/     LAST
ISN 0007          INTEGER OUT
ISN 0008          REAL MEMDLY,MXTIME
ISN 0009          REAL TIME
ISN 0010          COMMON/OBUF/SPH1(80,100),SABUFF(12,100),SXBUFF(12,100),
                 A SAREG(32,100),SXREG(32,100),SABREG(32,100),SCBIT(24,100),
                 B SCBBIT(24,100),SAFAC(15,100),SXFAC(15,100),SQ(16,100),
                 C SMEM(16,100),SIQ(4,100),SB(34,100),SS(10,100)
ISN 0011          INTEGER*2 CYCLE
                C              FILLS CYCLE POSITION IN OUTPUT BUFFER
ISN 0012          JT=CYCLE
                C
                C              FILL DISP REG,PH1, OUTPUT BUFFER
ISN 0013          DO 5 I=1,80
ISN 0014        5 SPH1(I,JT)=PH1(I)
                C              FILL BRANCH CONTROL OUTPUT BUFFER
ISN 0015          DO 6 I=1,8
ISN 0016          SB(I,JT)=0
ISN 0017          SB(I+8,JT)=0
ISN 0018          SB(I+16,JT)=0
```

300

L. Conway
Archives

```
ISN 0019          IF(ER(I).NE.0) SB(I,JT)=(ER(I)+240)*256
ISN 0021          IF(BE(I).NE.0) SB(I+8,JT)=(BE(I)+240)*256
ISN 0023          IF(ET(I).NE.0) SB(I+16,JT)=(ET(I)+240)*256
ISN 0025        6 CONTINUE
ISN 0026          DO 3 I=25,34
ISN 0027        3 SB(I,JT)=0
ISN 0028          IF(  BRXP.NE.0) SB(25,JT)=(BRXP+240)*256
ISN 0030          IF(  BRAP.NE.0) SB(26,JT)=(BRAP+240)*256
ISN 0032          IF(XHOLDT.NE.0) SB(27,JT)=(XHOLDT+240)*256
ISN 0034          IF(AHOLDT.NE.0) SB(28,JT)=(AHOLDT+240)*256
ISN 0036          IF(XFRCT .NE.0) SB(29,JT)=(XFRCT+240)*256
ISN 0038          IF(AFRCT .NE.0) SB(30,JT)=(AFRCT+240)*256
ISN 0040          IF(  XEP .NE.0) SB(31,JT)=(XEP+240)*256
ISN 0042          IF(  AEP .NE.0) SB(32,JT)=(AEP+240)*256
ISN 0044          IF(  BOSC.NE.0) SB(33,JT)=(BOSC+240)*256
ISN 0046          IF(  BNOP.NE.0) SB(34,JT)=(BNOP+240)*256
                C               FILL SKIP CONTROL OUTPUT BUFFER
ISN 0048          DO 4 I=1,8
ISN 0049          SS(I,JT)=0
ISN 0050          IF(SR(I).NE.0) SS(I,JT)=(SR(I)+240)*256
ISN 0052        4 CONTINUE
ISN 0053          SS(9,JT)=0
ISN 0054          SS(10,JT)=0
ISN 0055          IF(SKXP.NE.0) SS(9,JT)=(SKXP+240)*256
ISN 0057          IF(SKAP.NE.0) SS(10,JT)=(SKAP+240)*256
                C               FILL ABUFF OUTPUT BUFFER
ISN 0059          DO 10 I=1,NABUF
ISN 0060       10 SABUFF(I,JT)=ABUFF(I,1)
                C               FILL XBUFF OUTPUT BUFFER
ISN 0061          DO 11 I=1,NXBUF
ISN 0062       11 SXBUFF(I,JT)=XBUFF(I,1)
                C               FILL A AND X FACILITY OUTPUT BUFFER
ISN 0063          DO 9 I=1,NXFAC
ISN 0064        9 SXFAC(I,JT)=XFACSC(2,I,1)
ISN 0065          DO 19 I=1,NAFAC
ISN 0066          SAFAC(I,JT)=AFACSC(2,I,1)
ISN 0067       19 CONTINUE
                C               FILL Q AND MEM OUTPUT BUFFER
ISN 0068          DO 8 I=1,16
ISN 0069          SQ(I,JT)=Q(I,1)
ISN 0070        8 SMEM(I,JT)=MEMORY(I)
                C               FILL IQ OUTPUT BUFFER
ISN 0071          DO 7 I=1,4
ISN 0072        7 SIQ(I,JT)=IQ(I,1)
                C               FILL REG BUSY OUTPUT BUFFERS
ISN 0073          DO 12 I=1,32
ISN 0074           SAREG(I,JT)=ABUSY(I)
ISN 0075          SABREG(I,JT)=XBUSY(I)
ISN 0076           SXREG(I,JT)=XBUSY(I+32)
ISN 0077       12 CONTINUE
```

301

L. Conway
Archives

```
C               FILL COMPARE BIT BUSY OUTPUT BUFFERS
ISN 0078        DO 13 I=1,24
ISN 0079         SCBIT(I,JT)=XBUSY(I+64)
ISN 0080        SCBBIT(I,JT)=ABUSY(I+64)
ISN 0081     13 CONTINUE
ISN 0082        RETURN
ISN 0083        END
```

302

L. Conway
Archives

```
LEVEL 10 APR. 67                        OS/360  FORTRAN H                        DATE  67.307/18.23.51

          COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

      ISN 0002          SUBROUTINE XXCON
      ISN 0003          IMPLICIT INTEGER*2(A-Z)
      ISN 0004          COMMON        TIME,       IPAR1,      IPAR2,      IPAR3,
                       A AINPT,       NABUF,      ABUS(50),   XINPT,      NXBUF,
                       B XBUS(50),    IFADD,      IFDST,      IFRTN,      BRXP,
                       C BRAP,        ER(8),      BE(8),      ET(8),      NBBUF,
                       D AHOLDT,      XHOLDT,     AFRCT,      XFRCT,      BOSC,
                       E BNOP,        XEP,        AEP,        PH1(100),   PRINT,
                       F FSTADD,      NODOT,      NOPSC,      NOBUS,      NADSP,
                       G NXOSP
      ISN 0005          COMMON/RLS/   FIRST,      NAREGS,     NXREGS,     NABUS,
                       A NXBUS,       STATS,      ACON,       XCON,       AEMP,
                       B XEMP,        MXO,        AFULL(12),  XFULL(12),  AGO(12),
                       C XGO(12),     NAGO,       NXGO,       NATEST,     NXTEST,
                       D NAFAC,       NXFAC,      ABUSYZ,     ABUSY(200), XBUSYZ,
                       E XBUSY(200),  ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
                       F XSOR(12,200),ADEST(12,200),XDEST(12,200),          AFAC(12,15),
                       G XFAC(12,15), AFACSC(4,15,20),ARET,     XFACSC(4,15,20),XRET,
                       H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
                       I AOBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
                       J AFDLY(15),   XFDLY(15),  AFOBUS(15),  XFOBUS(15),  NSLOT,
                       K ABUPSZ,      ABUPS(200), XBUPS(200), ABUFUL(200), XBUFUL(200),
                       L Q(16,16),    SOBA(32,2), NQBUF,      NQTEST,     NQGO,
                       M QINPT,       QCON,       QEMP,       MBUSY,      MFREE,
                       N LOAD,        MEMDLY,     MEMORY(16), NBOX,       EAV,
                       O MXTIME,      OUTLVL,     IQ(4,16),   RTN,        LONGBR,
                       P SR(8),       ST(8),      SKXP,       SKAP,       NSBUF,
                       Q APASS(200),  XPASS(200), OUT(2),     JOB(6),     SSTOP,
                       R MEMCNT(16),  ABOX(15),   ABXBSY(10), XBOX(15),   XBXBSY(10)
      ISN 0006          COMMON/RLS/   LAST
      ISN 0007          INTEGER OUT
      ISN 0008          DIMENSION SORBSY(200),DESBSY(200)
      ISN 0009          REAL MEMDLY,MXTIME
      ISN 0010          REAL TIME
      ISN 0011          CALL CAUSE(XCON,TIME+1.0,0,0,0)
      ISN 0012          CALL CAUSE(XEMP,TIME+0.3,0,0,0)
      ISN 0013          CALL CAUSE(XRET,TIME+0.8,0,0,0)
      ISN 0014          NGO=0
      ISN 0015          QGO=0
      ISN 0016          DO 1 I=1,NXBUF
      ISN 0017        1 XGO(I)=0
                C - - - - - - -IF BNOP STATE, NOP ALL BDS OPS- - - - - - - - - - - - - - -
      ISN 0018          IF(BNOP.NE.1) GO TO 5
      ISN 0020          DO 4 I=1,NXBUF
      ISN 0021          IF(XBUFF(I,12).NE.1) GO TO 4
      ISN 0023          DO 2 K=1,NXREGS
      ISN 0024          XSOR(I,K)=0
      ISN 0025        2 XDEST(I,K)=0
      ISN 0026          DO 3 K=1,NXFAC
```

303

L. Conway
Archives

```
ISN 0027            XFAC(I,K)=0
ISN 0028          3 XOBUS(I,K)=0
ISN 0029            XBUFF(I,2)=0
ISN 0030            XBUFF(I,12)=0
ISN 0031          4 CONTINUE
ISN 0032          5 CONTINUE
                  C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                  C- - - - - - - - IF SKIP TAKEN,NOP ALL STARRED OPS UP TO 1ST EXEC SKIP - -
ISN 0033            DO 80 I=1,NXBUF
ISN 0034            IF(XBUFF(I,13).EQ.0) GO TO 79
ISN 0036            IF(XBUFF(I, 9).EQ.0) GO TO 84
ISN 0038            IF(XBUFF(I,11).EQ.0) GO TO 84
ISN 0040         79 CONTINUE
ISN 0041            IF(XBUFF(I,11).EQ.0) GO TO 80
ISN 0043            IF(XBUFF(I,9).EQ.0) GO TO 80
ISN 0045            DO 81 K=1,NXREGS
ISN 0046            XSOR(I,K)=0
ISN 0047         81 XDEST(I,K)=0
                  C               APPROX WAY TO HANDLE SKIPPED BRANCHES
                  C               LET GO AS UNSUCC BRANCH - NO S/D INTLKS
ISN 0048            IF(XBUFF(I,12).NE.1) GO TO 1081
ISN 0050            XBUFF(I,10)=0
ISN 0051            GO TO 80
ISN 0052       1081 CONTINUE
ISN 0053            DO 82 K=1,NXFAC
ISN 0054            XFAC(I,K)=0
ISN 0055         82 XOBUS(I,K)=0
ISN 0056            XBUFF(I,2)=0
ISN 0057            DO 83 K=9,15
ISN 0058         83 XBUFF(I,K)=0
ISN 0059         80 CONTINUE
ISN 0060         84 CONTINUE
                  C- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                  C               THIS EVENT SCANS XBUFF FOR INST WHICH CAN GO
                  C               SCAN FOR NXGO OUT OF NXTEST
ISN 0061            DO 10 REG=1,NXREGS
ISN 0062            SORBSY(REG)=0
ISN 0063         10 DESBSY(REG)=XBUSY(REG)
ISN 0064            DO 100 INS=1,NXTEST
ISN 0065            IF(XFULL(INS).EQ.0) GO TO 100
ISN 0067            IF(INS.EQ.1) GO TO 21
ISN 0069            DO 11 REG=1,NXREGS
ISN 0070            SORBSY(REG)=SORBSY(REG)+XSOR(INS-1,REG)
ISN 0071         11 DESBSY(REG)=DESBSY(REG)+XDEST(INS-1,REG)
ISN 0072            INSM1=INS-1
ISN 0073            DO 20 I=1,INSM1
                  C               PREV EXIT INTLKS ALL CODE BELOW
ISN 0074            IF(XBUFF(I,14).EQ.1) GO TO 100
                  C               PREV SKIP INTLKS ALL STARRED CODE BELOW
                  C               AND ALL SKIPS BELOW
```

304
L. Conway
Archives

```
ISN 0076            IF(XBUFF(I,13).EQ.0) GO TO 20
ISN 0078            IF((XBUFF(INS,13).EQ.1).OR.(XBUFF(INS,9).EQ.1)) GO TO 100
ISN 0080         20 CONTINUE
ISN 0081         21 CONTINUE
                C                IF EXIT, INTLK AGAINST PREV BRANCHES AND ER
ISN 0082            IF(XBUFF(INS,14).NE.1) GO TO 28
ISN 0084            IF(ER(BRXP).NE.1) GO TO 100
ISN 0086            IF(INS.EQ.1) GO TO 129
ISN 0088            DO 128 I=1,INSM1
ISN 0089            IF(XBUFF(I,12).EQ.1) GO TO 100
ISN 0091        128 CONTINUE
                C                EXIT PART OF OP GOES, MARK GO EXIT.
ISN 0092        129 CONTINUE
ISN 0093            XBUFF(INS,15)=1
ISN 0094         28 CONTINUE
ISN 0095            DO 22 REG=1,NXREGS
ISN 0096            IF((XSOR(INS,REG).EQ.1).AND.(DESBSY(REG).NE.0)) GO TO 100
ISN 0098            IF((XDEST(INS,REG).EQ.1).AND.(DESBSY(REG).NE.0)) GO TO 100
ISN 0100            IF((XDEST(INS,REG).EQ.1).AND.(SORBSY(REG).NE.0)) GO TO 100
ISN 0102         22 CONTINUE
                C                FIND FAC USED
ISN 0103            DO 25 FAC=1,NXFAC
ISN 0104            IF(XFAC(INS,FAC).NE.0) GO TO 26
ISN 0106         25 CONTINUE
                C                NO FAC USED. ISSUE OP
ISN 0107            FAC=0
ISN 0108         26 CONTINUE
                C - - - - - - - - TEST FOR SPECIAL OPS HERE - - - - - - - - - - - - - - -
ISN 0109            SPEC=0
                C                IF L/S TEST AVAIL OF QUEUE
ISN 0110            IF((XSOR(INS,89).NE.1).AND.(XDEST(INS,89).NE.1)) GO TO 27
ISN 0112            SPEC=1
ISN 0113            QPT=QINPT+QGO
ISN 0114            IF(QPT.GT.NQBUF) GO TO 100
ISN 0116         27 CONTINUE
                C                IF BRANCH,INTLK AGAINST PREV BRANCHES AND EHT AVAIL
ISN 0117            IF(XBUFF(INS,12).NE.1) GO TO 29
ISN 0119            IF(ER(BRXP).EQ.1) GO TO 100
                C                IF SHORT BRANCH,TEST LONGBR INTLK
ISN 0121            IF((XBUFF(INS,5).EQ.0).AND.(LONGBR.NE.0)) GO TO 100
ISN 0123            IF(INS.EQ.1) GO TO 231
ISN 0125            DO 230 I=1,INSM1
ISN 0126            IF(XBUFF(I,12).EQ.1) GO TO 100
ISN 0128        230 CONTINUE
ISN 0129        231 SPEC=1
ISN 0130         29 CONTINUE
                C                IF SKIP,INTLK AGAINST PREV NOGO STARRED OPS,SHT AVAILABLE
ISN 0131            IF(XBUFF(INS,13).NE.1) GO TO 132
ISN 0133            IF(SR(SKXP).EQ.1) GO TO 100
ISN 0135            IF(INS.EQ.1) GO TO 131
```


L. Conway
Archives

```
ISN 0137          DO 130 I=1,INSM1
ISN 0138          IF((XBUFF(I,9).EQ.1).AND.(XGO(I).NE.1)) GO TO 100
ISN 0140      130 CONTINUE
ISN 0141      131 SPEC=1
ISN 0142      132 CONTINUE
              C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
              C               IF NORMAL OR SPEC OP AND NGO =NXGO, DO NOT ISSUE
              C               IF REPLACE OR NOP, CAN ISSUE ANYWAY.
ISN 0143          IF(((FAC.NE.0).OR.(SPEC.NE.0)).AND.(NGO.EQ.NXGO)) GO TO 100
              C               IF NO FACS USED GO DIRECTLY TO 95
ISN 0145          IF(FAC.EQ.0) GO TO 95
              C               IF MULT IDENT FAC, GO TO SPEC HANDLING
ISN 0147          IF(XFAC(INS,FAC).GT.1) GO TO 49
              C               CHECK INBUS,FAC SLOT,OUTBUS INTLKS
              C               NO INBUS CONFLICTS IN X
ISN 0149          BOX=XBOX(FAC)
ISN 0150          IF(XBXBSY(BOX).EQ.1) GO TO 100
ISN 0152          DO 30 T=1,NSLOT
ISN 0153          IF((XFSLOT(FAC,T).EQ.1).AND.(XFACSC(1,FAC,T).EQ.1)) GO TO 100
ISN 0155       30 CONTINUE
ISN 0156          OBUS=XFOBUS(FAC)
ISN 0157          DELAY=XFDLY(FAC)
ISN 0158          IF((XOBUS(INS,OBUS).NE.0).AND.(XBUSSC(1,OBUS,DELAY).NE.0))
                 X GO TO 100
ISN 0160          IF((XOBUS(INS,OBUS+1).NE.0).AND.(XBUSSC(1,OBUS+1,DELAY).NE.0).AND.
                 X ((OBUS+1).LE.NXBUS)) GO TO 100
              C               SUCCESS.  MARK GO AND SET SHIFT CELLS
ISN 0162       31 CONTINUE
ISN 0163          XIBBSY(INBUS)=1
ISN 0164          XBXBSY(BOX)=1
ISN 0165          XBUFF1=XBUFF(INS,1)
ISN 0166          DO 32 T=1,NSLOT
ISN 0167          IF(XFSLOT(FAC,T).EQ.0) GO TO 32
ISN 0169          XFACSC(1,FAC,T)=1
ISN 0170          XFACSC(2,FAC,T)=XBUFF1
ISN 0171       32 CONTINUE
ISN 0172          XBUSSC(1,OBUS,DELAY)=XOBUS(INS,OBUS)
ISN 0173          XBUSSC(2,OBUS,DELAY)=XBUFF(INS,1)
ISN 0174          XBUSSC(3,OBUS,DELAY)=XBUFF(INS,2)
ISN 0175          IF(XOBUS(INS,OBUS+1).EQ.0) GO TO 95
ISN 0177          IF((OBUS+1).GT.NXBUS) GO TO 95
ISN 0179          XBUSSC(1,OBUS+1,DELAY)=XOBUS(INS,OBUS+1)
ISN 0180          XBUSSC(2,OBUS+1,DELAY)=XBUFF(INS,1)
ISN 0181          XBUSSC(3,OBUS+1,DELAY)=XBUFF(INS,2)
ISN 0182          GO TO 95
              C               SPEC ROUTINE TO HANDLE MULT IDENT FAC INTLK
ISN 0183       49 CONTINUE
              C               NO INBUS CONFLICTS IN X
ISN 0184          BOX=XBOX(FAC)
ISN 0185          IF(XBXBSY(BOX).EQ.1) GO TO 60
```


L. Conway
Archives

```
ISN 0187            DO 50 T=1,NSLOT
ISN 0188            IF((XFSLOT(FAC,T).EQ.1).AND.(XFACSC(1,FAC,T).EQ.1)) GO TO 60
ISN 0190         50 CONTINUE
ISN 0191            OBUS=XFOBUS(FAC)
ISN 0192            DELAY=XFDLY(FAC)
ISN 0193            IF((XOBUS(INS,OBUS).NE.0).AND.(XBUSSC(1,OBUS,DELAY).NE.0))
                   X GO TO 60
                C                SUCCESS
ISN 0195            DO 51 BUS=1,NXBUS
ISN 0196         51 IF(BUS.NE.OBUS) XOBUS(INS,BUS)=0
ISN 0198            GO TO 31
ISN 0199         60 CONTINUE
ISN 0200            FAC=FAC+1
ISN 0201            IF((FAC.GT.NXFAC).OR.(XFAC(INS,FAC).LE.1)) GO TO 100
ISN 0203            GO TO 49
ISN 0204         95 CONTINUE
ISN 0205            XGO(INS)=1
ISN 0206            IF((XSOR(INS,89).EQ.1).OR.(XDEST(INS,89).EQ.1)) QGO=QGO+1
                C                IF OP USES NO FACILITIES, AND IS NOT SPECIAL OP THEN IT
                C                IS A REPLACE OP, AND GOES WITHOUT INCREMENTING NGO.
ISN 0208            IF((FAC.NE.0).OR.(SPEC.NE.0)) NGO=NGO+1
ISN 0210        100 CONTINUE
                C  - - - - - - - - - - -  EXIT EXECUTION - - - - - - - - - - - - - - - - - -
                C                CHECK FOR NOGO EXITS TO SET XHOLDT
ISN 0211            XHOLDT=0
ISN 0212            DO 200 I=1,NXBUF
ISN 0213            IF((XBUFF(I,14).EQ.1).AND.(XBUFF(I,15).NE.1)) XHOLDT=1
ISN 0215        200 CONTINUE
                C                CHECK FOR GO EXIT,ET
ISN 0216            XFRCT=0
ISN 0217            DO 201 I=1,NXBUF
ISN 0218            IF(XBUFF(I,15).NE.1) GO TO 201
ISN 0220            IF(XBUFF(I,14).NE.1) GO TO 201
ISN 0222            XBUFF(I,14)=0
ISN 0223            XBUFF(I,15)=0
ISN 0224            IF(XBUFF(I,10).EQ.1) GO TO 202
ISN 0226        201 CONTINUE
ISN 0227            GO TO 300
                C                FOUND GO EXIT,ET. NOP AND MARK GO ALL CODE BENEATH IT.
                C                ALSO SET XFRCT.
ISN 0228        202 XFRCT=1
ISN 0229            IF(I.EQ.NXBUF) GO TO 300
ISN 0231            I=I+1
ISN 0232            DO 203 J=I,NXBUF
ISN 0233            XGO(I)=1
ISN 0234            DO 204 K=1,NXREGS
ISN 0235            XSOR(J,K)=0
ISN 0236        204 XDEST(J,K)=0
ISN 0237            DO 205 K=1,10
ISN 0238        205 XOBUS(J,K)=0
```


L. Conway
Archives

```
ISN 0239            XBUFF(I,2)=0
ISN 0240            DO 206 K=9,15
ISN 0241          206 XBUFF(J,K)=0
ISN 0242            DO 207 K=1,NXFAC
ISN 0243            XFAC(J,K)=0
ISN 0244          207 CONTINUE
ISN 0245          203 CONTINUE
ISN 0246          300 CONTINUE
                  C  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                  C                CALL BOSEX TO SAVE .1 BOS CONTROL TRIGGER VALUES.
ISN 0247            CALL BOSEX
                  C- - - - - - - - IF SKIP NOT TAKEN,REMOVE FLAGS FROM ALL OPS THRU 1ST SKIP
ISN 0248            DO 85 I=1,NXBUF
ISN 0249            IF(XBUFF(I,11).EQ.0) XBUFF(I,9)=0
ISN 0251            IF(XBUFF(I,13).EQ.1) GO TO 86
ISN 0253           85 CONTINUE
ISN 0254           86 CONTINUE
                  C- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
ISN 0255            RETURN
ISN 0256            END
```


L. Conway
Archives

```
COMPILER OPTIONS - NAME= MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

ISN 0002          SUBROUTINE XXEMP
ISN 0003          IMPLICIT INTEGER*2(A-Z)
ISN 0004          COMMON       TIME,       IPAR1,       IPAR2,       IPAR3,
                 A AINPT,      NABUF,      ABUS(50),    XINPT,       NXBUF,
                 B XBUS(50),   IFADD,      IFDST,       IFRTN,       BRXP,
                 C BRAP,       ER(8),      BE(8),       ET(8),       NBBUF,
                 D AHOLDT,     XHOLDT,     AERCT,       XERCT,       BOSC,
                 E BNOP,       XEP,        AEP,         PH1(100),    PRINT,
                 F FSTADD,     NODOT,      NOPSC,       NDBUS,       NADSP,
                 G NXDSP
ISN 0005          COMMON/RLS/  FIRST,      NAREGS,      NXREGS,      NABUS,
                 A NXBUS,      STATS,      ACON,        XCON,        AEMP,
                 B XEMP,       MXQ,        AFULL(12),   XFULL(12),   AGO(12),
                 C XGO(12),    NAGO,       NXGO,        NATEST,      NXTEST,
                 D NAFAC,      NXFAC,      ABUSYZ,      ABUSY(200),  XBUSYZ,
                 E XBUSY(200), ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
                 F XSOR(12,200),ADEST(12,200),XDEST(12,200),          AFAC(12,15),
                 G XFAC(12,15), AFACSC(4,15,20),ARET,      XFACSC(4,15,20),XRET,
                 H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
                 I AOBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
                 J AFOLY(15),  XFOLY(15),  AFOBUS(15),  XFOBUS(15),  NSLOT,
                 K ABUPSZ,     ABUPS(200), XBUPS(200),  ABUFUL(200), XBUFUL(200),
                 L Q(16,16),   SDBA(32,2), NQBUF,       NQTEST,      NQGO,
                 M QINPT,      QCON,       QEMP,        MBUSY,       MFREE,
                 N LOAC,       MEMDLY,     MEMORY(16),  NBOX,        EAV,
                 O MXTIME,     OUTLVL,     IQ(4,16),    RTN,         LONGBR,
                 P SR(8),      ST(8),      SKXP,        SKAP,        NSBUF,
                 Q APASS(200), XPASS(200), OUT(2),      JOB(6),      SSTOP,
                 R MEMCNT(16), ABOX(15),   ABXBSY(10),  XBOX(15),    XBXBSY(10)
ISN 0006          COMMON/RLS/  LAST
ISN 0007          INTEGER OUT
ISN 0008          COMMON/TAGS/D(256,70)
ISN 0009          REAL MEMDLY,MXTIME
ISN 0010          REAL TIME
ISN 0011          DO 100 INS=1,NXBUF
ISN 0012        5 IF(XGO(INS).EQ.0) GO TO 100
ISN 0014          IF(XFULL(INS).EQ.0) GO TO 100
                 C              ISSUE INS
                 C - - - - - - - - TEST FOR SPECIAL OPS HERE - - - - - - - - - - - - - - -
                 C              IF L/S SHIP TO QUEUE
ISN 0016          IF((XSCR(INS,89).NE.1).AND.(XDEST(INS,89).NE.1)) GO TO 7
ISN 0018          CALL CAUSE(EAV,TIME+1.0,0,0,0)
ISN 0019          IN=QINPT
ISN 0020          QINPT=QINPT+1
                 C              SET Q LETTER
ISN 0021          Q(IN,1)=XBUFF(INS,1)
                 C              SET Q A
ISN 0022          Q(IN,4)=XBUFF(INS,7)
                 C              SET Q X
```

309

L. Conway
Archives

```
ISN 0023            IF(Q(IN,4).NE.1) Q(IN,5)=1
                C               SET Q EFF ADD
ISN 0025            Q(IN,7)=XBUFF(INS,6)
                C               SET Q BOM
ISN 0026            Q(IN,6)=MOD(Q(IN,7),NBOX)+1
                C               SET Q LOAD
ISN 0027            IF(XSOR(INS,89).EQ.1) Q(IN,2)=1
                C               SET Q STORE
ISN 0029            IF(XDEST(INS,89).EQ.1) Q(IN,3)=1
                C               SET Q DATA VALID FOR X STORE
ISN 0031            IF((Q(IN,5).EQ.1).AND.(Q(IN,3).EQ.1)) Q(IN,9)=1
ISN 0033            IF(Q(IN,2).NE.1) GO TO 88
                C               SET Q DEST FOR LOADS
ISN 0035            DO 8 REG=1,NXREGS
ISN 0036            IF(XDEST(INS,REG).NE.0) Q(IN,15)=REG
ISN 0038          8 CONTINUE
ISN 0039         88 CONTINUE
                C               IF STORE A, GET DATA OR SET WAIT
ISN 0040            IF((Q(IN,3).NE.1).OR.(Q(IN,4).NE.1)) GO TO 7
ISN 0042            IF(SDBA(1,1).NE.1) GO TO 6
                C               DATA AVAIL
ISN 0044            Q(IN,9)=1
ISN 0045            DO 50 I=1,31
ISN 0046            SDBA(I,1)=SDBA(I+1,1)
ISN 0047         50 SDBA(I,2)=SDBA(I+1,2)
ISN 0048            SDBA(32,1)=0
ISN 0049            SDBA(32,2)=0
ISN 0050            GO TO 7
ISN 0051          6 CONTINUE
                C               DATA NOT AVAIL. SET FIRST FREE WAIT BIT
ISN 0052            DO 4 I=1,31
ISN 0053            IF(SDBA(I,2).EQ.1) GO TO 4
ISN 0055            SDBA(I,2)=1
ISN 0056            GO TO 7
ISN 0057          4 CONTINUE
ISN 0058            A=1
ISN 0059            B=20000
ISN 0060            C=102
ISN 0061            CALL TROUBL(A,B,C)
ISN 0062          7 CONTINUE
                C               ISSUE BRANCH OP
ISN 0063            IF(XBUFF(INS,12).NE.1) GO TO 200
                C               IF LONG BRANCH, SET LONGBR=1 TO INTLK SHORT BRANCH NXTCYC
ISN 0065            LONGBR=0
ISN 0066            IF((XBUFF(INS,5).NE.0).AND.(XBUFF(INS,2).NE.138)) LONGBR=1
                C               IF SUCC LONG BRANCH, SET LONGBR=2
ISN 0068            IF((LONGBR.EQ.1).AND.(XBUFF(INS,10).EQ.1))LONGBR=2
ISN 0070        200 CONTINUE
                C               ISSUE SKIP-INCR SKIP POINTER, SET SR
ISN 0071            IF(XBUFF(INS,13).NE.1) GO TO 60
```


L. Conway
Archives

```
ISN 0073            SR(SKXP)=1
ISN 0074            SKXP=SKXP+1
ISN 0075            IF(SKXP.GT.NSBUF) SKXP=1
ISN 0077         60 CONTINUE
                 C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
ISN 0078            OP = XBUFF(INS,2)
ISN 0079            REPL=D(OP,32)
                 C                  FIRST SET BUSY VECTOR
ISN 0080            DO 10 REG=1,NXREGS
                 C                  IS REG A DEST
ISN 0081            IF(XDEST(INS,REG).NE.1) GO TO 10
                 C                  IGNORE STORAGE AS DEST
ISN 0083            IF(REG.EQ.89) GO TO 10
                 C                  DOES XREG HAVE AN XBUREG
ISN 0085            IF(XBUPS(REG).NE.1) GO TO 9
                 C                  CAN THIS OP DO BACK-UP TO FRONT MOVE.
                 C                  I.E.,IS THIS A TO X MOVE OR A COMPARE OP.
ISN 0087            IF(REPL.EQ.0) GO TO 9
                 C                  IS XBUREG FULL
ISN 0089            IF(XBUFUL(REG).NE.1) GO TO 99
ISN 0091            XBUFUL(REG)=0
ISN 0092            ABUSY(REG)=0
ISN 0093            GO TO 10
ISN 0094         99 XPASS(REG)=XBUFF(INS,1)
ISN 0095          9 XBUSY(REG)=XBUFF(INS,1)
ISN 0096         10 CONTINUE
                 C                  REMOVE INS FROM BUFF
ISN 0097            XINPT=XINPT-1
ISN 0098            M=NXBUF-1
ISN 0099            IF(INS.EQ.NXBUF) GO TO 31
ISN 0101            DO 30 I=INS,M
ISN 0102            XGO(I)=XGO(I+1)
ISN 0103            XFULL(I)=XFULL(I+1)
ISN 0104            DO 25 J=1,25
ISN 0105         25 XBUFF(I,J)=XBUFF(I+1,J)
ISN 0106            DO 26 J=1,NXREGS
ISN 0107            XSOR(I,J)=XSOR(I+1,J)
ISN 0108         26 XDEST(I,J)=XDEST(I+1,J)
ISN 0109            DO 27 FAC=1,NXFAC
ISN 0110         27 XFAC(I,FAC)=XFAC(I+1,FAC)
ISN 0111            DO 28 BUS=1,NXBUS
ISN 0112         28 XOBUS(I,BUS)=XOBUS(I+1,BUS)
ISN 0113         30 CONTINUE
ISN 0114         31 CONTINUE
ISN 0115            XGO(NXBUF)=0
ISN 0116            XFULL(NXBUF)=0
ISN 0117            DO 125 J=1,25
ISN 0118        125 XBUFF(NXBUF,J)=0
ISN 0119            DO 126 J=1,NXREGS
ISN 0120            XSOR(NXBUF,J)=0
```


L. Conway
Archives

```
ISN 0121        126 XDEST(NXBUF,J)=0
ISN 0122            DO 127 FAC=1,NXFAC
ISN 0123        127 XFAC(NXBUF,FAC)=0
ISN 0124            DO 128 BUS=1,NXBUS
ISN 0125        128 XOBUS(NXBUF,BUS)=0
ISN 0126            GO TO 5
ISN 0127        100 CONTINUE
ISN 0128            RETURN
ISN 0129            END
```

312

L. Conway
Archives

```
LEVEL 5 DEC 66                          DATE  YS/360  FORTRAN H                          DATE  67.177/10.06.47

          COMPILER OPTIONS - NAME=  MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

     ISN 0002            SUBROUTINE XXRET
     ISN 0003            IMPLICIT INTEGER*2(A-Z)
     ISN 0004            COMMON        TIME,       IPAR1,      IPAR2,      IPAR3,
                        A AINPT,       NABUF,      ABUS(50),   XINPT,      NXBUF,
                        B XBUS(50),    IFADD,      IFDST,      IFRTN,      BRXP,
                        C BRAP,        ER(8),      BE(8),      ET(8),      NBBUF,
                        D AHOLDT,      XHOLDT,     AFRCT,      XFRCT,      BOSC,
                        E BNOP,        XEP,        AEP,        PH1(100),   PRINT,
                        F FSTADD,      NODOT,      NOPSC,      NOBUS,      NADSP,
                        G NXDSP
     ISN 0005            COMMON/RLS/   FIRST,      NAREGS,     NXREGS,     NABUS,
                        A NXBUS,       STATS,      ACON,       XCON,       AEMP,
                        B XEMP,        MXQ,        AFULL(12),  XFULL(12),  AGO(12),
                        C XGO(12),     NAGO,       NXGO,       NATEST,     NXTEST,
                        D NAFAC,       NXFAC,      ABUSYZ,     ABUSY(200), XBUSYZ,
                        E XBUSY(200),  ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
                        F XSOR(12,200),ADEST(12,200),XDEST(12,200),         AFAC(12,15),
                        G XFAC(12,15), AFACSC(4,15,20),ARET,    XFACSC(4,15,20),XRET,
                        H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
                        I AOBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
                        J AFDLY(15),   XFDLY(15),  AFOBUS(15),  XFOBUS(15),  NSLOT,
                        K ABUFSZ,      ABUFS(200), XBUFS(200), ABUFUL(200), XBUFUL(200),
                        L Q(16,16),    SDBA(32,2), NQBUF,      NQTEST,     NQGO,
                        M QINPT,       QCON,       QEMP,       MBUSY,      MERLE,
                        N LOAD,        MEMDLY,     MEMORY(16), NBOX,       EAV,
                        O MXTIME,      OUTLVL,     IQ(4,16),   RTN,        LONGBR,
                        P SR(8),       ST(8),      SKXP,       SKAP,       NSBUF,
                        Q APASS(200),  XPASS(200), OUT(2),     JOB(6),     SSTOP,
                        R MEMCNT(16),  ABOX(15),   ABXBSY(10), XBOX(15),   XBXBSY(10)
     ISN 0006            COMMON/RLS/   LAST
     ISN 0007            INTEGER OUT
     ISN 0008            REAL MEMDLY,MXTIME
     ISN 0009            REAL TIME
                    C              USES QBUS SHIFT CELLS TO BUS TO DESTINATIONS
                    C              PERFORMS ANY EXEC ACTIVITY ASSOC WITH RETURN
                    C              FIRST RESET THE WAITING VECTOR
     ISN 0010            DO 10 BUS=1,NXBUS
                    C              IGNORE L/S BUSES
     ISN 0011            IF((BUS.EQ.5).OR.(BUS.EQ.6)) GO TO 10
     ISN 0013            DEST=XBUSSC(1,BUS,1)
                    C              IF DEST NOT ABU GO HANDLE NORMALLY
     ISN 0014            IF(ABUFS(DEST).NE.1) GO TO 9
                    C              DEST IS ABU. SEE IF CORRESP AREG IS BUSY
                    C              IF SO, RETURN DEST TO IT, ELSE SET ABU BUSY.
     ISN 0015            IF(APASS(DEST).NE.0) GO TO 8
     ISN 0018            ABUFUL(DEST)=1
     ISN 0019            GO TO 10
     ISN 0020          8 ABUSY(DEST)=0
     ISN 0021            APASS(DEST)=0
```

313

L. Conway
Archives

```
ISN 0022          9 XBUSY(DEST)=0
ISN 0023         10 CONTINUE
                 C              PLACE ANY EXEC ACTIVITY HERE - - - - -
                 C - - - - - - - BRANCH EXECUTION - - - - - - - - - - - - - - - - - - - - -
                 C
                 C              CHECK FOR MOVE OF EHT POSITION
ISN 0024               IF((OERX.NE.1).OR.(OXEP.NE.1)) GO TO 200
                 C              MOVE BRXP
ISN 0026               BRXP=BRXP+1
ISN 0027               IF(BRXP.GT.NBBUF) BRXP=1
ISN 0029          200 CONTINUE
ISN 0030               IF((OERA.NE.1).OR.(OAEP.NE.1)) GO TO 201
                 C              MOVE BRAP
ISN 0032               BE(BRAP)=0
ISN 0033               ER(BRAP)=0
ISN 0034               ET(BRAP)=0
ISN 0035               BRAP=BRAP+1
ISN 0036               IF(BRAP.GT.NBBUF) BRAP=1
ISN 0038          201 CONTINUE
                 C
                 C              CHECK FOR RESETTING OF BNOP
ISN 0039               IF((OBNOP.EQ.1).AND.(OXEP.EQ.1)) BNOP=0
                 C
                 C              MAIN BRANCH EXECUTION ROUTINE
ISN 0041               IF(OEXBOS.NE.1) GO TO 250
ISN 0043               IF(BOSSUC.EQ.1) GO TO 240
ISN 0045               IF(OXEP.NE.1) GO TO 230
ISN 0047               IF(OBOSC.EQ.1) GO TO 290
ISN 0049               ER(OBRXP)=1
ISN 0050               ET(OBRXP)=0
ISN 0051               GO TO 290
ISN 0052          230 CONTINUE
ISN 0053               BE(OBRXP)=1
ISN 0054               ET(OBRXP)=0
ISN 0055               GO TO 290
ISN 0056          240 CONTINUE
ISN 0057               ER(OBRXP)=1
ISN 0058               ET(OBRXP)=1
ISN 0059               BNOP=1
ISN 0060               GO TO 290
ISN 0061          250 CONTINUE
ISN 0062               IF(OBEX .NE.1)GO TO 290
ISN 0064               IF(OXEP .NE.1)GO TO 290
ISN 0066               IF(OBOSC.EQ.1)GO TO 290
ISN 0068               ER(OBRXP)=1
ISN 0069               BE(OBRXP)=0
ISN 0070          290 CONTINUE
                 C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                 C              SHIFTS THE SHIFT CELLS
ISN 0071               DO 99 I=1,10
```

314

L. Conway
Archives

```
ISN 0072            XBXBSY(I)=0
ISN 0073         99 XIBBSY(I)=0
ISN 0074            SLOTM1=NSLOT-1
ISN 0075            DO 101 J=1,10
ISN 0076            DO 100 SLOT=1,SLOTM1
ISN 0077            XBUSSC(1,J,SLOT)=XBUSSC(1,J,SLOT+1)
ISN 0078            XBUSSC(2,J,SLOT)=XBUSSC(2,J,SLOT+1)
ISN 0079            XBUSSC(3,J,SLOT)=XBUSSC(3,J,SLOT+1)
ISN 0080        100 CONTINUE
ISN 0081            XBUSSC(1,J,NSLOT)=0
ISN 0082            XBUSSC(2,J,NSLOT)=0
ISN 0083            XBUSSC(3,J,NSLOT)=0
ISN 0084        101 CONTINUE
ISN 0085            DO 103 J=1,NXFAC
ISN 0086            DO 102 SLOT=1,SLOTM1
ISN 0087            XFACSC(1,J,SLOT)=XFACSC(1,J,SLOT+1)
ISN 0088            XFACSC(2,J,SLOT)=XFACSC(2,J,SLOT+1)
ISN 0089        102 CONTINUE
ISN 0090            XFACSC(1,J,NSLOT)=0
ISN 0091            XFACSC(2,J,NSLOT)=0
ISN 0092        103 CONTINUE
ISN 0093            RETURN
C - - - - - - - - - .1 ENTRY TO SET TRIGGERS USED IN BRANCH EXECUTION.- - - - -
ISN 0094            ENTRY BOSEX
C                   PRESERVES .1 VALUES OF VARIOUS BOS CONTROL TRIGGERS
ISN 0095            OLRA=ER(BRXP)
ISN 0096            OLRA=ER(BRAP)
ISN 0097            OBEX=BE(BRXP)
ISN 0098            OBEA=BE(BRAP)
ISN 0099            OETX=ET(BRXP)
ISN 0100            OETA=ET(BRAP)
ISN 0101            OBRXP=BRXP
ISN 0102            OBRAP=BRAP
ISN 0103            OXEP=XEP
ISN 0104            OAEP=AEP
ISN 0105            OBNJP=BNJP
C                   ESTABLISH AND SAVE TRUE BOSC CONDITION
ISN 0106            CBOS=0
ISN 0107            CEXIT=0
ISN 0108            CBOSC=0
ISN 0109            DO 300 I=1,NXBUF
ISN 0110            IF((XBUFF(I,12).EQ.1).AND.(XGO(I).EQ.0)) CBOS=1
ISN 0112            IF(XBUFF(I,14).NE.1) GO TO 300
ISN 0114            IF(CBOS.EQ.1) CBOSC=1
ISN 0116            CEXIT=1
ISN 0117            GO TO 301
ISN 0118        300 CONTINUE
ISN 0119        301 CONTINUE
ISN 0120            OBOSC=0
ISN 0121            IF((CEXIT.EQ.0).AND.((BOSC.EQ.1).OR.(CBOS.EQ.1))) OBOSC=1
```


L. Conway
Archives

```
      [illegible] LINECNT=50,SOURCE,EBCDIC,NOLIST,NODECK,LOAD [illegible]

ISN 0002          SUBROUTINE TSTEP(IEVENT)
ISN 0003          IMPLICIT INTEGER*2(A-Z)
ISN 0004          COMMON [illegible]            IPAR1,        IPAR2,        IPAR3 [illegible]
ISN 0005          REAL TIME [illegible]
ISN 0006          COMMON [illegible]            ISL,          ITL,          LINK(200),
                 A CTIME(200),  NEVENT(200), KOL1(200),   KOL2(200),    KOL3(200)
ISN 0007          REAL CTIME
ISN 0008          INTEGER*2 IEVENT
            C          SUBROUTINE TO STEP EVENTS IN CALENDAR
            C          ITL IS LOCATION OF FIRST EVENT IN CALENDAR
            C          ISL IS LOCATION OF FIRST AVAIL ROW IN CALENDAR
ISN 0009          ID=ITL
ISN 0010          ITL=LINK(ID)
ISN 0011          LINK(ID)=ISL
ISN 0012          ISL=ID
ISN 0013          TIME=CTIME(ID) [illegible]
ISN 0014          IPAR1=KOL1(ID) [illegible]
ISN 0015          IPAR2=KOL2(ID)
ISN 0016          IPAR3=KOL3(ID)
ISN 0017          IEVENT=NEVENT(ID)
ISN 0018          RETURN
ISN 0019          END
```

316

L. Conway
Archives

COMPILER OPTIONS - NAME= MAIN,OPT=02,LINECNT=50,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,NOID

```
ISN 0002            SUBROUTINE TROUBL(START,LSTOP,CODE)
ISN 0003            IMPLICIT INTEGER*2(A-Z)
ISN 0004            COMMON        TIME,       IPAR1,       IPAR2,       IPAR3,
                   A AINPT,       NABUF,      ABUS(50),    XINPT,       NXBUF,
                   B XBUS(50),    IFADD,      IFDST,       IFRTN,       BRXP,
                   C BRAP,        ER(8),      BE(8),       ET(8),       NBBUF,
                   D AHOLDT,      XHOLDT,     AFRCT,       XFRCT,       BOSC,
                   E BNOP,        XEP,        AEP,         PH1(100),    PRINT,
                   F FSTADD,      NODOT,      NOPSC,       NDBUS,       NADSP,
                   G NXDSP
ISN 0005            COMMON/RLS/   FIRST,      NAREGS,      NXREGS,      NABUS,
                   A NXBUS,       STATS,      ACON,        XCON,        AEMP,
                   B XEMP,        MXO,        AFULL(12),   XFULL(12),   AGO(12),
                   C XGO(12),     NAGO,       NXGO,        NATEST,      NXTEST,
                   D NAFAC,       NXFAC,      ABUSYZ,      ABUSY(200),  XBUSYZ,
                   E XBUSY(200),  ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
                   F XSOR(12,200),ADEST(12,200),XDEST(12,200),          AFAC(12,15),
                   G XFAC(12,15), AFACSC(4,15,20),ARET,      XFACSC(4,15,20),XRET,
                   H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
                   I AOBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
                   J AFDLY(15),   XFDLY(15),  AFOBUS(15),  XFOBUS(15),  NSLOT,
                   K ABUPSZ,      ABUPS(200), XBUPS(200), ABUFUL(200), XBUFUL(200),
                   L Q(16,16),    SDBA(32,2), NQBUF,      NQTEST,      NQGO,
                   M QINPT,       QCON,       QEMP,       MBUSY,       MFREE,
                   N LOAD,        MEMDLY,     MEMORY(16), NBOX,        EAV,
                   O MXTIME,      OUTLVL,     IQ(4,16),   RTN,         LONGBR,
                   P SR(8),       ST(8),      SKXP,       SKAP,        NSBUF,
                   Q APASS(200),  XPASS(200), OUT(2),     JOB(6),      SSTOP,
                   R MEMCNT(16),  ABOX(15),   ABXBSY(10), XBOX(15),    XBXBSY(10)
ISN 0006            COMMON/RLS/   LAST
ISN 0007            INTEGER OUT
ISN 0008            REAL MEMDLY,MXTIME
ISN 0009            REAL TIME
ISN 0010            INTEGER*2 START,LSTOP,CODE
ISN 0011            INTEGER L
ISN 0012            DIMENSION SAV(2)
ISN 0013            EQUIVALENCE(SAV(1),FIRST)
ISN 0014            LCODE=CODE
ISN 0015            END=LSTOP
ISN 0016            WRITE(6,100)TIME
ISN 0017            WRITE(6,101) LCODE
ISN 0018            CALL TMTU(OUT(1))
ISN 0019            WRITE(6,3333) OUT(1),OUT(2)
ISN 0020            IF(END.EQ.1) RETURN
ISN 0022            END=END-7
ISN 0023            DO 529 K=START,END,8
ISN 0024            K9=K+7
ISN 0025            DO 528 M=K,K9
ISN 0026            IF(SAV(M).NE.0) GO TO 527
```

317

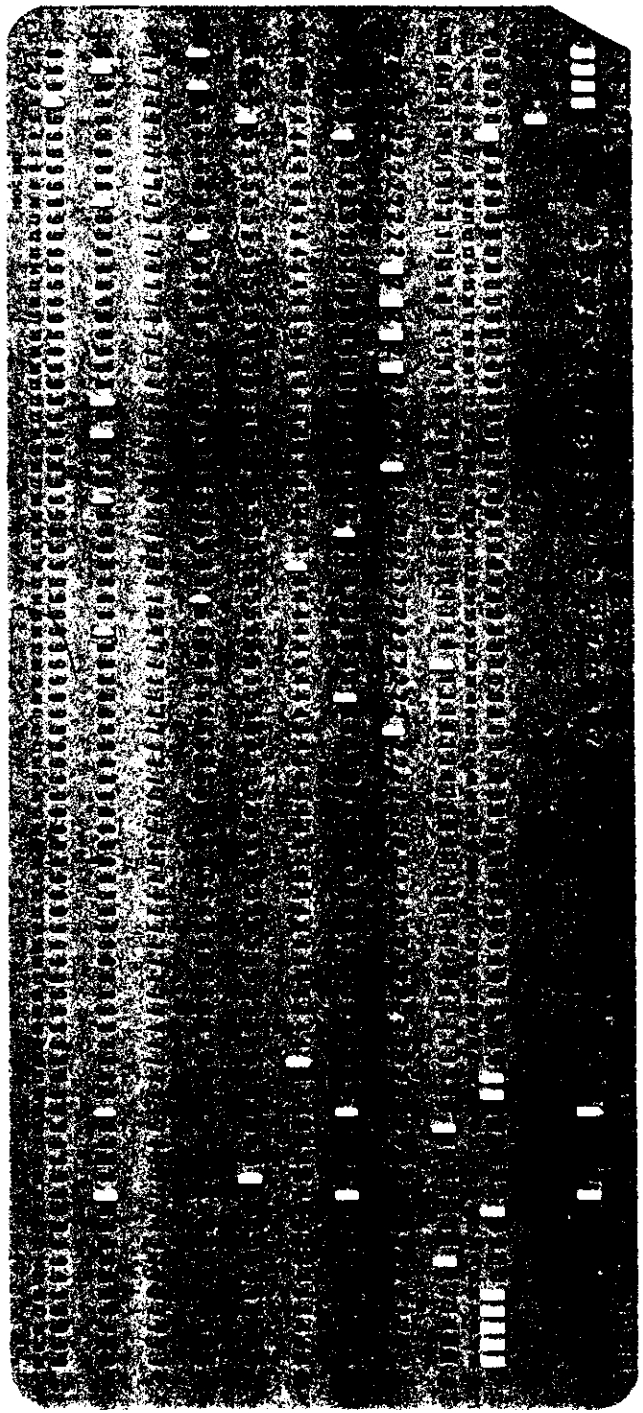
L. Conway Archives

```
ISN 0028          528 CONTINUE
ISN 0029              GO TO 529
ISN 0030          527 CONTINUE
ISN 0031              L=K
ISN 0032              WRITE(6,550)L,(SAV(J),J=K,K9)
ISN 0033          529 CONTINUE
ISN 0034              RETURN
ISN 0035          100 FORMAT(7H TIME =,F8.2)
ISN 0036          101 FORMAT(7H CODE =,I8)
ISN 0037          550 FORMAT(1X I6, 8(2X I8, 4X ))
ISN 0038         3333 FORMAT(19H TIME/DATE OF RUN =,2(1XZ8))
ISN 0039              END
```

318
L. Conway
Archives

```
      SUBROUTINE CAUSE(IEV,T,IP1,IP2,IP3)
      IMPLICIT INTEGER*2(A-Z)
      COMMON/CALEND/ ITL, ISL, LINK(200),
     1 CTIME(200), NEVENT(200), KOL1(200), KOL2(200), KOL3(200)
      REAL CTIME
      INTEGER*2 IEV,IP1,IP2,IP3
C                CAUSE ENTERS EVENTS ONTO CALENDAR
C                ITL IS LOCATION OF FIRST EVENT IN CALENDAR
C                ISL IS LOCATION OF FIRST AVAIL ROW IN CALENDAR
      NEXT=ITL
      GO TO 20
C                LOOP UNTIL GIVEN TIME IS LESS THAN NEXT ENTRY IN CALENDAR
   10 LAST=NEXT
      NEXT=LINK(NEXT)
   20 IF(T.GT.CTIME(NEXT))GO TO 10
      ID=ISL
      ISL=LINK(ISL)
      LINK(ID)=NEXT
C                SEE IF THIS EVENT WILL BE THE FIRST IN THE LIST
      IF(NEXT.EQ.ITL) GO TO 40
      LINK(LAST)=ID
   30 CTIME(ID)=T
      NEVENT(ID)=IEV
      KOL1(ID)=IP1
      KOL2(ID)=IP2
      KOL3(ID)=IP3
      RETURN
   40 ITL=ID
      GO TO 30
      END
```

L. Conway
Archives



320
L. Conway
Archives

MM-MS 3/0 F Correct

DTF FHF (best)

exp Mtx Mpy

- ACS-1 MPM SIMULATION PROGRAM - - -

(5) INPUT PROGRAM FOR THIS RUN = MM-MS

TIME/DATE OF RUN = 5A63CE76 0067271F

MACHINE PARAMETERS FOR THIS RUN - - -

(1) NUMBER OF A BUFFERS = 8 (4) NUMBER OF X BUFFERS = ~~32~~ 2 (7) NUMBER OF Q BUFFERS = 8

(2) NUMBER A OPS TESTED = ~~8~~ 6 (5) NUMBER X OPS TESTED = ~~32~~ 2 (8) NUMBER Q OPS TESTED = 8

(3) MAX A OPS ISS/CYCLE = ~~32~~ 2 (6) MAX X OPS ISS/CYCLE = ~~32~~ 2 (9) MAX Q OPS ISS/CYCLE = 2

(9) MINIMUN Q-MEM DELAY = 5.0

(10) NUMBER OF BOMS = 8

(11) NUMBER BRANCH REGS = 3 (12) NUMBER OF SKIP REGS = 4 (13) SIZE OF DO TABLE = 6

(14) NUMBER OF PSC REGS = ~~8~~

(15) NUMBER DISP BUSES = ~~(2)~~ 1

(16) MAX A OPS DSP/CYCLE = ~~4~~ 3 (17) MAX X OPS DSP/CYCLE = ~~4~~ 2

A FACILITIES - -	FA1	FA2	FM	FD	IA	IM	ID	C	L	S	?
REP TIME =	1	1	~~12~~	~~18~~	1	2	10	1	1	1	1
DELAY TIME =	3	4	~~34~~	~~210~~	2	5	15	1	~~2~~	~~2~~	~~2~~
INBUS =	2	1	3	1	1	2	2	1	2	3	1
BOX =	1	1	3	~~3~~	2	~~3~~	~~3~~	~~1~~	~~2~~	~~2~~	~~2~~
OUTBUS =	2	1	4	3	2	4	4	6	1	3	7

X FACILITIES - -	EA1	EA2	L	S	M	D	XA	C	SP
REP TIME =	1	1	1	1	2	8	1	1	1
DELAY TIME =	1	1	1	1	4	8	1	1	1
BOX =	1	2	3	~~5~~	~~5~~	~~3~~	~~3~~	~~3~~	~~3~~
OUTBUS =	5	6	1	3	2	2	7	10	8

321

L. Conway Archives

Figure 2-3. The Parameter Card Format

PARAMETER	MIN	TYP	MAX	COLS
(0) JØBNAME				1-6
(1) NABUF	1	8	12	9-10
(2) NATEST	1	8	NABUF	11-12
(3) NAGØ	1	3	3	13-14
(4) NXBUF	1	3	12	15-16
(5) NXTEST	1	3	NXBUF	17-18
(6) NXGØ	1	3	3	19-20
(7) NQBUF	1	8	16	21-22
(8) NQTEST	1	8	16	23-24
(9) NQGØ	1	2	NBØX	25-26
(10) NBØX	1	8	16	27-28
(11) NBBUF	1	3	8	29-30
(12) NSBUF	1	4	8	31-32
(13) NØDØT	1	6	16	33-34
(14) NØPSC	0	8	8	35-36
(15) NDBUS	1	2	2	37-38
(16) NADSP	1	4	NABUF	39-40
(17) NXDSP	1	3	NXBUF	41-42
(19) MXTIME		300.0		60-66 (F7.1)
(18) MEMDLY	2.0	5.0		68-71 (F4.1)
ØUTLVL	0	1	3	73-74
FSTADD	0	0		76-80

322
L. Conway Archives

MM-MS · 3/8 F

Correct
DTF FHF (test)
exp Mtx Mpy

- ACS-1 MPM SIMULATION PROGRAM - - -

INPUT PROGRAM FOR THIS RUN = MM-MS

TIME/DATE OF RUN = 5A63CE76 0067271F

MACHINE PARAMETERS FOR THIS RUN - - -

NUMBER OF A BUFFERS = 8 6 NUMBER OF X BUFFERS = ~~3~~ 2 NUMBER OF Q BUFFERS = 8

NUMBER A OPS TESTED = ~~8~~ 6 NUMBER X OPS TESTED = ~~3~~ 2 NUMBER Q OPS TESTED = 8

MAX A OPS ISS/CYCLE = ~~3~~ 2 MAX X OPS ISS/CYCLE = ~~3~~ 2 MAX Q OPS ISS/CYCLE = 2

MINIMUM Q-MEM DELAY = 5.0

NUMBER OF BOMS = 8

NUMBER BRANCH REGS = 3 NUMBER OF SKIP REGS = 4 SIZE OF DO TABLE = 6

NUMBER OF PSC REGS = ~~8~~

NUMBER DISP BUSES = ~~4~~ 1

MAX A OPS DSP/CYCLE = ~~4~~ 3 MAX X OPS DSP/CYCLE = ~~4~~ 2

A FACILITIES - -	FA1	~~FA2~~	FM	FD	IA	IM	ID	C	L	S	?
REP TIME =	1	~~1~~	~~1~~ 2	~~1~~ 6	1	2	10	1	1	1	1
DELAY TIME =	3	~~4~~	~~3~~ 4	~~9~~ 10	2	5	15	1	2	2	2
INBUS =	2	~~[illegible]~~	3	1	1	2	2	1	2	3	1
BOX =	1	~~[illegible]~~	3	3	2	3	3	1	2	2	2
OUTBUS =	2	~~1~~	4	3	2	4	4	6	1	3	7

X FACILITIES - -	EA1	EA2	L	S	M	D	XA	C	SP
REP TIME =	1	1	1	1	2	8	1	1	1
DELAY TIME =	1	1	1	1	4	8	1	1	1
BOX =	1	2	3	3	3	3	3	3	3
OUTBUS =	5	6	1	3	2	2	7	10	8

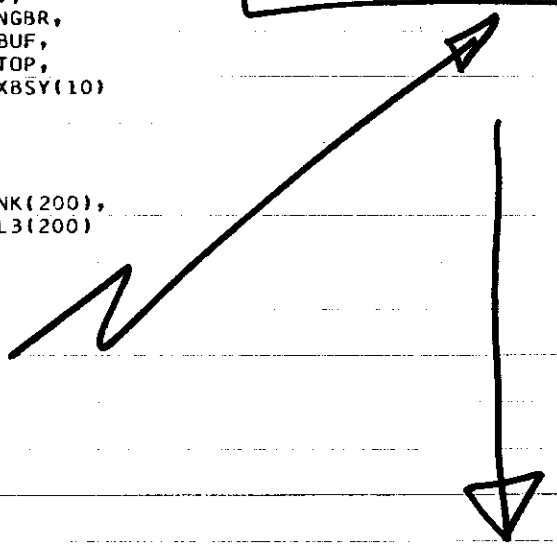
323

L. Conway
Archives

```
LEVEL 14 ( 1 JUN 67)                    OS/360  FORTRAN H                    DATE  68.089/09.18.20

          COMPILER OPTIONS - NAME=  MAIN,OPT=00,LINECNT=56,SOURCE,EBCDIC,NOLIST,DECK,LOAD,MAP,NOEDIT,ID,NOXREF
     ISN 0002            SUBROUTINE INIT
     ISN 0003            IMPLICIT INTEGER*2(A-Z)
     ISN 0004            DIMENSION COM(300)
     ISN 0005            DIMENSION SAV(20000)
     ISN 0006            COMMON       TIME,       IPAR1,       IPAR2,       IPAR3,
                        A AINPT,      NABUF,      ABUS(50),    XINPT,       NXBUF,
                        B XBUS(50),   IFADD,      IFDST,       IFRTN,       BRXP,
                        C BRAP,       ER(8),      BE(8),       ET(8),       NBBUF,
                        D AHOLDT,     XHOLDT,     AFRCT,       XFRCT,       BOSC,
                        E BNOP,       XEP,        AEP,         PH1(100),    PRINT,
                        F FSTADD,     NODOT,      NOPSC,       NDBUS,       NADSP,
                        G NXDSP
     ISN 0007            COMMON/RLS/  FIRST,      NAREGS,      NXREGS,      NABUS,
                        A NXBUS,      STATS,      ACON,        XCON,        AEMP,
                        B XEMP,       MXO,        AFULL(12),   XFULL(12),   AGO(12),
                        C XGO(12),    NAGO,       NXGO,        NATEST,      NXTEST,
                        D NAFAC,      NXFAC,      ABUSYZ,      ABUSY(200),  XBUSYZ,
                        E XBUSY(200), ABUFF(12,100),XBUFF(12,100),ASOR(12,200),
                        F XSOR(12,200),ADEST(12,200),XDEST(12,200),          AFAC(12,15),
                        G XFAC(12,15), AFACSC(4,15,20),ARET,      XFACSC(4,15,20),XRET,
                        H ABUSSC(4,10,20),AIBBSY(10),XBUSSC(4,10,20),XIBBSY(10),XFIBUS(15),
                        I AOBUS(12,10),XOBUS(12,10),AFSLOT(15,20),XFSLOT(15,20),AFIBUS(15),
                        J AFDLY(15),  XFDLY(15),  AFOBUS(15),  XFOBUS(15),  NSLOT,
                        K ABUPSZ,     ABUPS(200), XBUPS(200),  ABUFUL(200), XBUFUL(200),
                        L Q(16,16),   SDBA(32,2),  NQBUF,      NQTEST,      NQGO,
                        M QINPT,      QCON,       QEMP,        MBUSY,       MFREE,
                        N LOAD,       MEMDLY,     MEMORY(16),  NBOX,        EAV,
                        O MXTIME,     OUTLVL,     IQ(4,16),    RTN,         LONGBR,
                        P SR(8),      ST(8),      SKXP,        SKAP,        NSBUF,
                        Q APASS(200), XPASS(200), OUT(2),      JOB(6),      SSTOP,
                        R MEMCNT(16), ABOX(15),   ABXBSY(10),  XBOX(15),    XBXBSY(10)
     ISN 0008            COMMON/RLS/  LAST
     ISN 0009            INTEGER OUT
     ISN 0010            REAL MEMDLY,MXTIME
     ISN 0011            REAL TIME
     ISN 0012            COMMON /CALNDR/              ISL,         ITL,         LINK(200),
                        A CTIME(200), NEVENT(200), KOL1(200),  KOL2(200),   KOL3(200)
     ISN 0013            REAL CTIME
     ISN 0014            REAL X
     ISN 0015            INTEGER I
     ISN 0016            COMMON/TAGS/D(256,70)
     ISN 0017            EQUIVALENCE(COM(1),TIME),(X,CTIME(1))
     ISN 0018            EQUIVALENCE(SAV(1),FIRST)
                    C
                    C
                    C                ZERO ALL COMMON
     ISN 0019               DO 520 I=1,300
     ISN 0020           520 COM(I)=0
     ISN 0021               DO 525 I=1,20000
     ISN 0022           525 SAV(I)=0
     ISN 0023           526 CONTINUE
                    C
                    C
                    C                INITIALIZE THE CALENDAR
```

EXAMPLE OF INIT MODIFICATIONS TO CHANGE FACILITY STRUCTURE:



324
L. Conway Archives

```
ISN 0024            DO 92 ITL=2,199
ISN 0025         92 LINK(ITL)=ITL+1
ISN 0026            ISL=2
ISN 0027            ITL=1
ISN 0028            X=1.0E30
ISN 0029            TIME=0.0
                 C
                 C
                 C              INITIALIZE THE EVENT NUMBERS
ISN 0030            STATS=1
ISN 0031            MXO=2
ISN 0032            ACON=3
ISN 0033            XCON=4
ISN 0034            AEMP=5
ISN 0035            XEMP=6
ISN 0036            ARET=7
ISN 0037            XRET=8
ISN 0038            EAV=9
ISN 0039            QCON=10
ISN 0040            QEMP=11
ISN 0041            MBUSY=12
ISN 0042            MFREE=13
ISN 0043            LOAD=14
ISN 0044            RTN=15
                 C
                 C
                 C              SET UP STARTING EVENTS
ISN 0045            CALL CAUSE(STATS,TIME+0.0,0,0,0)
ISN 0046            CALL CAUSE( ACON,TIME+0.1,0,0,0)
ISN 0047            CALL CAUSE( XCON,TIME+0.1,0,0,0)
ISN 0048            CALL CAUSE( QCON,TIME+0.1,0,0,0)
ISN 0049            CALL CAUSE( MXO ,TIME+0.6,0,0,0)
                 C
                 C
                 C              INITIALIZE THE MACHINE PARAMETERS
ISN 0050            BRXP=1
ISN 0051            BRAP=1
ISN 0052            SKXP=1
ISN 0053            SKAP=1
ISN 0054            NAREGS=90
ISN 0055            NXREGS=90
ISN 0056            AINPT=1
ISN 0057            QINPT=1
ISN 0058            XINPT=1
ISN 0059            DO 50 I=1,32
ISN 0060            ABUPS(I)=1
ISN 0061         50 XBUPS(I)=0
ISN 0062            DO 51 I=33,89
ISN 0063            ABUPS(I)=0
ISN 0064         51 XBUPS(I)=1
ISN 0065            NSLOT=15
                 C              INITIALIZE AFAC TABLES
ISN 0066            NABUS=6
ISN 0067            NAFAC=10
```

325

L. Conway
Archives

```
ISN 0068            DO 10 I=1,10
ISN 0069         10 AFSLOT(I,3)=1
ISN 0070            DO 9 J=4,9
ISN 0071          9 AFSLOT(4,J)=1
ISN 0072            AFSLOT(6,4)=1
ISN 0073            DO 8 J=4,12
ISN 0074          8 AFSLOT(7,J)=1
ISN 0075            AFDLY(1)=3
ISN 0076            AFDLY(2)=4
ISN 0077            AFDLY(3)=3
ISN 0078            AFDLY(4)=9
ISN 0079            AFDLY(5)=2
ISN 0080            AFDLY(6)=5
ISN 0081            AFDLY(7)=15
ISN 0082            AFDLY(8)=1
ISN 0083            AFDLY(9)=1
ISN 0084             AFDLY(10)=1
ISN 0085            AFIBUS(1)=2
ISN 0086            AFIBUS(2)=1
ISN 0087            AFIBUS(3)=3
ISN 0088            AFIBUS(4)=1
ISN 0089            AFIBUS(5)=1
ISN 0090            AFIBUS(6)=2
ISN 0091            AFIBUS(7)=2
ISN 0092            AFIBUS(8)=1
ISN 0093            AFIBUS(9)=2
ISN 0094            AFIBUS(10)=3
ISN 0095            AFOBUS(1)=2
ISN 0096            AFOBUS(2)=1
ISN 0097            AFOBUS(3)=4
ISN 0098            AFOBUS(4)=3
ISN 0099            AFOBUS(5)=2
ISN 0100            AFOBUS(6)=4
ISN 0101            AFOBUS(7)=4
ISN 0102            AFOBUS(8)=6
ISN 0103            AFOBUS(9)=1
ISN 0104            AFOBUS(10)=3
ISN 0105            ABOX(1)=1
ISN 0106            ABOX(2)=2
ISN 0107            ABOX(3)=3
ISN 0108            ABOX(4)=4
ISN 0109            ABOX(5)=2
ISN 0110            ABOX(6)=4
ISN 0111            ABOX(7)=4
ISN 0112            ABOX(8)=5
ISN 0113            ABOX(9)=6
ISN 0114            ABOX(10)=7
                C           INITIALIZE XFAC TABLES
ISN 0115            NXBUS=10
ISN 0116            NXFAC=9
ISN 0117            DO 11 I=1,9
ISN 0118         11 XFSLOT(I,2)=1
ISN 0119            XFSLOT(5,3)=1
ISN 0120            DO 12 I=3,9
```


L. Conway Archives

IBM FORM NO. 413

```
ISN 0121        12 XFSLOT(6,I)=1
ISN 0122           XFDLY(1)=1
ISN 0123           XFDLY(2)=1
ISN 0124           XFDLY(3)=1
ISN 0125           XFDLY(4)=1
ISN 0126           XFDLY(5)=4
ISN 0127           XFDLY(6)=8
ISN 0128           XFDLY(7)=1
ISN 0129           XFDLY(8)=1
ISN 0130           XFDLY(9)=1
ISN 0131           XFOBUS(1)=5
ISN 0132           XFOBUS(2)=6
ISN 0133           XFOBUS(3)=1
ISN 0134           XFOBUS(4)=3
ISN 0135           XFOBUS(5)=2
ISN 0136           XFOBUS(6)=2
ISN 0137           XFOBUS(7)=7
ISN 0138           XFOBUS(8)=10
ISN 0139           XFOBUS(9)=8
ISN 0140           XBOX(1)=1
ISN 0141           XBOX(2)=2
ISN 0142           XBOX(3)=3
ISN 0143           XBOX(4)=4
ISN 0144           XBOX(5)=5
ISN 0145           XBOX(6)=5
ISN 0146           XBOX(7)=6
ISN 0147           XBOX(8)=7
ISN 0148           XBOX(9)=8
ISN 0149           NAFAC=11
ISN 0150           NABUS=7
ISN 0151           AFDLY(11)=1
ISN 0152           AFIBUS(11)=1
ISN 0153           AFOBUS(11)=7
ISN 0154           ABOX(11)=8
ISN 0155           AFSLOT(11,3)=1
ISN 0156           D(39,1)=1
ISN 0157           D(39,2)=1
ISN 0158           D(39,11)=1
ISN 0159           D(39,13)=1
ISN 0160           D(39,17)=1
ISN 0161           D(39,30)=0
ISN 0162           D(39,32)=1
ISN 0163           D(39,66)=1
ISN 0164           AFDLY(1)=4
ISN 0165           AFDLY(3)=4
ISN 0166           AFDLY(4)=15
ISN 0167           AFSLOT(3,4)=1
ISN 0168           DO 7 J=4,15
ISN 0169           AFSLOT(4,J)=1
ISN 0170           AFSLOT(7,J)=1
ISN 0171         7 CONTINUE
ISN 0172           NSLOT=18
ISN 0173           RETURN
ISN 0174           END
```

327

L. Conway Archives

AFSLOT(3,4) = 1
AFSLOT(4,10) = 1
AFDLY(3) = 4
AFDLY(4) = 10
AFDLY(9) = 2
AFDLY(10) = 2
AFDLY(11) = 2
ABOX(4) = 3
ABOX(6) = 3
ABOX(7) = 3
ABOX(8) = 1
ABOX(9) = 2
ABOX(10) = 2
ABOX(11) = 2
XBOX(4) = 3
XBOX(5) = 3
XBOX(6) = 3
XBOX(7) = 3
XBOX(8) = 3
XBOX(9) = 3

```
D0 991 I=56,66
D(I,56)=1
D(I,57)=0
991 CONTINUE
```

to get rid of FA2 had to change OP DECODE TAGS

A FACILITIES:

REP TIME → AFSLOT(I,J)
DELAY TIME → AFDLY(I)
INBUS → AFIBUS(I)
BOX → ABOX(I)
OUTBUS → AFOBUS(I)

X FACILITIES:

REP TIME → XFSLOT(I,J)
DELAY TIME → XFDLY(I)
BOX → XBOX(I)
OUTBUS → XFOBUS(I)

IBM FORM NO. 413

Date: November 29, 1967
From (location & U.S. mail address): Advanced Computing Systems, Menlo Park, California
Dept. & Bldg.: 988/031
Telephone Ext.:

IBM

Subject: Cover Letter for Preliminary Distribution of Logical Design Memorandum

Reference:

To: Mr. S. F. Anderson
Mr. B. O. Beebe
Dr. C. V. Freiman
Mr. M. E. Homan
Mr. B. J. Mooney
Mr. R. J. Robelen
Dr. H. Schorr
Dr. E. H. Sussenguth
Mr. W. P. Wissick

A memorandum describing basic ACS logical design conventions is enclosed.

On joining ACS engineering, I found that there was no single convenient source of this information. Some of the information was not documented in any available references.

Since most of the designers use different notations and conventions, it proved to be a surprisingly time consuming and confusing process to learn the precise details of this very simple basic material. Many of the designers related to me that they had had similar initial experiences.

At that time I made some notes for my own personal use. I have since formed these into a memorandum in the hope that it might prove useful to other newcomers to ACS engineering. It might also be useful to members of other ACS departments.

If you have any comments, criticisms, or discover any errors needing correction, please contact me about them. I will then be able to get the memorandum into shape so that it might be useful during the coming expansion of Dept. 988.

L. Conway

L. Conway

LC:aw

328

L. Conway Archives

November 29, 1967
Advanced Computing Systems
Menlo Park, California
988/031

Subject: ACS Logical Design Conventions: A Guide for the Novice

References:
1. ACS Circuit Manual, February 23, 1967.
2. ACS Packaging Manual, July, 1967.
3. DRKS User's Manual, R. T. Blosk, December 5, 1966.
4. McCluskey and Bartee, A Survey of Switching Circuit Theory, McGraw-Hill, 1962.

To: FILE

L. Conway

L. Conway

LC:aw

L. Conway Archives

CONTENTS

L. Conway
Archives

Introduction:

This memorandum describes the various rules and conventions for ACS logical design. The material presented is elementary in nature, but is basic to all ACS logical design.

A description is given of the logical functions of the ACS circuits available to the designer and of the various rules governing the use of these circuits in logical design. A number of different notations are in current use for writing the logical equations for these circuits and for drawing the diagrams of logical circuitry. Some of these different notations are illustrated and explained. Elementary logical design--the transformation from equations to circuits--is briefly described.

If we were designing in AND-OR logic with few restrictions, this memorandum would be unnecessary. However, we are usually designing with NOR-NOR or NOR-OR logic. The physical properties of the circuits force a number of restrictions in addition to simple fan-in and fan-out rules. The fact that designs eventually input a Design Record Keeping System (DRKS) has produced additional conventions and design notation.

These factors have led different designers to use different conventions for writing logical equations and drawing logic circuit diagrams, and to use different logical design techniques. It is true that at the time designs are input into DRKS, they all will be described in the same formal system. However, up to that time most designs will exist in the form of equations and diagrams in the "shorthand" of the originating designer. The newcomer may therefore become confused when attempting to decipher the designs of different engineers until he fully understands the fundamentals from which their different "shorthand" techniques originated.

These fundamentals are presented in this memorandum in the hope that they may assist the newcomer to ACS engineering in his first design efforts and serve as a reference for those outside of engineering who may wish to study some particular logical design in detail.

The newcomer should also study the listed references before undertaking any serious design. This memorandum was formulated from these references, but does not attempt to cover many important topics contained in them. Of particular importance is the information on circuit delays in the ACS Circuit Manual and information on wiring rules in both the ACS Circuit Manual and the ACS Packaging Manual. The DRKS User's Manual specifies the final form in which designs are to be placed.

L. Conway
Archives

The ACS Logical Circuits:

This section describes the logical functions of the circuits and connections available to the ACS logical designer. Truth tables and equations are given describing the logical functions. The various conventions, restrictions, and limitations of each circuit are listed.

The truth tables use 0 and 1 as symbols, and these are related to the actual physical voltages in the circuits as follows: 1 symbolizes positive (or ground), and 0 symbolizes negative voltages.

The Current Switch:

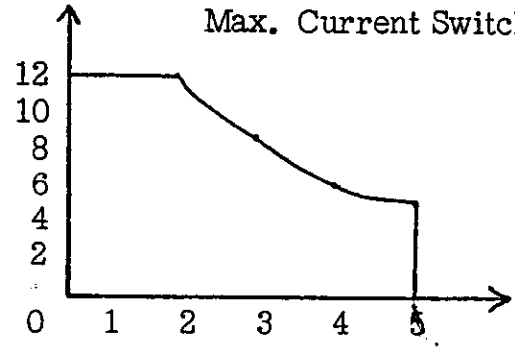
A	B	X	Y
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

$X = \bar{A} \cdot \bar{B}$

$Y = A + B$

Note the significance of the positions in the circuit symbol of the outputs X and Y. The top output X is the NOR of the inputs, and is often called the "out of phase" output. The bottom output Y is the OR of the inputs and is often called the "in phase" output. Note that $Y = \bar{X}$.

Fan-in: Current switch inputs are outputs of emitter followers or emitter follower dot circuits (see description of e.f. dot later in this section). The maximum number of inputs for a given current switch is a function of the maximum fan-in of those e.f. dot circuits forming the inputs. This function is as follows:

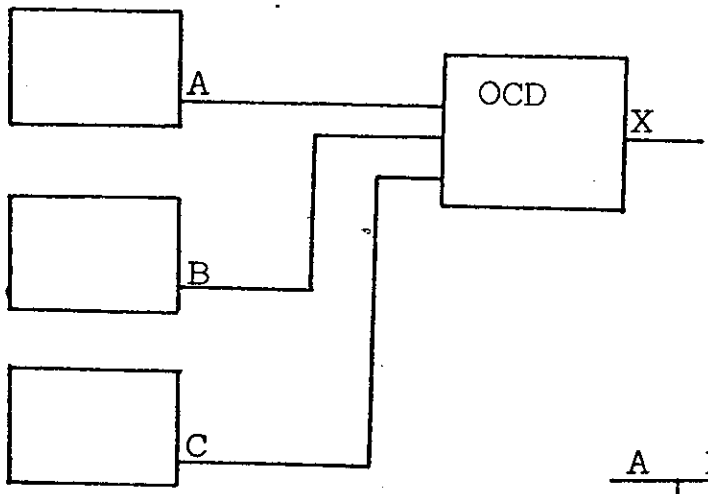


- 332
L. Conway Archives

For example, if the e.f. dots feeding a current switch had no more than two inputs each, then the current switch would have a maximum fan-in of 12. However if one of the e.f. dots had a fan-in of five, then the current switch would have a maximum fan-in of five.

Fan-out: The outputs always pass through emitter followers. The fan-out is thus determined by the fan-out of the emitter followers. The maximum fan-out of the emitter follower (emitter follower dot) is 12. See emitter follower dot description later in this section.

The Orthogonal Collector Dot:



$X = A \cdot B \cdot C$

Orthogonality Restriction:

No two inputs may be 0 (negative)

A	B	C	X
0	0	0	N.A.
0	0	1	N.A.
0	1	0	N.A.
0	1	1	0
1	0	0	N.A.
1	0	1	0
1	1	0	0
1	1	1	1

(N.A. = not allowed)

L. Conway Archives

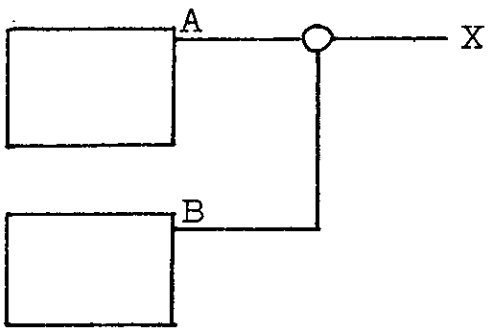
The orthogonal collector dot is the connection of collector outputs of current switches (the in phase outputs) before passing through an emitter follower. This connection performs the AND function--with the important restriction that no two of the inputs may be simultaneously negative. This is called the orthogonality restriction. In the above three input case the restriction requires that: $A \cdot B + A \cdot C + B \cdot C = 1$.

The ultimate physical restriction is somewhat weaker than the stated logical orthogonality restriction. A maximum time of .5 ns of non-orthogonality is allowed, which covers variations in signal delays. See Reference 1, Page 2.

Fan-in: ≤ 5

Fan-out: See fan-out for current switch. Same description applies here.

The Emitter Follower Dot:



A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

$X = A+B$

The emitter follower dot circuit is the "dotting" or connection of current switch outputs A and B after their emitter followers. The function performed is OR with no restrictions except fan-in and fan-out. Note that we might have a line connected to an e.f. dot which came from an emitter follower which followed a collector dot.

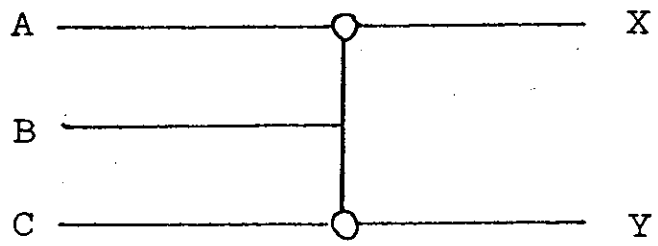
Fan-in: ≤ 5

Fan-out: ≤ 12 (try for ≤ 8)

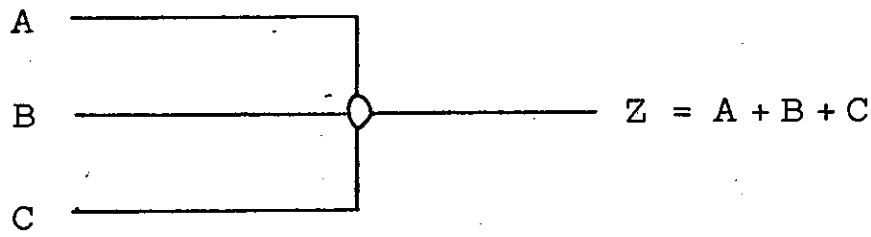
L. Conway Archives

Note: The meaning of "dot" in orthogonal collector dot and emitter follower dot is that the inputs are actually wired or connected together. Thus the O.C. Dot and E.F. Dot are not circuit elements, but are connections of wires which perform particular logical functions on the signals carried by those wires due to their locations in the circuitry (see Reference 1).

Therefore we cannot think of applying the same input to two separate dots. For example, the following diagram is incorrect for it shows B as an input to two separate E.F. Dots, treating these dots as independent circuit elements and expecting that X = A + B and Y = B + C:



Since the E.F. Dot is merely a connection of the inputs, the only possible interpretation of the E.F. Dot of A, B, C is that they are all wired together as follows:



L. Conway
Archives

Most beginning logical designers will have had considerable experience in design using AND, OR, and COMPLEMENT "gates" as circuit elements. It is natural for the designer to write logical equations for such designs using AND, OR, and COMPLEMENT logical operators. The primary content of switching theory consists of operations on logical functions expressed using these operators.

However in ACS the actual logic circuit implementation of a design is usually in NOR-NOR or NOR-OR logic.

It turns out that the usual OR-AND or AND-OR formulations of logic equations can be easily transformed and converted directly to the corresponding NOR-NOR or NOR-OR circuitry (see Section 5 for these techniques).

Therefore, for convenience most ACS designers express logical functions using OR, AND, and COMPLEMENT logical operators. The usual minimization techniques of switching theory may then be applied to these formulations before transformation into the final NOR-NOR or NOR-OR form (the circuit diagram itself).

The following different symbols for the logical operators are currently in use by different ACS designers:

AND(A, B): = $A \cdot B = AB = A \wedge B$

OR(A, B): = $A + B = A \vee B$

NOT(A): = $\bar{A} = A' = -A$

These variations in basic operator symbols from one designer to another should cause the newcomer no confusion.

There is one practice, stemming from the ultimate NOR-NOR or NOR-OR implementation of logical functions, which will definitely cause the newcomer confusion if it is not fully understood. It is a common practice in ACS to use two different symbols for complement in the same logic equations. Thus we may see both $\bar{A}$ and -A, or perhaps even $-\bar{A}$ in some equation. The reason some designers use both forms derives from the inversion of variables when using NOR-OR logic. One symbol is usually reserved for true logical complements and the other symbol (usually -) is used to mark variables or expressions which are complemented because they are at an intermediate point in the logic (see Section 5).

L. Conway Archives

It is easy for the newcomer to think that -A must mean something other than $\bar{A}$, perhaps having something to do with negative voltages. This happens easily because some designers also mark uncomplemented variables with + in some cases (using the symbol V for OR).

However, remember that -A is equivalent (logically) to $\bar{A}$, and that +A is equivalent (logically) to A. Some designers might argue otherwise, but that is because they have attached some additional heuristic values to these different symbols for complement in order to aid their design efforts. Thus, any difference between -A and $\bar{A}$ is only a heuristic difference, not a logical difference.

For example, the following equations all equate X with the same logical function of A, B, C:

$$\bar{X} = A \cdot B \cdot \bar{C}$$

$$-X = A . B . \bar{C}$$

$$+X = -(A \cdot B \cdot \bar{C})$$

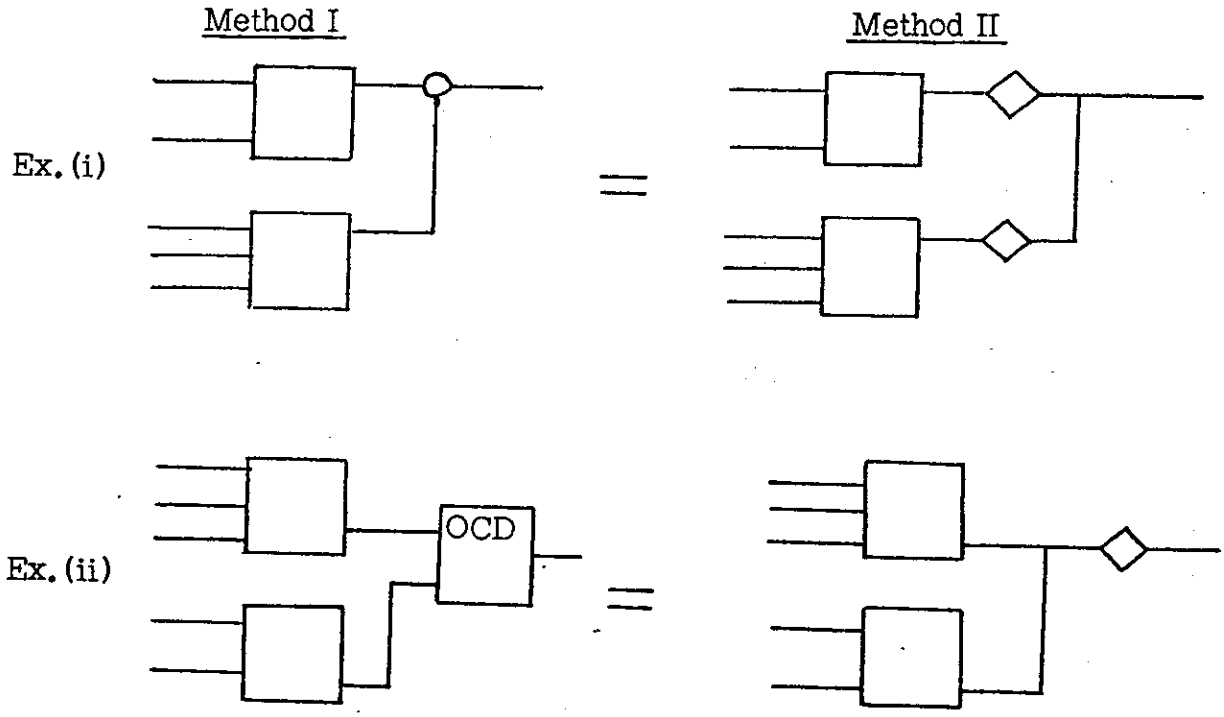
After gaining some experience with NOR-NOR and NOR-OR circuit implementations of logical functions, the newcomer may find that it aids him in his design efforts to use $\pm$ symbols in addition to the usual complement symbol.

It is not necessary to use these extra symbols and the corresponding heuristic techniques. They may assist those designers who prefer to design in an informal manner. One may, alternatively, design in a formal manner without ever using heuristics. However, all ACS designers should know about the techniques used by other designers and the resulting additional notation so that successful communication is possible between different designers.

L. Conway Archives

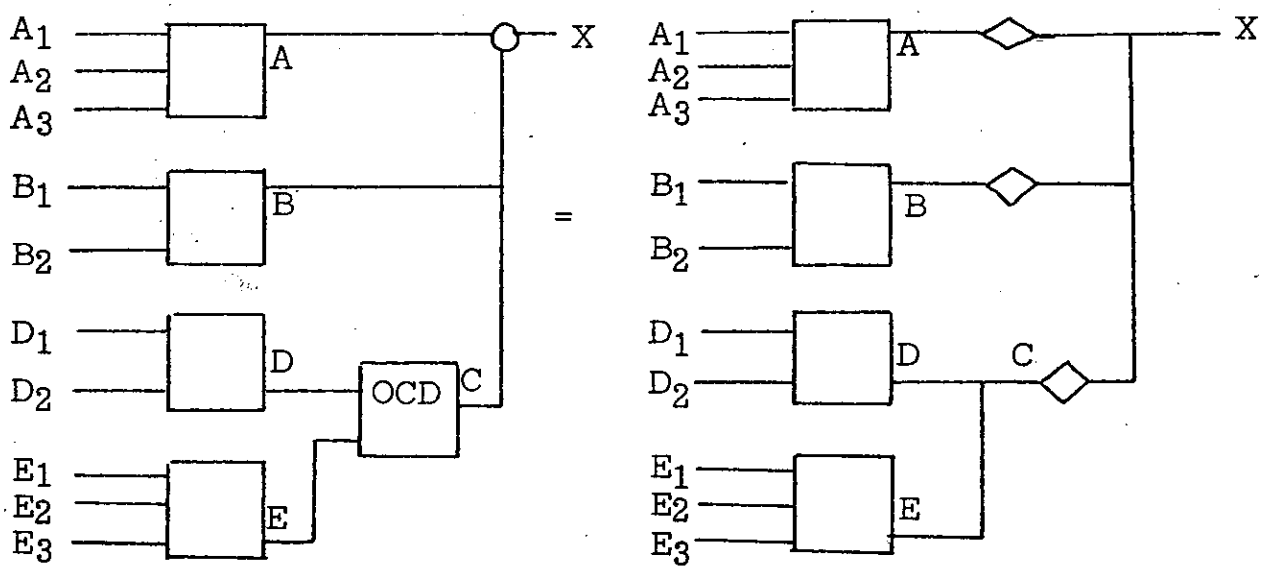
A number of different conventions are in current use for drawing logic circuitry composed of ACS circuits. Different designers may use different symbols for the basic circuits. Some designers indicate emitter followers while others do not.

Two methods are shown below which serve to illustrate some of the possible variations in circuit diagram techniques. The two methods differ primarily in the way in which the orthogonal collector dot is symbolized. When the O.C.D. is symbolized by a labelled block, it is not necessary to indicate emitter follower positions. However, if only a simple dot is used to symbolize O.C.D., then it is necessary to show emitter follower positions (symbol: ◊) in order to avoid confusing O.C.D. with emitter follower dot.



L. Conway Archives

Ex. (iii)



$$A = \bar{A}_1 \cdot \bar{A}_2 \cdot \bar{A}_3$$

$$B = \bar{B}_1 \cdot \bar{B}_2$$

$$D = D_1 + D_2$$

$$E = = E_1 + E_2 + E_3$$

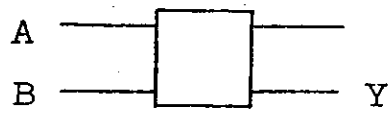
$$C = D \cdot E$$

$$X = A + B + C = A + B + D \cdot E$$

$$X = \bar{A}_1 \cdot \bar{A}_2 \cdot \bar{A}_3 + \bar{B}_1 \cdot \bar{B}_2 + (D_1 + D_2) \cdot (E_1 + E_2 + E_3)$$

In the examples shown above, the basic symbols for the current switch are all the same. Sometimes, however, designers will place a letter inside the current switch symbol to indicate the logical function that it performs. This practice may lead to considerable confusion for the newcomer for two reasons: (i) different function names are often used for the current switch by the same designer, (ii) the output phase of the switch to which the name refers is usually assumed to be obvious and is not explicitly indicated. Let us study these conventions in some detail to avoid confusion.

L. Conway
Archives



For the current switch shown, the output Y equals the OR of the inputs A, B:

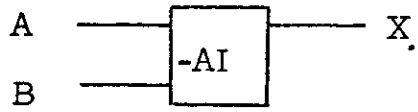
$$Y = A + B$$

Suppose we complement both sides of the equation to yield:

$$\overline{Y} = \overline{A} \cdot \overline{B}$$

We thus find that the complement of Y equals the AND of the complements of A, B. Now, even though this equation expresses Y as the same function of A, B, many designers call this the "MINUS AND" function. Thus one may see different current switches in the same circuit diagram labelled in both of the following ways:

These circuit symbols both stand for current switches and both perform exactly the same logical function on their inputs. Some designers choose to view them differently depending on whether or not complemented variables appear as inputs. This is another heuristic aid to the designer. Clearly it is not necessary to view the circuit element in these two different ways. It is just that some designers find that this technique assists them in their design efforts. Note that the output phase in the above examples to which the function name applies is found to be the "in"phase. This is not explicitly indicated, but is "obvious" because of the known function of the switch. This sort of duplicate naming can be carried further if desired. For example:



Here we have named the function as "MINUS AND INVERT." The meaning is that the output X is the complement of the MINUS AND function.

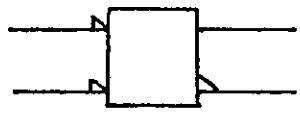
L. Conway
Archives

This duplicate naming of functions may sometimes be applied to the other circuit connections. The emitter follower dot performs an OR function and so may also be thought of as performing the "MINUS AND." The orthogonal collector dot performs the AND function and so may be thought of as performing a "MINUS OR" function.

It is important to note that "MINUS AND" and "MINUS OR" are not equivalent to the logical functions NAND and NOR. It is unfortunate that the use of MINUS (-) here conflicts with our previous definition of (-) as equivalent to complement. One might therefore be led to believe that MINUS AND (-A) is equivalent to $\overline{AND}$ (and thus equivalent to NAND), which it is not.

"MINUS AND" and "MINUS OR" may best be viewed by the beginner as merely other names for OR and AND, used by some designers for their heuristic value when circuit input variables are in complemented form.

There is another circuit diagram symbol which the newcomer will occasionally see and which is bound to confuse him. This is the "wedge" symbol appended to certain circuit block inputs/outputs. Wedges might be found on a current switch symbol as follows:



These wedges have no functional meaning to the logical designer. They do not change the identity or function of the circuit element. The wedges are normally produced by the DRKS system and automatically affixed to the circuit blocks appearing on the DRKS sheets. The wedges appear to be used primarily by CE's who service the hardware. Wedges appear mainly on the MACRO circuit blocks defined and used in DRKS. To quote Reference 3, Section 2.2.8.5:

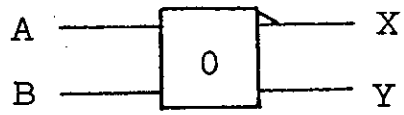
"Wedges will be printed in the edge of box print position for all input or output lines that are in the "down" signal condition when the logic block function is being performed. The designer need not draw these wedges on his diagram. They will be automatically inserted by DRKS, according to the block definition in the macro file, when the sheet is printed."

In other words, given a circuit block performing some logical function as stated by a logical equation, DRKS affixes wedges to those input and output lines which must be down (0; negative) when both sides of the equation are TRUE (1).

L. Conway Archives

Examples: note that although both examples use the same circuit, the wedge placement is different. This is because wedge placement depends on the statement of the function of the circuit. If we complement both sides of the equation defining the circuit, then the wedge placement changes.

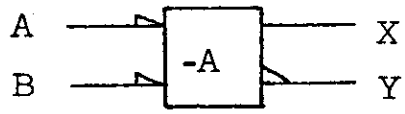
(i) Current Switch as an "OR"



$Y = A + B$

When both sides of the equation are TRUE (1), then Y must equal 1, neither A nor B must equal 0, and since $X = \overline{Y}$, then X must equal 0.

(ii) Current Switch as a "MINUS AND":



$\overline{Y} = \overline{A} \cdot \overline{B}$

When both sides of the equation are TRUE (1), then Y must equal 0, A must equal 0, B must equal 0, and since $X = \overline{Y}$, then X must equal 1.

Now, even though the wedges have no functional meaning, some designers may attach them to the circuit blocks in their circuit diagrams. This is especially true when MACRO circuit blocks are used. A reason for this is that the wedges can be used as a memory aid in locating particular inputs and outputs on the MACRO blocks which have many input/output lines. But remember that there is no additional information contained in the wedges. DRKS can produce them automatically when given the function of the block.

L. Conway
Archives

Elementary Logic Design

Logic design in ACS, and in any case where implementation will be made in real circuitry, is essentially an iterative procedure consisting of making a design, then testing that design against technological restrictions, then redesigning and retesting until a valid design is found.

First the logical functions to be implemented in the design are formulated in a set of logical equations. Then the set of equations is operated upon to minimize the logic according to some selected criteria such as number of circuits and/or number of circuit levels. Note that the minimization may be performed on the equations (which use AND, OR, NOT operators) even though the final implementation may be in NOR-NOR, or NOR-OR logic (see Reference 4, page 101).

Next, the minimized equations are examined to determine if all circuit restrictions are satisfied. These restrictions, such as fan-in and fan-out, can be checked while the design is still in the form of logical equations.

If the restrictions are not satisfied, we must iterate by going back and perhaps reformulating the equations and minimizing again, until equations are found which satisfy the restrictions.

At this point we can convert the equations directly into a logical circuit implementation. Descriptions of procedures, both formal and heuristic, for performing these conversions follow later in this section.

Now, if the design specification is beyond the preliminary stage and unlikely to be changed, then the circuitry must be checked against all the many and complex wiring and packaging rules. If the design cannot be wired or packaged as is, then additions or changes may have to be made, or perhaps another entire design iteration may be required.

Implementing Logic Equations in ACS Circuitry:

With a little experience a designer can directly sketch out the logic circuitry to implement some logical function. This is particularly easy to do if AND-OR or OR-AND logic circuits are used. For these cases the designer can place the equation for a function in "sum of products" or "product of sums" form and transform directly to a circuit diagram.

In the ACS technology, however, we have available only a restricted form of AND circuit (the orthogonal collector dot; inputs must be orthogonal). Thus OR-AND logic is seldom used. Instead, we normally use NOR-NOR or NOR-OR logic.

L. Conway Archives

The beginner should therefore learn the transformations for quickly and automatically drawing the circuit diagrams for NOR-NOR and NOR-OR logic implementing a logical function. This material is covered in detail in Reference 4, pages 94-102. A summary is presented here for reference:

Let us draw the logic circuitry to implement the function

$$f = (a + b)(b + \bar{d})(a + c) = ab + bc + a\bar{d}$$

Ex. (i): NOR-NOR logic circuit implementation:
(2 circuit levels: current switch to current switch)

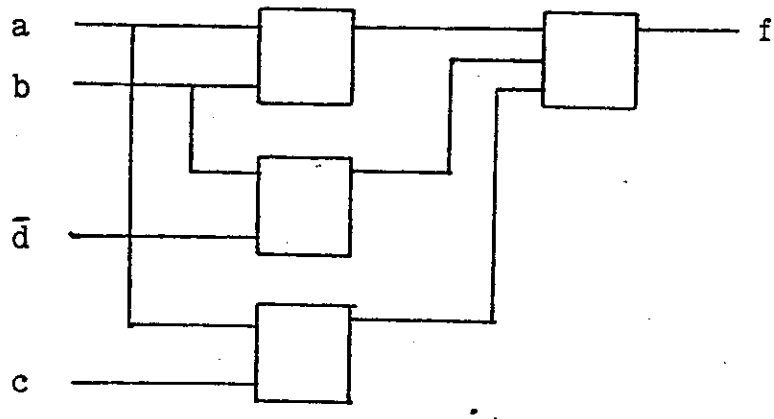
Step 1: Express function in product of sums form:

$$f = (a + b)(b + \bar{d})(a + c)$$

Step 2: Let NOR $(a, b) = \overline{(a + b)}$. Transform the equation to NOR-NOR form by simply replacing all OR, AND operators with NOR operators, leaving the variables in the original order and form:

$$f = \text{NOR}(\text{NOR}(a, b), \text{NOR}(b, \bar{d}), \text{NOR}(a, c))$$

Step 3: Draw the logic circuit diagram directly from the equation in Step 2.



Clearly we may proceed directly from Step 1 to Step 3. The NOR-NOR logic uses the same connections of circuits to implement a function as does OR-AND logic. We merely replace all OR and AND circuits with NOR circuits

L. Conway Archives

EX. (ii): NOR-OR logic circuit implementation:
(1 circuit level: current switch to E. F. Dot)

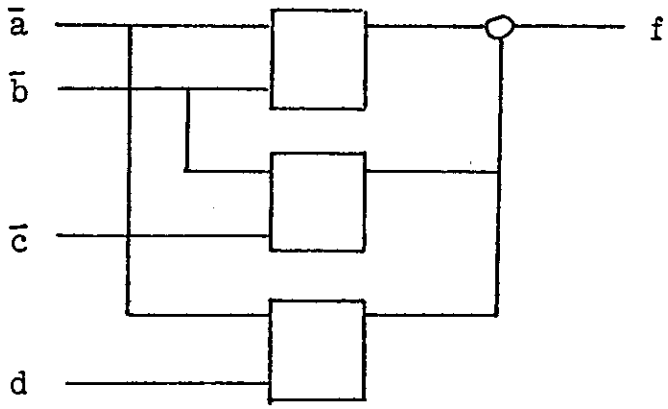
Step 1: Express function in sum of products form:

$$f = ab + bc + a\bar{d}$$

Step 2: Transform the equation to NOR-OR form by complementing each variable and replacing the AND operators with NOR operators:

$$f = \text{NOR}(\bar{a}, \bar{b}) + \text{NOR}(\bar{b}, \bar{c}) + \text{NOR}(\bar{a}, d)$$

Step 3: Draw the logic circuit diagram directly from the equation in Step 2:



Here also we see that it is easy to proceed directly to Step 3 from Step 1. The NOR-OR logic uses the same connections of circuits to implement a function as does AND-OR logic. We merely replace the AND circuits with NOR circuits and use the complementary inputs.

Heuristic Design Techniques:

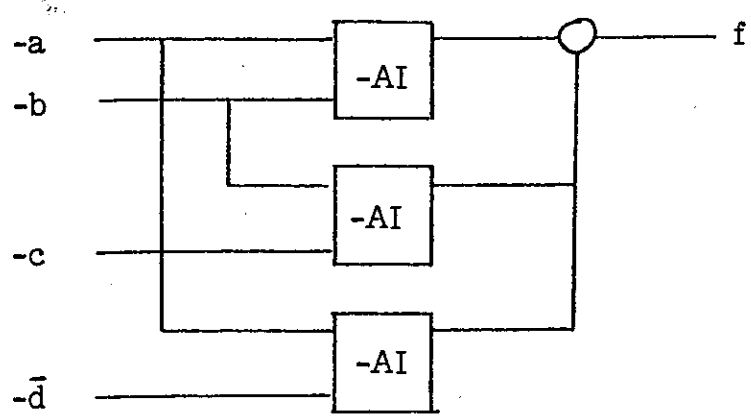
The extensive use of the NOR-OR logic has caused the evolution of many heuristic design practices, including the use of two different symbols for complementation and the duplicate naming of the logical function performed by the current switch.

To clarify all the points developed in this memorandum concerning heuristic design techniques, let us implement the same function f of the preceding examples in NOR-OR logic using one of the heuristic techniques rather than the formal, automatic procedure just described.

L. Conway Archives

Suppose we have available as inputs both phases of a, b, c, d, i.e., $\pm a$, $\pm b$, $\pm c$, $\pm d$ and wish to form $f = ab + bc + a\bar{d}$.

Using minus (-) inputs we can use "MINUS AND INVERT" circuits to obtain the terms ab, bc, and $a\bar{d}$. Then we can use the emitter follower dot to OR these terms.



Clearly this is the same circuit as that developed in the preceding formal NOR-OR example. However, here the designer is thinking directly in terms of pseudo AND-OR logic by renaming the functions of his circuit elements and making a sequence of appropriate complementations.

The beginner is warned not to attempt to imitate such techniques at first. The heuristic techniques, used by the novice as though they were formal methods, will prove far more unwieldy and confusing than the previously illustrated formal techniques. The novice using these heuristics will put a great deal of effort into the essentially trivial process of forming circuit diagrams from logic equations.

When the time comes that the designer has a good "feeling for" NOR-NOR, NOR-OR logic design, he may then find that some of the existing heuristic techniques are useful. Experienced ACS designers can sometimes find "tricky" implementations using these techniques which have less delay or lower circuit count than those derived by formal approaches. This occurs especially when both the O.C. Dot and E.F. Dot are used in the implementation.

L. Conway
Archives

Date: October 31, 1967
From (location & U.S. mail address): Advanced Computing Systems, Menlo Park, California
Dept. & Bldg.: 988/031
Telephone Ext.: 252

IBM

Subject: A Proposed ACS Logic Simulation System (LSS)

Reference:

1. Specifications for Input and Output of ACS/TALES Simulator, A. G. Auch, Dept. B24, SDD Poughkeepsie, September 20, 1967.
2. TALES - ACS Simulation Capability, A. G. Auch, Dept. B24, SDD Poughkeepsie, August 15, 1967.
3. ACS AP #67-115, MPM Timing Simulation, L. Conway, August 25, 1967.
4. ACS AP #66-022, ACS Simulation Technique, D. P. Rozenberg, L. Conway, R. H. Riekert, March 15, 1966.

To: File

L. Conway

L. Conway

LC:aw

347

L. Conway Archives

CONTENTS

348

L. Conway Archives

Introduction

This memorandum describes a proposed ACS Logic Simulation System(LSS). This system has been only tentatively defined. The purpose of this memorandum is to set down the current thinking and stimulate some feedback from potential users, potential implementers, and other critics on the feasibility and utility of such a system and on the practical details of its implementation and use.

The purpose of the proposed LSS is to provide a mechanism for aiding the debugging of the logical design of the ACS-1. The logical designer may know that for a given section of logic circuitry a certain set of inputs should produce a particular set of outputs (for a given initial internal state) according to the "system level" description of the design which he implemented in the logic circuitry. The LSS will provide a means of inserting the circuit inputs into a logic simulator which simulates the action of the circuitry on these signals and then compares the resulting output with the output expected by the designer. Any mismatches would indicate a logical design error in the circuit (see fig. 1)

A group in Poughkeepsie can provide ACS with a package of programs capable of performing the logic simulation. The ACS designer would provide input to these programs indicating the particular partition of the machine to be simulated and the input-output lines on the interface of this partition. The programs would use this input to extract from the DRKS files the detailed description of the logic of the partition selected. The designer would then need to apply a sequence of inputs to the logic simulator corresponding to a proper sequence of input-output line signals at the interface of the partition. The programs would simulate the logic operating on the input signals and mark any mismatches in the logic output and expected output. The designer would then use these mismatches to debug his logic design.

A major obstacle to the practical application of this proposed system is the difficulty of generating the I/O signals at the partition interface. It does not appear to be at all practical, or even feasible, for the logic designers to generate by hand all the correct test patterns necessary to "moderately" debug all the partitions of the machine.

A method has been proposed to solve this problem by providing a programmed means of automatically generating these interface I/O signals. A detailed timing simulator now exists for the MPM (ref. 3). This simulator times the activity of all MPM hardware, as described at a system level, during the execution of an input program.

L. Conway
Archives

Now, suppose we wish to use the LSS to study and debug a particular partition of the MPM. We could carefully define the interface of that partition and rewrite the appropriate sections of the timing simulator such that (i) the same interface existed in the timer as in the logic circuitry, (ii) the same "system" level description is used in the timer to describe the partition that was used to formulate the logical design of the partition, (iii) provide for output to suitable files of the timing simulator interface signals during each simulated cycle of execution.

The timer thus modified could become a practical source of the I/O signals needed to drive the LSS. The timer would have to accurately reflect the MPM only at and within the interface of the partition to be studied. Any errors in this system description would be discovered early in the debugging process. After this phase, many selected programs could be run on the timer to yield as many interface signal sets as are necessary to debug the logic design of the partition to the required level (see fig. 2).

The timer could also assist the designer of the partition in his efforts to find a particular bug when the LSS indicates a mismatch in outputs. The timing charts produced by the timer will give a concise picture of the state of the machine at a system level in the region of time surrounding and including the cycle in which the bug occurred. This may help to determine if the bug is at the level of system specification or logic circuit implementation. Both the timer and LSS can provide the states of specified triggers within the partition and a comparison of these can aid the designer in debugging.

In the following sections of this memorandum some of the details of this proposed LSS system are described and questions are raised which must be answered before any serious development of the system can begin.

The main point to keep in mind is that there are two levels of simulation involved in this scheme -- the detailed simulation of the logic circuitry of a design and the system level simulation of the same design. This two level simulation technique for debugging logic circuitry was originally proposed to ACS in August, 1966 by G. T. Paul. The technique now appears to be feasible because of the availability of an adequate logic simulator and ACS experience with the current timing simulator.

Comments and criticisms are invited, especially on questions concerning the feasibility of the system, its utility to the ACS logic designers, its cost relative to any alternative systems, and the various practical problem of its implementation and use.

L. Conway
Archives

FIG1. THE BASIC IDEA OF LSS:

APPLY SAME INPUT TO BOTH LEVELS OF SIMULATION AND COMPARE OUTPUTS. IF OUTPUTS ARE DIFFERENT THEN ERROR EXISTS IN LOGIC DESIGN.

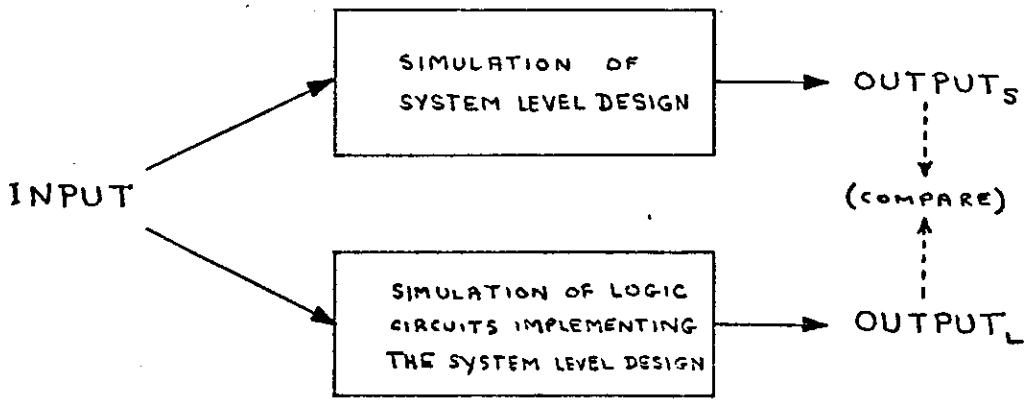
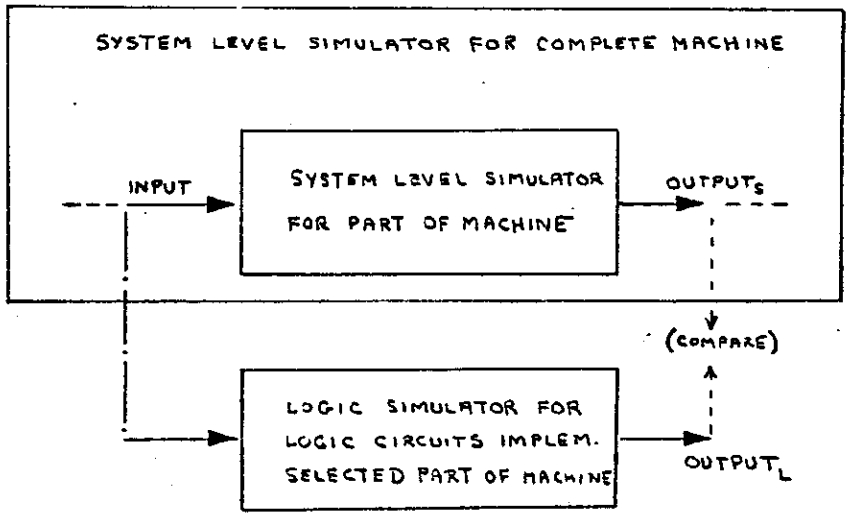


FIG 2. AUTOMATIC GENERATION OF SYSTEM LEVEL INPUT/OUTPUT, LOGIC SIMULATOR INPUT:

SYSTEM LEVEL DESIGN IS IMBEDDED IN SYSTEM LEVEL SIMULATION OF ENTIRE MACHINE. WHEN THIS SIMULATOR RUNS WE AUTOMATICALLY GENERATE (AND SAVE) THE I/O AT THE DESIGN INTERFACE. WE MAY LATER APPLY THESE INPUTS TO THE LOGIC SIMULATOR FOR THE SAME DESIGN AND COMPARE THE LOGIC OUTPUTS WITH THE SYSTEM LEVEL OUTPUTS.



L. Conway Archives

The LSS Programs

In this section the programs forming the LSS are identified and described. The relationships between the various programs and the designers input and output to the system is described. This specification was developed from information contained in ref. 1 and the notion of using the timing simulator to drive the LSS. This specification is very tentative in nature.

The simulation of the logic of a portion of the ACS-1 machine operating on a sequence of inputs may be viewed as occurring in three distinct phases within LSS.

The first phase is the selection of the specific partition of the machine to be studied and the specification of the I/O interface for this partition. The designer will specify the partition and interface in a card input deck. This deck is used by the LSS to extract the detailed information describing the logic circuitry of the partition from the DRKS files and DRKS rules. The program performing this extraction is termed the Simulation Interface Program (SIP), and is to be written by the Poughkeepsie people.

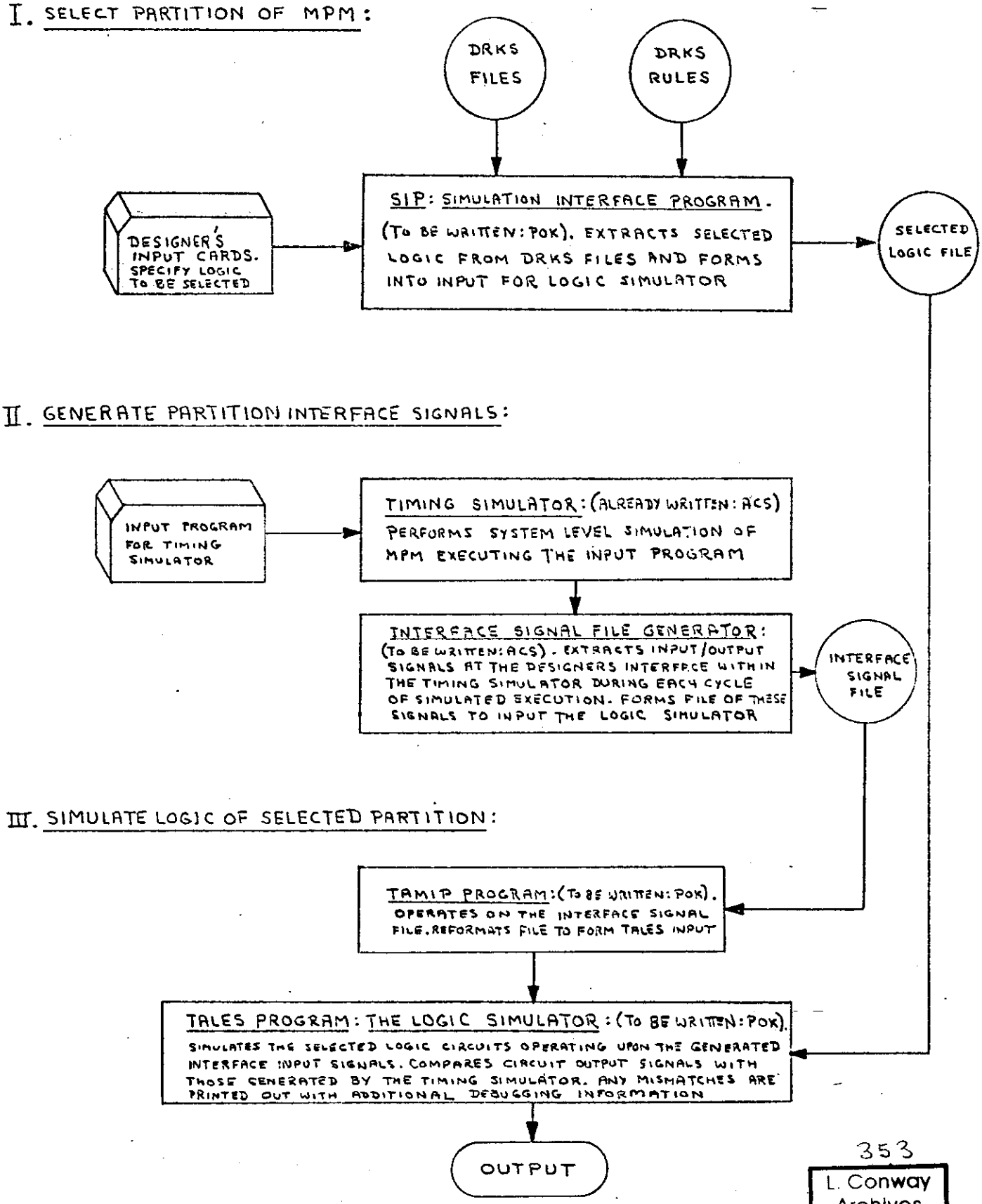
The next phase of the LSS simulation is the generation of a sequence of interface signals for the selected partition. This is done by running ACS program on the modified timing simulator. Once the designer has assisted in forming the proper timing simulator specification for his partition, the production of these interface signals requires no more effort by him. Many programs exist which run on the timer. The designer would merely select those programs which might best be applied to debugging his particular section of the machine. An addition must be made to the existing timing simulator to extract and file the proper interface signals during each cycle of simulated time. Let us call this the interface signal file generator. This program would be written here at ACS.

The final phase of the LSS run is to perform the logic simulation itself. This is done by a program to be called TALES, which is to be developed by the Poughkeepsie group. The interface signal files produced by the timer-interface file generator programs are processed by a reformatting program called TAMIP (also to be written by Poughkeepsie) and then input the TALES logic simulator. The TALES simulator uses the logic files formed by the SIP program to perform the proper logical functions on the input signals to yield interface output signals for each simulated cycle. If the logic simulator output signals differ from the expected output signals produced by the timing simulator, an output listing to this effect will be produced and certain information printed to assist the designer in finding the cause of the mismatch.

In figure 3 the functions of the three phases of LSS are illustrated by flow-charting the relations between the designer's input, the various LSS programs, the DRKS files, and the various LSS internal files.

L. Conway Archives

FIG 3. THE ACS LOGIC SIMULATION SYSTEM



L. Conway Archives

Possible Procedures for Use

So far we have examined the overall functions of the LSS and identified the component programs and files. All of this is very tentative. In this section let us explore some of the many different possibilities which exist for organizing and using the LSS system, and identify those areas which are only tentatively defined and need to be worked on.

Many questions and alternative approaches are outlined which must be resolved before the system can be considered feasible, useful, and economical. Criticism on these specific questions from everyone concerned is needed to formulate the answers to these questions.

Most of these questions center on the organization and management of the system, i.e., what technical form should the system have in order to be usable by the designer? For example, how do we partition the machine, how large or small should the partitions be, and how do we select the interfaces? How should the designers specify the system level description of their partition?

(i) Partitioning the MPM: How large or small should a partition be? From an organizational and system simulator point of view, the larger the better. If a partition is too large, however, the designers may have a difficult time in debugging the logic. This problem might be eased by placing certain triggers internal to a partition in the set of outputs the designer can check. If the partitions are too small and thus many in number, we will have difficulty in managing the study--there will be too many interfaces, and some of them may be inconvenient to specify at the system level.

It seems undesirable to have a single partition so large or so chosen that two different design groups design sections of the partition. The utility of the LSS system is increased by having formal interfaces between the various groups of designers, to allow a successful segmentation of the design. It is natural that the interfaces between design groups would also be interfaces in the system level simulator in LSS.

An approach to choosing partition size might be the following: choose the partitions as large as is possible subject to the following constraints, (i) the boundaries of the various design groups, (ii) the maximum amount of logic which the logic simulator will handle. It is likely that the second limit will usually be met first. This raises the question of whether the logic simulator (TALES)

L. Conway
Archives

can handle a large enough partition for the LSS to be practical. This question is quantitatively studied (section 4) later in this memorandum, and the answer currently appears to be yes.

(ii) Selecting the Interface: Suppose we wish to formulate a partition of the MPM whose approximate size and boundaries are known. We face the problem of selecting the exact interface that is to exist between this partition and the rest of the machine. This is the problem of selecting an interface which is reasonable both in the logic and in the system level of description. The problems involved in doing this do not appear to be serious if the partition is large, for then certain natural boundaries (the phases) within the MPM may be chosen as interfaces. If the partitions must be very small and many in number, we will have serious problems for the system level description as a whole will become much more detailed and unmanageable. We might not be able to simulate on a cycle by cycle basis, but have to generate and check interface signals at many different times within a machine cycle.

(iii) Describing a Partition: In order to correctly generate the interface signals for a given partition, the timing simulator must accurately reflect the system level description of that partition. An important question to be answered is how is the detailed system level description of a partition to be formed, in what language, and by whom? There is a wide range of possibilities.

Method (a). The designers could give a verbal, nonformal description of their partition to a programmer who would formalize the description by writing the code which performs the system level simulation. This is probably not adequate because it would be too difficult to maintain the description. The designers would have no direct link to the formal description when they desired to make a change.

Method (b). The designers could produce a "semi-formal" description of their partition by creating a combination of flow charts, diagrams, and written description which attempted to document as accurately as possible (outside a formal language) all the details of their design. A programmer could use documents of this type as a direct basis for his coding of the system level simulation. This at least solves the problem of maintenance of the program. A change in a flow chart could fairly easily point to the necessary corresponding change in the simulator code. Even with this method, serious problems arise (even more serious if using Method (a)). Since the designers would not themselves have a complete, formal description at a system level of the thing they have designed, many errors are bound to occur in the system description--errors which would be difficult to debug.

L. Conway
Archives

Method (c). We might go a step further in the specification of a partition by the designers and require that they help formulate and have access to a complete, formal description of their partition at the system level. This could be done by having the designers partitipate actively in the production of the formal description. The obvious choice of a language for formal description is the simulation language used in the timing simulation program. This language is an "elementary form" of "Simscript," and is written in FORTRAN (see ref. 4).

The designers could produce the flow charts, etc., as in Method (b), but then assist in the production of the system simulation code to the extent that they would fully understand and be able to modify (with programming assistance) the system level description.

The system simulation code would then be the formal description for the designer. It would be easy for the designer to introduce changes into the formal description.

Method (d). We can go one step further and require that the designers independently produce a formal system description of their partitions in some language common to all the design groups. This is a goal to strive for in later design efforts. It seems impractical at the present time, however, because of (1) the time required to educate the designers in some formal language, (2) the even greater time required for them to gain "programing" experience--the experience needed to use the language to describe their design at the proper system level. Most logic designers probably conceptualize their design not as a system description being implemented in some logic circuitry, but as the logic circuit implementation itself. That this is likely is indicated by the current lack of detailed system descriptions within engineering and the current wealth of logic circuit diagrams.

Considering the methods (a), (b), (c) and (d) outlined above, it would appear that the most useful and feasible method for currently producing the necessary system level descriptions for the LSS is Method (c).

(iv) Selecting the Partition in the Logic: When we have selected and described a partition at the system level, we face the problem of selecting the same partition at the logic circuit level. The description of the logic circuits is formal and is contained in the DRKS files. The Poughkeepsie group will write the SIP program which actually extracts the logic design of a partition and forms the file to input the logic simulator.

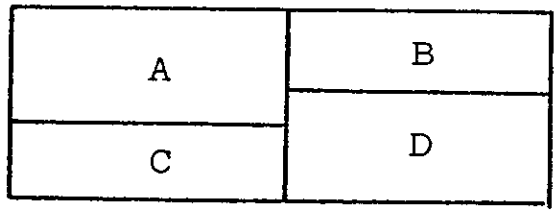
L. Conway
Archives

The designer's input to specify the logic to be selected by the SIP program has been tentatively defined in reference 1. There will have to be a study by all concerned to produce a specification of the SIP input conventions. Once the procedures for use of the LSS system have been defined, it would be desirable to specify input conventions for SIP which are the simplest possible in nature which meet the needs of the LSS. The smaller and simpler the interface between ACS designers and Poughkeepsie programs the better.

(v) Sequence of Partitions to be Studied: An important property of the proposed LSS using the existing timing simulator as a starting point in the system level description is that the debugging of one partition may proceed independently of that of another partition. We can thus choose a sequence of partitions to be debugged which corresponds to the schedule of design of the partitions.

We could have chosen not to use the timer, but to apply Method (d) of the previous section and develop a formal and accurate system level description of the whole machine. Let us examine some of the problems within this scheme and thus learn the advantages of using the timer.

Suppose the machine could be divided into four partitions:



We could have the designers write the programs described A, B, C, and D and then run these as an accurate timing simulator, obtaining input and output signals at the interfaces.

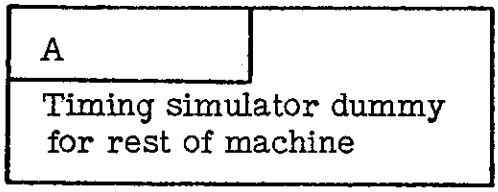
The problem with this is that the system level programs must all exist and be reasonably debugged before the whole system level simulation will run. Of course the individual partition programs could be run separately to yield partition outputs for a given set of partition inputs. But this does not solve the original problem affecting the feasibility of logic simulation--the difficulty of generating by hand all the input-output patterns. It only half solves the problem.

L. Conway
Archives

Another difficulty with this approach is that we would be heavily committed to whatever techniques were chosen to implement Method (d).

Clearly we do not need to face these problems and uncertainties. The existing timing simulator can be used to circumvent them as follows:

We chose for LSS debugging the first partition whose design is "completed." Suppose this is partition A.



We already have a working, debugged timing simulator which simulates an approximation to the whole MPM. We write and place into the timer (replacing existing code) the the description of partition A at the system level. Now the remainder of the timer serves as a dummy machine which can properly interact with partition A once the system description of A is debugged. Now we may not get exactly the same feedback from the dummy portion of the machine that we would get from the eventual real machine, but this does not matter. We will get valid feedback which will properly drive partition A. We will automatically get both inputs and outputs of A every cycle while the simulated machine runs an input program.

This allows a considerable degree of freedom in the planning of the debugging process. We may debug the partitions independently and in sequence if we so desire. It is likely that the various partitions will be ready for debugging at different times. We could schedule the debugging to correspond to these design schedules. We would not be committed to the first procedures chosen to debug the first available partition. If a method proves unsatisfactory on the first partition, we can modify our procedures for handling later partitions.

By using this method we can proceed only as far as we choose in applying LSS to debugging the logic. We do not need to determine in advance how much of the logic is to be debugged this way. Some sections of the machine may remain in dummy (original timing simulator) form. Some sections of logic such as functional units (adders, multipliers) clearly can have their logic simulator input-output signals formed by hand or by special programs of much simpler form than system level simulators.

L. Conway
Archives

Note that the timing simulator can eventually become an exact system level simulator of the whole machine if that end is desired. This method does not preclude that possibility. Indeed, this method offers a practical means of achieving that end in a step by step approach rather than attempting it directly.

(vi) Debugging a Partition: How does the designer use LSS to uncover bugs in the logic design? Let us consider various procedures which might help in the debugging process.

An important consideration in the debugging of a partition is the selection of some appropriate input programs for the system simulator. We wish to run programs on the timer which exercise as fully as possible the system logic of the partition under study, in order to debug that partition as fully and efficiently as possible. This selection process is yet to be developed.

A question which arises here is how far should the debugging of a partition proceed using LSS. This is a function of input program choice, the available computer time and manpower available for debugging. This question must be studied fully in order to estimate the performance of the LSS system compared to its cost.

An important potential function of LSS which must be explored and developed is that of providing the designer with information to assist his debugging effort in addition to the mere indication of an output mismatch.

One possibility, easily implemented, is to make available to the designer the timing charts produced by the timing simulator (see ref. 3) for the LSS run under study. It has proven possible, with some practice, for individuals to use the timing charts to follow completely the system level functioning of the MPM. The designer would thus have available to him a concise description of the states and functioning of the whole machine in the region of time surrounding and including the cycle in which a bug was found in his partition.

Another possibility is to have the timer and the logic simulator both provide as output the contents of important registers and triggers within a partition in addition to those on the partition interface. This would be especially important if the partition is a large one. Of course we would have to have the timer quantities behave exactly as the logic circuits in order for this to work. This might provide a practical way of allowing large partition size, yet

L. Conway Archives

feasible debugging. As an example, suppose a large section of phase 1 of the MPM is to be contained in one partition. It would be very useful in the debugging process if the designer had access to the values of such things as NFA, HISTORY TABLE, DO TABLE, etc., in both levels of simulation (i.e., as "interface output quantities"). Usually these important internal quantities of a partition could be easily made to function exactly the same at both simulation levels.

(vii) Other Modes of Use: During the specification and development of the LSS system we must identify and meet the requirements for any other possible uses of the system and its components.

An example of this is the need to allow manual insertion of interface signals into the Poughkeepsie programs in order to perform the debugging of isolated sections of design for which manual signal insertion is adequate. Examples of such design areas where manual or special program generation of the interface signals is possible are functional units such as adders, multipliers, dividers, etc.

Another function the system might perform is the generation of files suitable for hardware debugging at a later time.

L. Conway
Archives

Requirements for Development

The hardware, software, computer time and personnel required to develop, use and maintain the LSS system must be estimated to determine if the system is feasible and economical.

It has been determined that the ACS Mod. 75 computer will have adequate hardware for both the Poughkeepsie programs and the ACS timer-interface signal generator program.

Yet to be explored are possible work schedules, documentation requirements, and forms of communication needed between ACS and Poughkeepsie. It appears possible for the LSS development to proceed without altering engineering design schedules, if a proper scheme of development is chosen. Of course the time required for the designers to specify the system descriptions of their design areas will add to the design schedule time, but it appears likely that this system description will be necessary whether LSS is implemented or not. The requirements for maintenance of the system are yet to be determined. These depend on the role the designers play in specifying and maintaining the specifications of their partitions.

There are two important considerations which strongly affect the feasibility and economics of LSS. These are the computer time required to simulate and the memory requirements of simulation (determines maximum partition size).

Reference 2 indicates that a few seconds of Mod. 75 time would be required for the TALES program to perform the logic simulation of one machine cycle for the largest partition it could handle. The ACS system level simulation of the whole machine will run at a rate of approximately 10 to 15 machine cycles/second on the Mod. 75.

Thus it appears likely that the feasibility of LSS is not impacted by the computer time requirements. The required time is down in the range where the human time and effort in debugging the results would probably be a stronger limitation than available machine time. Of course these machine time requirements could be heavy ones and thus it is very important that the logic simulator (TALES) be made as efficient as possible, for the running of TALES will probably be the major cost of LSS.

Let us now consider the question of memory requirements and their determination of the maximum partition size.

L. Conway
Archives

P. Shivdasani has formulated the following study of this question, based on verbal communications with the Poughkeepsie group. His result of 56K ACS circuits as the maximum partition size indicates that we can choose partitions large enough for LSS to be practical (see section 3(i)).

(i) Storage capacity, S, in K bytes, required to run the logic simulator is

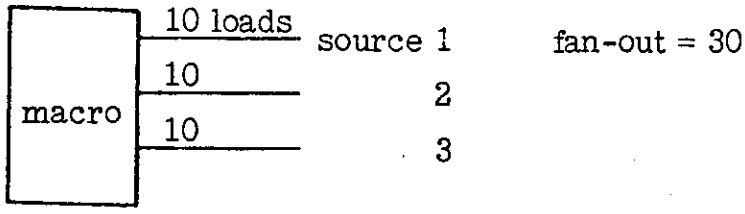
$$S = 98 + 2L\,(10 + \text{avg. fan-in} + \text{avg. fan-out})$$

where L = # of nets to be simulated (in thousands)

Also the fan-out from a block (macro, U. L. or dot) is

$$= \sum_{i=1}^{n} (\text{source}_i \cdot \text{load}_i) \qquad \leq 31$$

Thus



Another 200K bytes must be allowed for the worst case op. system.

There is also an absolute limit of 32K on L due to the present simulation programs.

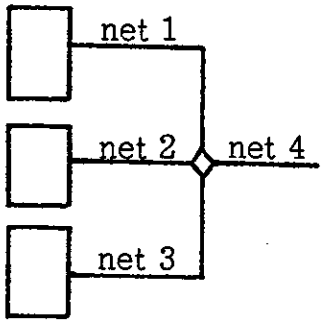
Thus if we assume L = 32
fan-out = 31
fan-in = 15

We have S = 3882 K bytes which will easily be handled by the two LCS's ACS has on order.

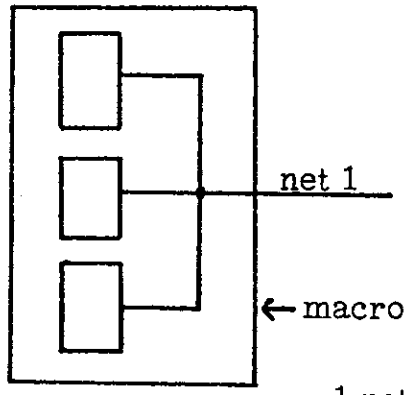
L. Conway Archives

(ii) Nets:

A net is defined as a logic source feeding any number of sinks. Thus in U. L. representation each U. L. block leading to a dot is a net.



4 nets



1 net

It is important, then to try and define as many macros as possible.

(iii) Assume 32K nets as maximum partition. Find equivalent in ACS circuits.

a) Let X be the number of circuits corresponding to these nets.

b) Assume 80% of the circuits can be represented in macros and the remaining 20% need a unit logic representation in DRKS.

c) Also assume each macro contains 5 circuits and has two source outputs.

Then nets due to macros = $\left(\frac{.8\,X}{5}\right) 2$

d) Assume an average dot of 4 in U. L. Then we have 5 nets for every 4 circuits.

Or nets due to U. L. = $\left(\frac{.2X}{4}\right) 5$

$$\frac{1.6\,X}{5} + \frac{X}{4} = 32,000$$

or $X = \frac{32,000}{.57} = 56K$ circuits

363

L. Conway Archives

(iv) DRKS does not handle macros made up of U. L. blocks from different portions of the same chip, let alone different chips. So if a high number of U. L. blocks is being dotted externally, the above capability will be desirable to keep the net count down.

L. Conway
Archives

Additional Benefits of LSS

There are some additional benefits which might result from implementing the proposed LSS system.

The formal specification of the machine at a system level would give the various design groups a chance to uncover many system level design errors before the logic itself is tested for bugs.

This formal system level description would be useful to many others in ACS.

Of course this description would have to be maintained by the designers to reflect all design changes. If maintained and the timing simulator reflects the description accurately, then the LSS could be used later to generate the interface signals for hardware circuit debugging.

Also, an accurate timing simulator would be very useful to the compiler and system programmers and to any ACS customers who wish to optimize hand code.

L. Conway
Archives

TO: L. Conway
Dept. 988
IBM - ACS
2800 Sand Hill Road
Menlo Park, California

Note: If you have any comments, questions, criticisms or ideas concerning the proposed LSS system, jot them down in the space below and mail this page as indicated above.

- -

366

L. Conway
Archives

August 6, 1968
Advanced Computing Systems
Menlo Park, California
988/031
Ext. 391

Subject: The Computer Design Process: A Proposed Plan for ACS

References:

1. ACS AP #66-022, ACS Simulation Technique, D. P. Rozenberg, L. Conway, R. H. Riekert, March 15, 1966.
2. ACS AP #67-115, MPM Timing Simulation, L. Conway, August 25, 1967.
3. A Proposed Logic Simulation System. L. Conway, ACS Dept. 988, October 31, 1967.
4. System Simulation Program in ACS Engineering, P. Shivdasani, ACS Dept. 988, April 24, 1968.
5. Proposal for a Design Procedure for the ACS System, Uno R. Kodres, July 19, 1968 (memorandum to D. P. Rozenberg).
6. Preliminary Description of Traceback and Simulation in ACS Fault Isolation, D. G. Keehn, August, 1968.

Memorandum to: File

L. Conway

L. Conway

LC:aw

L. Conway Archives

August 6, 1968

The Computer Design Process: A Proposed Plan for ACS

by: L. Conway

Distribution

Mr. J. G. Adler
Dr. G. M. Amdahl
Mr. S. F. Anderson
Dr. R. F. Arnold
Mr. A. M. Baptiste
Mr. B. O. Beebe
Mr. R. T. Blosk
Mr. J. Earle
Mr. A. F. Fitch--B73/959 Pok.
Dr. H. Fleisher--C14/704 Pok.
Dr. H. Freitag--12/234 Yorktown
Mr. R. R. Hanko
Mr. L. J. Hasbrouck
Mr. F. B. Jones
Dr. D. G. Keehn
Mr. L. E. King--B57/979 Pok.
Mr. J. D. Kyffin
Mr. R. Litwiller
Dr. R. E. Love
Mr. B. C. Madden
Mr. B. J. Mooney
Mr. M. O. Paley
Mr. J. F. Parsons
Mr. R. E. Pickett
Mr. R. J. Robelen
Dr. D. P. Rozenberg
Mr. P. Shivdasani
Dr. E. H. Sussenguth
Mr. G. E. Werner
Mr. E. L. Willette
Mr. W. P. Wissick
Mr. J. J. Zasio

L. Conway
Archives

CONTENTS

L. Conway
Archives

INTRODUCTION

For many years, computer designers have proposed the use of various levels of simulation for design specification, verification and evaluation. Simulation and automation have been applied to some phases of the design process in a number of past projects.

At the present time, in ACS, we feel that we have sufficient practical experience in system simulation and design automation to propose a workable system plan for the whole computer design process.

This plan has as its key element the specification of the system-level design in a high-level simulator. All following phases of design are viewed as implementations of this system specification.

Details of this plan are presented including initial design studies using timing simulation, design specification in a high-level simulator, logic design verification by comparing two levels of simulation, design automation and finally, hardware checkout and maintenance.

Design automation eliminates routine human effort in the later design phases. Simulation allows creative human effort where it is important--in the initial system level planning and evaluation. Rather than being merely a sideline in the design process, simulation can be and should be viewed as the natural medium of expression of the computer designer. A designer who can quickly generate working models of his ideas can get the feedback necessary for real design improvements. Adequate programming tools are now available to the designer for this purpose.

This memorandum presents a brief description of all the phases and components of the design process as it might exist in ACS. Much of this material is well established practice, and thus the memorandum could serve as an introductory tutorial document on this subject.

The purpose of this memorandum is to make certain specific suggestions concerning important aspects of the planning, implementation and operation of the total design process. The most important of these suggestions are

370

L. Conway
Archives

(i) The careful planning of the design process itself is as necessary for success of the project as is the careful planning of the computer design. The design process should be planned as one integrated system. If the separate phases are planned by different groups of people, the result will be an ineffective overall plan with serious difficulties at the interfaces of the phases.

(ii) The plans produced should be carefully documented and maintained and made available to all designers. A common terminology would then develop for all the many design phases, simulation and design automation programs, design languages, etc., and better understanding and communication would develop across design group boundaries.

(iii) It is strongly urged that the output of the Architecture department be a formal, high-level description of the computer in the form of a running simulator of the system architecture. This simulator would have to be maintained and modified as the design proceeded into later phases. This simulator would, in effect, be the design of the machine with all later phases viewed as implementations of the design. The use of a high-level language for this description is emphasized to insure that the system description be readable and intelligible to all designers. With the design formalized at a high level the prediction of performance, modification, debugging and general understanding of the design would be greatly simplified and improved. Many of the essential functions in the total design process proposed in this memorandum are completely dependent upon the existence of this high-level system architecture simulator.

(iv) The design should be carefully "partitioned" at the earliest possible point in the design process (i.e., in Architecture) into functional segments that will be manageable by later design groups. Although it may be possible for a small group of people to design and comprehend the entire computer at the architectural level, it is not possible at later levels of design. The computer must be divided or partitioned among a number of groups of logic designers. If this partitioning is done in architecture along functional lines, the interfaces between partitions can be kept narrow and simple. These interfaces must be formally specified

L. Conway
Archives

in the high-level simulator and maintained throughout later phases of design.

The design process described in this memorandum, including the above suggestions and the many programs implementing the process, is not just a speculation as to what might be a good way to do things in the distant future. There is considerable practical experience within ACS with the various components of the process.

L. Conway
Archives

THE OVERALL DESIGN PROCESS

Let us now identify and define the fundamental stages of the overall design process. Then in the following sections of the memorandum each stage will be described in some detail.

The design and production of the computer passes through four rather distinct stages. The stages are identified by their final production of a "formal description" of the computer in a particular "language." The output of one stage is the input to the succeeding stage. Each stage of the process may be thought of as implementing or redescribing the design of the prior stage in a lower level language.

These stages are as follows (see Figure 1 for a visualization of the process):

System Architecture: This is the planning of the structure and function of the computer system, developed from a consideration of predicted market conditions and technology. The plan is developed to the level of detail of system description such that the complete function of the system is specified. The formal description produced by the architecture group would be a running system level simulation program written in a high-level language. The design would be carefully partitioned along functional lines into formally specified partitions with fairly narrow interfaces between them. The architectural design would consist of (i) variables and arrays in the high-level language symbolizing the various registers and control latches of the machine, and (ii) algorithms in the language expressing the functioning of the control latches and the flow of data between registers and functional units on a cycle to cycle basis.

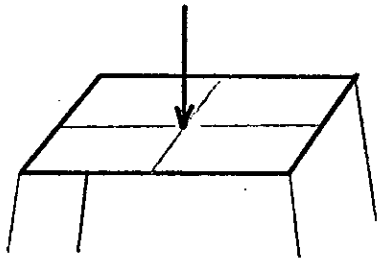
Logic Design and Engineering: The logic designers and engineers implement the structure and function of the architectural design in the logic circuitry and physical package of the chosen technology. The logic designer identifies and implements all the latches specified in the architectural design and designs combinational logic circuitry to connect the latches and implement the algorithms of the architectural design. This logic design must then be mapped onto real physical circuitry. This involves the selection of a circuit chip on which a given logic circuit is to be found, and the placement of that chip on a particular MCM on a board. The interconnections between all such chips, MCM's and boards must be specified. The output of

L. Conway Archives

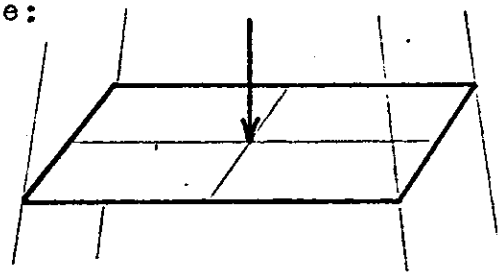
FIGURE 1: VISUALIZING THE STAGES OF THE COMPUTER DESIGN PROCESS:

Each stage produces a partitioned description of the machine design in a formal language. Each stage implements the design of the preceding stage in a lower level language, with the design then containing more detail but performing the same function. The partitions can pass thru the process independantly.

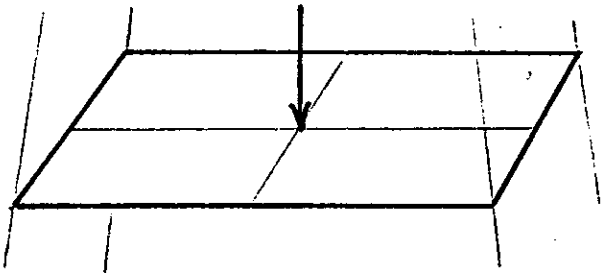
SYSTEM ARCHITECTURE: Produces the system level description of the machine: a system simulation program:



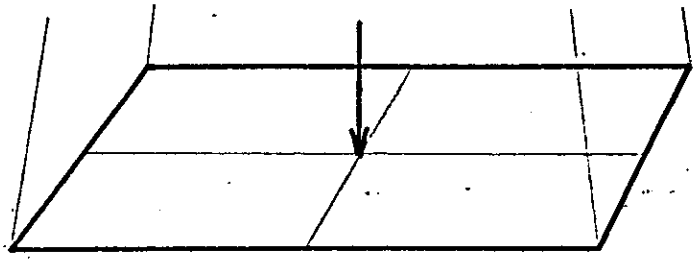
LOGIC DESIGN AND ENGINEERING: Produces the logic design and circuit placement and interconnections, specified in the DRKS language:



DESIGN AUTOMATION: Produces the physical files, a complete physical specification of the machine including wiring, bonding.



PROCESS AUTOMATION: Produces the wired circuit boards composing the computer:



L. Conway Archives

this design phase is a formal specification of the logic design, placement, and interconnections in the input language to the Design Record Keeping System (DRKS), which stores the design in a set of computer files. An alternative logic description language is now in development.

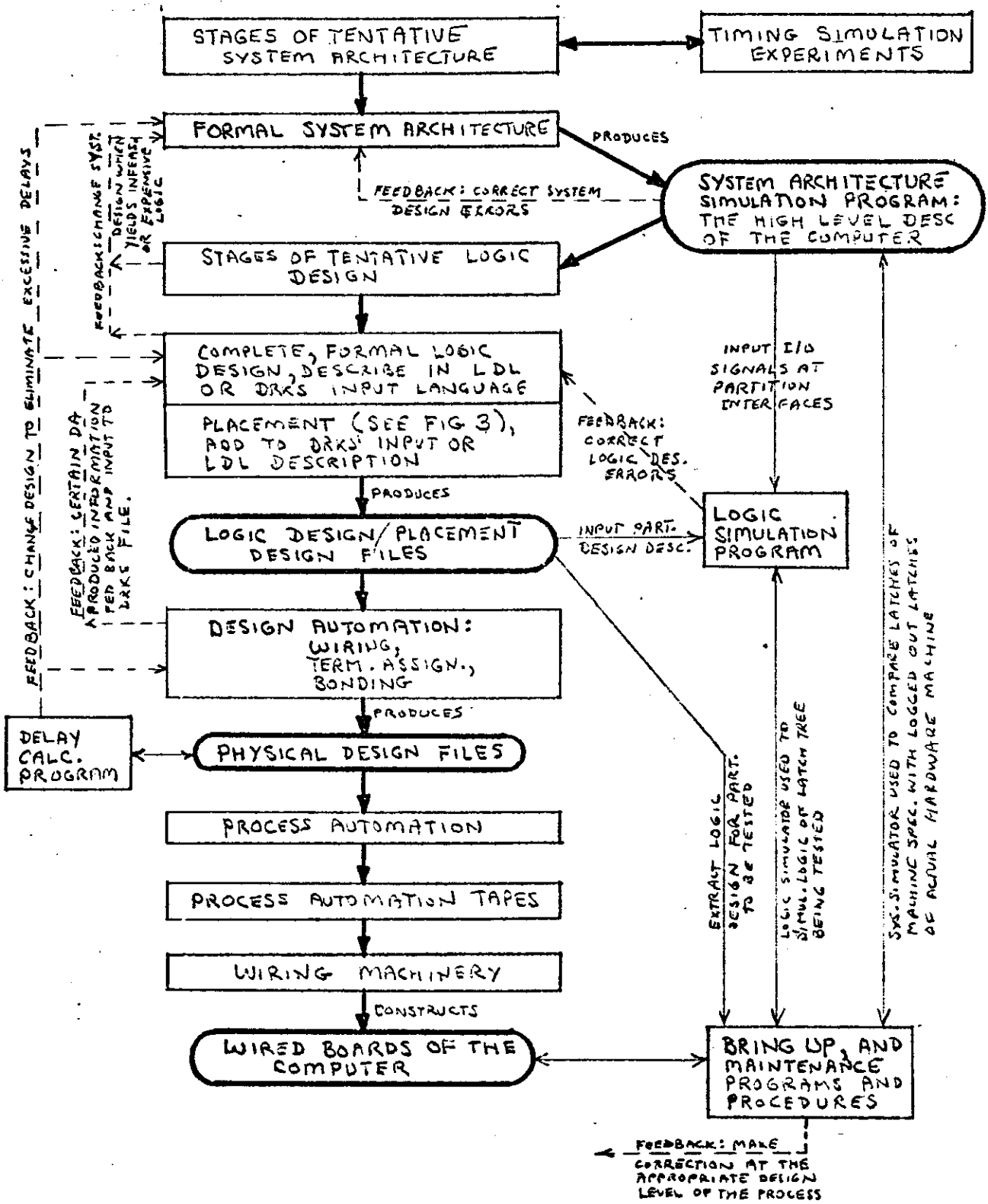
Design Automation: In the design automation phase a set of computer programs operate upon the design filed in DRKS to produce as output a complete physical description of the computer. This is done on a board by board basis. Note that in the DRKS system the various pads which must be interconnected to form a net are specified. However the actual route of wiring to connect these points is not. This wiring of all the nets on a board is computed by a wiring program. The pattern for bonding the wires to the pads is completed, and terminating resistors are assigned. The result of this design automation phase is a set of computer files which contain the complete physical description of all the boards of which the computer is composed.

Process Automation: We now have a complete physical description of all the boards. But how do we actually wire a board; what sequence of wire placements should we make? We must compute an orderly and feasible sequence of wire placements to be made by wiring machinery. The process automation programs operate on the physical files to produce a set of tapes which drive the wiring machinery through the proper sequence of operations to wire the boards of the computer. The output of this phase is the physical computer itself.

We are now ready to study the design process in more detail. Figure 2 is a flow chart of the stages of the design process which indicates the various computer programs used at each stage and the interaction of the various stages. This flow chart serves as a basis for the detailed descriptions of each stage which follow in the later sections of this memorandum.

L. Conway
Archives

FIG.2. FLOWCHART OF THE COMPUTER DESIGN PROCESS:



L. Conway Archives

SYSTEM ARCHITECTURE

The function of the system architecture phase of design is to produce a system-level specification of the machine. In the design process as described in this memorandum this specification is to be in the form of a running system simulation program.

Tentative System Design: The development of a system design which effectively meets cost and performance requirements calls for considerable experimentation with tentative system designs. The design will thus pass through these tentative, experimental phases until the experiments indicate that it is satisfactory. Then the design can be completely placed into a formal description.

Now, how can one experiment with a tentative computer design? It turns out that this is well established in ACS--by using a timing simulation program. See Reference 2 for a description of a past timing simulation effort, and Reference 1 for the simulation technique used in that effort.

The timing simulator is written at essentially the same level of description as the later system-level simulator and using the same simulation technique. However, it can be simpler and quicker to write because it does not require a data flow. Only the timing of control operations is relevant to timing simulation. The input to the timing simulator is the stream of instructions to be processed by the simulated computer, and the output of the simulator is a chart of the activities in the various machine registers, initiated by the instructions being processed, as a function of time. The detailed model of the proposed control structure can thus be tested quite accurately to predict performance and uncover design bottlenecks.

In order for timing simulation to really interact with and affect the system design, the simulator must be running while the system design is in development. This is only possible if

(i) The system architects really want a simulator, believe in its value, and help in its production.

(ii) The timing simulator is written in a high-level language. This will make algorithm production and documentation much easier than would assembly coding. Also, the timing simulator would be consistent with and a basis for the later system simulator.

L. Conway Archives

(iii) The architects participate in its writing.

If the simulator writer(s) must form all the detailed algorithms specifying a tentative design, then the simulator will lag the design by many months, perhaps 4 to 6 months. However, if the architects specify their tentative design in detail, then the coding of these designs would be a far simpler process and might lag specification by only one or two months.

This simulator should be partitioned along the same lines as the machine and interfaces identified early in the design processes. Then the separate partitions could be designed independently with unspecified partitions modeled in the simulator by dummy subroutines which roughly approximate the function of those partitions. In this way the entire machine can be simulated as early as possible even though some sections are not completely designed. Studies can then be made on those sections which have been designed.

Formal System Design: When timing simulation experiments indicate that the system design is satisfactory and unlikely to change greatly, the construction of a complete system simulator describing that design can begin.

The design will already have been partitioned. Engineers from the logic design groups assigned to implement these partitions could work along with the architects to write the system simulator. This simulator must be carried uniformly to the latch level of detail in order to be useful in later stages of design. The engineers could see that this requirement is met and that all algorithms specified for latch to latch operations in one cycle could probably be implemented in combinatorial logic without breaking the machine cycle.

There is experience in ACS with this sort of simulation, where a number of engineers write the program rather than having a simulation programmer do it. See Reference 4.

Note that this production of the system level design by both architects and engineers blurs the traditional boundary between the two functions. Both groups of designers work on the system level design, but from different orientations.

When the system description is complete, it can be run as a simulator and the design debugged at this level by running many actual programs on the "computer." As the later stages of design are completed,

L. Conway Archives

much information will be fed back to the architectural stage and force revisions in the system description. For example, many algorithms will not turn out to be realizable in logic in one cycle, and will have to be respecified, changing the system description. This system description must be accurately maintained if the design process as described in this memo is to function properly.

The availability of an accurate, maintained system level simulator will result in:

(i) Accurate performance prediction--potential users, compiler writers, etc., can run code on this simulator and predict machine performance and optimize their programs.

(ii) The logic design of the machine will proceed directly from the high-level description and thus will progress more rapidly and with better communication between design groups working on different partitions.

(iii) An effective logic simulation can be performed to compare the logic design of a partition with the system specification of that partition. The system level simulator can produce the input/output signals on the partition interface which can then be used to "drive" the logic simulator. More will be said about this very important logic simulation later in this memorandum.

(iv) Accurate system simulation plus accurate logic simulation will make possible the implementation of a very effective maintenance plan. This will be described later in this memo. See also Reference 6.

The significance and importance of the system level simulator cannot be overemphasized. It must be produced and maintained for the proposed scheme to work. The higher the level at which a design is formally specified, the easier it is for everyone involved to fully understand the design, experiment with it, and change and debug that design.

This system level simulator should really be viewed as "the machine." All later design and automation of design and manufacture should be viewed as implementations of the system design.

L. Conway Archives

LOGIC DESIGN AND ENGINEERING

This stage of the design process produces an implementation of the structure and function of the architectural design in the logic circuitry and physical package of the chosen technology.

In a manner similar to the system design, the logic design and engineering pass through two phases: (i) a tentative phase where attempts are made at implementation, often resulting in revisions being made in the system design, and (ii) a formal phase where the formal description of the logic and physical placement is produced.

Tentative Logic Design: When a partition of the system has completed tentative system design and is ready to be formalized in the system level simulator, then the tentative logic design of that partition may begin. The tentative logic design is the attempt at implementation of the system partition in logic circuitry and package. These early attempts will fail because many of the system algorithms will not be realizable in one machine cycle of logic. A strong interaction must exist between those persons producing the formal system specification and the logic designers. The tentative logic design efforts must feed back enough information such that the formal system description will have most of the algorithms checked for feasibility of implementation in logic and package without breaking the machine cycle time. For this reason it is suggested that at least one of the logic designers who works on the tentative logic design of a partition also work along with the architect for that partition and participate in the formation of the system level description. In this way the partition of the system will not only reflect architectural requirements, but will be implementable, as described, in logic.

These early, tentative logic design and placement efforts will probably be specified nonformally. The designs at this stage are traditionally sketched out as logic circuit diagrams on "yellow sheets." Rough approximations of circuit placement can be made, and then estimates of delays and circuit counts can be generated. These estimates will be fed back, and perhaps modify the system design and/or the logic design.

380

L. Conway Archives

Formal Logic Design and Placement: When tentative logic design studies have produced sufficient feedback to finalize the system design, then the formal logic design and placement can begin. The formal logic design must implement in logic circuitry the function of the system design. The behavior of a partition of the machine, as seen at its interfaces, must be the same at both levels of design, system and logic.

There are two aspects to this implementation of the system design: the implementation of the system function in logic circuitry and the mapping of that circuit design onto real hardware.

Currently the logic design phase is done by the designer with no computer assistance. The mapping of the logic design onto hardware and the placement of the different levels of hardware may be done in part, or perhaps entirely by computer programs.

The mapping or partitioning of logic circuitry onto hardware and the placement of levels of hardware involves the following levels: logic circuitry maps onto circuit chips, circuit chips are placed on MCM's, and MCM's are placed on the board.

There are a number of possible techniques that might be used to accomplish the placement which involve varying amounts of computer assistance to the designer. Some methods being considered for ACS use are

(i) In current use is a method where the designer must partition the logic onto chips by hand, and then a sequence of computer programs places the chips on MCM's on the board.

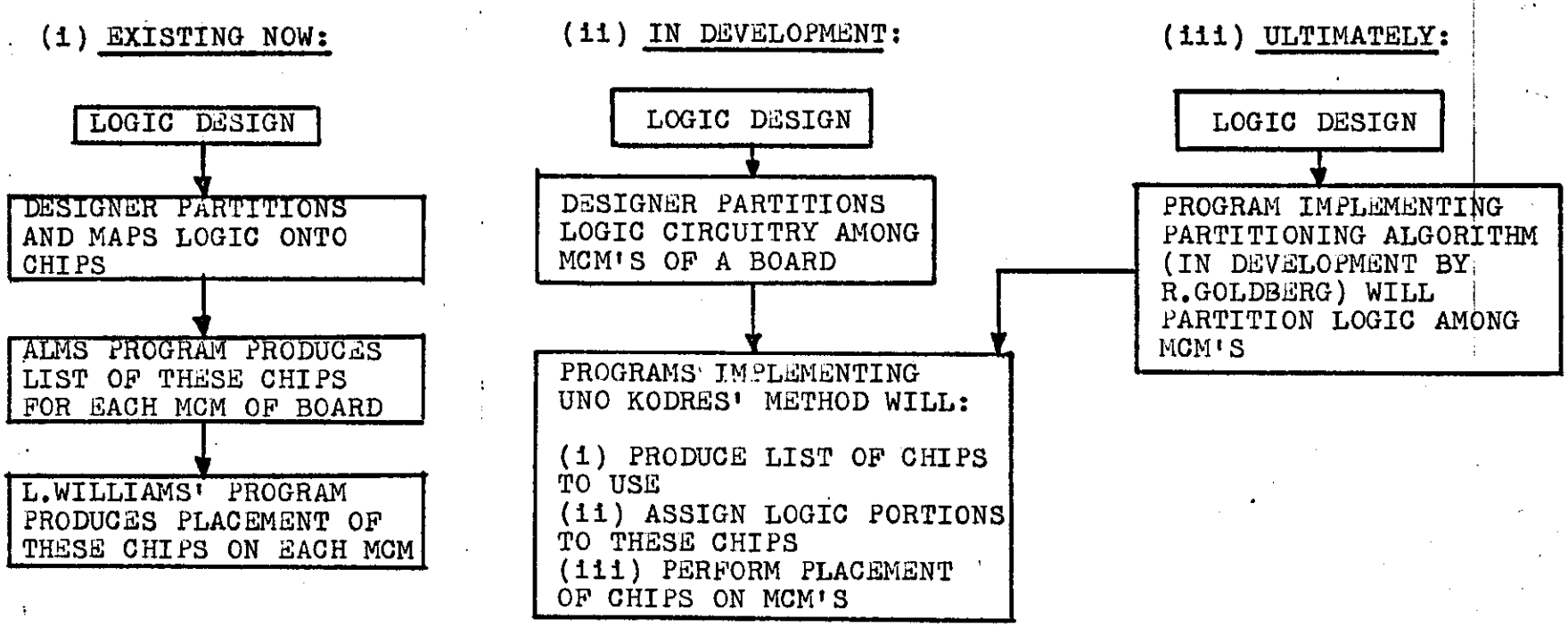
(ii) In development is a placement system which will require that the designer merely partition the logic among MCM's. The selection of chips, assignment of logic to chips and placement of chips on MCM's on the board would be accomplished by computer programs. See Reference 5 which summarizes Dr. U. Kodres' work in this area.

(iii) It may eventually be possible to have the partitioning of logic among MCM's be automated also, thus automating the entire partitioning and placement process. Mr. R. Goldberg is working on this partitioning algorithm. Also, Research has developed a program, ALMS, which may be applicable.

These three placement schemes are summarized in the flow charts in Figure 3.

L. Conway Archives

FIG.3. POSSIBLE PLACEMENT TECHNIQUES:



L. Conway Archives

Formal Description of Logic Design/Placement: The output of the formal logic design and placement is a formal description of the design at this level. The language in which this description may be placed is the DRKS input language. DRKS is the design record keeping system which files the logic design and placement information.

An unfortunate aspect of the DRKS language is that it imposes a totally arbitrary level of partitioning on the design description: the ALD sheet (logic diagram sheet). The design is input to DRKS by drawing logic diagrams on sheets of a fixed size and then describing the drawing by statements in the DRKS language.

This partitioning onto sheets is usually too fine to correspond to any useful design partition. The designers' partition of the machine and even various functional entities within that partition will contain logic circuitry requiring many, many ALD sheets to describe. The language used to input DRKS is awkward to use, and describes the sheets rather than the logic directly. The statements of the language are usually formulated by someone other than the designer, who merely sketches the sheets.

It is strongly suggested that an alternative Logic Description Language (LDL) be developed and used so that the designer can more easily specify his logic design in a formal language. In this way the processing and understanding of the logic designs might be improved greatly. Dr. J. Cocke has proposed a tentative version of such a language. Dr. R. Love, Mr. P. Shivdasani and I are now working on completing the specification of this language.

An important reason for the use of sheets as the formal logic design description has been the traditional use of these sheets by CE's who maintain the hardware. As we shall see later in this memorandum (Section 6), the importance of the sheets may be reduced because their use by CE's can be minimized by using improved maintenance methods.

If the ALD sheet were needed, perhaps in some central maintenance facility, a form of ALD sheet could be generated by program from the design files formed from LDL input. Thus there is no real reason for requiring that the design be specified by sheets initially.

Another development which might really de-emphasize the importance of ALD's is the possible use of prototype sheets. This plan involves the use of a very limited total number of chip-types. Each chip would be described by a prototype sheet. There would thus be only a limited number of possible sheet types. These could be stored as macros in a file. A design would be described by program statements

L. Conway
Archives

indicating the interconnection of such chips. No actual sheet input would be necessary as the sheet would be implied by chip type. Thus the logic could easily be described by a simple form of LDL. Appropriate ALD sheets could be very easily generated by program on those rare occasions when someone really needed to look at them.

Logic Simulation: When the logic design of a partition of the machine has been completed and formally described, it is very desirable to verify that the logic design correctly implements the architectural specification of the partition before going any further into the design automation and process automation phases. An error found at this stage will be much easier to correct than if found later on.

This verification of the logic design is performed using a logic simulation program. A partition of the design can be simulated on this program. Input signals are supplied at its interface and the logic simulator produces the output signals at the interface.

The major problem in this sort of logic simulation is the generation of test cases of interface input signals and expected output signals. The generation of a large enough set of such signals to moderately debug a partition of logic would be a very costly process if done manually. It would probably be possible to generate only a rather small number of such tests.

There is a solution to this problem. If the system level simulator and logic description of a partition are really different levels of description of the same entity, then they should behave the same at the partition interface. Thus it would be possible to run a program on the system level simulator and store all the I/O signals on a partition's interface while the program is running. Then these signals could be used to input and compare against the logic of the partition when it runs on the logic simulator. In this way many tests could be automatically generated. The tests would be consistent over the whole machine; if we debugged the logic of all partitions on a given program, then when we put all partitions together later, they might all function properly together when running that program.

This idea of using two levels of simulation to debug the logic design has been extensively studied and described in an earlier memorandum. See Reference 3.

Figures 4 and 5 graphically portray the idea of a Logic Simulation System (LSS) using two simulators: a system simulator which provides input/output signals for the partition which runs on a logic simulator.

384

L. Conway Archives

FIG4. THE BASIC IDEA OF LSS:

APPLY SAME INPUT TO BOTH LEVELS OF SIMULATION AND COMPARE OUTPUTS. IF OUTPUTS ARE DIFFERENT THEN ERROR EXISTS IN LOGIC DESIGN.

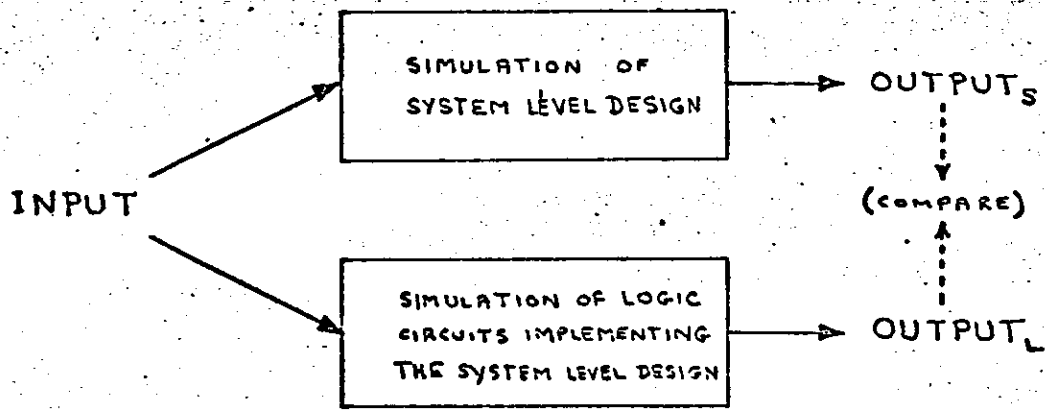
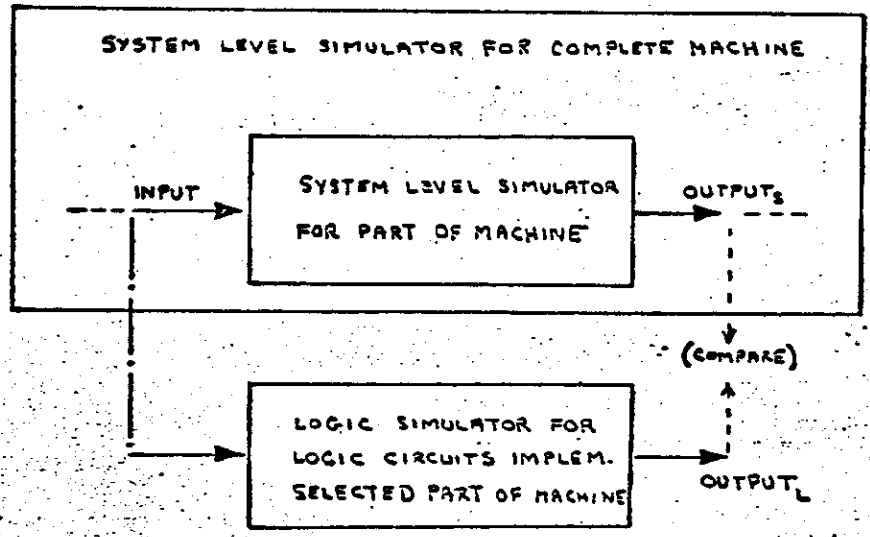


FIG5. AUTOMATIC GENERATION OF SYSTEM LEVEL INPUT/OUTPUT, LOGIC SIMULATOR INPUT:

SYSTEM LEVEL DESIGN IS IMBEDDED IN SYSTEM LEVEL SIMULATION OF ENTIRE MACHINE. WHEN THIS SIMULATOR RUNS WE AUTOMATICALLY GENERATE (AND SAVE) THE I/O AT THE DESIGN INTERFACE. WE MAY LATER APPLY THESE INPUTS TO THE LOGIC SIMULATOR FOR THE SAME DESIGN AND COMPARE THE LOGIC OUTPUTS WITH THE SYSTEM LEVEL OUTPUTS.



L. Conway Archives

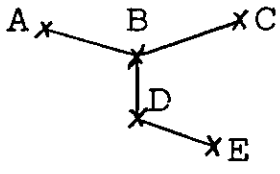
DESIGN AND PROCESS AUTOMATION

Suppose we now have a verified logic design along with physical placement information resident in the DRKS files. There is still a long way to go before the machine can actually be constructed. The remainder of the design process is completely automated, however.

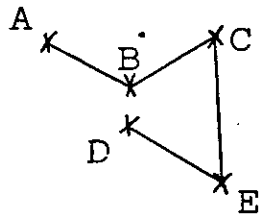
The steps in the design automation process are as follows (greatly simplified):

(i) The records describing the logic design and placement for a board are selected from the DRKS files.

(ii) The nets on the board must now be wired. This involves determining the best path for wiring together the points of a net subject to the wiring rule constraints. For example, given that points A, B, C, D, E must be wired together we must decide whether to wire as in (a), (b) or some other way.



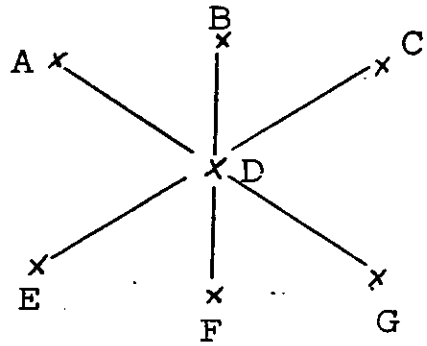
(a)



(b)

(iii) When the wiring has been calculated for the nets, we must assign the location of terminating resistors for the nets.

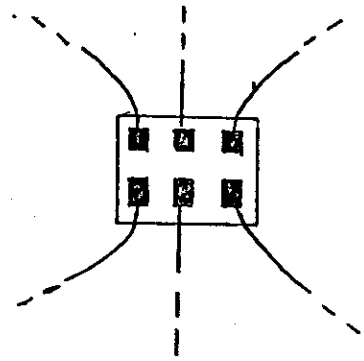
(iv) Suppose we have wired A, B, C, D, E, F, G together as follows:



L. Conway Archives

We must now decide how to bond the wires on each pad of the net. In the above example, D would be bonded as follows:

D:



The actual DA programming becomes somewhat involved because a situation may arise in the later stages of processing which cannot yield a solution, and this will have to be fed back to the earlier phases and a new pass made through the DA programs.

After the design automation is completed, we have in a "physical file" the complete physical specification of the boards of the machine.

At this point we have sufficient information to perform delay calculations to determine the circuit and wiring delays in various paths through the machine. Computer programs can be written to perform these calculations. Excessive delays will necessitate design changes.. This raises an interesting point: We have proposed four formal specification levels for the design. Thus, we can envision four levels of design simulation: system, logic, "A-C" logic including delays, and finally actual running hardware.

Unfortunately, the "A-C" logic simulation, including physical delays, is not really feasible for a machine of the size we are designing. Even the usual logic simulation must be partitioned, and the AC logic simulation includes much more detail. So all we can do at this level is delay calculations on paths through the hardware. It is of theoretical interest however to note that with sufficient machine power a simulation at the physical level could be performed and make this stage of the proposed process similar to the preceding stages in the use of simulation to verify the design.

L. Conway Archives

The phase in the design process which results in the production of actual hardware is the process automation phase. After appropriate reformatting, the information in the physical file describing a board is input to the process automation programs. These programs produce as output the tapes which drive the wiring machinery which actually constructs the boards of the computer.

Now, how can the boards (or MCM's) produced by the process automation be debugged? Even if the design at the system level and logic design level is error free, defects or errors may have been introduced in the manufacture or wiring of the circuitry.

It is possible to partially debug the hardware in an economical manner by using the two levels of simulators to generate test signals.

The signals could be generated as follows: The system simulator can produce input signals for the logic simulator while running a particular program. This would be done for the logic simulation of the partition of the machine which contains the hardware to be tested (usually the hardware would be a small subset of a partition). All of the signals internal to the partition are generated during the logic simulation. Thus the signals at the interface of the hardware to be tested could be extracted, and filed, while running the logic simulator.

Of course this method of debugging is only partial. Not all possible input-output test patterns would be generated for the hardware. However, this is a very special form of partial debugging: the same program could be run on the system simulator to generate tests for all hardware components. Thus, although only partially debugged, the hardware will run that particular program when it is all put together.

The key point to note is that the partial debugging is uniform over the whole machine. Of course many programs could be run--the number depending on the economics of the situation. Diagnostic programs could be used for this hardware test generation. Then the machine, when constructed, would run the diagnostics to isolate residual hardware errors under normal maintenance procedures.

Note that if each piece of hardware were very thoroughly, but not completely, debugged with traditional methods, there would be no assurance that any program would run when the pieces were put together.

L. Conway
Archives

Thus, the partial, but uniform, test generation could be a very economical method of quickly getting hardware to the point where it will run at least some programs when integrated into the whole machine.

This could serve as a basis for planning the bring-up of the machine.

L. Conway
Archives

MAINTENANCE

The design process is not completed with the wiring and construction of the computer. A bring-up of the computer must be accomplished and the machine must be maintained. Bring-up may uncover design errors at any of the stages of design. In addition to the correction of hardware failures, maintenance will involve the installation of engineering changes. Thus, both of these activities involve cycling back through the design process and both are strongly tied into the network of simulation and automation programs used in the design process.

At this time the bring-up process has not been completely defined. However, a complete maintenance procedure has been defined by Dr. D. G. Keehn (See Reference 6). This plan will be briefly described here to indicate how it depends upon the simulation programs. Some leads to ways of planning bring-up might be uncovered in this maintenance plan. The scheme functions as follows:

(a) Diagnostic programs running on the ACS computer detect an error. The program causing the error is identified.

(b) The error producing diagnostic program is repeated on both the ACS computer and on the system architecture simulator running on a smaller diagnostic computer. The ACS computer's latches are logged out each cycle and compared to the latches of the simulator. The failing latch and cycle of failure are identified.

(c) A traceback program is run on the diagnostic computer, operating on the logic files, to find all latches which could set/reset the failing latch in one cycle. This is the latch tree of the failing latch.

(d) All scopeable points in the logic of the selected latch tree are found from the design files and output by another program running on the diagnostic computer.

(e) The logic of the latch tree is extracted from the design files. A logic simulation of the latch tree is performed for the cycles of interest: the cycle preceding failure and the failing cycle. The scopeable point values are output for these cycles.

L. Conway Archives

(f) A technician can now scope the ACS machine at the appropriate points and compare the values with the above values for the cycles of interest. This will isolate the point of error.

(g) The technician then decides what unit of hardware to pull and replace in order to correct the failure.

There are some very interesting operational characteristics in this maintenance plan:

(i) The diagnostic computer can be physically distant from the ACS machine being repaired with communication between the two locations handled by teleprocessing. Thus, one central diagnostic computer and maintenance system could maintain several ACS machines in the field.

(ii) The person repairing the machine in the field need not be a CE in the usual sense. He could be a technician instead, for no knowledge of the functioning of computer logic would be required to perform repair work.

(iii) Because of (ii), it is clear that the distribution of ALD sheets to many CE's in the field would not be necessary. The significance of these sheets is thus greatly reduced.

This particular maintenance plan has significant advantages over previous plans. These advantages are bought at a price: dependence on the existence of accurate system architecture and logic simulators.

L. Conway
Archives

CONCLUSIONS

We have now covered all the phases of the design process in some detail. For the sake of simplicity and brevity, the presentation has treated these phases as separate activities which follow each other in a serial manner.

The actual design situation is obviously far more complex and requires careful planning, scheduling and management of human and machine resources. There are three factors in the process (not fully developed in this initial memorandum) which lead to this additional complexity:

(i) Design phases do not follow serially, but overlap in time. For example, the tentative logic design may be proceeding while the formal system specification is still in process.

(ii) There is a relative independence of the design of different partitions. We might be far along in the design process on one partition of the machine, but only experimenting at the system level with another partition.

(iii) There is consistent feedback (as indicated in Figure 2) from later phases of design to earlier phases. Very often the design at a given phase cannot be feasibly or economically implemented at a later stage and must be modified.

Therefore this basic plan for the design process must be made considerably more detailed and account for these additional complexities before it is really a working plan for the process.

This elaboration of the plan will have to await the feedback produced by this memorandum.

In conclusion, it is felt that the suggestions proposed in this memorandum, especially the fundamental uses of the system simulation program, can lead to a workable system plan for the whole computer design process if they are properly elaborated and detailed.

A key factor in reaching this conclusion is the existence of practical experience within ACS in the separate phases of the plan.

It is hoped that this memorandum will stimulate discussion and new ideas on this subject. Your comments and criticisms concerning the various suggestions made herein are welcomed by the author.

392

L. Conway
Archives