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:**

<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>PAGE:</th>
</tr>
</thead>
<tbody>
<tr>
<td>i. Overview of Archive Papers and Documents.</td>
<td>i. 1</td>
</tr>
<tr>
<td>ii. Letter to Dr. Smotherman, January 2, 1999.</td>
<td>ii. 1</td>
</tr>
<tr>
<td>1. &quot;Dynamic Instruction Scheduling&quot;, February 23, 1966.</td>
<td>001</td>
</tr>
<tr>
<td>2. &quot;ACS Simulation Technique&quot;, March 15, 1966.</td>
<td>022</td>
</tr>
<tr>
<td>3. &quot;Dual Arithmetic on ACS-1&quot;, May 1, 1967.</td>
<td>051</td>
</tr>
<tr>
<td>6. MPM Architecture and Simulator Reference Notebook, as of August 1967.</td>
<td>093</td>
</tr>
<tr>
<td>7. Timing Simulator Source Code Listings, as of August 1967.</td>
<td>211</td>
</tr>
</tbody>
</table>

* 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.
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.

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.

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.


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


093  A Unit Interlock Simulation: A primer based on the sort of code used in the Timing
Simulator. Hardware diagrams, flowcharts and code are condensed from the actual
simulator, and give the essentials of A-Interlocks for a simpler "ACS-like" machine.
Also constitutes a tutorial on the micro-architecture of the A-Unit Interlocks.

103  Facility Structure:
Some details of the XFAC's, AFAC's, INBUS #'s, OUTBUS #'s, delays;

111  OP Decode Tags:
Contains tabulation (unary) of all decode tags for the 227 instructions,
I.e., the internal claims on facilities, busses, etc., for all OPs,
in a 256 by 70 table for the instruction set of April 17, 1967.

143  Various flowcharts and notes:
Definitions of simulator Common Variables; LJ indexing of A-SD's, X-SD's.
More on the decode tags, format of XBUFF and ABUFF.
Bussing of OPs to A and X Buffers.
Format of Execution Simulator output cards; Example of Output.

152  Various architectural and simulator details:
Block diagram of machine's major dynamic instruction modules.
Flow charts for key functional module routines.
"Event running times within the cycle", in 0.1's of a machine cycle.
Stack to Register timing: key difference between A and X stack algorithms,
bussing and facilities.
"Full Bypassing" timing; "No Bypassing" timing.
Common Vars, "Revised 18 May 1967", Common Vars, "Before Revision".

168  Memory timing details:
Memo to file by G. T. Paul re "MPM-BLCU Interface for Store OPS", 5-24-67,
with diagrams by M.E.H., G. P., 5-17-67, revised 6-7-67.
Memory Timing Diagram; Routines re memory instructions.
Instruction fetching overview.
Handling the Back-Up Registers - overview.
M. Homan's notes re Back-Up Logic, as of about a year earlier: 7-25-66.

189  Skips, Branches and Exits:
SKIP instruction overview; Execution of EXIT instruction -overview.
BRANCH and EXIT Handling, complete details of, in a coordinated, hand-written
"memo" of 3-27-67 by B. O. B. (?), along with similar memo re "old branch info"
by B. O. B. dated 3-17-67, followed by detailed timing diagrams.
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.


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, flowcharts 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.
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].
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.
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.
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

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

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

The order in which the instructions comprising a program are to be executed 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)

\[
\begin{align*}
R_1 + R_2 & \rightarrow R_3 \\
R_1 \times R_4 & \rightarrow R_5 \\
R_6 + R_2 & \rightarrow R_7 \\
R_3 \times R_6 & \rightarrow R_8
\end{align*}
\]

might well be preferable to

\[
\begin{align*}
R_1 + R_2 & \rightarrow R_3 \\
R_6 + R_2 & \rightarrow R_7 \\
R_1 \times R_4 & \rightarrow R_5 \\
R_3 \times R_6 & \rightarrow R_8
\end{align*}
\]

if the adder and multiplier were independent functional units.
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].
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 mechanism. (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.
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:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Source Matrix</th>
<th>Destination Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $R_3 + R_4 \rightarrow R_7$</td>
<td>1 2 3 4 5 6 7 8</td>
<td>1 2 3 4 5 6 7 8</td>
</tr>
<tr>
<td>2. $R_7 \times R_2 \rightarrow R_4$</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>3. $R_1 + R_2 \rightarrow R_5$</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>4. $R_8 \div R_1 \rightarrow R_8$</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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.
(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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Source Matrix</th>
<th>Destination Matrix</th>
<th>Busy Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_3 + R_4 \rightarrow R_7$</td>
<td>1 2 3 4 5 6 7 8</td>
<td>1 2 3 4 5 6 7 8</td>
<td>1 1 1</td>
</tr>
<tr>
<td>$R_7 \times R_2 \rightarrow R_4$</td>
<td>1 1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$R_1 + R_2 \rightarrow R_5$</td>
<td>1 1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$R_8 \div R_1 \rightarrow R_8$</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

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.
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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Source Matrix</th>
<th>Destination Matrix</th>
<th>Busy Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $R_3 + R_4$</td>
<td>$R_7$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. $R_7 \times R_2$</td>
<td>$R_4$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. $R_8 + R_1$</td>
<td>$R_8$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. $R_6 - R_3$</td>
<td>$R_3$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
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 mechanism, 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.
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.
THE SEQUENCING OF STORAGE ACCESSES

Another area where dynamic scheduling can be of value is the sequencing of accesses interleaved storage. Such storage is characterized 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 determined 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 \((S_A, D_A)\), 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
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.
### 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. \( R_4 - R_1 \rightarrow R_2 \)

### STORAGE ACCESS SCHEDULER

- **SA**
  1. 1:53 \( R_3 \)
  2. '91' 2:29
  3. 3:25 \( R_1 \)
  4. --

- **DA**

- **BA**

---

**Fig. 4** Example of a 4 Register, 4 Storage Box Machine
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.
### INSTRUCTION SCHEDULER

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>D</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R₁ + R₂ R₃</td>
<td>1 2 3 4 S</td>
<td>1 2 3 4 S</td>
</tr>
<tr>
<td>2</td>
<td>S[R₁ + 2] R₃</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>R₂ S[R₁ + 1]</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>R₁ × R₃ R₁</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>R₄ - R₁ R₂</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>R₂ S[9]</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### STORAGE ACCESS SCHEDULER

<table>
<thead>
<tr>
<th></th>
<th>Sₐ</th>
<th>Dₐ</th>
<th>Bₐ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1:53 R₃</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>2</td>
<td>3:25 R₁</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4:57 R₄</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Fig. 5. The Example of Fig. 4 One Cycle and One Bus Slot Later*
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 be set 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.
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 insofar 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 corresponding functional 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\textsuperscript{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 assuming 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.
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.
**Fig. 5.** Example of Multiple Instructions per Cycle Initiation—Cycle 1.

**Fig. 6.** Example of Multiple Instructions per Cycle Initiation—Cycle 2.
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.
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.
REFERENCES


MEMORANDUM TO:

SUBJECT: ACS Simulation Technique

D. P. Rozenberg

L. Conway

R. H. Riekert

cc: Architectural Distribution List

March 15, 1966
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.
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 (ll) 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.
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.
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 calendar of future events as the related cause statements are encountered. The calendar is ordered according to increasing time of occurrence. When an event terminates (i.e., the event subroutine terminates), the ordering of the calendar 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 calendar. 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 calendar.

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 calendar 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 calendar, 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 calendar 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
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 calendar 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 calendar manipulation. The system variables are those vari-
ables in terms of which the programmer describes his simulation
model.

The calendar consists of six single indexed arrays which are indexed
by the same pointer. Thus the calendar 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 calendar manipulation.
Assume that the calender is in the state given in figure 1.
<table>
<thead>
<tr>
<th>Index</th>
<th>Link</th>
<th>Time</th>
<th>Event Reference</th>
<th>Par 1</th>
<th>Par 2</th>
<th>Par 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>10^{30}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1.0</td>
<td>Event 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>2.0</td>
<td>Event 17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1.5</td>
<td>Event 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4.0</td>
<td>Event 9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>3.0</td>
<td>Event 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ISL=7, ITL=2

FIGURE 1
Assume that the next encountered utility call is

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

The result is shown in figure 2.

<table>
<thead>
<tr>
<th>Index</th>
<th>Link</th>
<th>Time</th>
<th>Event Reference</th>
<th>Par 1</th>
<th>Par 2</th>
<th>Par 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>10^30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1.0</td>
<td>Event 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>2.0</td>
<td>Event 17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1.5</td>
<td>Event 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4.0</td>
<td>Event 9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>3.0</td>
<td>Event 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>3.25</td>
<td>Event 12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ITL=2, ISL=8

**FIGURE 2**
If the next encountered utility call is:

```
CALL TSTEP
```

The result is given in figure 3.

<table>
<thead>
<tr>
<th>Index</th>
<th>Link</th>
<th>Time</th>
<th>Event Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>$10^{30}$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>2.0</td>
<td>Event 17</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1.5</td>
<td>Event 3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4.0</td>
<td>Event 9</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>3.0</td>
<td>Event 5</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>3.25</td>
<td>Event 12</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ITL=4, ISL=2**

**FIGURE 3**
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 straightforward 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.

<table>
<thead>
<tr>
<th>Event Names</th>
<th>Subroutine Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE</td>
<td>X MOVE</td>
</tr>
<tr>
<td>GENER</td>
<td>X GENER (A)</td>
</tr>
<tr>
<td>DELAY</td>
<td>X DELAY</td>
</tr>
<tr>
<td>PROC</td>
<td>X PROC (X,Y,Z)</td>
</tr>
<tr>
<td>FINIS</td>
<td>X FINIS</td>
</tr>
</tbody>
</table>

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
```

- SYSTEM INITIATION STATEMENTS
- CALENDER INITIALIZATION STATEMENTS

MOVE = 1
GENER = 2
-DELAY = 3
PROC = 4
FINIS = 5
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
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 calendar 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
```
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 control 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.
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:

```
Generator of Memory Requests → Memory Queue → Memory Boxes
```

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.
$IBFTC GENER

SUBROUTINE XGENER

COMMON X

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 X

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
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 __ __ __
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

MCCYCC

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

$IBFTC MCCYCC

SUBROUTINE XMCCYCC (BOXNO)
COMMON -- 
C AT MEMORY CYCLE COMPLETION, FREE BOX
MEMBSY (BOXNO) = 0
RETURN
END
STATS

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

$IBFTE 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
including the statement number 1000 in MAIN are the instruction necessary to cycle thru the events in the calendar.

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), QMNPNT, INUMB
INTEGER QM,QMNPNT
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 ZEROS. 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
```
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
REFERENCES


Appendix A

Listings of utility routines for Method 1
SURORoutine 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)

CAUSE ENTERS EVENTS ONTO CALENDAR

ITL IS LOCATION OF FIRST EVENT IN CALENDAR
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
30 IF (NEXT.EQ. ITL) GO TO 40
   LINK(LAST)=ID
   CTIME(ID)=T
   NEVENT(ID)=IEV
   KOL1(ID)=IP1
   KOL2(ID)=IP2
   KOL3(ID)=IP3
   RETURN
40 ITL=ID
GO TO 30
END
SUBROUTINE REMOVE(EVENT,TIME,I,J,K)
COMMON TIME,IPAR1,IPAR2,IPAR3,ITL,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=XLINK(NEXT)
IF(NEXT.EQ.0) GO TO 30
IF(NEXT(NEXT).NE.EVENT) GO TO 10
WE FOUND EVENT
CTIME=CTIME(NEXT)
I=KOL1(NEXT)
J=KOL2(NEXT)
K=KOL3(NEXT)
XLINK(LAST)=XLINK(NEXT)
XLINK(NEXT)=ISL
ISL=NEXT
RETURN
C EVENT IS FIRST IN LIST
20 CONTINUE
CTIME=CTIME(NEXT)
I=KOL1(NEXT)
J=KOL2(NEXT)
K=KOL3(NEXT)
ITL=XLINK(ITL)
XLINK(NEXT)=ISL
ISL=NEXT
RETURN
C EVENT NOT PRESENT
30 CONTINUE
CTIME=TIME
I=0
J=0
K=0
RETURN
END
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
Appendix B

Listings of utility routines for Method 2
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
25 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
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

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
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

IN=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
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:

\[
\begin{pmatrix}
A_1 \\
A_2
\end{pmatrix}
\begin{pmatrix}
\phi \\
\phi
\end{pmatrix}
\begin{pmatrix}
B_1 \\
B_2
\end{pmatrix} \rightarrow
\begin{pmatrix}
C_1 \\
C_2
\end{pmatrix} =
\begin{pmatrix}
A_1 \phi + B_1 \\
A_2 \phi + B_2
\end{pmatrix}
\]

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.
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).
Dr. J. E. Bertram
May 1, 1967
Page 3

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
May 12, 1967
Advanced Computing Systems
Menlo Park
986/031

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
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.
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)

<table>
<thead>
<tr>
<th>Step Description</th>
<th>Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage bus to IB's</td>
<td>0</td>
</tr>
<tr>
<td>IB to dispatch register</td>
<td>1</td>
</tr>
<tr>
<td>Dispatch to contender</td>
<td>1</td>
</tr>
<tr>
<td>Contender to functional unit</td>
<td>1 (2 in A-unit)</td>
</tr>
</tbody>
</table>

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:

1. interlock tests on BRANCH cycle 1
2. compute eba, successful/unsuccessful cycle 2
3. 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.
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.

<table>
<thead>
<tr>
<th>if top DO entry is</th>
<th>correct</th>
<th>incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IB</td>
<td>IB</td>
</tr>
<tr>
<td>if required instructions in</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsuccessful branch exit</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>successful branch exit</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>no exit (normal sequence)</td>
<td>1</td>
<td>2*</td>
</tr>
</tbody>
</table>

*pathological case (hence unimportant)

IX. Next-fetch mechanism

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

X. Computation dependent SK I P s

The following sequence of instructions illustrates the problem

\[ A^3 \leftarrow \text{any A instruction} \]
\[ C_2 \leftarrow A^3 \geq A^{10} \]

SKIP if \( C_2 \) or \( C_{30} \)

* any A instruction

The data/control sequence is

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

\[ A^3 \leftarrow \text{any A instruction} \]
\[ C_2 \leftarrow A^3 \geq A^{10} \]
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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>3/1 and 4/1</td>
</tr>
<tr>
<td>MPY</td>
<td>3/1</td>
</tr>
<tr>
<td>DIV</td>
<td>10/7 or 10/8</td>
</tr>
<tr>
<td>CMP</td>
<td>1/1</td>
</tr>
</tbody>
</table>

Floating point, 96-bit

<table>
<thead>
<tr>
<th>Operation</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>4/1</td>
</tr>
<tr>
<td>MPY</td>
<td>5/3</td>
</tr>
<tr>
<td>DIV</td>
<td>17/14</td>
</tr>
<tr>
<td>CMP</td>
<td>1/1 (maybe 2/1)</td>
</tr>
</tbody>
</table>

Floating point, mixed

<table>
<thead>
<tr>
<th>Operation</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPY</td>
<td>3/1</td>
</tr>
<tr>
<td>DIV</td>
<td>10/7</td>
</tr>
</tbody>
</table>

Integer

<table>
<thead>
<tr>
<th>Operation</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>2/1</td>
</tr>
<tr>
<td>MPY</td>
<td>4/2</td>
</tr>
</tbody>
</table>

Index integers

<table>
<thead>
<tr>
<th>Operation</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>1/1</td>
</tr>
<tr>
<td>MPY</td>
<td>4/2 (improve to 3/1)</td>
</tr>
</tbody>
</table>

Shift, logic, moves

<table>
<thead>
<tr>
<th>(A and X)</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1/1</td>
</tr>
</tbody>
</table>


Subject: MPM Timing Simulation

Reference:
1. ACS AP #68-022, 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
CONTENTS

Introduction 0-1
The Unroller 1-1
The Timing Simulator 2-1
Current Job Running Procedures 3-1
Table of Implemented Instructions 4-1
Planned Modifications 5-1
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.
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 (S 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.
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
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.
EXAMPLE: UNROLLER INPUT DECK

<table>
<thead>
<tr>
<th>LOOP:</th>
<th>CGEX</th>
<th>2,4,3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAND</td>
<td>2,2,0,0</td>
<td>2LOOP,*</td>
</tr>
<tr>
<td>CGEN</td>
<td>1,1,2</td>
<td></td>
</tr>
<tr>
<td>AXK</td>
<td>3,3,0,0</td>
<td></td>
</tr>
<tr>
<td>AN</td>
<td>1,1,8</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>8,0,0,1000</td>
<td></td>
</tr>
<tr>
<td>AN</td>
<td>2,2,9</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>9,0,0,2000</td>
<td></td>
</tr>
<tr>
<td>SKOR</td>
<td>1,1 (*,2DUMMY)</td>
<td></td>
</tr>
<tr>
<td>MN*</td>
<td>1,1,2</td>
<td></td>
</tr>
<tr>
<td>EXIT</td>
<td>STA</td>
<td>1,0,0,1000</td>
</tr>
<tr>
<td>STOP</td>
<td>END</td>
<td></td>
</tr>
</tbody>
</table>

CORRESPONDING UNROLLER OUTPUT DECK

<table>
<thead>
<tr>
<th>0</th>
<th>CGEX</th>
<th>02</th>
<th>04</th>
<th>03</th>
<th>00000</th>
<th>000</th>
<th>00000</th>
<th>1</th>
<th>87</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BAND</td>
<td>02</td>
<td>02</td>
<td>00</td>
<td>00000</td>
<td>100</td>
<td>00000</td>
<td>3</td>
<td>139</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>CGEN</td>
<td>01</td>
<td>01</td>
<td>02</td>
<td>00000</td>
<td>100</td>
<td>00000</td>
<td>4</td>
<td>79</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>AXK</td>
<td>03</td>
<td>03</td>
<td>00</td>
<td>00000</td>
<td>100</td>
<td>00000</td>
<td>6</td>
<td>76</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>AN</td>
<td>01</td>
<td>01</td>
<td>08</td>
<td>00000</td>
<td>100</td>
<td>00000</td>
<td>7</td>
<td>166</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>LA</td>
<td>08</td>
<td>00</td>
<td>00</td>
<td>01000</td>
<td>100</td>
<td>01000</td>
<td>9</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>AN</td>
<td>02</td>
<td>02</td>
<td>09</td>
<td>00000</td>
<td>100</td>
<td>00000</td>
<td>10</td>
<td>166</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>LA</td>
<td>09</td>
<td>00</td>
<td>00</td>
<td>02000</td>
<td>100</td>
<td>02000</td>
<td>12</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>SKOR</td>
<td>01</td>
<td>01</td>
<td>02</td>
<td>00000</td>
<td>100</td>
<td>00000</td>
<td>13</td>
<td>150</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>MN*</td>
<td>01</td>
<td>01</td>
<td>02</td>
<td>00000</td>
<td>101</td>
<td>00000</td>
<td>14</td>
<td>178</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>EXIT</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00000</td>
<td>100</td>
<td>00000</td>
<td>0</td>
<td>199</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>CGEX</td>
<td>02</td>
<td>04</td>
<td>03</td>
<td>00000</td>
<td>000</td>
<td>00000</td>
<td>1</td>
<td>87</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>BAND</td>
<td>02</td>
<td>02</td>
<td>00</td>
<td>00000</td>
<td>100</td>
<td>00000</td>
<td>3</td>
<td>139</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>CGEN</td>
<td>01</td>
<td>01</td>
<td>02</td>
<td>00000</td>
<td>100</td>
<td>00000</td>
<td>4</td>
<td>79</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>AXK</td>
<td>03</td>
<td>03</td>
<td>00</td>
<td>00000</td>
<td>100</td>
<td>00000</td>
<td>6</td>
<td>76</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>AN</td>
<td>01</td>
<td>01</td>
<td>08</td>
<td>00000</td>
<td>100</td>
<td>00000</td>
<td>7</td>
<td>166</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>LA</td>
<td>08</td>
<td>00</td>
<td>00</td>
<td>01000</td>
<td>100</td>
<td>01000</td>
<td>9</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>AN</td>
<td>02</td>
<td>02</td>
<td>09</td>
<td>00000</td>
<td>100</td>
<td>00000</td>
<td>10</td>
<td>166</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>SKOR</td>
<td>01</td>
<td>01</td>
<td>02</td>
<td>00000</td>
<td>100</td>
<td>00000</td>
<td>13</td>
<td>150</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>MN*</td>
<td>01</td>
<td>01</td>
<td>02</td>
<td>00000</td>
<td>111</td>
<td>00000</td>
<td>14</td>
<td>178</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>EXIT</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00000</td>
<td>110</td>
<td>00000</td>
<td>0</td>
<td>199</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>CGEX</td>
<td>02</td>
<td>04</td>
<td>03</td>
<td>00000</td>
<td>010</td>
<td>00000</td>
<td>1</td>
<td>87</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>BAND</td>
<td>02</td>
<td>02</td>
<td>00</td>
<td>00000</td>
<td>010</td>
<td>00000</td>
<td>3</td>
<td>139</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>CGEN</td>
<td>01</td>
<td>01</td>
<td>02</td>
<td>00000</td>
<td>010</td>
<td>00000</td>
<td>4</td>
<td>79</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>AXK</td>
<td>03</td>
<td>03</td>
<td>00</td>
<td>00000</td>
<td>010</td>
<td>00000</td>
<td>6</td>
<td>76</td>
<td>2</td>
</tr>
<tr>
<td>25</td>
<td>AN</td>
<td>01</td>
<td>01</td>
<td>08</td>
<td>00000</td>
<td>010</td>
<td>00000</td>
<td>7</td>
<td>166</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>LA</td>
<td>08</td>
<td>00</td>
<td>00</td>
<td>01000</td>
<td>010</td>
<td>01000</td>
<td>9</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>27</td>
<td>AN</td>
<td>02</td>
<td>02</td>
<td>09</td>
<td>00000</td>
<td>010</td>
<td>00000</td>
<td>10</td>
<td>166</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>LA</td>
<td>09</td>
<td>00</td>
<td>00</td>
<td>02000</td>
<td>010</td>
<td>02000</td>
<td>12</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>29</td>
<td>SKOR</td>
<td>01</td>
<td>01</td>
<td>02</td>
<td>00000</td>
<td>011</td>
<td>00000</td>
<td>14</td>
<td>178</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>MN*</td>
<td>01</td>
<td>01</td>
<td>02</td>
<td>00000</td>
<td>010</td>
<td>00000</td>
<td>15</td>
<td>199</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>EXIT</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00000</td>
<td>010</td>
<td>00000</td>
<td>17</td>
<td>9</td>
<td>2</td>
</tr>
</tbody>
</table>
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).
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 "STOP" 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, NAGQ:** 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, NXGQ:** Similarly for X unit Contending Stack.
NQBUF, NQTEST, NQGØ: Similarly for Data Memory Queue.

NBØX: Number of memory bombs.

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.
Figure 2-1. Timer Input Track Card Format

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 35
   branch op. Applies from and including branch
   op to and including EXIT op.
8. Skip Flagged ops bit. Indicates skip state. 36
   Applies to op after skip to and including
   next skip
9. Skip Flag 37
10. Effective address accessed (LOAD/STORE) 41-45
11. Address of next instruction to be executed 48-52
13. Long Op = 2, Short Op = 1 60

Figure 2-2. Timer Input Deck Format

Example: Two PROGRAMS PRG1 and TEST to be timed:
Figure 2-3. The Parameter Card Format

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

(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.
(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.
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).
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
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.
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
**Arithmetic Load Instruction G7:** At time = 7 we find the instruction \texttt{LAT G, 0, 31, 136} which is marked by a "G", in the instruction trace section of the output. We observe from the trace that \texttt{G} was fetched into \texttt{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 \texttt{IB(1) \rightarrow DSPX1}, \texttt{IB(1) \rightarrow DSPA1}. At that time \texttt{G} is present in \texttt{DSPX1(7)}, \texttt{DSPX1(8)}, and in \texttt{DSPA1(7)}, \texttt{DSPA1(8)}. \texttt{G} is a long OP and takes two of the 24-bit positions in the dispatchers.

Let us follow the A unit activity of \texttt{G} first. We note that at time = 8 \texttt{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 \texttt{G} in A BUFFER (1). This part of \texttt{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 \texttt{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 \texttt{G}. Since three other X OPS precede \texttt{G} in DSPX1 at time = 8, and at most 3 ops may be dispatched to the X BUFFER per cycle, \texttt{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 \texttt{G} to enter.

We now find that \texttt{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 \texttt{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 \texttt{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 \texttt{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.
Figure 2-6. Timing of Memory Activity of Example G7

17 - G ARRIVES AT QUEUE
18 - EFF ARRIVES AT QUEUE
19 - TEST QUEUE ENTRY + DECIDE G CAN GO TO MEMORY
20 - BUS TO MEMORY
21 - G IN MEMORY
22 - MEMORY MARKED BUSY WITH G
23 - BUS TO REGISTER

MEMDLY = 5.0

ARRIVE AT REG
TURN WAITING BITS OFF
INPUT PROGRAM FOR THIS RUN = CR-FS
TIME/DATE OF RUN = 4D72C8FE C067194F

MACHINE PARAMETERS FOR THIS RUN

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of A Buffers</td>
<td>8</td>
</tr>
<tr>
<td>Number of X Buffers</td>
<td>3</td>
</tr>
<tr>
<td>Number of Q Buffers</td>
<td>8</td>
</tr>
<tr>
<td>Number A Ops Tested</td>
<td>8</td>
</tr>
<tr>
<td>Number X Ops Tested</td>
<td>3</td>
</tr>
<tr>
<td>Number Q Ops Tested</td>
<td>8</td>
</tr>
<tr>
<td>Max A Ops Iss/Cycle</td>
<td>3</td>
</tr>
<tr>
<td>Max X Ops Iss/Cycle</td>
<td>3</td>
</tr>
<tr>
<td>Max Q Ops Iss/Cycle</td>
<td>2</td>
</tr>
<tr>
<td>Minimum Q-Mem Delay</td>
<td>5.0</td>
</tr>
<tr>
<td>Number of BOMS</td>
<td>8</td>
</tr>
<tr>
<td>Number of Branch Regs</td>
<td>3</td>
</tr>
<tr>
<td>Number of Skip Regs</td>
<td>4</td>
</tr>
<tr>
<td>Size of DQ Table</td>
<td>6</td>
</tr>
<tr>
<td>Number of PSC Regs</td>
<td>8</td>
</tr>
<tr>
<td>Number Disp Buses</td>
<td>2</td>
</tr>
<tr>
<td>Max A Ops DSP/Cycle</td>
<td>4</td>
</tr>
<tr>
<td>Max X Ops DSP/Cycle</td>
<td>4</td>
</tr>
</tbody>
</table>

A FACILITIES

<table>
<thead>
<tr>
<th>Field</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>FA1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FA2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FM</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>FD</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>IA</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>IM</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>ID</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>L</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

X FACILITIES

<table>
<thead>
<tr>
<th>Field</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EA2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>L</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>S</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>M</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>XA</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>TIME</td>
<td>AXK</td>
<td>LAT</td>
<td>MXK</td>
<td>AXK</td>
<td>LAT</td>
<td>MXK</td>
<td>AXK</td>
<td>LAT</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>1.0</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td>2.0</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
</tr>
<tr>
<td>3.0</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>100</td>
</tr>
<tr>
<td>4.0</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>5.0</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>100</td>
<td>110</td>
<td>120</td>
</tr>
<tr>
<td>6.0</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>100</td>
<td>110</td>
<td>120</td>
<td>130</td>
</tr>
<tr>
<td>7.0</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>100</td>
<td>110</td>
<td>120</td>
<td>130</td>
<td>140</td>
</tr>
<tr>
<td>8.0</td>
<td>80</td>
<td>90</td>
<td>100</td>
<td>110</td>
<td>120</td>
<td>130</td>
<td>140</td>
<td>150</td>
</tr>
<tr>
<td>9.0</td>
<td>90</td>
<td>100</td>
<td>110</td>
<td>120</td>
<td>130</td>
<td>140</td>
<td>150</td>
<td>160</td>
</tr>
<tr>
<td>10.0</td>
<td>100</td>
<td>110</td>
<td>120</td>
<td>130</td>
<td>140</td>
<td>150</td>
<td>160</td>
<td>170</td>
</tr>
<tr>
<td>11.0</td>
<td>110</td>
<td>120</td>
<td>130</td>
<td>140</td>
<td>150</td>
<td>160</td>
<td>170</td>
<td>180</td>
</tr>
<tr>
<td>12.0</td>
<td>120</td>
<td>130</td>
<td>140</td>
<td>150</td>
<td>160</td>
<td>170</td>
<td>180</td>
<td>190</td>
</tr>
<tr>
<td>13.0</td>
<td>130</td>
<td>140</td>
<td>150</td>
<td>160</td>
<td>170</td>
<td>180</td>
<td>190</td>
<td>200</td>
</tr>
<tr>
<td>14.0</td>
<td>140</td>
<td>150</td>
<td>160</td>
<td>170</td>
<td>180</td>
<td>190</td>
<td>200</td>
<td>210</td>
</tr>
<tr>
<td>15.0</td>
<td>150</td>
<td>160</td>
<td>170</td>
<td>180</td>
<td>190</td>
<td>200</td>
<td>210</td>
<td>220</td>
</tr>
<tr>
<td>16.0</td>
<td>160</td>
<td>170</td>
<td>180</td>
<td>190</td>
<td>200</td>
<td>210</td>
<td>220</td>
<td>230</td>
</tr>
<tr>
<td>17.0</td>
<td>170</td>
<td>180</td>
<td>190</td>
<td>200</td>
<td>210</td>
<td>220</td>
<td>230</td>
<td>240</td>
</tr>
<tr>
<td>18.0</td>
<td>180</td>
<td>190</td>
<td>200</td>
<td>210</td>
<td>220</td>
<td>230</td>
<td>240</td>
<td>250</td>
</tr>
<tr>
<td>19.0</td>
<td>190</td>
<td>200</td>
<td>210</td>
<td>220</td>
<td>230</td>
<td>240</td>
<td>250</td>
<td>260</td>
</tr>
<tr>
<td>20.0</td>
<td>200</td>
<td>210</td>
<td>220</td>
<td>230</td>
<td>240</td>
<td>250</td>
<td>260</td>
<td>270</td>
</tr>
<tr>
<td>TIME</td>
<td>U</td>
<td>W</td>
<td>Y</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22.00</td>
<td>72 AXK</td>
<td>74 AXK</td>
<td>76 EXIT</td>
<td>41 MN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23.00</td>
<td></td>
<td>2 2 0</td>
<td>0 0 0</td>
<td>11 9 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24.00</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25.00</td>
<td>3</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26.00</td>
<td>74</td>
<td>76</td>
<td>41</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27.00</td>
<td>76</td>
<td>210</td>
<td>199</td>
<td>178</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28.00</td>
<td>210</td>
<td>76</td>
<td>111</td>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29.00</td>
<td>210</td>
<td>76</td>
<td>111</td>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TIME</th>
<th>Q</th>
<th>S</th>
<th>U</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>33.00</td>
<td>42 LAT</td>
<td>44 LAT</td>
<td>46 AXK</td>
<td>48 MN</td>
</tr>
<tr>
<td>34.00</td>
<td>9 0 5</td>
<td>10 0 3</td>
<td>5 5 0</td>
<td>12 7 8</td>
</tr>
<tr>
<td>35.00</td>
<td>156</td>
<td>80</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>36.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>37.00</td>
<td>376</td>
<td>86</td>
<td>150</td>
<td>881</td>
</tr>
<tr>
<td>38.00</td>
<td>44</td>
<td>46</td>
<td>48</td>
<td>56</td>
</tr>
<tr>
<td>39.00</td>
<td>15</td>
<td>211</td>
<td>199</td>
<td>178</td>
</tr>
<tr>
<td>40.00</td>
<td>15</td>
<td>211</td>
<td>111</td>
<td>101</td>
</tr>
<tr>
<td>41.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TIME</th>
<th>E</th>
<th>F</th>
<th>H</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>42.00</td>
<td>56 MN</td>
<td>57 LAT</td>
<td>59 LAT</td>
<td>61 AN</td>
</tr>
<tr>
<td>43.00</td>
<td>13 5 6</td>
<td>5 0 5</td>
<td>6 0 3</td>
<td>2 12 2</td>
</tr>
<tr>
<td>44.00</td>
<td>0</td>
<td>136</td>
<td>82</td>
<td>0</td>
</tr>
<tr>
<td>45.00</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>46.00</td>
<td>150</td>
<td>436</td>
<td>88</td>
<td>90</td>
</tr>
<tr>
<td>47.00</td>
<td>57</td>
<td>59</td>
<td>61</td>
<td>62</td>
</tr>
<tr>
<td>48.00</td>
<td>178</td>
<td>211</td>
<td>211</td>
<td>166</td>
</tr>
<tr>
<td>49.00</td>
<td>101</td>
<td>211</td>
<td>211</td>
<td>101</td>
</tr>
<tr>
<td>50.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TIME</th>
<th>K</th>
<th>L</th>
<th>N</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>51.00</td>
<td>62 AN</td>
<td>63 MN</td>
<td>64 LAT</td>
<td>66 LAT</td>
</tr>
<tr>
<td>52.00</td>
<td>11 1 1</td>
<td>14 3 4</td>
<td>3 0 2</td>
<td>4 0 4</td>
</tr>
<tr>
<td>53.00</td>
<td>0</td>
<td>136</td>
<td>256</td>
<td>82</td>
</tr>
<tr>
<td>54.00</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>55.00</td>
<td>840</td>
<td>88</td>
<td>436</td>
<td>88</td>
</tr>
<tr>
<td>56.00</td>
<td>63</td>
<td>64</td>
<td>66</td>
<td>68</td>
</tr>
<tr>
<td>57.00</td>
<td>166</td>
<td>178</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>58.00</td>
<td>101</td>
<td>211</td>
<td>211</td>
<td>211</td>
</tr>
<tr>
<td>59.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TIME</th>
<th>P</th>
<th>Q</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>60.00</td>
<td>41 MN</td>
<td>42 LAT</td>
<td>48 MN</td>
<td>70 AXK</td>
</tr>
<tr>
<td>61.00</td>
<td>11 9 10</td>
<td>9 0 5</td>
<td>5 5 0</td>
<td>4 4 0</td>
</tr>
<tr>
<td>62.00</td>
<td>0</td>
<td>196</td>
<td>60</td>
<td>2</td>
</tr>
<tr>
<td>63.00</td>
<td>100</td>
<td>210</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>64.00</td>
<td>5</td>
<td>76</td>
<td>5</td>
<td>210</td>
</tr>
<tr>
<td>65.00</td>
<td>74</td>
<td>76</td>
<td>41</td>
<td>0</td>
</tr>
<tr>
<td>66.00</td>
<td>76</td>
<td>210</td>
<td>199</td>
<td>178</td>
</tr>
<tr>
<td>67.00</td>
<td>210</td>
<td>76</td>
<td>111</td>
<td>101</td>
</tr>
<tr>
<td>68.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TIME</th>
<th>E</th>
<th>F</th>
<th>H</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>69.00</td>
<td>56 MN</td>
<td>57 LAT</td>
<td>59 LAT</td>
<td>61 AN</td>
</tr>
<tr>
<td>70.00</td>
<td>13 5 6</td>
<td>5 0 5</td>
<td>6 0 3</td>
<td>2 12 2</td>
</tr>
<tr>
<td>71.00</td>
<td>0</td>
<td>136</td>
<td>82</td>
<td>0</td>
</tr>
<tr>
<td>72.00</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>73.00</td>
<td>150</td>
<td>436</td>
<td>88</td>
<td>90</td>
</tr>
<tr>
<td>74.00</td>
<td>57</td>
<td>59</td>
<td>61</td>
<td>62</td>
</tr>
<tr>
<td>75.00</td>
<td>178</td>
<td>211</td>
<td>211</td>
<td>166</td>
</tr>
<tr>
<td>76.00</td>
<td>101</td>
<td>211</td>
<td>211</td>
<td>101</td>
</tr>
<tr>
<td>77.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TIME</th>
<th>K</th>
<th>L</th>
<th>N</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>78.00</td>
<td>62 AN</td>
<td>63 MN</td>
<td>64 LAT</td>
<td>66 LAT</td>
</tr>
<tr>
<td>79.00</td>
<td>11 1 1</td>
<td>14 3 4</td>
<td>3 0 2</td>
<td>4 0 4</td>
</tr>
<tr>
<td>80.00</td>
<td>0</td>
<td>196</td>
<td>256</td>
<td>136</td>
</tr>
<tr>
<td>81.00</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>82.00</td>
<td>840</td>
<td>88</td>
<td>436</td>
<td>88</td>
</tr>
<tr>
<td>83.00</td>
<td>63</td>
<td>64</td>
<td>66</td>
<td>68</td>
</tr>
<tr>
<td>84.00</td>
<td>166</td>
<td>178</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>85.00</td>
<td>101</td>
<td>211</td>
<td>211</td>
<td>211</td>
</tr>
<tr>
<td>86.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TIME</th>
<th>E</th>
<th>F</th>
<th>H</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>87.00</td>
<td>56 MN</td>
<td>57 LAT</td>
<td>59 LAT</td>
<td>61 AN</td>
</tr>
<tr>
<td>88.00</td>
<td>13 5 6</td>
<td>5 0 5</td>
<td>6 0 3</td>
<td>2 12 2</td>
</tr>
<tr>
<td>89.00</td>
<td>0</td>
<td>136</td>
<td>82</td>
<td>0</td>
</tr>
<tr>
<td>90.00</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>91.00</td>
<td>210</td>
<td>58</td>
<td>496</td>
<td>90</td>
</tr>
<tr>
<td>92.00</td>
<td>57</td>
<td>59</td>
<td>61</td>
<td>62</td>
</tr>
<tr>
<td>93.00</td>
<td>199</td>
<td>178</td>
<td>111</td>
<td>101</td>
</tr>
<tr>
<td>94.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TIME</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>95.00</td>
<td>63 MN</td>
</tr>
<tr>
<td>96.00</td>
<td>14 3 4</td>
</tr>
<tr>
<td>97.00</td>
<td>0</td>
</tr>
<tr>
<td>98.00</td>
<td>100</td>
</tr>
<tr>
<td>99.00</td>
<td>10</td>
</tr>
<tr>
<td>M</td>
<td>64 LAT</td>
</tr>
<tr>
<td>----</td>
<td>--------</td>
</tr>
<tr>
<td>O</td>
<td>66 LAT</td>
</tr>
<tr>
<td>Q</td>
<td>68 AN</td>
</tr>
<tr>
<td>R</td>
<td>69 AN</td>
</tr>
<tr>
<td>S</td>
<td>70 AXK</td>
</tr>
<tr>
<td>U</td>
<td>72 AXK</td>
</tr>
<tr>
<td>W</td>
<td>74 AXK</td>
</tr>
<tr>
<td>Y</td>
<td>76 EXIT</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TIME= 46.00**

**TIME= 47.00**

**TIME= 48.00**

**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**
### AEP

<table>
<thead>
<tr>
<th>BUSC</th>
<th>BNOP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SKIP CONTROL

- **SR 1**
- **SR 2**
- **SR 3**
- **SR 4**

### A BUFFER

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>QAGP</td>
<td>1</td>
<td>K</td>
<td>M</td>
<td>G</td>
<td>Z</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>PPPP</td>
<td>2</td>
<td>C</td>
<td>S</td>
<td>W</td>
<td>E</td>
<td>L</td>
<td>F</td>
</tr>
<tr>
<td>PPPP</td>
<td>3</td>
<td>K</td>
<td>S</td>
<td>W</td>
<td>E</td>
<td>L</td>
<td>F</td>
</tr>
<tr>
<td>PFF</td>
<td>4</td>
<td>C</td>
<td>S</td>
<td>W</td>
<td>E</td>
<td>L</td>
<td>F</td>
</tr>
<tr>
<td>PPPP</td>
<td>5</td>
<td>K</td>
<td>S</td>
<td>W</td>
<td>E</td>
<td>L</td>
<td>F</td>
</tr>
<tr>
<td>QAG</td>
<td>6</td>
<td>K</td>
<td>S</td>
<td>W</td>
<td>E</td>
<td>L</td>
<td>F</td>
</tr>
<tr>
<td>PPPP</td>
<td>7</td>
<td>K</td>
<td>S</td>
<td>W</td>
<td>E</td>
<td>L</td>
<td>F</td>
</tr>
<tr>
<td>PFF</td>
<td>8</td>
<td>K</td>
<td>S</td>
<td>W</td>
<td>E</td>
<td>L</td>
<td>F</td>
</tr>
</tbody>
</table>

### X BUFFER

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>AEGG</td>
<td>C1</td>
<td>E1</td>
</tr>
<tr>
<td>GGGG</td>
<td>C2</td>
<td>E2</td>
</tr>
<tr>
<td>GGGG</td>
<td>C3</td>
<td>E3</td>
</tr>
</tbody>
</table>

### A FACILITIES

<table>
<thead>
<tr>
<th>FA</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>J</td>
<td>K</td>
<td>W</td>
<td>P</td>
<td>W</td>
<td>P</td>
<td>W</td>
<td>P</td>
<td>W</td>
<td>P</td>
</tr>
</tbody>
</table>

### X FACILITIES

<table>
<thead>
<tr>
<th>EA</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>G</td>
<td>C</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>Q</td>
<td>Z</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>S</td>
<td>W</td>
</tr>
</tbody>
</table>

### MEMORY QUEUE (D)

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAE</td>
<td>CQ</td>
<td>AM</td>
<td>QGF</td>
</tr>
<tr>
<td>CQ</td>
<td>M</td>
<td>Q</td>
<td>GF</td>
</tr>
<tr>
<td>AM</td>
<td>Q</td>
<td>GF</td>
<td></td>
</tr>
<tr>
<td>QGF</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### MEMORY QUEUE (1)

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456 789</td>
<td>A</td>
<td>G</td>
<td>C</td>
</tr>
<tr>
<td>123456 789</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>123456 789</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>123456 789</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

### MEMORY

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>AkR</td>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>C</td>
<td>2</td>
</tr>
</tbody>
</table>
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.
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.
<table>
<thead>
<tr>
<th>OP</th>
<th>OP</th>
<th>OP</th>
<th>OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACH</td>
<td>X</td>
<td>CEQKK</td>
<td>X</td>
</tr>
<tr>
<td>ACL</td>
<td>X</td>
<td>CGED</td>
<td>X</td>
</tr>
<tr>
<td>ADN</td>
<td>X</td>
<td>CGEI</td>
<td>X</td>
</tr>
<tr>
<td>ADR</td>
<td>X</td>
<td>CGEN</td>
<td>X</td>
</tr>
<tr>
<td>ADU</td>
<td>X</td>
<td>CGEX</td>
<td>X</td>
</tr>
<tr>
<td>AI</td>
<td>X</td>
<td>CGEXK</td>
<td>X</td>
</tr>
<tr>
<td>AN</td>
<td>X</td>
<td>CMEQD</td>
<td>X</td>
</tr>
<tr>
<td>ANDA</td>
<td>X</td>
<td>CMEQN</td>
<td>X</td>
</tr>
<tr>
<td>ANDC</td>
<td>X</td>
<td>CMGED</td>
<td></td>
</tr>
<tr>
<td>ANDX</td>
<td>X</td>
<td>CMGEN</td>
<td>X</td>
</tr>
<tr>
<td>AR</td>
<td>X</td>
<td>CNTAA</td>
<td>X</td>
</tr>
<tr>
<td>AU</td>
<td>X</td>
<td>CNTAX</td>
<td>X</td>
</tr>
<tr>
<td>AX</td>
<td>X</td>
<td>CNTDA</td>
<td>X</td>
</tr>
<tr>
<td>AXC</td>
<td>X</td>
<td>CNTDX</td>
<td>X</td>
</tr>
<tr>
<td>AXK</td>
<td>X</td>
<td>CNTT</td>
<td>X</td>
</tr>
<tr>
<td>BAND</td>
<td>X</td>
<td>CUGEXK</td>
<td>X</td>
</tr>
<tr>
<td>BEQ</td>
<td>X</td>
<td>CVF</td>
<td>X</td>
</tr>
<tr>
<td>BFAF</td>
<td>X</td>
<td>CVI</td>
<td>X</td>
</tr>
<tr>
<td>BFOF</td>
<td>X</td>
<td>CVN</td>
<td>X</td>
</tr>
<tr>
<td>BOR</td>
<td>X</td>
<td>CVS</td>
<td>X</td>
</tr>
<tr>
<td>BTAF</td>
<td>X</td>
<td>CVS</td>
<td>X</td>
</tr>
<tr>
<td>BTOF</td>
<td>X</td>
<td>CVF</td>
<td>X</td>
</tr>
<tr>
<td>BU</td>
<td>X</td>
<td>CVN</td>
<td>X</td>
</tr>
<tr>
<td>BXOR</td>
<td>X</td>
<td>CVS</td>
<td>X</td>
</tr>
<tr>
<td>CBA</td>
<td>X</td>
<td>DDN</td>
<td>X</td>
</tr>
<tr>
<td>CBMA</td>
<td>X</td>
<td>DDR</td>
<td>X</td>
</tr>
<tr>
<td>CBMX</td>
<td>X</td>
<td>DMN</td>
<td>X</td>
</tr>
<tr>
<td>CBX</td>
<td>X</td>
<td>DMX</td>
<td>X</td>
</tr>
<tr>
<td>CBQD</td>
<td>X</td>
<td>DMR</td>
<td>X</td>
</tr>
<tr>
<td>CBQQ</td>
<td>X</td>
<td>DNX</td>
<td>X</td>
</tr>
<tr>
<td>CBQQ</td>
<td>X</td>
<td>DXK</td>
<td>X</td>
</tr>
<tr>
<td>CEBQ</td>
<td>X</td>
<td>DXK</td>
<td>X</td>
</tr>
<tr>
<td>CEQK</td>
<td>X</td>
<td>DXK</td>
<td>X</td>
</tr>
<tr>
<td>OP</td>
<td>OP</td>
<td>OP</td>
<td>OP</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>MRC</td>
<td>X</td>
<td>SCAN</td>
<td>SPF</td>
</tr>
<tr>
<td>MSX</td>
<td>SCH</td>
<td>X</td>
<td>SPI</td>
</tr>
<tr>
<td>MSXZ</td>
<td>SCL</td>
<td>X</td>
<td>SPX</td>
</tr>
<tr>
<td>MTX</td>
<td>SDN</td>
<td>SR</td>
<td>X</td>
</tr>
<tr>
<td>MU</td>
<td>X</td>
<td>SDR</td>
<td>STA</td>
</tr>
<tr>
<td>MX</td>
<td>X</td>
<td>SDU</td>
<td>STA</td>
</tr>
<tr>
<td>MXA</td>
<td>X</td>
<td>SHA</td>
<td>STA</td>
</tr>
<tr>
<td>MXC</td>
<td>X</td>
<td>SHAC</td>
<td>STA</td>
</tr>
<tr>
<td>MXK</td>
<td>X</td>
<td>SHD</td>
<td>STATH</td>
</tr>
<tr>
<td>MXP</td>
<td>X</td>
<td>SHDC</td>
<td>STD</td>
</tr>
<tr>
<td>MXS</td>
<td>SHDX</td>
<td>STDH</td>
<td>X</td>
</tr>
<tr>
<td>MXSO</td>
<td>SHDXC</td>
<td>STDHAA</td>
<td>X</td>
</tr>
<tr>
<td>MXT</td>
<td>SHX</td>
<td>X</td>
<td>STDHBA</td>
</tr>
<tr>
<td>MZT</td>
<td>SHXC</td>
<td>STDHCA</td>
<td>X</td>
</tr>
<tr>
<td>NOP</td>
<td>X</td>
<td>SIAC</td>
<td>STMA</td>
</tr>
<tr>
<td>ORA</td>
<td>X</td>
<td>SIO</td>
<td>STMS</td>
</tr>
<tr>
<td>ORC</td>
<td>X</td>
<td>SIX</td>
<td>STMZ</td>
</tr>
<tr>
<td>ORX</td>
<td>X</td>
<td>SIXC</td>
<td>STMZA</td>
</tr>
<tr>
<td>PAUSE</td>
<td>SKFAF</td>
<td>X</td>
<td>STXH</td>
</tr>
<tr>
<td>PI</td>
<td>SKFOF</td>
<td>X</td>
<td>SU</td>
</tr>
<tr>
<td>RND</td>
<td>SKTOF</td>
<td>X</td>
<td>SWA</td>
</tr>
<tr>
<td>RX</td>
<td>X</td>
<td>SXXOR</td>
<td>X</td>
</tr>
<tr>
<td>RXK</td>
<td>X</td>
<td>SN</td>
<td>SX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SNF</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SNI</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SNX</td>
<td></td>
</tr>
</tbody>
</table>
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 consecu- tively 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.
A UNIT INTLK SIMULATION:

Enclosed is a sample of the sort of code used in the timing simulator. The code is condensed from actual simulation 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:

Sketch of the "Hardware" Arrays  P1
Simplified Flowchart of XACON  P2
Simplified Flowchart of XAEMP  P3
Code (commented) for XACON  P4
Code (commented) for XAEMP  P6
Actual A UNIT FAC, BUS TABLES  P8
Sketch of A UNIT FAC, Buses  P9
SUBROUTINE XACON : SIMPLIFIED FLOWCHART

THIS ROUTINE SCANS THE STACK FOR PARTS WHICH CAN GO — I.E., PASS ALL INTLK TESTS. IT MARKS SUCH OP "GO" AND SETS CERTAIN BUSY AND SHIFT CELL PATTERNS.

CAUSE(ACON, T+1.)
CAUSE(AEMP, T+8)
N60 = 0
INS = 1
AC0(j) = 0, j=1,8

TEST OP(INS) S/D INTLK
NO INTLK
TEST WAITING BIT INTLK
NO IN-TBUS CONFLICT
TEST FACILITY BUSY FREE
TEST OUTBUS CONFLICT
NO IN

OP CAN GO: MARK 0,0
SET INBUS BUSY BIT.
SET FUNC. BK BUSY PATTERN INTO SHIFT CELL
SET OBUS BUSY BIT INTO SHIFT CELL (AND DEST)

N60 = N60 + 1
IF N60 > 3 [RETURN]

INS = INS + 1
IF INS > 8 [RETURN]
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 outflowing to registers. Then shifts the shift cells.

INS = 1

OP(INS) GOES?

YES

FIND BEST REG AND MARK BUSY

REMOVE OP FROM STACK AND BUBBLE UP THE STACK

NO

INS = INS + 1

IS INS > 8?

NO

YES

EXAMINE FRONT ROW OF CBS SHIFT CELLS FOR BUSY TO BEST. RESET ACCRF WAITING BITS

SHIFT THE SHIFT CELLS

RETURN
Subroutine XACON
COMMON -- -
CALL CAUSE (AGSN, TIME + 1.0, 0, 0, 0)
CALL CAUSE (AEMP, TIME + 0.8, 0, 0, 0)
DO 1 I = 1, 8
1 AGΦ(I) = 0
NGΦ = 0
C Scan the A contender stack for ins = 1 to 8 to find ops
C which can go - mark them with AGΦ(INS) = 1.
DO 100 INS = 1, 8
IF (AFULL(INS), EQ.0) GO TO 100
IF (INS, EQ.1) GO TO 21
INSM1 = INS - 1
C Test the source-dest interlocks
DO 20 I = 1, INSM1
DO 20 REG = 1, NAREGS
IF ((ASOR(INS, REG), EQ.1), AND, (ADEST(I, REG), EQ.1)) GO TO 100
IF ((ADEST(INS, REG), EQ.1), AND, (ASOR(I, REG), EQ.1)) GO TO 100
IF ((ADEST(INS, REG), EQ.1), AND, (ADEST(I, REG), EQ.1)) GO TO 100
20 CONTINUE
21 CONTINUE
C Test waiting bit interlock
DO 22 REG = 1, NAREGS
IF ((ASOR(INS, REG), EQ.1), AND, (ABUSY(REG), EQ.1)) GO TO 100
IF ((ADEST(INS, REG), EQ.1), AND, (ABUSY(REG), EQ.1)) GO TO 100
22 CONTINUE
C Find facility used by op (ins)
DO 25 FAC = 1, NAFAC
IF (AFAC(INS, FAC), NE,0) GO TO 26
25 CONTINUE
26 CONTINUE
C See if INBUS req'd by FAC is busy
INBUS = AFINBUS(FAC)
IF (AIBUSY(INBUS), EQ.1) GO TO 100
C See if facility box req'd is busy
BOX = ABOX(FAC)
IF (ABXBSY(BOX), EQ.1) GO TO 100
C TEST FAC BUSY REQUIREMENTS AGAINST SHIFT CELL CONTENTS
C WHICH INDICATE BUSY CONDITIONS SET BY PRIOR OPS.
    D0 30 T=1,NSLPT
    IF((AFSLPT(FAC,T).EQ.1).AND.(AFACSC(FAC,T).EQ.1)) GO TO 100
    30 CONTINUE
C TEST FOR PUTBUS CONFLICTS BY COMPARING BUS/TIME READ
C AGAINST CONTENTS OF SHIFT CELLS
    A8BUS = AF8BUS(FAC)
    DELAY = AF8LY(FAC)
    IF((A8BUS(IN8BUS).NE.0).AND.(ABUSSC(D8BUS,DELAY).NE.0))
        X N8 TO 100
C IF REACH THIS POINT, ALL TESTS PASSED, AND OP CAN GO
C SO MARK GO, AND SET APPROP. PATTERNS IN SHIFT CELLS.
    A8BSY(IN8BUS)=1
    A8XBSY(IN8BUS)=1
    D0 32 T=1,NSLPT
    IF(AFSLPT(FAC,T).EQ.0) AFACSC(FAC,T)=1
    32 CONTINUE
    ABUSSC(D8BUS,DELAY)=A8BUS(IN8BS,A8BUS)
    AG8(IN8)=1
    N8 = N8 + 1
    IF(N8.EQ.NAG8) RETURN
100 CONTINUE
RETURN
END
SUBROUTINE XAEMP

COMMON --

C THIS ROUTINE FIRST SCANS THE A CONTENDER STACK FOR
C OPS MARKED GΦ. IT ISSUES THE OPS BY SETTING THE
C WAITING BITS AND REMOVING THE OP FROM THE STACK.

DΦ 100 INS = 1,8
5 IF (AGΦ(INS),EQ.0) GO TO 100
   IF (AFULL(INS),EQ.0) GO TO 100

C GΦ (INS) GOES - FIRST SET REG WAITING BITS
DΦ 10 REG = 1, NAREGS
   IF (ADEST(INS, REG),NE.1) GO TO 10
   ABUSY (REG) = 1
10 CONTINUE

C NOW REMOVE THE GΦ FROM THE STACK AND BUBBLE UP STACK.
   AINPT = AINPT - 1
   IF (INS, EQ. 8) GO TO 31
DΦ 30 I = INS, 7
   AGΦ(I) = AGΦ(I+1)
   AFULL(I) = AFULL(I+1)
DΦ 25 J = 1,25
25 ABUFF(I, J) = ABUFF(I+1, J)
DΦ 26 J = 1, NAREGS
   AΣΦR(I, J) = AΣΦR(I+1, J)
26 ADEST(I, J) = ADEST(I+1, J)
DΦ 27 FAC = 1, NAFAC
27 AFAC(I, FAC = AFAC(I+1, FAC)
DΦ 28 BUS = 1, NABUS
28 AΦBUS(I, BUS) = AΦBUS(I+1, BUS)
30 CONTINUE
31 AGΦ(8) = 0
   AFULL(8) = 0
DΦ 125 J = 1,25
125 ABUFF(8, J) = 0
DΦ 126 J = 1, NAREGS
   ADEST(8, J) = 0
126 AΣΦR(8, J) = 0

L. Conway
Archives

(CONT.)
D0 127 FAC=1,NAFAC

127 AFAC (8,FAC) = 0
D0 128 BUS=1,NABUS
128 ABUS (8,BUS) = 0
G0,T05
100 CONTINUE

C
C THE NEXT FUNCTION OF THE ROUTINE IS TO CONTROL THE WAITING BITS - RESET WHEN INDICATED BY SHIFT CELL ENTRIES, THEN SHIFT THE SHIFT CELLS.
C
D0 210 BUS=1,NABUS
DEST = ABUSSC ( BUS,1)
ABUSY (DEST) = 0
210 CONTINUE

C NOW SHIFT THE SHIFT CELLS
D0 299 I = 1,10
AGXBSY (I) = 0
299 AISBSY (I) = 0
SLSTM1 = NSLST - 1
D0 301 J = 1,10
D0 300 SLST = 1,SLSTM1
300 ABUSSC ( J,SLST) = ABUSSC ( J,SLST+1)
301 ABUSSC ( J,NSLST) = 0
D0 303 J = 1,NAFAC
D0 302 SLST = 1,SLSTM1
302 AFACSCC (J,SLST) = AFACSC ( J,SLST+1)
303 AFACSC ( J,NSLST) = 0
RETURN
END
A FACILITIES, BUSES

AFAC

AFSLΦT(I,J)

AFDLY

AFIBUS

AFΩBUS

AFPS

ARRIVE AT
FAC TWO
CYCLES AFTER
SCAN: i.e.
ONE CYCLE
FOR INUSHING

3 4 3 9 2 5 15 1 1 1

2 1 3 1 1 2 2 1 2 3

1 2 3 4 2 4 4 5 6 7

2 1 4 3 2 4 4 6 1 3
X FACILITIES, BUSES

Note: No Inbus Conflicts. Assume all inbus-s to all functional units.

XFAC

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD1</td>
<td>ADD2</td>
<td>LGG</td>
<td>SH</td>
<td>MPY</td>
<td>DIV</td>
<td>XT</td>
<td>CMP</td>
<td>5</td>
</tr>
</tbody>
</table>

XFSLΦT(I,J)

<table>
<thead>
<tr>
<th>I</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

XFDLY

| 1 | 1 | 1 | 1 | 4 | 8 | 1 | 1 | 1 |

XΦBUS

| 5 | 6 | 1 | 3 | 2 | 2 | 7 | 0 | 8 |

XΦAX

| 1 | 2 | 3 | 4 | 5 | 5 | 6 | 7 | 8 |

NOTE: LGG contains ADD, SUB, ,, LGG , etc.
FACILITIES, BUSES

<table>
<thead>
<tr>
<th>AFAC</th>
<th>ADD3</th>
<th>ADD4</th>
<th>MUX</th>
<th>FLV</th>
<th>IADD</th>
<th>IMPY</th>
<th>IDIV</th>
<th>CMP</th>
<th>LOG-SH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
AFSLOT(i,j) = \begin{cases} 0 & i < 0 \\ 0 & i = 0 \\ 0 & i > 0 \end{cases}
\]

\[
I = \begin{cases} 0 & \text{(current time)} \\ 0 & \text{(bus slot)} \\ 1 & \text{(arrives at AFAC)} \end{cases}
\]

ARRIVE AT FAC TWO CYCLES AFTER SCAN: i.e., ONE CYCLE FOR INBUSING

AFDLY

<table>
<thead>
<tr>
<th>3</th>
<th>4</th>
<th>3</th>
<th>9</th>
<th>2</th>
<th>5</th>
<th>15</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
</table>

AFIBUS

<table>
<thead>
<tr>
<th>2</th>
<th>1</th>
<th>3</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
</table>

ABOX

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>2</th>
<th>4</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
</table>

AFIBUS

<table>
<thead>
<tr>
<th>2</th>
<th>1</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>4</th>
<th>4</th>
<th>6</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
</table>
## List of Functional Units & Their Performance

### Arithmetic Units (Ref. A-Unit Data Flow)

<table>
<thead>
<tr>
<th>Unit</th>
<th>Rate/Delay</th>
<th>Single Precision</th>
<th>Double Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SP CMP</strong></td>
<td>1/1</td>
<td>Compare</td>
<td>Compare</td>
</tr>
<tr>
<td><strong>INT +</strong></td>
<td>1/2</td>
<td>Add</td>
<td>Add</td>
</tr>
<tr>
<td><strong>DP CMP</strong></td>
<td>1/4</td>
<td>Compare</td>
<td>Compare</td>
</tr>
<tr>
<td><strong>SP FL +</strong></td>
<td>1/4</td>
<td>Add</td>
<td>Add</td>
</tr>
<tr>
<td><strong>DP FL +</strong></td>
<td>1/4</td>
<td>Add</td>
<td>Add</td>
</tr>
<tr>
<td><strong>SP FL ÷</strong></td>
<td>7/10 or 8/10 (Eng. Goal 7/9)</td>
<td>Divide</td>
<td>Divide</td>
</tr>
<tr>
<td><strong>MP FL ÷</strong></td>
<td>n/a</td>
<td>Mixed</td>
<td>Mixed</td>
</tr>
<tr>
<td><strong>DP FL ÷</strong></td>
<td>4/17 (Eng. Goal 14/16)</td>
<td>General</td>
<td>General</td>
</tr>
<tr>
<td><strong>SP INT ÷</strong></td>
<td>9/14 (Eng. Goal 9/13)</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td><strong>MP INT ÷</strong></td>
<td>n/a</td>
<td>Mixed</td>
<td>Mixed</td>
</tr>
<tr>
<td><strong>SP FL ×</strong></td>
<td>1/3</td>
<td>Multiply</td>
<td>Multiply</td>
</tr>
<tr>
<td><strong>MP FL ×</strong></td>
<td>1/3</td>
<td>Mixed</td>
<td>Mixed</td>
</tr>
<tr>
<td><strong>DP FL ×</strong></td>
<td>3/5</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td><strong>SP INT ×</strong></td>
<td>2/4</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td><strong>MP INT ×</strong></td>
<td>2/4</td>
<td>Mixed</td>
<td>Mixed</td>
</tr>
</tbody>
</table>

### Logic

- **SP FL +** 1/3

### Shift

- **SHIFT** 1/1

### SWAP

- **SWAP** 1/1
LIST OF FUNCTIONAL UNITS, CONTINUED.

INDEX UNITS (Ref. X-UNIT DATA FLOW)

\{\begin{align*}
\text{COMPARE} & \quad 1/1 \sim \text{RATE/DELAY} \\
\text{SHIFT} & \quad 1/1 \\
\text{MULTIPLY} & \quad 2/4. \\
\text{DIVIDE} & \quad 8/8 \text{ AVERAGE TO } 13/13 \text{ MAX. (DATA DEPENDENT)} \\
\text{ADD} & \quad 1/1 \quad \text{(INCLUDING EBA ADD IF REQ)} \\
\text{SUBTRACT} & \quad 1/1 \\
\text{LOGIC} & \quad 1/1 \\
\text{EBA SELECT} & \quad 1/1 \quad \text{(SELECT } x^{k+1} \text{ or } x^k \text{)} \\
\text{ADD 1 (EAI)} & \quad 1/1 \\
\text{ADD 2 (EA2)} & \quad 1/1 \\
\text{MOVE SELECT} & \quad 1/1 \quad \text{(TRANSMIT)} \\
\text{STORE SELECT} & \quad 1/1
\end{align*}\}

108
<table>
<thead>
<tr>
<th></th>
<th>LXH</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

L. Conway Archives
<p>| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 33 | STMZ  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 34 | STMZA |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 35 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 36 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 37 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 38 | MXA  | X  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 39 | MAX  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 40 | MKL  | X  | X  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 41 | MKR  | X  | X  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 42 | MLX  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 43 | MXS  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 44 | MSX  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 45 | MSXZ |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 46 | MSX^φ |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 47 | MXC  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 48 | MCX  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 49 | MLC  | X  | X  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 50 | MRC  | X  | X  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 51 | MXP  | X  | X  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 52 | MKP  | X  | X  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 53 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 54 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 55 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 56 | AN   | X  | X  | X  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 57 | ADN  | X  | X  | X  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 58 | AR   | X  | X  | X  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 59 | ADR  | X  | X  | X  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 60 | AU   | X  | X  | X  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 61 | ADU  | X  | X  | X  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 62 | SN   | X  | X  | X  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 63 | SDN  | X  | X  | X  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 64 | SR   | X  | X  | X  | X  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   | SDR | SU  | SDU | MN  | MDN | MR  | MDR | MU  | MDU | MMN | MMU | DN  | DDN | DR  | DDR | DMN | DMR | RND | SPF | SNF | CVS | CVF | AI  | SI  | MI  | MMI | DI  | DMI | ACL | SCL |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|</p>
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>97</td>
<td>ACH</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>SCH</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>SPI</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>SNI</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>CVN</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>CVI</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>AX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>SX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>MX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>DRX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>DX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>RX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>AXC</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>AXK</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>MXK</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>DRXK</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>DXK</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>R XK</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>118</td>
<td>SPX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>SNX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>121</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>122</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>CGEN</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>124</td>
<td>CEQN</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>CGED</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>CEQD</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>CMGEN</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>CMSEQN</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CMGED</td>
<td>CMEOQD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-------</td>
<td>--------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>131</td>
<td>CGEI</td>
<td>XX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>CEGI</td>
<td>XX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>133</td>
<td>CUEGI</td>
<td>XX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>134</td>
<td>CGEX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>CEGX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>CUUEX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>CGEXK</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>CEGXK</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>CUEEX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>CBE</td>
<td>XX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>CBMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>CBX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>CBMX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>CBT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>146</td>
<td>SHA</td>
<td>XX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>148</td>
<td>SHX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>149</td>
<td>SHAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>SHXC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>SHD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>152</td>
<td>SHDX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>SHDC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>154</td>
<td>SHDNC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>155</td>
<td>SWA</td>
<td>XX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>SWX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>157</td>
<td>IFA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>158</td>
<td>IFX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>IFZA</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>IFZX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

L. Conway Archives
<table>
<thead>
<tr>
<th>Page</th>
<th>SIA</th>
<th>SIX</th>
<th>SIAC</th>
<th>SIXC</th>
<th>SID</th>
<th>SIDC</th>
</tr>
</thead>
<tbody>
<tr>
<td>161</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>162</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>163</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>164</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>165</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>166</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>167</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>168</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>169</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>171</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>172</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>173</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>174</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>175</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>176</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>177</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>178</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>179</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>181</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>182</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>183</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>184</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>185</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>186</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>187</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>188</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>189</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>191</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>192</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>ANDA</th>
<th>TÅFA</th>
<th>FÅFA</th>
<th>ØRA</th>
<th>TØFA</th>
<th>FØFA</th>
<th>EQA</th>
</tr>
</thead>
<tbody>
<tr>
<td>170</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>171</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>172</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>173</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>174</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>175</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>176</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>177</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>178</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>179</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>181</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>182</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>183</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>184</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>185</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>186</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>187</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>188</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>189</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>191</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>192</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>XØRC</td>
<td>XNTT</td>
<td>XNTAA</td>
<td>XNTDA</td>
<td>XNTAX</td>
<td>XNTDX</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>193</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>194</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>195</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>196</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>197</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>198</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>199</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>201</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>202</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>203</td>
<td>BAND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>204</td>
<td>BTAFL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>205</td>
<td>BFRAF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>206</td>
<td>BTF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>207</td>
<td>BFQF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>208</td>
<td>BEQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>209</td>
<td>BXØR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td>BU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>211</td>
<td>EXIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>212</td>
<td>EXITL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>213</td>
<td>EXITA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>214</td>
<td>EXITP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>215</td>
<td>SKAND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>216</td>
<td>SKTAFL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>217</td>
<td>SKFAFL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>218</td>
<td>SKØR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>219</td>
<td>SKTF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>SKFØF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>221</td>
<td>SKQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>222</td>
<td>SKXØR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>223</td>
<td>IVIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>224</td>
<td>NØP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>225</td>
<td>PAUSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>226</td>
<td>PI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>227</td>
<td>SCAN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>228</td>
<td>SVC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>229</td>
<td>SVR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td>IC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>231</td>
<td>IR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>232</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>233</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>234</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>235</td>
<td>SIΦ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>236</td>
<td>HΙΦ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>237</td>
<td>TCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>238</td>
<td>MTX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>239</td>
<td>MXT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td>M2T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>241</td>
<td>MΦT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>242</td>
<td>ITUMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>243</td>
<td>ITUMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>244</td>
<td>IDA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>245</td>
<td>LDA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>246</td>
<td>LDHAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>247</td>
<td>LDHBA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>248</td>
<td>LDHCA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>249</td>
<td>LDHDA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>STDHAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>251</td>
<td>STDHBA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>252</td>
<td>STDHCA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>253</td>
<td>STDHDA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>254</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>255</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>---</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>1</td>
<td>LHX</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>LX</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>LXA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>STXH</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>STX</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>STXA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>LXC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>LXCA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>LAH</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>LA</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>LAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>STAH</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>STA</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>STAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>LDH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>LD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>STDH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>STD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>LATH</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>LAT</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>STATH</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>STAT</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>LL</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>LR</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>STL</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>STR</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>LMX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>STMX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>LMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>STMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>LMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>STMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>33</td>
<td>STMZ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>STMZA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>MXA</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>MAX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>MKL</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>MKR</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>MLX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>MXS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>MSX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>MSXZ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>MXSφ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>MXC</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>MCX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>MLC</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>MRC</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>MXP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>MXP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>AN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>ADN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>AR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>ADR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>AU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>ADU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>SN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>SDN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>SR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>SDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>SU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>SDU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>MN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>MDN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>MR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>MDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>MU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>MDU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>MMN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>MMU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>DN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>DDN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>DR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>DDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>DMN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>DMR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>RND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>SPF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>SNF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>CVS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>CVF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>87</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>AI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>SI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>MI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>MMI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>DI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>DMI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>ACL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>SCL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMGED</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMEQD</td>
<td>X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGEI</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CEQI</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CUGEI</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGEX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CEQX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CUGEX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGEXK</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CEQXK</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CUGEXK</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBA</td>
<td>X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBMA</td>
<td>X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBMX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHA</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHAC</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHXC</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHDX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHDC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHD XC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWA</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFA</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFZA</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFZX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>193</td>
<td>XORC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>194</td>
<td>CNTT</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>195</td>
<td>CNTAA</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>196</td>
<td>CNTDA</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>197</td>
<td>CNTAX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>198</td>
<td>CNTDX</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>199</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>201</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>202</td>
<td>BAND</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>203</td>
<td>BTAF</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>204</td>
<td>BFAF</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>205</td>
<td>BFR</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>206</td>
<td>BFVF</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>207</td>
<td>BEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>208</td>
<td>BXOR</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>209</td>
<td>BU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td>EXIT</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>211</td>
<td>EXTL</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>212</td>
<td>EXITA</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>213</td>
<td>EXITP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>214</td>
<td>SKAND</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>215</td>
<td>SKTAF</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>216</td>
<td>SKFAF</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>217</td>
<td>SKFR</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>218</td>
<td>SKVF</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>219</td>
<td>SKVF</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>SKFVF</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>221</td>
<td>SKEQ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>222</td>
<td>SKXOR</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>223</td>
<td>IVIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>224</td>
<td>NOP</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

L. Conway Archives
<table>
<thead>
<tr>
<th>Page</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>STMZ</td>
</tr>
<tr>
<td>34</td>
<td>STMZA</td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>MXA</td>
</tr>
<tr>
<td>39</td>
<td>MAX</td>
</tr>
<tr>
<td>40</td>
<td>MKL</td>
</tr>
<tr>
<td>41</td>
<td>MKR</td>
</tr>
<tr>
<td>42</td>
<td>MLX</td>
</tr>
<tr>
<td>43</td>
<td>MXS</td>
</tr>
<tr>
<td>44</td>
<td>MSX</td>
</tr>
<tr>
<td>45</td>
<td>MSXZ</td>
</tr>
<tr>
<td>46</td>
<td>MXSφ</td>
</tr>
<tr>
<td>47</td>
<td>MXC</td>
</tr>
<tr>
<td>48</td>
<td>MXC</td>
</tr>
<tr>
<td>49</td>
<td>MLC</td>
</tr>
<tr>
<td>50</td>
<td>MRC</td>
</tr>
<tr>
<td>51</td>
<td>MXP</td>
</tr>
<tr>
<td>52</td>
<td>MKP</td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>AN</td>
</tr>
<tr>
<td>57</td>
<td>ADN</td>
</tr>
<tr>
<td>58</td>
<td>AR</td>
</tr>
<tr>
<td>59</td>
<td>ADR</td>
</tr>
<tr>
<td>60</td>
<td>AU</td>
</tr>
<tr>
<td>61</td>
<td>ADU</td>
</tr>
<tr>
<td>62</td>
<td>SN</td>
</tr>
<tr>
<td>63</td>
<td>SDN</td>
</tr>
<tr>
<td>64</td>
<td>SR</td>
</tr>
</tbody>
</table>

L. Conway
Archives
<table>
<thead>
<tr>
<th>No.</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>129</td>
<td>CMGED</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>CMMEQD</td>
<td></td>
</tr>
<tr>
<td>131</td>
<td>CGEI</td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>CEQI</td>
<td></td>
</tr>
<tr>
<td>133</td>
<td>CUGEI</td>
<td></td>
</tr>
<tr>
<td>134</td>
<td>CGEX</td>
<td>1</td>
</tr>
<tr>
<td>135</td>
<td>CEQX</td>
<td>1</td>
</tr>
<tr>
<td>136</td>
<td>CUGEX</td>
<td>1</td>
</tr>
<tr>
<td>137</td>
<td>CGEXK</td>
<td>1</td>
</tr>
<tr>
<td>138</td>
<td>CEQXX</td>
<td>1</td>
</tr>
<tr>
<td>139</td>
<td>CUGEXK</td>
<td>1</td>
</tr>
<tr>
<td>140</td>
<td>CBA</td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>CBMA</td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>CBX</td>
<td>1</td>
</tr>
<tr>
<td>143</td>
<td>CBMX</td>
<td></td>
</tr>
<tr>
<td>144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td></td>
<td></td>
</tr>
<tr>
<td>146</td>
<td></td>
<td></td>
</tr>
<tr>
<td>147</td>
<td>SHA</td>
<td></td>
</tr>
<tr>
<td>148</td>
<td>SHX</td>
<td>1</td>
</tr>
<tr>
<td>149</td>
<td>SHAC</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>SHXC</td>
<td>1</td>
</tr>
<tr>
<td>151</td>
<td>SHD</td>
<td></td>
</tr>
<tr>
<td>152</td>
<td>SHDX</td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>SHDC</td>
<td></td>
</tr>
<tr>
<td>154</td>
<td>SHDXC</td>
<td></td>
</tr>
<tr>
<td>155</td>
<td>SWA</td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>SWX</td>
<td>1</td>
</tr>
<tr>
<td>157</td>
<td>IFA</td>
<td></td>
</tr>
<tr>
<td>158</td>
<td>IFX</td>
<td>1</td>
</tr>
<tr>
<td>159</td>
<td>IFZA</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>IFZX</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>161</td>
<td>SIA</td>
<td></td>
</tr>
<tr>
<td>162</td>
<td>SIX</td>
<td></td>
</tr>
<tr>
<td>163</td>
<td>SIA C</td>
<td></td>
</tr>
<tr>
<td>164</td>
<td>SIX C</td>
<td></td>
</tr>
<tr>
<td>165</td>
<td>SID</td>
<td></td>
</tr>
<tr>
<td>166</td>
<td>SID C</td>
<td></td>
</tr>
<tr>
<td>167</td>
<td></td>
<td></td>
</tr>
<tr>
<td>168</td>
<td></td>
<td></td>
</tr>
<tr>
<td>169</td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>ANDA</td>
<td></td>
</tr>
<tr>
<td>171</td>
<td>TAFA</td>
<td></td>
</tr>
<tr>
<td>172</td>
<td>FAFA</td>
<td></td>
</tr>
<tr>
<td>173</td>
<td>ØRA</td>
<td></td>
</tr>
<tr>
<td>174</td>
<td>TOFA</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>FAFA</td>
<td></td>
</tr>
<tr>
<td>176</td>
<td>EQA</td>
<td></td>
</tr>
<tr>
<td>177</td>
<td>ØRA</td>
<td></td>
</tr>
<tr>
<td>178</td>
<td>AND X</td>
<td></td>
</tr>
<tr>
<td>179</td>
<td>TAFX</td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>FAFX</td>
<td></td>
</tr>
<tr>
<td>181</td>
<td>ØRX</td>
<td></td>
</tr>
<tr>
<td>182</td>
<td>TOFX</td>
<td></td>
</tr>
<tr>
<td>183</td>
<td>ØFX</td>
<td></td>
</tr>
<tr>
<td>184</td>
<td>EQX</td>
<td></td>
</tr>
<tr>
<td>185</td>
<td>XØRX</td>
<td></td>
</tr>
<tr>
<td>186</td>
<td>ANDC</td>
<td></td>
</tr>
<tr>
<td>187</td>
<td>TAFC</td>
<td></td>
</tr>
<tr>
<td>188</td>
<td>FAFC</td>
<td></td>
</tr>
<tr>
<td>189</td>
<td>ØRC</td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>TOFC</td>
<td></td>
</tr>
<tr>
<td>191</td>
<td>ØFC</td>
<td></td>
</tr>
<tr>
<td>192</td>
<td>EQC</td>
<td></td>
</tr>
</tbody>
</table>
193  XØRC
194  CNTT
195  CNTAA
196  CNTDA
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
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LXH</td>
</tr>
<tr>
<td>2</td>
<td>LX</td>
</tr>
<tr>
<td>3</td>
<td>LXA</td>
</tr>
<tr>
<td>4</td>
<td>STXH</td>
</tr>
<tr>
<td>5</td>
<td>STX</td>
</tr>
<tr>
<td>6</td>
<td>STXA</td>
</tr>
<tr>
<td>7</td>
<td>LXC</td>
</tr>
<tr>
<td>8</td>
<td>LXCA</td>
</tr>
<tr>
<td>9</td>
<td>LAH</td>
</tr>
<tr>
<td>10</td>
<td>LA</td>
</tr>
<tr>
<td>11</td>
<td>LAA</td>
</tr>
<tr>
<td>12</td>
<td>STAH</td>
</tr>
<tr>
<td>13</td>
<td>STA</td>
</tr>
<tr>
<td>14</td>
<td>STAB</td>
</tr>
<tr>
<td>15</td>
<td>LDH</td>
</tr>
<tr>
<td>16</td>
<td>LD</td>
</tr>
<tr>
<td>17</td>
<td>STDH</td>
</tr>
<tr>
<td>18</td>
<td>STD</td>
</tr>
<tr>
<td>19</td>
<td>LATH</td>
</tr>
<tr>
<td>20</td>
<td>LAT</td>
</tr>
<tr>
<td>21</td>
<td>STATH</td>
</tr>
<tr>
<td>22</td>
<td>STAT</td>
</tr>
<tr>
<td>23</td>
<td>LL</td>
</tr>
<tr>
<td>24</td>
<td>LR</td>
</tr>
<tr>
<td>25</td>
<td>STL</td>
</tr>
<tr>
<td>26</td>
<td>STR</td>
</tr>
<tr>
<td>27</td>
<td>LMX</td>
</tr>
<tr>
<td>28</td>
<td>STMX</td>
</tr>
<tr>
<td>29</td>
<td>LMA</td>
</tr>
<tr>
<td>30</td>
<td>STMA</td>
</tr>
<tr>
<td>31</td>
<td>LMS</td>
</tr>
<tr>
<td>32</td>
<td>STMS</td>
</tr>
<tr>
<td>Year</td>
<td>Abbreviation</td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
</tr>
<tr>
<td>65</td>
<td>SDR</td>
</tr>
<tr>
<td>66</td>
<td>SU</td>
</tr>
<tr>
<td>67</td>
<td>SDU</td>
</tr>
<tr>
<td>68</td>
<td>MN</td>
</tr>
<tr>
<td>69</td>
<td>MDN</td>
</tr>
<tr>
<td>70</td>
<td>MR</td>
</tr>
<tr>
<td>71</td>
<td>MDR</td>
</tr>
<tr>
<td>72</td>
<td>MU</td>
</tr>
<tr>
<td>73</td>
<td>MDU</td>
</tr>
<tr>
<td>74</td>
<td>MMN</td>
</tr>
<tr>
<td>75</td>
<td>MMU</td>
</tr>
<tr>
<td>76</td>
<td>DN</td>
</tr>
<tr>
<td>77</td>
<td>DDN</td>
</tr>
<tr>
<td>78</td>
<td>DR</td>
</tr>
<tr>
<td>79</td>
<td>DDR</td>
</tr>
<tr>
<td>80</td>
<td>DMN</td>
</tr>
<tr>
<td>81</td>
<td>DMR</td>
</tr>
<tr>
<td>82</td>
<td>RND</td>
</tr>
<tr>
<td>83</td>
<td>SPF</td>
</tr>
<tr>
<td>84</td>
<td>SNF</td>
</tr>
<tr>
<td>85</td>
<td>CVS</td>
</tr>
<tr>
<td>86</td>
<td>CVF</td>
</tr>
<tr>
<td>87</td>
<td></td>
</tr>
<tr>
<td>88</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>AI</td>
</tr>
<tr>
<td>90</td>
<td>SI</td>
</tr>
<tr>
<td>91</td>
<td>MI</td>
</tr>
<tr>
<td>92</td>
<td>NMI</td>
</tr>
<tr>
<td>93</td>
<td>DI</td>
</tr>
<tr>
<td>94</td>
<td>DMI</td>
</tr>
<tr>
<td>95</td>
<td>ACL</td>
</tr>
<tr>
<td>96</td>
<td>SCL</td>
</tr>
</tbody>
</table>

56 57 58 59 60 61 62 63 64 65 66 67 68 69 70
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>97</td>
<td>ACH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>SCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>SPI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>SNI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>CVN</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>CVI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>105</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>AX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>SX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>MX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>DRX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>DX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>RX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>AXC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>AXK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>MXK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>DRXK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>DXK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>RXK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>118</td>
<td>SPX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>SNX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>121</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>122</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>CGEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>124</td>
<td>CEQN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>CGED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>CEQD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>CMGEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>CMEQN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>FAI FA2 FM FD IA IM ID C L S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>129 CMGED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>130 CMEDQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>131 CGEI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>132 CEQI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>133 CUQEI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>134 CGEX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>135 CEQX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>136 CUQEX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>137 CGEXX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>138 CEQXX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>139 CUGEXX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>140 CBA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>141 CBMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>142 CBX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>143 CBMX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>144</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>145</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>146</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>147</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>148</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>149</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>151</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>152</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>153</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>154</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>155</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>156</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>157</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>158</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>159</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>160</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page</td>
<td>Symbol</td>
<td>Row(s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>----------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>161</td>
<td>SIA</td>
<td>56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>162</td>
<td>SIX</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>163</td>
<td>SIAC</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>164</td>
<td>SIXC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>165</td>
<td>SID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>166</td>
<td>SIDC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>167</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>168</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>169</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>ANDA</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>171</td>
<td>TAF</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>172</td>
<td>FAFA</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>173</td>
<td>ORA</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>174</td>
<td>TFOA</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>FOFA</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>176</td>
<td>EQA</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>177</td>
<td>XORA</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>178</td>
<td>ANDX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>179</td>
<td>TAFX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>FAFX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>181</td>
<td>ORX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>182</td>
<td>TFRX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>183</td>
<td>FRFX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>184</td>
<td>EFX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>185</td>
<td>XORX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>186</td>
<td>ANDC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>187</td>
<td>TAF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>188</td>
<td>FAFC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>189</td>
<td>ORC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>TFOFC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>191</td>
<td>FOFC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>192</td>
<td>EQC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page</td>
<td>Code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>225</td>
<td>PAUSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>226</td>
<td>PI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>227</td>
<td>SCAN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>228</td>
<td>SVC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>229</td>
<td>SVR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td>IC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>231</td>
<td>IR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>232</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>233</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>234</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>235</td>
<td>SIQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>236</td>
<td>HIQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>237</td>
<td>TCH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>238</td>
<td>MTX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>239</td>
<td>MXT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td>M2T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>241</td>
<td>M2OT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>242</td>
<td>ITUMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>243</td>
<td>ITUMP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>244</td>
<td>IDA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>245</td>
<td>LDA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>246</td>
<td>LDHAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>247</td>
<td>LDHAS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>248</td>
<td>LDHCA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>249</td>
<td>LDHDA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>STDHAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>251</td>
<td>STDHBA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>252</td>
<td>STDHCA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>253</td>
<td>STDHDA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>254</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>255</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Common/RLS/Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAREGS</td>
<td>Number of registers</td>
</tr>
<tr>
<td>NABUS</td>
<td>Number of buses</td>
</tr>
<tr>
<td>AFULL(12)</td>
<td>Full register number</td>
</tr>
<tr>
<td>AGOP(12)</td>
<td>General purpose register</td>
</tr>
<tr>
<td>NAGOP</td>
<td>Number of AGOP registers</td>
</tr>
<tr>
<td>NATTEST</td>
<td>Number of test registers</td>
</tr>
<tr>
<td>NABUF</td>
<td>Number of buffer registers</td>
</tr>
<tr>
<td>ABUSYSZ</td>
<td>Size of buffer registers</td>
</tr>
<tr>
<td>ABUSY(200)</td>
<td>Size of buffer register (200)</td>
</tr>
<tr>
<td>ABUFFER(12,100)</td>
<td>Buffer register (12,100)</td>
</tr>
<tr>
<td>ASFR(12,200)</td>
<td>Source register (12,200)</td>
</tr>
<tr>
<td>PDEST(12,200)</td>
<td>Destination register (12,200)</td>
</tr>
<tr>
<td>AFAC(12,10)</td>
<td>FAC (12,10)</td>
</tr>
<tr>
<td>AFACSC(4,10,20)</td>
<td>FAC (4,10,20)</td>
</tr>
<tr>
<td>ABUSSC(4,10,20)</td>
<td>BUS (4,10,20)</td>
</tr>
<tr>
<td>ABUSY(10)</td>
<td>BUSY (10)</td>
</tr>
<tr>
<td>AFBUS(10)</td>
<td>BUS (10)</td>
</tr>
<tr>
<td>AFSLUT(10,20)</td>
<td>SLUT (10,20)</td>
</tr>
<tr>
<td>AFIBUS(10)</td>
<td>BUS (10)</td>
</tr>
<tr>
<td>AFQBUS(10)</td>
<td>BUS (10)</td>
</tr>
<tr>
<td>AFDLY(10)</td>
<td>DLY (10)</td>
</tr>
<tr>
<td>NAFAC</td>
<td>FAC (10)</td>
</tr>
<tr>
<td>NSLUT</td>
<td>SLUT (10)</td>
</tr>
<tr>
<td>QC(16,16)</td>
<td>Control register (16,16)</td>
</tr>
<tr>
<td>SDGA(32,2)</td>
<td>DGA (32,2)</td>
</tr>
<tr>
<td>NQBUF</td>
<td>Quantity of buffer</td>
</tr>
<tr>
<td>NQTEST</td>
<td>Number of test registers</td>
</tr>
<tr>
<td>NQGP</td>
<td>GP register number</td>
</tr>
<tr>
<td>QINPT</td>
<td>Input pointer</td>
</tr>
<tr>
<td>MEMDLY</td>
<td>Memory delay</td>
</tr>
<tr>
<td>MEMORY(16)</td>
<td>Memory size (16)</td>
</tr>
<tr>
<td>NQX</td>
<td>Quantity of X</td>
</tr>
<tr>
<td>MXTIME</td>
<td>Maximum time</td>
</tr>
<tr>
<td>OUTLVL</td>
<td>Output level</td>
</tr>
<tr>
<td>IQ(4,16)</td>
<td>Index register (4,16)</td>
</tr>
<tr>
<td>LONGER</td>
<td>Longer register</td>
</tr>
<tr>
<td>SR(8), ST(8)</td>
<td>Register (8, 8)</td>
</tr>
<tr>
<td>SKAP, SKXP</td>
<td>Skip register (X, X)</td>
</tr>
<tr>
<td>NSBUF</td>
<td>Number of buffer registers</td>
</tr>
</tbody>
</table>

# A, X Source-Dest. Regs(etc.)

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full trigger on buffer positions</td>
</tr>
<tr>
<td>GP trigger</td>
</tr>
<tr>
<td>MAX &amp; A, X issue per cycle</td>
</tr>
<tr>
<td>A, X posns tested</td>
</tr>
<tr>
<td>A, X buffer posns</td>
</tr>
</tbody>
</table>

### Dummy 0th Reg Busy Trigger

- AREG, XREG waiting triggers
- A BUFFER, X BUFFER
- Source tags for AREG, XREG dest

### Facilities Used by A, X

- A, X FACILITY SLOT BUSY SHIFT CELLS
- A, X OUTBUS BUSY SHIFT CELLS
- A, X INBUS BUSY VECTOR
- A, X OUTBUS DEST FOR GP IN BUFF
- SLOT BUSY PATTERN FOR A, X FACS IN BUS FOR A, X FACS
- OUTBUS " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " 

# A, X Facs

- Memory queue
- A STORE data buffer
- # of Q points in model
- # of Q posns tested for go
- MAX & Q posns go/cycle
- Q input pointer
- Q TO REG MEMORY DELAY
- MEMORY BOXES (BUSY)
- MEMORY BOXES
- MAX. RUN TIME. See main output level code. See stats inst mem queue
- Issued long branch trigger
- Skip res, skip taken
- Skip X,A pointer
- # skip ring positions
COMMON/RLS/VARIABLES (CONT)

APASS(200), XPASS(200)
PUT(2)
JOB(6)
STOP
MEMANOT(16)
ABOX(10), XBOX(10)
ABXESY(10), XBXESY(10)

THE REGISTER PASS BITS (BU-FRONT)
TIME/DATE
CHARACTERS IDENT CURRENT JOB
STOP CONDITION TRIGGER
CNT, QUIT, MEM. REGS. FOR STOP CND.
A, X FAC BOX NUMBER TABLES
A, X FAC BOX BUSY TRIGGERS
INDEXING  ASOR(I,J), ADEST(I,J), ABUSY(I,J)

i.e. A-sources-destinations:

\[\begin{align*}
J = 1 & : & A(0) \\
J = 32 & : & A(31) \\
J = 33 & : & XB(0) \\
J = 64 & : & XB(31) \\
J = 65 & : & CB(0) \\
J = 88 & : & CB(23) \\
\end{align*}\]

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

i.e. X-sources-destinations:

\[\begin{align*}
J = 1 & : & AB(0) \\
J = 32 & : & AB(31) \\
J = 33 & : & X(0) \\
J = 64 & : & X(31) \\
J = 65 & : & C(0) \\
J = 88 & : & C(31) \\
J = 89 & : & STORAGE \\
J = 90 & : & \text{\ldots} \\
\end{align*}\]
The Decode Table \( D(256,50) \)

OPS AND TAGS AS IN OLD MPM SIM PROGRAM
DECODE TABLE. 198 OPS.

\[ D(i,j) \]

\[
\begin{array}{cccc}
\text{J} & \text{X operation} & \text{A} & \text{BFS} \\
1 & \text{X operation} & \text{X operation} & \text{X operation} \\
2 & A & " " & " " \\
3 & A(I) = \text{Source} & A(J) = \text{DEST} \\
4 & A(J) = \text{Source} & A(I+1) = \text{DEST} \\
5 & A(I+1) = \text{Source} & A(I) = \text{DEST} \\
6 & A(I) = \text{Source} & A(J+1) = \text{Source} \\
7 & A(J+1) = \text{Source} & A(K) = \text{Source} \\
8 & A(K) = \text{Source} & A(K+1) = \text{Source} \\
9 & X(I) = \text{Source} & X(J) = \text{DEST} \\
10 & X(J) = \text{DEST} & X(I+1) = \text{Source} \\
11 & X(I+1) = \text{Source} & X(I+1) = \text{DEST} \\
12 & X(I+1) = \text{DEST} & X(I) = \text{Source} \\
13 & X(I) = \text{Source} & X(J) = \text{Source} \\
14 & X(J) = \text{DEST} & X(K) = \text{Source} \\
15 & X(K) = \text{Source} & AB(I) = \text{DEST} \\
16 & AB(I) = \text{DEST} & C(I) = \text{Source} \\
17 & C(I) = \text{Source} & C(J) = \text{Source} \\
18 & C(J) = \text{Source} & STORAGE = \text{Source} \\
19 & STORAGE = \text{Source} & STORAGE = \text{DEST} \\
20 & STORAGE = \text{DEST} & ILLEGAL OP TAG \\
21 & ILLEGAL OP TAG & \\
22 & & \\
23 & & \\
24 & & \\
25 & & \\
26 & & \\
27 & & \\
28 & & \\
29 & & \\
30 & & \\
\end{array}
\]
( D(256, 50), cont.)

31  INDEX ADDER 1  (For EAS)
32  INDEX ADDER 2  (For EAT)
33  LOG  (ADD, SUB, CMP, LOG)
34  SHIFTER
35  MPY
36  DIV
37  X TO A
38  CMP
40

41  ADD3
42  ADD4
43  FMPY
44  FDIV
45  IADD
46  IMPY
47  FDIV
48  CMP
49  LOG
50  SHIFT
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 ÛP
8. X ÛP
9. EXIT FLAG (SKIP FLAG)
10. BRANCH SUCCE.
11. SKIP SUCCE.
12. B ÛP
13. S ÛP
14. BEÛP
15. "EXIT GOES ALONE THIS CYCLE" TAG
16
17
18
19
20
21
22
23
24
25
THE $\phi_1 - \phi_3$ INTERFACE

I Busing of Ops to A4X Buffers:

Subject to, # of A or X Ops in 18'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 simulate this // action sequentially:

\[ \text{XBUS} \xrightarrow{\text{XINPT}} \text{XBUFF} \]
\[ \text{AUS} \xrightarrow{\text{AINPT}} \text{ABUFF} \]

$\phi_1$ examines input pointers to see if $AINPT > NABUF$, $XINPT > NXBUFF$. If not, it can place A or X op in ABUS or XBUS and call BUSTOA, BUS TO X subroutine to bus op to 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:

<table>
<thead>
<tr>
<th></th>
<th>A Buff Input Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>AINPT</td>
<td></td>
</tr>
<tr>
<td>XINPT</td>
<td>X &quot; &quot; &quot; &quot;</td>
</tr>
<tr>
<td>NABUF</td>
<td># of A Buffers</td>
</tr>
<tr>
<td>NXBUF</td>
<td>&quot; &quot; X &quot;</td>
</tr>
<tr>
<td>ABUS (50):</td>
<td>The A Bus</td>
</tr>
<tr>
<td>XBUS (50):</td>
<td>&quot; X &quot;</td>
</tr>
</tbody>
</table>

BUSTOA : SUBR. TO BUS ABUS TO ABUFF
BUSTOX : " " " X " " X "

(These update AINPT, XINPT)
## Format of Exec-Sim Output Cards

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>Starting Col.</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSADD</td>
<td>1</td>
<td>1X15</td>
</tr>
<tr>
<td>MNEM(2)</td>
<td>7</td>
<td>1XA6</td>
</tr>
<tr>
<td>II</td>
<td>14</td>
<td>2XI2</td>
</tr>
<tr>
<td>IJ</td>
<td>18</td>
<td>1XI2</td>
</tr>
<tr>
<td>IK</td>
<td>21</td>
<td>1XI2</td>
</tr>
<tr>
<td>ILIT</td>
<td>24</td>
<td>2X15</td>
</tr>
<tr>
<td>BRANCH</td>
<td>31</td>
<td>4X1L</td>
</tr>
<tr>
<td>SKIP</td>
<td>36</td>
<td>1L</td>
</tr>
<tr>
<td>IEKILT</td>
<td>37</td>
<td>1L</td>
</tr>
<tr>
<td>ACCADD</td>
<td>38</td>
<td>3x15</td>
</tr>
<tr>
<td>NXADD</td>
<td>46</td>
<td>2X15</td>
</tr>
<tr>
<td>OPNUM</td>
<td>53</td>
<td>2X13</td>
</tr>
<tr>
<td>LENGTH</td>
<td>58</td>
<td>2X1L</td>
</tr>
<tr>
<td>XOP</td>
<td>61</td>
<td>1L</td>
</tr>
<tr>
<td>ROOP</td>
<td>62</td>
<td>1L</td>
</tr>
<tr>
<td>BOP</td>
<td>63</td>
<td>1L</td>
</tr>
<tr>
<td>SOP</td>
<td>64</td>
<td>1L</td>
</tr>
<tr>
<td>BEOP</td>
<td>65</td>
<td>1L</td>
</tr>
</tbody>
</table>
S/D INTLK MATRICES
ABUSY
AGUFF (12,100)

AFAC5C (4,15,20)
2 LETTER
1 BUSY
AFAC (12,15)
CONTAINS FAC BUSY PATTERN
AFIBUS (15)
IBUS USED BY FAC
AFDLY (15)
TIME ØBUS USED BY FAC
AFØBUS (15)
ØBUS USED BY FAC
**Block Diagram of Busva (BusvaX)**

(Bus op to Abuff, Decode op tags)

1. **Enter**
2. Move op from Abus to Abuff (Ainpt)
3. Decode source-dest interlock tags from tag table
4. Remove tags for zero registers
5. Decode facility use tags

Forfac used, get Qbus implied and scan Adest for first dest. Place this in A.bus (I,Qbus). Continue scan of depts. If find another assume op has double dest and place second dest in A.bus I,Qbus+1

Ainpt = Ainpt + 1

Return
Block Diagram of ACQN (XCNQ)

(Scan for 1st ACG where ATTEST which can be)

ENTER

CAUSE ACGN (TIME + 1.0)
CAUSE ACGN (TIME + 0.8)
CAUSE ACGN (TIME + 0.8)
INS = 1
ALL AGE(z) = 0
NGQ = 0

AFULL (INS) = 0?
YES

INS > NATEST?
YES

NGQ = NAG0

NO

100

RETURN

100

INS = INS + 1

NGQ = NAG0

NO

NO

Fails

SOURCE-DEST BUSY INLKS?
YES

20

Succeeds

CURR-PREV SOURCE-DEST INLKS?
YES

FAIL

MARK GQ.
MOVE INBU, OUTBU, FAC
PATTERNS INTO SHIFT CELLS

NO

MULT

FAC USED BY OP(INS) SINGLE
OR MULT IDENT?
YES

30

SINGLE

INBUS, FAC SWIT, OR OUTBUS
FOR FAC INLKS?
NO

FAIL

26

Succeeds

NO

FAIL

FAC # > 1? STILL ON MULTFAC
YES

49

INBUS, FAC SWIT, OR OUTBU
FOR FAC INLKS?
NO

FAIL

60

MARK THIS FAC TO 1,
ALL OTHER FACS = 1 TO 0

NO

MARK GQ.
MOVE INBU, OUTBU, FAC
PATTERNS INTO SHIFT CELLS

This section handles test for
FACs for ops using one FAC
of a set of multiple IDENT. FACs
(i.e. one of two threat adders)

L. Conway
Archives

Insert tests for spec
facilities here
(LOAD + STORE, etc)
(BRANCH, EXIT, etc.)
Block Diagram of REmp (XEmp) (at 9 Time)

(Issue the 04 op5)

ENTER

INS = 1

INS > NABUF

AFull(ins) = 0 or A60 (ins) = 0

NO

(Turn 6Op5)

DO X REG = 1, NAREGS

IF (AFull(ins) .NE. 1) Go to X

C

IF (AFull(ins) .NE. 1) Go to Y

C

C

If (ABufol(ins) .NE. 1) Go to Y

ABufol(reg) = 0

X

ABusy(reg) = 0

C

Go to X

Y

ABusy(reg) = 1

C

Continue

Remove Ins from ABuff:

Ainpt = Ainpt - 1

Bubble up external ins:

Acy (S), AFull (S), ASfR (S, S)

ADest (S, S), AFac (S, S), ABus (S, S)

+ 0 all NABuf posns
**Block Diagram of ARET (XRET)**

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

**ENTER**

**RETET BUSY VECTORS, HANDLING BUREST**

\[ D \phi \times 5 = 1, 10 \]

**DEST**: ABUSSC (1, 1, 1)

- **C** (COMPARISON)
  - IF DEST NOT BU, G\( \phi \) HANDLE normally
  - IF (XBUSPS (DEST, NE. 1)) G\( \phi \) TO Y
- **C** (CONDITIONAL)
  - IF S\( \phi \), RETURN DEST TO IT. OTHERWISE JUST SET XBU BUSY.
  - IF (XBUSY (DEST). EQ. 1) G\( \phi \) TO Z
  - \( \Phi \) BUFS (DEST) = 0
  - G\( \phi \) TO X
- **Z**
  - XBUSY (DEST) = 0
- **Y**
  - A BUSY (DEST) = 0
  - CONTINUE

**AS FNC OF RETET DEST, MAY HAVE TO INCLUDE OTHER EXEC ACTIVITY HERE**

**SHIFT THE SHIFT CELLS**

**RETURN**
THE EVENT RUNNING TIMES WITHIN THE CYCLE:

STATS

ACMN  XCMN  QCMN

MXN

REMP

MFREE

ARET  XRET  QEMP  MBUSY

LOAD

XEMP IF GO
FILL SAME CYCLE

XEMP IF GO
FILL NEXT CYCLE
STACK TO REG TIMING

KEY DIFFERENCE BETWEEN

A and X Stack Algorithm
and busing, facilities.

X overlaps intlk & busing
to get results at reg at
earliest time. Does not
anticipate result arrival for
issuance of next op.

A has busynq 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.
<table>
<thead>
<tr>
<th>18582</th>
<th>18838</th>
<th>18902</th>
<th>18903</th>
<th>18904</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q(16,16)</td>
<td>SDGA(32,2)</td>
<td>NQBUF</td>
<td>NQTEST</td>
<td>NQGFL</td>
</tr>
<tr>
<td>18905</td>
<td>18906</td>
<td>18907</td>
<td>18908</td>
<td>18909</td>
</tr>
<tr>
<td>QINPT</td>
<td>QCQN</td>
<td>QEMP</td>
<td>MBUSY</td>
<td>MFREE</td>
</tr>
<tr>
<td>18910</td>
<td>18911</td>
<td>18912</td>
<td>18929</td>
<td>18930</td>
</tr>
<tr>
<td>LOAD</td>
<td>MEMDLY</td>
<td>MEMORY(16)</td>
<td>NSFX</td>
<td>EAV</td>
</tr>
<tr>
<td>18931</td>
<td>18933</td>
<td>18934</td>
<td>18998</td>
<td>18999</td>
</tr>
<tr>
<td>MXTIME</td>
<td>QUTLVL</td>
<td>IQ(4,16)</td>
<td>RTN</td>
<td>WONGBR</td>
</tr>
<tr>
<td>19000</td>
<td>19008</td>
<td>19016</td>
<td>19017</td>
<td>19018</td>
</tr>
<tr>
<td>SR(8)</td>
<td>ST(8)</td>
<td>SKXP</td>
<td>SKAP</td>
<td>NSBUF</td>
</tr>
<tr>
<td>19019</td>
<td>19219</td>
<td>19419</td>
<td>19423</td>
<td>19429</td>
</tr>
<tr>
<td>RPASS(200)</td>
<td>XPASS(200)</td>
<td>QUT(8)</td>
<td>J98(6)</td>
<td>SSFLP</td>
</tr>
<tr>
<td>19446</td>
<td>19461</td>
<td>19471</td>
<td>19486</td>
<td>19496</td>
</tr>
<tr>
<td>ABFAX(15)</td>
<td>A8X85Y(10)</td>
<td>KBFAX(15)</td>
<td>KBFAX(15)</td>
<td>X8X85Y(10)</td>
</tr>
<tr>
<td></td>
<td>COMMON/RLS/ Revised</td>
<td>MAY 18 1967</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---------------------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>FIRST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>NAREGS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>NXREGS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>NABUS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>NXBUS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>STATS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>ACφN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>XCON</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>AMϕM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>XEMϕ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>FILL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>AFULL(12)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>XFULL(12)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>AGϕ(12)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>XGϕ(12)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>NAGϕ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>NXGϕ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>NATEST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>NXTEST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>NAFAC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>NXFAC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>ABUSYZ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>ABUSY(200)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>XBUSYZY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>XBUSY(200)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>ABUFF(1200)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>XBUFF(1200)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>ASϕR(2400)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>XSϕR(2400)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>ADEST(2400)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>XDEST(2400)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>AFAC(180)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>XFAC(180)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>AFAC SC(1200)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>ARET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>XFAC SC(1200)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>ABUSSC(800)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>ABUSY(10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>XBUSSC(800)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>X18SSY(10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>XFBUS(15)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>AFBUS(120)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>XFBUS(120)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>XFSLOT(300)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>XFDSL(300)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>AFDSL(15)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>XFDLS(15)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>AF ϕBUS(15)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>X ϕBUS(15)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>NSELϕT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>ABUSPZ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>ABUSPS(200)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>XBUSPS(200)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>ABUSFB(200)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>XBUSFB(200)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>ABUSF(200)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>XBUSF(200)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

L. Conway Archives
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FIRST NAREGS</td>
<td>NXREGS</td>
<td>NABUS</td>
<td>NABUS</td>
<td>STATS</td>
<td>ACON</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>XCON</td>
<td>AEMP</td>
<td>XEMP</td>
<td>FILL</td>
<td>AFULL(12)</td>
<td>XFULL(12)</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>AGF(12)</td>
<td>X60(12)</td>
<td>NACO</td>
<td>NX6P</td>
<td>VATEDT</td>
<td>NTEST</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>267</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>NAPAC</td>
<td>NXPAC</td>
<td>ABUSY2</td>
<td>ABUSY(200)</td>
<td>XBUSZ2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>268</td>
<td>468</td>
<td>1668</td>
<td>2868</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>XBUSY(200)</td>
<td>ABUFF(1200)</td>
<td>XBUFF(1200)</td>
<td>ASOR(2400)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>5268</td>
<td>7668</td>
<td>10068</td>
<td>12468</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>X5OR(2400)</td>
<td>ADEST(2400)</td>
<td>XDEST(2400)</td>
<td>AFAC(120)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>12588</td>
<td>12708</td>
<td>13508</td>
<td>13509</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>XFACE(120)</td>
<td>AFACSC(800)</td>
<td>ARET</td>
<td>XFACEC(800)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>14309</td>
<td>14310</td>
<td>15110</td>
<td>15120</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>XRET</td>
<td>ABUSC(800)</td>
<td>ABBUSY(10)</td>
<td>XBUSC(800)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>15920</td>
<td>15930</td>
<td>15940</td>
<td>16060</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>XBBUSY(10)</td>
<td>XFBUS(10)</td>
<td>ABBUS(120)</td>
<td>XBUS(120)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>16180</td>
<td>16380</td>
<td>16580</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>AFSLOT(200)</td>
<td>XFSLOT(200)</td>
<td>AFSSB(10)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>16590</td>
<td>16600</td>
<td>16610</td>
<td>16620</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>AFPLY(10)</td>
<td>XPDLY(10)</td>
<td>ABUS(10)</td>
<td>XBUS(10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>16630</td>
<td>16631</td>
<td>16632</td>
<td>16832</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>NSSLT</td>
<td>ABUP3</td>
<td>ABUS(200)</td>
<td>XBUP3(200)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>17032</td>
<td>17232</td>
<td>163</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Description</td>
<td>Address</td>
<td>Description</td>
<td>Address</td>
<td>Description</td>
<td>Address</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------</td>
<td>---------</td>
<td>----------------------</td>
<td>---------</td>
<td>----------------------</td>
<td>---------</td>
<td>----------------------</td>
</tr>
<tr>
<td>17432</td>
<td>Q(16,16)</td>
<td>17688</td>
<td>SDBA(32,2)</td>
<td>17752</td>
<td>NQBUF</td>
<td>17754</td>
<td>NQGYP</td>
</tr>
<tr>
<td>17755</td>
<td>QINPT</td>
<td>17756</td>
<td>QCQN</td>
<td>17757</td>
<td>QEMP</td>
<td>17758</td>
<td>MBUSY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17760</td>
<td>LOAD</td>
<td>17761</td>
<td>MEMDLY</td>
<td>17763</td>
<td>MEMORY(16)</td>
<td>17778</td>
<td>NBFX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17781</td>
<td>MXTIME</td>
<td>17783</td>
<td>OUTLVL</td>
<td>17784</td>
<td>IQ(4,16)</td>
<td>17848</td>
<td>RTN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17850</td>
<td>SR(8)</td>
<td>17858</td>
<td>ST(8)</td>
<td>17866</td>
<td>SKXP</td>
<td>17867</td>
<td>SKAP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18269</td>
<td>APASS(200)</td>
<td>18269</td>
<td>XPASS(200)</td>
<td>18269</td>
<td>OUT(2)</td>
<td>18273</td>
<td>SSTOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18290</td>
<td>ABFX(10)</td>
<td>18300</td>
<td>BGXBSY(10)</td>
<td>18310</td>
<td>XBFX(10), XGXBXY(10)</td>
<td>18320</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ONLY CHANGE: $A_6 = 3$
Bussing with
SEP x, ÷
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
MPM-BLCU Interface for Store Ops

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

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.
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 bus designated by the name at the top of the STO BUS bus 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 bus column is as follows:

<table>
<thead>
<tr>
<th>V</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>V</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

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 BUS₂ and the second on A DATA BUS₁ to the corresponding Request Stack. The name of the slot of the data buffer to be filled is taken from the top
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
Parameters:

- **NQBUF**: total # fillable Q POSNS
- **NQTEST**: # posn's tested for Gφ
- **NQGφ**: # OPS MAX Gφ / CYCLE

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

**TIME DELAY** Q TO REG = MEMDLY

**EX**: MEMDLY = 5.0
SUBROUTINE ARCON

ALGORITHM LOADS OUT OF ORDER WITH GOM IN TLC
STORES IN ORDER

CALL CAUSE (QCON, TIME + 1.0, 0, 0, 0, 0)
CALL CAUSE (QEMT, TIME + 0.8, 0, 0, 0, 0)
N60 = 0

D(1) I = 1, NQBUF

L 1 Q(3, 16) = 0
D(100) IN5 = 1, NQTEST
IF (Q(I, 8). EQ. 0) G0 T0 100
IF (INS .EQ. 1) G0 T0 11
INSM1 = INS - 1
D(10) I = 1, INS M1

C IF PREV INST TO SAME GOM, N60P
IF (Q(I, 6). EQ. Q(IN5, 6)) G0 T0 100
IF (Q(I, 3). EQ. 1). AND. (Q(3, 16). EQ. 0)) G0 T0 100
CONTINUE

C IF STORE AND DATA NOT AVAIL, N60P
11 IF (Q(IN, 9). EQ. 1). AND. (Q(INS, 9). EQ. 0)) G0 T0 100
MARK G0
Q(IN, 16) = 1
N64 = N60 + 1
IF (N60, GT, NQ60) RETURN

100 CONTINUE
RETURN
END
SUBROUTINE XQEMP

D0 100 INS=1, NQBUF

5 IF(Q(INS, 16).EQ.0) GO TO 100
   IF(Q(INS, 8).EQ.0) GO TO 100
   ISSUE INS TO MEMORY
   BYM = 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, BYM, L, 0)
   CALL CAUSE(MFREE, TIME+MEMDLY-1, BYM, 0, 0)
   IF LOAD, CAUSE DEST LOAD IN MEMDLY CYCLES
   IF(Q(INS, 2).EQ.1) CALL CAUSE(LOAD, TIME+MEMDLY, DEST, A, X)
   REMOVE INS FROM Q
   QINPT = QINPT - 1
   M = NQBUF - 1
   IF(INS.EQ.INS.QBUF) GO TO 31
   D0 30 I = INS, M
   D0 30 J = 1, 16
   Q(I, J) = Q(I+1, J)
30   CONTINUE
31   CONTINUE
   D0 32 J = 1, 16
   Q(NQBUF, J) = 0
32   CONTINUE
   GO TO 5
100   CONTINUE
   RETURN
   END
SUBROUTINE GRQ

C
ENTRY XM BUSY
BOM = IPAR1
L = IPAR2
MEMORY (BOM) = L
RETURN

C
ENTRY XM FREE
BOM = IPAR1
MEMORY (BOM) = 0
RETURN

C
ENTRY XLOAD
DEST = IPAR1
IF (ABUPS (DEST).NE. 1) 60 T 9
IF (ABUSY (DEST).EQ. 1) 60 T 8
ABFULC (DEST) = 1
RETURN
8 ABUSY (DEST) = 0
9 XBUSY (DEST) = 0
RETURN

C
ENTRY XEAV
DO 10 I = 1, NQ, BUF
IF (Q (I, 8).EQ. 1) 60 T 10
Q (I, 8) = 1
RETURN
10 CONTINUE
A = 1
B = 20000
C = 101
CALL TRUL (A, B, C)
RETURN
IN XACQN

C --- TEST FOR SPEC OPS HERE ---
C IF STORE A TEST AVAIL OR INBUS (STOREBUS)
C IF AVAIL, SET BUSY. IF NOT, GO TO 100

IF (ADEST (INS, 89).NE.1) GO TO 27
IF (AIIBSY (3). EQ.1) GO TO 100
AIIBSY (3) = 1
GO TO 95

C 27 CONTINUE ---

---

IN XAEMP

C --- TEST FOR SPEC OPS HERE ---
C IF STORE A SHIP DATA TO BUFFER OR Q
C DEP. ON STATE OF BUFFER
C IF (ADEST (INS, 89).NE.1) GO TO 7
C IF (SDBA (1, 2). NE.1) GO TO 2
C NY STA WAITING. SET DATA IN SDBA
D$ 3 I = 1, 32
C IF (SDBA (2, 1). EQ.1) GO TO 3
C SDBA (1, 1) = 1
C CONTINUE
C GO TO 7

C STA WAITING. DATA TO Q, SHIFT SDBA
C CONTINUE
D$ 4 I = 1, 31
C SDBA (1, 1) = SDBA (I + 1, 1)
C SDBA (3, 1) = 0
C SDBA (3, 2) = 0
C SD$ 50 INS = 1, NO BUF
C IF (Q (INS, 3). NE.1) GO TO 50
C IF (Q (INS, 4). NE.1) GO TO 50
C IF (Q (INS, 9). EQ.1) GO TO 50
C Q (INS, 9) = 1
C GO TO 7

C 50 CONTINUE
IN XCON

C ---- TEST FOR SPEC. OPS HERE ----

IF L/5, TEST AVAILABILITY OF QUEUE
   IF (X59R(INS, 89), NE, 1)) AND (XDEST(INS, 89), NE, 1) 697727
   IF (QINPT, GT, NQBUF) 60 TO 100
   27 CONTINUE

IN XXEMP

C ---- TEST FOR SPEC. OPS HERE ----

IF L/5, SHIP TO QUEUE
   IF (X59R(INS, 89), EQ, 1)) AND (XDEST(INS, 89), NE, 1) 697727
   CALL CAUSE(EAV, TIME+1.0, Q, 0, 0)
   IN = QINPT
   IN = QINPT + 1
   Q(IN, 1) = XBUFF(INS, 1)
   Q(IN, 2) = XBUFF(INS, 2)
   IF (Q(IN, 2), NE, 1) Q(IN, 3) = 1
   Q(IN, 7) = XBUFF(INS, 6)
   Q(IN, 4) = MOD(Q(IN, 7), NBOX) + 1
   IF (X59R(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, 7) = 1
   CONTINUE

   88 IF STORE A, GET DATA OR SET WAIT
      IF (Q(IN, 3), NE, 1) OR (Q(IN, 4), NE, 1) 60 TO 88
      IF (XDEST(INS, REG), EQ, 1) Q(IN, 15) = REG
      CONTINUE

   50 50 I = 1, 31
      50 J = 1, 2
      SDBA(I, J) = SDBA(I+1, J)
      SDBA(32, 1) = 0
      SDBA(32, 2) = 0
      60 TO 70
6 CONTINUE
   DATA NOT AVAIL. SET FIRST FREE WAIT OUT
   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 TRFUBL(A, B, C)
7 CONTINUE
   }
   C ————————————————————————
C —— IGNORE STORAGE AS BEST
   IF (REG, EQ. 89) GO TO 10
   IN X4XRET
C —— IGNORE L/S BUSES
   IF ((BUS.EQ.5).OR.(BUS.EQ.6)) GO TO 10
STORE A

XEMP

AEMP

IF DATA REG AVAILABLE ISSUE

1. If waiting_CNT > 0
   INCR waiting_CNT

2. If waiting_CNT < 0
   Set DATAV and
   Set_CNT = CNT + 1

equiv to

1'. If SDBA(1,2) = 1
   or SDBA(1,1) + (1,2) = 0
   Set 1ST_AVAIL_WAIT (1,2)

2'. If SDBA(1,1) = 1
   Set DATAV on Q
   Shift up SDBA

L. Conway Archives
I. Interface with IQ:

1. IFADD, IFDST, IFRTN
   IFADD = INS. FETCH ADDRESS. IFDST = 13 DEST and will be
   non-zero if valid request present at end of cycle (is
   reset on third request). IFRTN = place DEST here on
   completion of full INST FETCH.

II. QCENJ After DA runs, run IQ. Compare 4 int/k
    against GO DATA AREA BAMS. Issue up to 4 by
    marking GO.

III. QEMP After DA issue, run IQ issue. Cause
     many, MEREF at appro. times with L=+. Remove
     orissued. If last op issued, cause RTN at appro.
     time to place Dest into IFRTN. (See below-filling)

IV. BTN Event to set:
    DEST = IPAR
    IFRTN = DEST

V. Filling of IQ:

IN QEMP:
   After QEMP
   empties IQ examine interface to see if
   request present. If so, and if IQ empty,
   place request in IQ expanding address and
   decoding BAMS setting letter, etc.
   Then zero incoming interface.
1. **Qp issue**:
   Issuance of op whose dest has no 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 a `repl`, check if back-up is full. If not, normal handling. If so, don't set `recovery` but set `bufull = 0` and `bufbusy = 0` (cover to move off back-up to `front`).

**Summary**:

IN `remp`, `xemp`  

- QP YES  
- QP HAS ABU REG?  
- ABU REG?  
- BUFULL?  
- ABUFULL = 0?  
- ABUSY = 0?  
- NORMAL ISSUE: SET AREGBUSY

2. **Data arrives at a `dest`**:

IN `pret`, `xpret`  

- Data arrives at `dest`  
- IS THE ADEST AXBU REG?  
- NORMAL ARRIVAL: AREGBUSY = 0  
- XREGBUSY?  
- BUFULL = 1?  
- AREGBUSY = 0?  
- ABUSY = 0?
Handling the Back-Up REGs (Cont)

Relevant Variables:

\[ \text{XBUFS (200)} \] ____________

\[ \text{XBUFSU (200)} \] ____________

\[ \text{ADEST (200)} \] A | X | C | S

\[ \text{XDEST (200)} \] A | B | X | C | S

\[ \text{ABUVS (200)} \] ____________

\[ \text{ABUVS (200)} \] ____________

Example:

Does ADEST have BUREG? \( \equiv \) Is ABUVS = 1?

Is ADEST AXYBUREG? \( \equiv \) Is XBUFS = 1?
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.

REG BUSY
  = 0 = 1 WAIT

BU FULL
  = 0 = 1

REG BUSY - 1
PASS - 1

BU FULL - 0
BU BUSY - 0

REG BUSY - 0
REG DATA PASS - 0
REG BUSY - 0
BU BUSY - 0

LOADA INX

(TREAT AS ANY OP WITH BU AS DEST)

BU BUSY
  = 0 = 1 WAIT

LOAD GOES

BU BUSY - 1
Back Up Register

BU Busy
BU Full
Pass

wait for all work to finish

Pass

yes = 1

BU Full

Pass ← 1

Front ← BU

BU Full ← 0
BU Busy ← 0

load in X

BU Busy

yes = 0

BU Busy ← 1

wait

BU Busy

Pass ← 0

Front ← BU

BU Full ← 1

Pass ← 0

BU Busy ← 0
Notes:
1) data arrival must follow load
2) load, replace in any order
3) replace, replace illegal as long outstanding use (ie data wired)
at soft interrupt: both load 2 executed and data returned
  hence only 000 possible
at hard interrupt: all 5 states possible:
  some misas P=2 split possible
  missing data misses L=0 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 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 (i)
(ii) IF ST = 0, remove skip flag at end of cycle (scan)
EXECUTION OF EXIT INSTRUCTIONS

(C A X UNITS)

I  INTLK + Qφ: 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 NφGφ EXITS IN STACK. IF ANY, SET
    XHOLDT(AHOLDT) TO ONE. OTHERWISE ZERφ

    (ii) ALSO AT END OF .1 SCAN, CHECK FOR
    Gφ EXIT WITH ET. IF ANY, SET XFRCT(AFRC)
    TO ONE. OTHERWISE ZERφ.

    (iii) IF ET EXECUTION PERF BY MERELY
    GOING AT END OF CYCLE. Thus INTLK ON
    QPS BELOW IS GONE AT NXT CYCLE.

    (iv) IF ET, NφP AND MARK Gφ AT END
    OF .1 SCAN ALL QPS BELOW EXIT.
### 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)

<table>
<thead>
<tr>
<th>Long Bos</th>
<th>INTK, Bus</th>
<th>ADD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Test Cond</td>
<td>EBA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Short Bos</th>
<th>Dest INTK</th>
<th>Out Bus INTK</th>
<th>INTK, Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Test Cond</td>
<td>EBA</td>
<td></td>
</tr>
</tbody>
</table>

- 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.)

### II. Exit Inst INTK

A. All Exit Inst's Have Common Source (EHT)

### III. Special INTK

A. All Code Below an Exit Flag Is INTK'd.

### IV. Exit Execution

A. Contender

1. Contender
2. Cycle
3. Set ENT
4. Reset Exit Flag
5. NOP Code
6. Reset Exit Flag

---

191

L. Conway
Archives
Exit History Table

<table>
<thead>
<tr>
<th>ER</th>
<th>BE</th>
<th>ET</th>
<th>EBA</th>
</tr>
</thead>
</table>

ER = Exit Resolved
BE = Branch Executed
ET = Exit Taken
EBA = Low Order 3 Bits of the Effective Branch Address.

VI Definitions:
1) $\text{BOSC} = \left[ \text{Contender Exit} \land (\text{Disp Bos Ahead of 1st Exit Disp Exit} \lor \text{Any Contender Bos}) \right] \lor \left[ \text{Contender Exit} \land \text{Any Contender Bos Ahead of 1st Contender Exit} \right]

2) XEP - Exit Passed to X Unit Dispatchers = SEE FLOW-CHART

3) BNOP -

\[ \text{BNOP} \]

\[ \text{RST Valid Bit on All Disp if Contender Bos} \]
BRANCH IN X

- ONE BO's PER CYCLE

<table>
<thead>
<tr>
<th>CYC 1</th>
<th>CYC 2</th>
<th>CYC 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BoS (Conditional): K Field = 0  \{ SET EHT \}
BoS (Unconditional): \{ \}
BoS (Conditional): K Field \# 0  IF E_T

NOTE: SETTINGS OF EXIT HEADER TABLE (E_T, SET BITS)
IMPLIES EBA VALID.

- THE SERIES OF THE EHT (E_T) IS DONE ONLY WHEN THE
  LAST BO'S FIRST PRIOR TO THE EXIT IS REGARDED TO BE
  UNSUCCESSFUL.

EXIT EXPLANATION

WHEN EHT IS COMBINED WITH ALL, NDR, AXC

1) EXIT, IF NOT, STAY OR
2) AXX, NDR, AND THROUGH OF EHT BE ENTERED
3) WITH COMPARE EXIT EHT UNTIL EXIT TO END.
If ER at BUS-XFER Time, Enter at CYC. 3A (ER*ET) or 3B (ER-ET)
CASE I

Branch resolved (taken) before exit is found.

1st Bus
- Xfer 1st Bus
- To Disp

2nd Bus (EBA Word)
- Hold 2nd Bus
- Xfer 2nd Bus
- To Disp

Both Disp's go empty
- Disp code -> Cont.

CASE II

Branch resolved (taken) after exit is found.

1st Bus
- Xfer 1st Bus
- To Disp

2nd Bus (EBA Word)
- Hold 2nd Bus
- Hold
- Xfer 2nd Bus
- To Disp

* Bubble-Up & Enter
* New code in cont

197
L. Conway
Archives
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. & res. Inc.)
   1. If no contender exit:
      A. ER ET - RST 1st Exit in Each Disp (Only one Disp can have Exit)
      B. If no 2nd Disp Exit - Release Bus Xfer Hold
   2. ER ET
      A. Reset 2nd 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 ER
      Always Intk 2nd Exit & All Code Below
   2. ER ET
      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. ER ET
      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

198
REPLACES

X-UNIT

MAX OPS MUST MATCH DX FIELDS WITH X-BYPASS VECTOR ALONG WITH S/D INTERLOCK
IF A-X BUFFER "FULL" & BUFFER ID MATCHES DX FIELDS MOVE A-X BUFFER CONTENTS THROUGH BACK-UP BUSS TO X STACK
IF A-X BUFFER ID DOESN'T MATCH DX FIELD SET X-BYPASS VECTOR & X REG. WAITING VECTOR

A-UNIT

MAX OPS INTERLOCK CREATION A-X BUFFER "FULL"
DATA SENT ON A-X DATA BUSS ALONG WITH DX ID'S IF UNINTERLOCKED
IF DXID & X-BYPASS BUSY VECTOR
MATCH THEN A-X BUFFER BYPASSED
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 BUS & A-X BUFFER "FULL" ACTIVATED.
Load Index Count (UXC) i -> j field {special
multi-step op
Store Multiple Index i+1 -> j field {n

Shifts (Half-word format) j, k -> jk field {This can be
expanded with logic

x+1 -> j field

Shifts (full word format) j -> i field

Index Arithmetic (Shift Content) k + jk field {This can be
expanded to add}
Set Positive Index
Set Negative Index

Branch

k < i0

More Location to Index (MAX) j, k -> jk field {This can be
also (EXITL)
X-UNIT

More Condition Bit to Index Bit i -> j

1/30/67
1:15 P.M.
L. Conway
Archives
### HEXADECIMAL ARITHMETIC

#### ADDITION TABLE

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>0A</td>
<td>0B</td>
<td>0C</td>
<td>0D</td>
<td>0E</td>
<td>0F</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>0A</td>
<td>0B</td>
<td>0C</td>
<td>0D</td>
<td>0E</td>
<td>0F</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>0A</td>
<td>0B</td>
<td>0C</td>
<td>0D</td>
<td>0E</td>
<td>0F</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>0A</td>
<td>0B</td>
<td>0C</td>
<td>0D</td>
<td>0E</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>0A</td>
<td>0B</td>
<td>0C</td>
<td>0D</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>0A</td>
<td>0B</td>
<td>0C</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>0A</td>
<td>0B</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>0A</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
</tr>
</tbody>
</table>

#### MULTIPLICATION TABLE

|   | 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  |     |     |     |     |     |     |     |     |     |
| 4 |     |     | 08  | 0C  | 14  | 18  |     |     |     |     |     |     |     |     |     |
| 5 |     |     |     | 0A  | 0F  | 1E  | 23  | 28  | 33  |     |     |     |     |     |     |
| 6 |     |     |     |     | 0C  | 12  | 18  | 24  |     |     |     |     |     |     |     |
| 7 |     |     |     |     |     | 0E  | 15  |     |     |     |     |     |     |     |     |
| 8 |     |     |     |     |     |     | 10  |     |     |     |     |     |     |     |     |
| 9 |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| A |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| B |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| C |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| D |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| E |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| F |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
### Appendix D. Powers of Two Table

<table>
<thead>
<tr>
<th>$2^n$</th>
<th>2$^{-n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
</tr>
<tr>
<td>4</td>
<td>0.125</td>
</tr>
<tr>
<td>8</td>
<td>0.0625</td>
</tr>
<tr>
<td>16</td>
<td>0.03125</td>
</tr>
<tr>
<td>32</td>
<td>0.015625</td>
</tr>
<tr>
<td>64</td>
<td>0.0078125</td>
</tr>
<tr>
<td>128</td>
<td>0.00390625</td>
</tr>
<tr>
<td>256</td>
<td>0.001953125</td>
</tr>
<tr>
<td>512</td>
<td>0.0009765625</td>
</tr>
<tr>
<td>1024</td>
<td>0.00048828125</td>
</tr>
<tr>
<td>2048</td>
<td>0.000244140625</td>
</tr>
<tr>
<td>4096</td>
<td>0.0001220703125</td>
</tr>
<tr>
<td>8192</td>
<td>0.00006103515625</td>
</tr>
<tr>
<td>16384</td>
<td>0.000030517578125</td>
</tr>
<tr>
<td>32768</td>
<td>0.0000152579578125</td>
</tr>
<tr>
<td>65536</td>
<td>0.0000076289928571429</td>
</tr>
<tr>
<td>131072</td>
<td>0.000003814697265625</td>
</tr>
<tr>
<td>262144</td>
<td>0.0000019071993265625</td>
</tr>
<tr>
<td>524288</td>
<td>0.000000953628987503125</td>
</tr>
<tr>
<td>1048576</td>
<td>0.000000476937153203125</td>
</tr>
<tr>
<td>2097152</td>
<td>0.0000002384185995125</td>
</tr>
<tr>
<td>4194304</td>
<td>0.000000119209295125</td>
</tr>
<tr>
<td>8388608</td>
<td>0.00000005960464475609375</td>
</tr>
<tr>
<td>16777216</td>
<td>0.0000000298023223885625</td>
</tr>
<tr>
<td>33554432</td>
<td>0.0000000149011611968476875</td>
</tr>
<tr>
<td>67108864</td>
<td>0.000000007450580596923828125</td>
</tr>
<tr>
<td>134217728</td>
<td>0.00000000372529094619140625</td>
</tr>
<tr>
<td>268435456</td>
<td>0.00000000186264514923099703125</td>
</tr>
<tr>
<td>536870912</td>
<td>0.000000000931322574615478515625</td>
</tr>
<tr>
<td>1073741824</td>
<td>0.0000000004656612873077392578125</td>
</tr>
<tr>
<td>2147483648</td>
<td>0.00000000023283064365386962890625</td>
</tr>
<tr>
<td>4294967296</td>
<td>0.000000000116415321826934814453125</td>
</tr>
<tr>
<td>8589934592</td>
<td>0.0000000000582076609134874072265625</td>
</tr>
<tr>
<td>17179869184</td>
<td>0.00000000002910383045673370361328125</td>
</tr>
<tr>
<td>34359738368</td>
<td>0.000000000014650226253166563125</td>
</tr>
</tbody>
</table>

*L. Conway Archives*
**PROGRAM SIZE = 370,000 bytes**

1. Reduce PROG to 500 x 30:
   - will easily yield = 30,000 bytes

2. Make TAGS 1 byte instead of 1/2 word:
   - will yield \( 256 \times 70 / 2 = 8960 \) bytes
   - with some prog problems

3. Make QBUF 1 byte instead of 1/2 word. This will take some experimenting but yields:
   - \( 35800 / 2 = 17,900 \) bytes

**TOTAL POSSIBLE REDUCTION BY COMMON**

**REDUCTION = 56,860 bytes**
# Appendix E. Hexadecimal-Decimal Conversion Table

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

<table>
<thead>
<tr>
<th>HEXADECIMAL</th>
<th>DECIMAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 to FFF</td>
<td>0000</td>
</tr>
<tr>
<td>000 to 4095</td>
<td>0000</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>HEXADECIMAL</th>
<th>DECIMAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>4096</td>
</tr>
<tr>
<td>2000</td>
<td>8192</td>
</tr>
<tr>
<td>3000</td>
<td>12388</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0020</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0030</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0040</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0050</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0060</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0070</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0080</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0090</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00A0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00B0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00C0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00D0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00E0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00F0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1020</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1030</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1040</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1050</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1060</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1070</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1080</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1090</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10A0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10B0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10C0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10D0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10E0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10F0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>16334</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>20484</td>
</tr>
<tr>
<td>0000</td>
<td>24578</td>
</tr>
<tr>
<td>0000</td>
<td>28672</td>
</tr>
<tr>
<td>0000</td>
<td>32768</td>
</tr>
<tr>
<td>0000</td>
<td>36864</td>
</tr>
<tr>
<td>0000</td>
<td>40960</td>
</tr>
<tr>
<td>0000</td>
<td>45056</td>
</tr>
<tr>
<td>0000</td>
<td>49152</td>
</tr>
<tr>
<td>0000</td>
<td>53248</td>
</tr>
<tr>
<td>0000</td>
<td>57344</td>
</tr>
<tr>
<td>0000</td>
<td>61440</td>
</tr>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>----</td>
<td>---</td>
</tr>
<tr>
<td>200</td>
<td>0512</td>
</tr>
<tr>
<td>210</td>
<td>0520</td>
</tr>
<tr>
<td>220</td>
<td>0523</td>
</tr>
<tr>
<td>230</td>
<td>0525</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

... (Table continues with similar patterns)
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>800</td>
<td>1350</td>
<td>1357</td>
<td>1528</td>
<td>1539</td>
<td>1540</td>
<td>1541</td>
<td>1542</td>
<td>1543</td>
<td>1544</td>
<td>1545</td>
<td>1546</td>
<td>1547</td>
<td>1548</td>
<td>1549</td>
<td>1550</td>
</tr>
<tr>
<td>810</td>
<td>1552</td>
<td>1553</td>
<td>1554</td>
<td>1555</td>
<td>1556</td>
<td>1557</td>
<td>1558</td>
<td>1559</td>
<td>1560</td>
<td>1561</td>
<td>1562</td>
<td>1563</td>
<td>1564</td>
<td>1565</td>
<td>1566</td>
</tr>
<tr>
<td>820</td>
<td>1568</td>
<td>1569</td>
<td>1570</td>
<td>1571</td>
<td>1572</td>
<td>1573</td>
<td>1574</td>
<td>1575</td>
<td>1576</td>
<td>1577</td>
<td>1578</td>
<td>1579</td>
<td>1580</td>
<td>1581</td>
<td>1582</td>
</tr>
<tr>
<td>830</td>
<td>1584</td>
<td>1585</td>
<td>1586</td>
<td>1587</td>
<td>1588</td>
<td>1589</td>
<td>1590</td>
<td>1591</td>
<td>1592</td>
<td>1593</td>
<td>1594</td>
<td>1595</td>
<td>1596</td>
<td>1597</td>
<td>1598</td>
</tr>
<tr>
<td>840</td>
<td>1600</td>
<td>1601</td>
<td>1602</td>
<td>1603</td>
<td>1604</td>
<td>1605</td>
<td>1606</td>
<td>1607</td>
<td>1608</td>
<td>1609</td>
<td>1610</td>
<td>1611</td>
<td>1612</td>
<td>1613</td>
<td>1614</td>
</tr>
<tr>
<td>850</td>
<td>1615</td>
<td>1616</td>
<td>1617</td>
<td>1618</td>
<td>1619</td>
<td>1620</td>
<td>1621</td>
<td>1622</td>
<td>1623</td>
<td>1624</td>
<td>1625</td>
<td>1626</td>
<td>1627</td>
<td>1628</td>
<td>1629</td>
</tr>
<tr>
<td>860</td>
<td>1632</td>
<td>1633</td>
<td>1634</td>
<td>1635</td>
<td>1636</td>
<td>1637</td>
<td>1638</td>
<td>1639</td>
<td>1640</td>
<td>1641</td>
<td>1642</td>
<td>1643</td>
<td>1644</td>
<td>1645</td>
<td>1646</td>
</tr>
<tr>
<td>870</td>
<td>1647</td>
<td>1648</td>
<td>1649</td>
<td>1650</td>
<td>1651</td>
<td>1652</td>
<td>1653</td>
<td>1654</td>
<td>1655</td>
<td>1656</td>
<td>1657</td>
<td>1658</td>
<td>1659</td>
<td>1660</td>
<td>1661</td>
</tr>
<tr>
<td>880</td>
<td>1662</td>
<td>1663</td>
<td>1664</td>
<td>1665</td>
<td>1666</td>
<td>1667</td>
<td>1668</td>
<td>1669</td>
<td>1670</td>
<td>1671</td>
<td>1672</td>
<td>1673</td>
<td>1674</td>
<td>1675</td>
<td>1676</td>
</tr>
<tr>
<td>890</td>
<td>1677</td>
<td>1678</td>
<td>1679</td>
<td>1680</td>
<td>1681</td>
<td>1682</td>
<td>1683</td>
<td>1684</td>
<td>1685</td>
<td>1686</td>
<td>1687</td>
<td>1688</td>
<td>1689</td>
<td>1690</td>
<td>1691</td>
</tr>
<tr>
<td>900</td>
<td>1692</td>
<td>1693</td>
<td>1694</td>
<td>1695</td>
<td>1696</td>
<td>1697</td>
<td>1698</td>
<td>1699</td>
<td>1700</td>
<td>1701</td>
<td>1702</td>
<td>1703</td>
<td>1704</td>
<td>1705</td>
<td>1706</td>
</tr>
<tr>
<td>910</td>
<td>1707</td>
<td>1708</td>
<td>1709</td>
<td>1710</td>
<td>1711</td>
<td>1712</td>
<td>1713</td>
<td>1714</td>
<td>1715</td>
<td>1716</td>
<td>1717</td>
<td>1718</td>
<td>1719</td>
<td>1720</td>
<td>1721</td>
</tr>
<tr>
<td>920</td>
<td>1722</td>
<td>1723</td>
<td>1724</td>
<td>1725</td>
<td>1726</td>
<td>1727</td>
<td>1728</td>
<td>1729</td>
<td>1730</td>
<td>1731</td>
<td>1732</td>
<td>1733</td>
<td>1734</td>
<td>1735</td>
<td>1736</td>
</tr>
<tr>
<td>930</td>
<td>1737</td>
<td>1738</td>
<td>1739</td>
<td>1740</td>
<td>1741</td>
<td>1742</td>
<td>1743</td>
<td>1744</td>
<td>1745</td>
<td>1746</td>
<td>1747</td>
<td>1748</td>
<td>1749</td>
<td>1750</td>
<td>1751</td>
</tr>
<tr>
<td>940</td>
<td>1752</td>
<td>1753</td>
<td>1754</td>
<td>1755</td>
<td>1756</td>
<td>1757</td>
<td>1758</td>
<td>1759</td>
<td>1760</td>
<td>1761</td>
<td>1762</td>
<td>1763</td>
<td>1764</td>
<td>1765</td>
<td>1766</td>
</tr>
<tr>
<td>950</td>
<td>1767</td>
<td>1768</td>
<td>1769</td>
<td>1770</td>
<td>1771</td>
<td>1772</td>
<td>1773</td>
<td>1774</td>
<td>1775</td>
<td>1776</td>
<td>1777</td>
<td>1778</td>
<td>1779</td>
<td>1780</td>
<td>1781</td>
</tr>
<tr>
<td>960</td>
<td>1782</td>
<td>1783</td>
<td>1784</td>
<td>1785</td>
<td>1786</td>
<td>1787</td>
<td>1788</td>
<td>1789</td>
<td>1790</td>
<td>1791</td>
<td>1792</td>
<td>1793</td>
<td>1794</td>
<td>1795</td>
<td>1796</td>
</tr>
<tr>
<td>970</td>
<td>1797</td>
<td>1798</td>
<td>1799</td>
<td>1800</td>
<td>1801</td>
<td>1802</td>
<td>1803</td>
<td>1804</td>
<td>1805</td>
<td>1806</td>
<td>1807</td>
<td>1808</td>
<td>1809</td>
<td>1810</td>
<td>1811</td>
</tr>
<tr>
<td>980</td>
<td>1812</td>
<td>1813</td>
<td>1814</td>
<td>1815</td>
<td>1816</td>
<td>1817</td>
<td>1818</td>
<td>1819</td>
<td>1820</td>
<td>1821</td>
<td>1822</td>
<td>1823</td>
<td>1824</td>
<td>1825</td>
<td>1826</td>
</tr>
<tr>
<td>990</td>
<td>1827</td>
<td>1828</td>
<td>1829</td>
<td>1830</td>
<td>1831</td>
<td>1832</td>
<td>1833</td>
<td>1834</td>
<td>1835</td>
<td>1836</td>
<td>1837</td>
<td>1838</td>
<td>1839</td>
<td>1840</td>
<td>1841</td>
</tr>
<tr>
<td>999</td>
<td>1842</td>
<td>1843</td>
<td>1844</td>
<td>1845</td>
<td>1846</td>
<td>1847</td>
<td>1848</td>
<td>1849</td>
<td>1850</td>
<td>1851</td>
<td>1852</td>
<td>1853</td>
<td>1854</td>
<td>1855</td>
<td>1856</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>E00</td>
<td>3581</td>
<td>3585</td>
<td>3586</td>
<td>3587</td>
<td>3583</td>
<td>3589</td>
<td>3590</td>
<td>3591</td>
<td>3592</td>
<td>3593</td>
<td>3594</td>
<td>3595</td>
<td>3596</td>
<td>3597</td>
<td>3598</td>
</tr>
<tr>
<td>E10</td>
<td>3600</td>
<td>3601</td>
<td>3602</td>
<td>3603</td>
<td>3604</td>
<td>3605</td>
<td>3606</td>
<td>3607</td>
<td>3608</td>
<td>3609</td>
<td>3610</td>
<td>3611</td>
<td>3612</td>
<td>3613</td>
<td>3614</td>
</tr>
<tr>
<td>E20</td>
<td>3616</td>
<td>3617</td>
<td>3618</td>
<td>3619</td>
<td>3620</td>
<td>3621</td>
<td>3622</td>
<td>3623</td>
<td>3624</td>
<td>3625</td>
<td>3626</td>
<td>3627</td>
<td>3628</td>
<td>3629</td>
<td>3630</td>
</tr>
<tr>
<td>E30</td>
<td>3632</td>
<td>3633</td>
<td>3634</td>
<td>3635</td>
<td>3636</td>
<td>3637</td>
<td>3638</td>
<td>3639</td>
<td>3640</td>
<td>3641</td>
<td>3642</td>
<td>3643</td>
<td>3644</td>
<td>3645</td>
<td>3646</td>
</tr>
<tr>
<td>E40</td>
<td>3648</td>
<td>3649</td>
<td>3650</td>
<td>3651</td>
<td>3652</td>
<td>3653</td>
<td>3654</td>
<td>3655</td>
<td>3656</td>
<td>3657</td>
<td>3658</td>
<td>3659</td>
<td>3660</td>
<td>3661</td>
<td>3662</td>
</tr>
<tr>
<td>E50</td>
<td>3664</td>
<td>3665</td>
<td>3666</td>
<td>3667</td>
<td>3668</td>
<td>3669</td>
<td>3670</td>
<td>3671</td>
<td>3672</td>
<td>3673</td>
<td>3674</td>
<td>3675</td>
<td>3676</td>
<td>3677</td>
<td>3678</td>
</tr>
<tr>
<td>E60</td>
<td>3680</td>
<td>3681</td>
<td>3682</td>
<td>3683</td>
<td>3684</td>
<td>3685</td>
<td>3686</td>
<td>3687</td>
<td>3688</td>
<td>3689</td>
<td>3690</td>
<td>3691</td>
<td>3692</td>
<td>3693</td>
<td>3694</td>
</tr>
<tr>
<td>E70</td>
<td>3696</td>
<td>3697</td>
<td>3698</td>
<td>3699</td>
<td>3700</td>
<td>3701</td>
<td>3702</td>
<td>3703</td>
<td>3704</td>
<td>3705</td>
<td>3706</td>
<td>3707</td>
<td>3708</td>
<td>3709</td>
<td>3710</td>
</tr>
<tr>
<td>E80</td>
<td>3712</td>
<td>3713</td>
<td>3714</td>
<td>3715</td>
<td>3716</td>
<td>3717</td>
<td>3718</td>
<td>3719</td>
<td>3720</td>
<td>3721</td>
<td>3722</td>
<td>3723</td>
<td>3724</td>
<td>3725</td>
<td>3726</td>
</tr>
<tr>
<td>E90</td>
<td>3728</td>
<td>3729</td>
<td>3730</td>
<td>3731</td>
<td>3732</td>
<td>3733</td>
<td>3734</td>
<td>3735</td>
<td>3736</td>
<td>3737</td>
<td>3738</td>
<td>3739</td>
<td>3740</td>
<td>3741</td>
<td>3742</td>
</tr>
<tr>
<td>E100</td>
<td>3744</td>
<td>3745</td>
<td>3746</td>
<td>3747</td>
<td>3748</td>
<td>3749</td>
<td>3750</td>
<td>3751</td>
<td>3752</td>
<td>3753</td>
<td>3754</td>
<td>3755</td>
<td>3756</td>
<td>3757</td>
<td>3758</td>
</tr>
<tr>
<td>E110</td>
<td>3760</td>
<td>3761</td>
<td>3762</td>
<td>3763</td>
<td>3764</td>
<td>3765</td>
<td>3766</td>
<td>3767</td>
<td>3768</td>
<td>3769</td>
<td>3770</td>
<td>3771</td>
<td>3772</td>
<td>3773</td>
<td>3774</td>
</tr>
<tr>
<td>E120</td>
<td>3776</td>
<td>3777</td>
<td>3778</td>
<td>3779</td>
<td>3780</td>
<td>3781</td>
<td>3782</td>
<td>3783</td>
<td>3784</td>
<td>3785</td>
<td>3786</td>
<td>3787</td>
<td>3788</td>
<td>3789</td>
<td>3790</td>
</tr>
<tr>
<td>E130</td>
<td>3792</td>
<td>3793</td>
<td>3794</td>
<td>3795</td>
<td>3796</td>
<td>3797</td>
<td>3798</td>
<td>3799</td>
<td>3800</td>
<td>3801</td>
<td>3802</td>
<td>3803</td>
<td>3804</td>
<td>3805</td>
<td>3806</td>
</tr>
<tr>
<td>E140</td>
<td>3808</td>
<td>3809</td>
<td>3810</td>
<td>3811</td>
<td>3812</td>
<td>3813</td>
<td>3814</td>
<td>3815</td>
<td>3816</td>
<td>3817</td>
<td>3818</td>
<td>3819</td>
<td>3820</td>
<td>3821</td>
<td>3822</td>
</tr>
<tr>
<td>E150</td>
<td>3824</td>
<td>3825</td>
<td>3826</td>
<td>3827</td>
<td>3828</td>
<td>3829</td>
<td>3830</td>
<td>3831</td>
<td>3832</td>
<td>3833</td>
<td>3834</td>
<td>3835</td>
<td>3836</td>
<td>3837</td>
<td>3838</td>
</tr>
</tbody>
</table>

---

L. Conway Archives
LEVEL 2 FEB 67

DS/360 FORTRAN H

DATE 67.278/15, 54.28

COMPILER OPTIONS - NAME = MAIN, OPT = 02, L IMEINT = 50, SOURCE, ERGIC, NOLIST, DECK, LOAD, MAP, NOEDIT, NOID

ISN 0002
IMPLICIT INTEGER*2(A-Z)

ISN 0003
COMMON TIME, XPAR1, XPAR2, XPAR3,
A AINPT, NDAUP, ABUS(50), XINPT, NXBUF,
B NABUS(50), IFADD, IFDST, IFRTN, BRXPT,
C ERAP, ERB(8), ERT(8), NAG, NBUP,
D AHOLDT, KXODT, AFRCT, XFRCT, BOSC,
E GNODP, XFP, AFP, PHI(100), PRINT,
F NSTD, NODH, NDPS, NODBUS, NADSP,
G NXDSP

ISN 0004
COMMON/RLS/ FIRST, NAREGS, NXRREGS, NNRBUS,
A NXBUS, STAS, ACON, XCON, AEMP,
B KEMP, MXD, AFULL(12), XFULL(12), AG(12),
C XGCD(12), NAGD, NXGD, NATEST, NXFTEST,
D NAFAC, NXFAC, ABUSY, ABUSY(200), XBUSY(200),
E XBUSY(200), ABUFF(12, 100), XBUFF(12, 100), ASR(12, 200),
F XSR(12, 200), ADEST(12, 200), XDEST(12, 200), AFAC(12, 15),
G XFRAC(12, 15), AFRAC(14, 15, 20), AFR(14, 15, 20), XFRAC(12, 15),
H AURUSC(4, 10, 20), XURUSC(10), XURUSC(4, 10, 20), XURUSC(10), XURUSC(10),
I AURUS(12, 15), XURUS(12, 15), AURUS(12, 15), XURUS(12, 15),
J AFDLY(15), XFDLY(15), AFDBUSY(15), XFDLY(15), NSLLOT,
K AKRUSZ, XKRPUS(200), XKRPUS(200), AFRULF(200), XFRULF(200),
L Q16(16), SDQ(16), QDBUF, NQDBUF, NQGO,
M QNF(15), QCON, QEMP, MBUS, MBUS,
N LOAD, MEMOY, MEMORY(16), MBUS, EAV,
O MXTIME, OUTVL, IQ(4, 16), EAV, RXN, LONG8R,
P SRT(8), IST(8), SKP, SKP, NSBUF,
Q ASPASS(200), XPASS(200), OUT(2), JORD(16), SSTOP,
R MEMCNTR(16), ABOX(15), XBOX(15), XBXBY(10)

ISN 0005
COMMON/RLS/ LAST

ISN 0006
INTEGER OUT

ISN 0007
REAL MEMOY, MXTIME

ISN 0008
REAL TIME

ISN 0009
EXTERNAL FINIS

ISN 0010
CALL ABNREFINIS

ISN 0011
CONTINUE

ISN 0012
CALL INIT

ISN 0013
CALL JSTART(ENRUN)

ISN 0014
CALL INTMPI

ISN 0015
IF(CHRUN(=0.1) STOP

ISN 0017
1000 CALL TSTEP(EVENT)

ISN 0018
IF(IFSSEP(=Q.1) GO TO 1111

ISN 0019
IF(IFOUTVL, EQ.0.9) WRITE(6, 900) EVENT

ISN 0022
IF(IFTIME, GT, MXTIME) GO TO 999

ISN 0024
GO TO (12, 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

L. Conway Archives

211
| ISN 0028 | 2 CONTINUE |
| ISN 0029 | CALL XLOAD |
| 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 XXCON |
| ISN 0054 | GO TO 1000 |
| ISN 0055 | 11 CONTINUE |
| ISN 0056 | CALL XXEMP |
| ISN 0057 | GO TO 1000 |
| ISN 0058 | 12 CONTINUE |
| ISN 0059 | CALL XBUSY |
| ISN 0060 | GO TO 1000 |
| ISN 0061 | 13 CONTINUE |
| ISN 0062 | CALL XFREE |
| 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 XRIN |
| 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 |
COMMON PHASE 1

DOT1 - pointer to first entry of DO Table
DOT2 - pointer to next available entry - if equal to DOT2, 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)

SKD, SKCE, SKXS - 3 variables for skip control - not needed in Phase I
SKAV, SKAC, SKAS - 3 variables for skip control - not needed in Phase I

CYCL - count of cycles spent in PHASE
KY - condition code indicating path through exception routines (see table for KY)
SY - variable indicating X entry (231) or A entry (233) 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
XIE - 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 B address from PSC or NEAR

DPA - fetch variable - contains address for match with addresses on cards
DEN - fetch variable - contains low value of position within IB being loaded 0-7 for current

DOT - variable indicating length of DO Table - usually set to 6

SBN -
XIER - holding area for XIE to keep it current for dispatcher
AIER - holding area for AIC to keep it current for dispatcher

PTS - 3 pointers within OP area for moving IB's to dispatch positions

XEXT - trigger set to properly process unresolved units that have been dispatched from X
AEXT - trigger set to properly...
**DO Table Entries**

- **DOLE**: first entry
- **DOLES**: next available
- **DOC**: current level

**DOIB** - 2 - pointer to EB 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 address when in X-Disp
**DOXP** - 2 - current pointer for X-ops (only used in Disp) - contains IB address when in A-Disp

**LDOY** - 1 - whisker: this level is valid
**LDDV** - 1 - - - the data for this IB is in OP area
**LOC KD** - 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, 20

---

**IB Entries** - 12

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

**Hist Table** is a push-up with oldest IB indicated by top entry & newest by bottom entry.
| ISN 0024 | LDC (DOTL)=0 |
| ISN 0025 | LDS (DOTL)=0 |
| ISN 0026 | DOTL=DOTL+1 |
| ISN 0027 | IF (DOTL GT DOTL) DOTL=1 |
| ISN 0029 | IF (DOTL LE DUSL) GOTO 912 |

**C** CHECK FOR RETURN OF REQUESTED 1B

| ISN 0031 | 915 CONTINUE |
| ISN 0032 | IF (IPRTN.EQ.0) GOTO 925 |
| ISN 0034 | LIBW(IPRTN)=0 |
| ISN 0035 | PBUF(4)=IPRTN |
| ISN 0036 | IPRTN=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 (CNEQ.NE.0) GOTO 942 |
| ISN 0043 | IF (XX.NE.0) GOTO 930 |
| ISN 0045 | IF (IFDST.NE.0) GOTO 930 |
| ISN 0047 | IF (IDEV(NPSC)=0) GOTO 916 |
| ISN 0049 | NNFA=NNFA+8 |
| ISN 0050 | IF (INPSC.EQ.0) GOTO 916 |
| ISN 0052 | DD 926 I-1, NOPSC |
| ISN 0053 | IF (LPSV(I),EQ.0) GOTO 926 |
| ISN 0055 | IF (NFA.NE.PSCAT(I)) GOTO 926 |
| ISN 0057 | NNFA=PSCB(I) |
| ISN 0058 | SEQ=0 |

**C** CHECK FOR 1B'S FOR REQUEST

| ISN 0060 | 916 CONTINUE |
| ISN 0061 | NFA=NNFA |

**C** SCAN 1B'S FOR REQUEST

| ISN 0062 | DD 917 I = 1, 12 |
| ISN 0063 | IF (IBA(I),EQ,NFA) GOTO 920 |
| ISN 0065 | 917 CONTINUE |
| ISN 0066 | I = 1 |
| ISN 0067 | TACL=HIST(I) |

**C** REQUEST INSTRUCTION FETCH

| ISN 0068 | IFADD=NFA/I |
| ISN 0069 | IFDST=IBCL |
| ISN 0070 | LBITW(IBCL)=5 |

**C** SET NEXT DO ENTRY

| ISN 0071 | 918 CONTINUE |
| ISN 0072 | XX=1 |
| ISN 0073 | PBUF(4)=NFA |
| ISN 0074 | PBUF(2)=IBCL |
| ISN 0075 | PBUF(3)=DUSL |
| ISN 0076 | IBA(I,IBCL)=NFA |
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   GOTo(DOSL)=IBCL
ISN 0082   LDEV(DOSL)=1
ISN 0083   LDCMD(DOSL)=0
ISN 0084   LDDQ(DOSL)=SEQ
ISN 0085   LDDV(DOSL)=0
ISN 0086   LDAM(DOSL)=0
ISN 0087   LDSEL(DOSL)=0
ISN 0088   LDRX(DOSL)=0
ISN 0089   LDXF(DOSL)=0
ISN 0090   DOSL=DOSL+1
ISN 0091   IF (DOSL.GT.DOT) DOSL=1
ISN 0092   GOTO 930

C REQUEST IN IB'S - SET-UP

ISN 0094   920 CONTINUE
ISN 0095   IF (LIBV(1).EQ.0) GOTO 917
ISN 0096   IBCL=1
ISN 0097   DD 921 I=1,12
ISN 0098   IF (HIST(I).EQ.IBCL) GOTO 918
ISN 0101   921 CONTINUE
ISN 0102   GOTO 930

C

ISN 0103   930 CONTINUE
ISN 0104   IF (INPSC.LT.2) GOTO 935
ISN 0105   IF (LPSTV(INPSC).EQ.0) GOTO 935
ISN 0106   MOVE PROGRAM ENTRY INTO PSC
ISN 0107   K=1
ISN 0108   INPSC=1
ISN 0109   DD 932 I=1,N
ISN 0110   IF (LPSTV(I).EQ.0) K=1
ISN 0111   CONTINUE
ISN 0112   932 CONTINUE
ISN 0113   DD 933 I=K,N
ISN 0114   INPSC(I)=INPSC(I+1)
ISN 0115   INPSCB(I)=INPSCB(I+1)
ISN 0116   LPSTV(I)=LPSTV(I+1)
ISN 0117   CONTINUE
ISN 0118   933 CONTINUE
ISN 0119   LPSTV(INPSC)=0
ISN 0120   GOTO 935

C

ISN 0121   941 CONTINUE
ISN 0122   XEX=0
ISN 0123   AXE=0
ISN 0124   EBXS=XEXS
ISN 0125   EBXA=XEXA
ISN 0126   EBX=B=XEXB
ISN 0127   XAF=0
ISN 0128  GOTO 945

ISN 0129  942 CONTINUE
ISN 0130  AE=0
ISN 0131  EBX=AE
ISN 0132  EBX=AE
ISN 0133  EBX=AE
ISN 0134  XAF=1
ISN 0135  GOTO 945

C

ISN 0136  945 CONTINUE

SCAN PSC & INVALIDATE MATCHES

ISN 0137  EBA=(EBX+8)/8
ISN 0138  IF (EBX,NE.0) EBA=EBX/8
ISN 0140  IF (INOPSC.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.EBA) GOTO 955
ISN 0151  946 CONTINUE

ISN 0152  IF (EBX.IE.0) GOTO 950

INSERT BRANCH ENTRY INTO PSC

ISN 0154  943 CONTINUE
ISN 0155  IF (IN.EQ.1) GOTO 949
ISN 0157  IF (LPSVIN.EQ.0) GOTO 949
ISN 0159  K=1
ISN 0160  DO 947 I=1,N2
ISN 0161  IF (LPSV(I).EQ.0) K=1
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  LPSVIN(LPSV(I+1))
ISN 0168  948 CONTINUE
ISN 0169  949 CONTINUE
ISN 0170  LPSVIN=1
ISN 0171  PSCA(I)=EBX
ISN 0172  PSCB(I)=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=DOYL
ISN 0181  951 CONTINUE
L. Conway
Archives
C C MOVE X-OP TO XBUF AND GO TO NEXT

ISN 0069 23 CONTINUE
ISN 0070 IF (XEXT.NE.0) GOTO 25
ISN 0072 LBX(NCTX)=OP(PTR+23)
ISN 0073 XBUS(11)=OP(PTR+23)*256
ISN 0074 XBUS(12)=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 0090 OP(PTR+16)=0
ISN 0091 GOTO 30

ISN 0092 25 CONTINUE
ISN 0093 XEXT=1
ISN 0094 IF (OP(PTR+20).NE.0) GOTO 130
ISN 0095 XEXT=0
ISN 0096 OP(PTR+16)=0
ISN 0097 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

ISN 0105 30 DDXP(DOCL)=DDXP(DOCL)+OP(PTR+15)
ISN 0106 IF (DDXP(DOCL).GT.7) LDXF(DOCL)=1
ISN 0108 XIC=XIC+OP(PTR+15)
ISN 0109 GOTO 111

C C STEP OVER SPACE (III) TO NEXT ENTRY

ISN 0110 34 DDXP(DOCL)=DDXP(DOCL)+1
ISN 0111 IF (DDXP(DOCL).GT.7) LDXF(DOCL)=1
ISN 0113 XIC=XIC+1
ISN 0114 GOTO 111

223
L. Conway
Archives
216 IF (DOCL.GT.19) GOTO 230

20 DOCL=20

212 GOTO 212

C CHECK OP ENTRY

C

70 PTR=(((DOAP*(DOCL*25)+1*(DOCL-11)*23*8))

C BUS= 'A'

BUS=193

IF (OP(PTR+25).EQ.0) GOTO 84

C

71 CONTINUE

C

71 CONTINUE

ISN 0175 IF (AINPT.GT.NABUF) GOTO 230

C

ISN 0177 IF (NCTA.LT.1) GOTO 230

C

ISN 0179 IF (OP(PTR+20).NE.0) GOTO 85

C

ISN 0181 IF (OP(PTR+24).NE.0) GO TO 75

C

C MOVE A-OP TO ABUF AND GO TO NEXT

C

73 CONTINUE

C

ISN 0183 IF (NEXT.NE.0) GOTO 75

C

ISN 0184 LBA(NCTA)=OP(PTR+23)

C

ISN 0185 ABUS(11)=OP(PTR+23)*256

C

ISN 0187 ABUS(2)=OP(PTR+14)

C

ISN 0188 ABUS(3)=OP(PTR+15)

C

ISN 0189 ABUS(4)=OP(PTR+6)

C

ISN 0190 ABUS(5)=OP(PTR+7)

C

ISN 0191 ABUS(6)=OP(PTR+12)

C

ISN 0192 ABUS(7)=OP(PTR+17)

C

ISN 0193 ABUS(8)=OP(PTR+16)

C

ISN 0194 ABUS(9)=OP(PTR+11)

C

ISN 0195 ABUS(10)=OP(PTR+9)

C

ISN 0196 ABUS(11)=OP(PTR+10)

C

ISN 0197 ABUS(12)=OP(PTR+18)

C

ISN 0198 ABUS(13)=OP(PTR+19)

C

ISN 0199 ABUS(14)=OP(PTR+20)

C

ISN 0200 NCTA=NCTA-1

C

ISN 0201 CALL BUSTOA

C

ISN 0203 IF (OP(PTR+24).NE.0) GOTO 75

C

ISN 0205 OP(PTR+17)=0

C

ISN 0206 GOTO 80

C

ISN 0207 75 CONTINUE

C

ISN 0208 AEXT=1

C

ISN 0209 IF (OP(PTR+20).NE.0) GOTO 230

C

ISN 0211 AEXT=0

C

ISN 0212 OP(PTR+17)=0

C

L. Conway Archives
ISN 0213  IF (OP(PTR+9).EQ.0) GOTO 80
ISN 0215  AIC=OP(PTR+13)
ISN 0216  LDADF(DOCL)=1
ISN 0217  GOTO 211

C
C STEP BY OP LENGTH (1,2) TO NEXT ENTRY

C
ISN 0218  DOAP(DOCL)=DOAP(DOCL)+OP(PTR+15)
ISN 0219  IF (DOAP(DOCL).GT.7) LDADF(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  DOAP(DOCL)=DOAP(DOCL)+1
ISN 0224  IF (DOAP(DOCL).GT.7) LDADF(DOCL)=1
ISN 0226  AIC=AIC+1
ISN 0227  GOTO 211

C
ISN 0228  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=DOAP(DOCL)
ISN 0238  GO TO 75

C
ISN 0239  CONTINUE
ISN 0240  AIC=AIC
ISN 0241  IF ((IAEP.NE.0).AND.((ER(BRAP).NE.0)) AEP=0

C
C UPDATE DISP A-FLOW

C
ISN 0243  IF (LDEV(19).EQ.0) GOTO 620
ISN 0245  IF (LDADF(19).EQ.0) GOTO 640
ISN 0247  IF (LDEV(20).EQ.0) GOTO 620
ISN 0249  IF (LDADF(20).EQ.0) GOTO 630
ISN 0251  CONTINUE
ISN 0252  AFI=MODUS
ISN 0253  ADSP=19
ISN 0254  LDEV(19)=0
ISN 0255  LDEV(20)=0
ISN 0256  GOTO 60
ISN 0257  CONTINUE
ISN 0258  J=19
ISN 0259  K=20

L. Conway
Archives
ISN 0260  DDIB(J)=DDIB(K)
ISN 0261  DOST(J)=DOST(K)
ISN 0262  DOPAP(J)=DOPAP(K)
ISN 0263  DDXPI(J)=DDXPI(K)
ISN 0264  LDEV(J)=LDEV(K)
ISN 0265  LDDV(J)=LDDV(K)
ISN 0266  LDAM(J)=LDAM(K)
ISN 0267  LDARF(J)=LDARF(K)
ISN 0268  LDXW(J)=LDXW(K)
ISN 0269  LDXF(J)=LDXF(K)
ISN 0270  LODK(J)=LOCKD(K)
ISN 0271  LSEQ(J)=LSEQ(K)
ISN 0272  PTJ=(J-1)*200
ISN 0273  PTK=(K-1)*200
ISN 0274  DO 633 I=1,200

ISN 0275  633 UPTJ+I=OPTK+I
ISN 0276  635 CONTINUE

ISN 0277  AIC=AIC/R*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     MOVE IB TO DISP PER DO ENTRY
C
C     CHECK THE DEC ORDER LEVEL

ISN 0286  60 DCL=DUTL
ISN 0287  IF ((AEP.NE.0).OR.(AMOLDT.NE.0)) GOTO 210
ISN 0288  61 IF (LDEV(DCL).EQ.0) GOTO 62
ISN 0289  IF (LDAF(DCL).EQ.0) GOTO 63
ISN 0290  62 DCL=DOCL+1
ISN 0291  IF (DOCL.GT.DOT) DOCL=1
ISN 0292  IF (DOCL.NE.DOCL) GOTO 61
ISN 0293  GOTO 210

C     CHECK ADDRESS FOR LEVEL

ISN 0294  63 CONTINUE
ISN 0295  IF (LOCDT(DOCL).NE.0) GOTO 67
ISN 0296  IBCL=DDIB(DOCL)
ISN 0297  IF (IBCL.DEC.0) GOTO 68
ISN 0298  IF (LOSEQ(DOCL).NE.0) GOTO 67
ISN 0299  GOTO 210

C     DATA ADDRESS CHECKED

ISN 0300  67 LOCDT(0DCL)=1

227
ISN 0309  DO0T(DOC1) = 0
ISN 0310  DO0AP(DOC1) = DO0T(DOC1)
ISN 0311  IDCL = DO0B(DOC1)

C CHECK IF WAIT & LIMIT

C
ISN 0312  IF (LIBW(1BCL), NE, 0) GOTO 210

C CHECK IF DATA VALID

C
ISN 0314  IF (LDDV(DOC1), NE, 0) GOTO 68
ISN 0316  LDDV(DOC1) = 1
ISN 0317  SV = 193
ISN 0318  CALL FETCH

C
ISN 0319  IF (KNT(97), NE, 0) GOTO 210
ISN 0321  IF (LDDV(DOC1), EQ, 0) GOTO 210
ISN 0323  68 LDAM(DOC1) = 1
ISN 0324  LDAF(DOC1) = 1

C
ISN 0325  J = ADSP
ISN 0326  K = DOC1
ISN 0327  DO18(J) = DO18(K)
ISN 0328  DO0AP(J) = DO0T(K)
ISN 0329  DO0T(J) = K
ISN 0330  DDXP(J) = TBA(DO1B(K))
ISN 0331  LDEV(J) = LDEV(K)
ISN 0332  LDEV(J) = LDEV(K)
ISN 0333  LDAM(J) = 0
ISN 0334  LDAF(J) = 0
ISN 0335  LDW(J) = LDW(K)
ISN 0336  LDXF(J) = LDXF(K)
ISN 0337  LDCD(J) = LDCD(K)
ISN 0338  LDSEQ(J) = LDSEQ(K)
ISN 0339  PTJ = (J - 1) * 200
ISN 0340  DO 69 I = 1, 200
ISN 0342  69 DO 69 I = 1, 200
ISN 0343  OP = OP + 1
ISN 0344  ADF = ADF + 1
ISN 0345  AIC = AIC + 1
ISN 0346  DO 66 I = 20, 200, 25
ISN 0347  66 IF (OP(PJ) + 1), 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 0355  IF (LDAF(J), NE, 0) GOTO 705
ISN 0358  K=(J-1)*200
ISN 0359  DO 702 I=1,8
ISN 0360  IF (DP(K+17).EQ.0) GOTO 701
ISN 0362  IF (T(IPTK+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 (DP(K+24).NE.0) GOTO 710
ISN 0373  AEP=1
ISN 0374  DP(K+24)=1
ISN 0375  710 IF (AEP.NE.0) GOTO 712
ISN 0377  DP(K+20)=0
ISN 0378  IF (LDXF(J).NE.0) GOTO 712
ISN 0380  AEX=1
ISN 0381  AEX5=DP(K+9)
ISN 0382  AEX8=DP(K+13)
ISN 0383  AEXA=LDXF(J)
ISN 0384  712 CONTINUE
ISN 0385  IF (IXEP.NE.0).AND.(ERIBRX).NE.0) XEP=0

C
C C UPDATE DISP X-FLOW
C
ISN 0387  IF (LDEVII7.EQ.0) GOTO 420
ISN 0389  IF (LDXF(17).EQ.0) GOTO 440
ISN 0391  IF (LDEVII21.EQ.0) GOTO 420
ISN 0393  IF (LDXF(18).EQ.0) GOTO 430
ISN 0395  420 CONTINUE
ISN 0396  XFIL=NBUS
ISN 0397  XDSP=17
ISN 0398  LDEVII7=0
ISN 0399  LDEVII21=0
ISN 0400  GOTO 10
ISN 0401  430 CONTINUE
ISN 0402  J=17
ISN 0403  K=18
ISN 0404  DO18(J)=DO18(K)
ISN 0405  D3ST(J)=D3ST(K)
ISN 0406  D3AP(J)=D3AP(K)
ISN 0407  DUXP(J)=DUXP(K)
ISN 0408  LDEV(J)=LDEV(K)
ISN 0409  LDVF(J)=LDVF(K)
ISN 0410  LDAM(J)=LDAM(K)
ISN 0411  LDAT(J)=LDAT(K)
ISN 0412  LDXM(J)=LDXM(K)
ISN 0413  LUXF(J)=LUXF(K)
ISN 0414  LDCKD(J)=LDCKD(K)
ISN 0415  LDSEQ(J)=LDSEQ(K)
ISN 0416     PTJ=(J-1)*200
ISN 0417     PTK=(K-1)*200
ISN 0418     DO 433 I=1,200
ISN 0419     433 OP(PTJ+1)=OP(PTK+1)
ISN 0420     433 CONTINUE
ISN 0421
ISN 0422
ISN 0423
ISN 0424     LDENV(18)=0
ISN 0425     GOTO 10
ISN 0426     440 CONTINUE
ISN 0427     IF (LDENV(18).NE.0) GOTO 110
ISN 0429     GOTO 435
C
MOVE IB TO DISP PER DO ENTRY
C
CHECK THE DEC ORDER LEVEL
C
ISN 0430     10 DOCL=DOCL
ISN 0431     IF (XEP.NE.0).OR.(XHOLD.NE.0)) GOTO 110
ISN 0433     11 IF (LDENV(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.DOCL) DOCL=1
ISN 0440     IF (DOCL.NE.DOCL) GOTO 11
ISN 0442     GOTO 110
C
CHECK ADDRESS FOR LEVEL
C
ISN 0443     13 CONTINUE
ISN 0444     IF (LOCKD(DOCL).NE.0) GOTO 17
ISN 0446     IBCI=DOIB(DOCL)
ISN 0447     IF (IIBC/(IBI(IBL)/(IBI/8))) GOTO 17
ISN 0449     IF (LDSEQ(DOCL).NE.0) GOTO 17
ISN 0451     LOSEQ(DOCL)=1
ISN 0452     ASA=XIC
ISN 0453     SY=231
ISN 0454     CALL SEARCH
ISN 0455     GOTO 110
C
DATA ADDRESS CHECKED
C
ISN 0456     17 LOCKD(DOCL)=1
ISN 0457     DOST(DOCL)=0
ISN 0458     DDX(DOCL)=DOST(DOCL)
ISN 0459     IBCI=DOIB(DOCL)
C
CHECK IB WAIT & LIMIT
C
ISN 0460     IF (LIBW(IBCL).NE.0) GOTO 110
C CHECK IF DATA VALID
C
ISN 0462 IF (LDV(DC), NE, 0) GOTO 18
ISN 0464 LDV(DC) = 1
ISN 0465 SY = 231
ISN 0466 CALL FETCH

ISN 0467 IF (KNT(J), NE, 0) GOTO 110
ISN 0469 IF (LDV(DC), EQ, 0) GOTO 110
18 LDW(DC) = 1
ISN 0472 LDW(DC) = 1

ISN 0473 J = XSP
ISN 0474 K = DC
ISN 0475 DOIB(J) = DOIB(K)
ISN 0476 DOXP(J) = DOXP(K)
ISN 0477 DOCT(J) = K
ISN 0478 DQAP(J) = IB(DOIB(K))
ISN 0479 LDAP(J) = LDAP(K)
ISN 0480 LDV(J) = LDV(K)
ISN 0481 LDAM(J) = LDAM(K)
ISN 0482 LDAD(J) = LDAD(K)
ISN 0483 LDWM(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 DD 19 1 = 1, 200
ISN 0490 OP(PTJ + 1) = OP(PTJ + 1)
ISN 0491 XFIJ = XFIK - 1
ISN 0492 XSP = XSP + 1
ISN 0493 XIC = XIC + 1
ISN 0494 DD 16 I = 20, 200, 25
ISN 0495 IF (UPP(PTJ + 1), NE, 0) GOTO 110
ISN 0497 IF (XFIK, NE, 0) GOTO 12

110 CONTINUE
ISN 0499 BOSC = 0
ISN 0501 J = 17
ISN 0502 505 CONTINUE
ISN 0503 IF (LDAP(J), EQ, 0) GOTO 510
ISN 0505 IF (LDXF(J), NE, 0) GOTO 510
ISN 0507 K = (J - 1) * 200
ISN 0508 DD 504 I = 1, 8
ISN 0509 IF TPK(K-16), EQ, 0) GOTO 503
ISN 0511 IF TPK(K+20), NE, 0) GOTO 515
ISN 0513 IF TPK(K+18), EQ, 0) GOTO 503

231 L. Conway Archives
ISN 0515   IF (BNOP.EQ.0) GOTO 502
ISN 0517   DPI(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 (DPI(K+24).NE.0) GOTO 516
ISN 0529   XEP=1
ISN 0530   DPI(K+24)=1
ISN 0531   516  IF (XEP.NE.0) GOTO 520
ISN 0533   DPI(K+20)=0
ISN 0534   IF (LDAFTJ).NE.0) GOTO 520
ISN 0536   XEX=1
ISN 0537   XEX=XEX+DPI(K+9)
ISN 0538   XEXX=DPI(K+13)
ISN 0539   XEXA=DQAP(J)
ISN 0540   520  CONTINUE
ISN 0541   911  CONTINUE
C        LOAD IB'S
C
ISN 0542   IF (PHEND.GT.0) GOTO 930
ISN 0544   CALL PH18S
ISN 0545   930  CONTINUE
ISN 0546   CYCL=CYCL+1
ISN 0547   DO 999  I=1,100
ISN 0548   999  PHII(1)=0
ISN 0549   PHII(100)=PHEND
ISN 0550   IF (PHEND.GT.1) GOTO 931
ISN 0552   CALL PH1MAP
C
ISN 0553   931  CONTINUE
ISN 0554   IF ((KNT18).EQ.0).OR.(KNT19).EQ.0)) GOTO 932
ISN 0556   IF (LDEV17).OR.LDEV18).OR.LDEV197).OR.LDEV120) GOTO 932
ISN 0558   PHEND=2
ISN 0559   932  CONTINUE
ISN 0560   RETURN
ISN 0561   END
L Conway
Archives
C

TO USE UNROLLER OUTPUT, SET INOP FROM JTRACE

ISN 0042
    GO 131 I=1,5
    INOP(I)=JTRACE(INSLOC, 1)

ISN 0043 131 CONTINUE

ISN 0044
    MN1=JTRACE(INSLOC, 21)

ISN 0045
    MN2=JTRACE(INSLOC, 22)

ISN 0046
    MN3=JTRACE(INSLOC, 23)

ISN 0047
    MN4=JTRACE(INSLOC, 24)

ISN 0048
    MN5=JTRACE(INSLOC, 25)

ISN 0049
    MN6=JTRACE(INSLOC, 26)

ISN 0050

ISN 0051
    INSLOC=INSLOC+1

ISN 0052
    INOP(25)=1

ISN 0053
    IF (INOP(14).EQ.099) GOTO 123

ISN 0054
    INOP(16)=0

ISN 0055
    INOP(17)=0

ISN 0056
    INOP(18)=0

ISN 0057
    INOP(19)=0

ISN 0058
    INOP(20)=0

ISN 0059
    INOP(21)=0

ISN 0060
    INOP(16)=INOP(14)+1

ISN 0061
    INOP(17)=INOP(14)+2

ISN 0062
    INOP(18)=INOP(14)+3

ISN 0063
    INOP(19)=INOP(14)+29

ISN 0064
    INOP(20)=INOP(14)+31

ISN 0065
    IF (INOP(17).NE.1) (INOP(10)=0

ISN 0066
    PNT=MOD(INOP(11),26)+1

ISN 0067
    INOP(23)=LETTER(PNT)

ISN 0068 123 CONTINUE

ISN 0069
    IF (PRINT.GT.1) GOTO 121

ISN 0070
    WRITE(6,802) INOP(23), INOP(1), MN1, MN2, MN3, MN4, MN5, MN6.

ISN 0071
    X = INOP(1), I=5,17, PNT

ISN 0072 121 CONTINUE

ISN 0073
    IF (INOP(14).EQ.099) GOTO 152

ISN 0074
    IF (INOP(11).NE.DFA) GOTO 125

ISN 0075
    CONTINUE

ISN 0076
    PNT=PNT+DEN*25

ISN 0077
    ON 122 I=1,25

ISN 0078
    OP(PTR+1)=INOP(1)

ISN 0079
    CONTINUE

ISN 0080
    INOP(15)=0

ISN 0081
    CONTINUE

ISN 0082
    INOP(17)=0

ISN 0083
    CONTINUE

ISN 0084
    PTR=PAR+1

ISN 0085
    DFM=DEN+OP(PTR+15)

ISN 0086
    DFA=DFA+OP(PTR+15)

ISN 0087
    IF (DEN.LE.7) GOTO 120

ISN 0088
    RETURN

235

L. Conaway

Archives
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 (INOP11).LT.DFA1) GOTO 127
ISN 0096  IF (INOP11).GE.DFA2) GOTO 127
ISN 0098  DFA=INOP11
ISN 0099  DFN=MOD(DFA,8)
ISN 0100  GOTO 126
C
ISN 0101  127 CONTINUE
ISN 0102  KY=3
ISN 0103  KNT(1)+KNT(3)+1
ISN 0104  128 CONTINUE
ISN 0105  IF (PAR,NE,0) RETURN
ISN 0107  Lenovo(DOCL)=0
ISN 0108  Lenovo(DOCL)=0
ISN 0109  RETURN
C
ISN 0111  150 CONTINUE
ISN 0112  KY = 10
ISN 0113  GOTO 2000
ISN 0114  152 CONTINUE
ISN 0115  IF (SY.EQ.23) GOTO 157
ISN 0117  IF (SY.EQ.193) GOTO 154
ISN 0119  KY=6
ISN 0120  GOTO 123
ISN 0121  153 KNT(9)=1
ISN 0122  KY=6
ISN 0123  GOTO 123
ISN 0124  154 KNT(9)=1
ISN 0125  KY=5
ISN 0126  GOTO 129
C
C END OF INPUT DATA
C
ISN 0111  157 CONTINUE
ISN 0112  KY = 10
ISN 0113  GOTO 2000
ISN 0114  152 CONTINUE
ISN 0115  IF (SY.EQ.23) GOTO 157
ISN 0117  IF (SY.EQ.193) GOTO 154
ISN 0119  KY=6
ISN 0120  GOTO 123
ISN 0121  153 KNT(9)=1
ISN 0122  KY=6
ISN 0123  GOTO 123
ISN 0124  154 KNT(9)=1
ISN 0125  KY=5
ISN 0126  GOTO 129
C
C SEARCH ROUTINE
C SEARCH 18'S FOR ADDRESS
C
ISN 0127  C ENTRY SEARCH
C
ISN 0128  KY=9
ISN 0129  SCT=SCT+1
ISN 0130  ASH=ASH/8*8
ISN 0131  DO 201 I=1,12
ISN 0132  IF (ASH.EQ.18A111) GOTO 210
ISN 0135 201 CONTINUE

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

REQUEST INSTRUCTION FETCH

C

ISN 0139  IFDST2=IFDST
ISN 0140  IF (IFDST2.NE.7) LHV(IFDST2)=0
ISN 0141  LHV(IFDST2)=1
ISN 0142  LHV(IFDST2)=5
ISN 0143  ISAV(IFDST2)=ASA/R#8
ISN 0144  IF (IFDST2.EQ.7) PRUE(1)=ASA
ISN 0145  IF (IFDST2.EQ.7) PRUE(2)=IBCL

C

ISN 0146  PRUE(3)=DOCL

C

ISN 0150  206 CONTINUE
ISN 0151  XX=1
ISN 0152  NFA=IFDST
ISN 0153  IF (LDXW(DOCL).OR.LDAW(DOCL)) GOTO 205

C

ISN 0154  205 CONTINUE
ISN 0155  DOLS(DOCL)=IBCL
ISN 0156  LDOLV(DOCL)=0
ISN 0157  LOGV(DOCL)=0
ISN 0158  DOCL=DOCL

C

ISN 0159  203 CONTINUE
ISN 0160  DOLS(DOCL)=1
ISN 0161  IF (DOLS.GT.DOCL) DOLS=1
ISN 0162  IF (DOLS.EQ.DOCL) GOTO 204
ISN 0163  DOLS(DOCL)=0
ISN 0164  DOLS(DOCL)=0
ISN 0165  GOTO 203

C

ISN 0166  204 CONTINUE
ISN 0167  DOLS(DOCL)=1
ISN 0168  IF (DOLS.GT.DOCL) DOLS=1
ISN 0169  RETURN

C

ISN 0170  205 CONTINUE
ISN 0171  KY = 15
ISN 0172  GOTO 2000

C

FOUR-WORD ADDRESS MATCH - SET POINTER TO ENTRY

C

ISN 0173  210 CONTINUE
ISN 0174  IF (LHV(1).EQ.0) GOTO 201
ISN 0175  IF (LDXW(DOCL).OR.LDAW(DOCL)) GOTO 215
ISN 0180   I1C1 = 1
ISN 0181   IF (I1C1 .EQ. HIST(12)) GOTO 214
ISN 0183   DO 211 I = 1,11
ISN 0184   IF (I1C1 .EQ. HIST(11)) GOTO 212
ISN 0186   211 CONTINUE
ISN 0187   212 CONTINUE
ISN 0188   DO 213 J = 1,11
ISN 0189   213 HIST(J) = HIST(J+1)
ISN 0190   HIST(12) = I1C1
ISN 0191   214 CONTINUE
ISN 0192   GOTO 216
ISN 0193   215 CONTINUE
ISN 0194   KY = 14
ISN 0195   GOTO 2050
ISN 0196
C
C
ENTRY INTPH1
C
INITIALIZE PHASI COMMON
ISN 0197   DO 12 I = 1,433
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   PHN = 0
C
C
SET INITIAL VALUES
ISN 0204   DOTL = 1
ISN 0205   DMSL = 1
ISN 0206   DMT = 1
ISN 0207   IT (NMOD.TE.01) DOT = NMOD
C
C
AND SET XIC, AIC, ETC.
ISN 0209   PNT = 1
ISN 0210   AIC = PSFAADD
ISN 0211   XIC = AIC
ISN 0212   NFA = AIC
ISN 0213   AICR = AIC
ISN 0214   XICR = XIC
ISN 0215   IRA(1) = NFA
ISN 0216   LEV(1) = 1
ISN 0217   LEV(1) = 0
ISN 0218   LDEV(1) = 1
ISN 0219   DMTB(1) = 1
ISN 0220   DMSL = 2
ISN 0221   IFADD = NFA/2
ISN 0222   IFPS = 1

238
ISN 0223  J=1
ISN 0224    DO 11 I=1,12
ISN 0225  J=J+1
ISN 0226    IF (J.GT.17) J=1
ISN 0227    11 HIST(I)=J
ISN 0228    RETURN

ROUTINE TO DUMP VARIABLES AND BUFFERS

ISN 0230      ENTRY DUFER
ISN 0231      GOTO 2000

ISN 0232  2011 FORMAT (*2XY='I12,' DOOT='I12,' DOSL='I12,' DOL='I12,'  
       * * IRA='I12,' HIS='I12,' 2X,'XIC='I6,4(1XII),' AIC='I6,4(1XII),
       * * ASA='I6')
ISN 0233  2012 FORMAT (* DDN='9,4(1XII),6(1XIII)')
ISN 0234  2013 FORMAT (* DIBA='9,12(1XIII),1(1XI)')
ISN 0235  2014 FORMAT (* 12(1XI)')
ISN 0236  2015 FORMAT (*.BAR='9,3(1XII),1X16)')
ISN 0237  2016 FORMAT (*. ***; 9(1XI6),/)
ISN 0238  2017 FORMAT (* CYCL XINPT NBUF AINPT NABUF SY SCT',
       * * FCT, PTR')
ISN 0239  2018 FORMAT (*. IB,ST,AP,XP, EV,CKD,SEQ,OV,AW,AF,WX, XF')
ISN 0240  2019 FORMAT (* C KNT= * 101(1XII)')

ISN 0241      2020 CONTINUE
ISN 0242      WRITE (6,2011) KY, DOOT, DOL, DOCL, DOLC, IRA, HISL, XIC, BRXP, SKXV,
       * SKC, SKXS, ALC, BRAP, SKAV, SKAC, SKAS, ASA
ISN 0243      WRITE (6,2017)
ISN 0244      WRITE (4,2016) CYCL, XINPT, NBUF, AINPT, NABUF, SY, SCT, FCT, PTR
ISN 0245      WRITE (6,2020) (KNT(I),I=1,10)

ISN 0246      J=1
ISN 0247      DO 2004 I=1,12
ISN 0248      WR(1) = (BAI1)
ISN 0249      WR(I+1) = (BAI1)
ISN 0250      WR(I+2) = (BAI1)
ISN 0251      J=J+3
ISN 0252      2003 CONTINUE

ISN 0253      WRITE (6,2013) (WR(I),I=1,36)

ISN 0254      I=DOT-1
ISN 0255      WRITE (6,2019)
ISN 0256      DO 2001 I=1,DOIT,3
ISN 0257      DO 2002 K=1,36,12
ISN 0258      I=I+1
ISN 0259      IF (I.GT.DOIT) I=1

L. Conway
Archives
L. Conway
Archives
ISN 0317 1003 K=K+25
ISN 0318 1004 CONTINUE

ISN 0319 PH1(21)=0
ISN 0320 IF (LOFV(19).EQ.0) GOTO 1006
ISN 0322 IF (LOAF(19).NE.0) GOTO 1006
ISN 0324 PHI(21)=LETR(D0IT(19)+1)
ISN 0325 K=3609
ISN 0327 DO 1005 J=23,30
ISN 0328 IF (OP(K+17), EQ, 0) GOTO 1005
ISN 0330 PHI(J)=OP(K+23)
ISN 0331 IF (UP(K+15), EQ, 1) GOTO 1005
ISN 0333 PHI(J+1)=PHI(J)
ISN 0334 1005 K=K+25
ISN 0335 1006 CONTINUE
ISN 0336 PHI(33)=PHI(33)
ISN 0337 PHI(33)=0
ISN 0338 IF (LOFV(20), EQ, 0) GOTO 1008
ISN 0340 IF (LOAF(20), NE, 0) GOTO 1008
ISN 0342 PHI(33)=LETR(D0IT(20)+1)
ISN 0343 PHI(33)=LETR(D0IT(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 PHI(J)=31(K+23)
ISN 0349 IF (OP(K+15), EQ, 1) GOTO 1007
ISN 0351 PHI(J+1)=PHI(J)
ISN 0352 1007 K=K+25
ISN 0353 1008 CONTINUE
ISN 0354 PHI(41)=0
ISN 0355 DO 1009 J=1,12
ISN 0356 PHI(J+1)=LETR(HIST(J)+1)
ISN 0357 1009 CONTINUE
ISN 0358 PHI(56)=LETR(D0ITL)
ISN 0359 PHI(56)=LETR(D0ITL+1)
ISN 0360 DO 1010 J=1,100
ISN 0361 IF (LOFV(3), NE, 0) PHI(J+56)=LETR(D0ITR(J)+1)
ISN 0362 1010 CONTINUE
ISN 0363 PSUF(4)=KY
ISN 0364 KY=0
ISN 0365 J=66
ISN 0366 P+1(J+1)=LETR(PBUF(3)+1)
ISN 0367 PHI(J+2)=LETR(PBUF(1+1))
ISN 0368 PHI(J+6)=LETR(PBUF(4)+1)
ISN 0369 PHI(J+5)=LETR(PBUF(3)+1)
ISN 0370 J=71
ISN 0371 DO 1012 T=1,6
ISN 0372 IF (LPSV(11), PHI(J+1)=LETR(138)
ISN 0373 1012 CONTINUE
ISN 0374 J=80

L. Conway
Archives
ISN 0377  PHI(J+1)=LBA(4)
ISN 0378  PHI(J+2)=LBA(3)
ISN 0379  PHI(J+3)=LBA(2)
ISN 0380  PHI(J+4)=LBA(1)
ISN 0381  PHI(J+5)=LBA(4)
ISN 0382  PHI(J+6)=LBA(3)
ISN 0383  PHI(J+8)=LBA(2)
ISN 0384  PHI(J+9)=LBA(1)
ISN 0385  PHI(J+11)=LETTRANSC+1)
ISN 0386  PHI(J+12)=LETTRAXEP+1)
ISN 0387  PHI(J+13)=LETTRAXEP+1)
ISN 0388  ON 1011 K=1,100
ISN 0389  1011 PHI(K)=PHI(K)+256
ISN 0390  IF (PRINT,45,0) RETURN
ISN 0391  WRITE (*,XIO) CYCL,((PHI[I],I=1,100)
ISN 0392  S10 FORMAT (*,PHI[I],I=1,100)
ISN 0393  RETURN
ISN 0394  END
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, NOD

ISN 0002  SUBROUTINE UNROLL
ISN 0003  IMPLICIT INTEGER*2(J)
ISN 0004  IMPLICIT INTEGER*2(R)
ISN 0005  DIMENSION JSMTAB(200), KSYMAD(200)
ISN 0006  DIMENSION JABSOP(300), JCDTYP(300), JASTCL(300)
ISN 0007  DIMENSION JCARDN(300), JACSLC(300), JROWNUM(300)
ISN 0008  DIMENSION JQOLP(300)
ISN 0009  DIMENSION JSDBB(300)
ISN 0010  DIMENSION JIN(300,80)
ISN 0011  DIMENSION JTEMP(6), JEND(4)
ISN 0012  DIMENSION JTEMP(7)
ISN 0013  DIMENSION JIREG(2), JKREG(2)
ISN 0014  DIMENSION JHFLD(5)
ISN 0015  DIMENSION JSTOP(6)
ISN 0017  DATA JZERO/*S*/
ISN 0019  DATA JBLANK, JSMCLN/*S*, Z*/
ISN 0020  DATA JASTER/*S*/
ISN 0021  DATA JLPARN, JRPARN, JCUMMA/*S*, C*, R*, Z*/
ISN 0022  COMMON /AKE2/ JN8B(36), JOPDCE(6,256), JS1DB(256),
          #JITYPE (256), JEXITF(300),
ISN 0023  DIMENSION RMTOT(200), RNUMSF(200), RLPWRT(200),
          XROWCRT(200), RSTYTP(200), RNRSMSF(200), JACDNO(200)
ISN 0024  COMMON/AREA4/ JN(80), JS, JS
ISN 0025  COMMON/PROG/ JTRACE(1000, 301), INSLOC
ISN 0026  INTEGER*2 INSLOC
ISN 0027  DIMENSION JNUNUM(10)

C  INITIALIZE PROG TRACE AREA TO 0
C  DO 2000 LII = 1,1000
C  DO 2000 LK = 1,130
C  JTRACE(LII,LK) = 0
C  INTR = 1
C  INSLOC = 1
C  WRITE(6,3000)
C  WRITE(6,3001)
C  JDECK = 1

C  INITIALIZE ONE ENTRY IN JSMTAB
C  III = 1
C  DO 60 LJJ = 1,8
C  JSMTAB(III,LJJ) = JBLANK
C  INITIALIZE
C  IJK = 0
C  LDC = 0
C  M = 1
C  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    RCNSF(LLL)=0
ISN 0051    RLPNTR(LLL)=0
ISN 0052    ROWCRT(LLL)=1
ISN 0053    RSPFYP(LLL)=1
ISN 0054    RNMSF(LLL)=1

ISN 0055  2005 CONTINUE
ISN 0056  DO 2006 LLL=1,300
ISN 0057    JEXITP(LLL)=0
ISN 0058  2006 CONTINUE

C

ISN 0059  10 IF(IJK>IKK) GO TO 11
ISN 0060    IF(IJK.LT.300) GO TO 11
ISN 0061    WRITE(6,3)
ISN 0062    STOP
ISN 0063

ISN 0064  11 LCARDN(IJK) = IJK
C CHECK FOR COMMENT CARD

ISN 0065  12 READ (5,10) JN
ISN 0066    IF (JN(J1) .EQ. JASTER ) GO TO 12

ISN 0067  1 = 1
ISN 0068    CALL BLNKCK(I1,INT)
ISN 0069    I= INT
ISN 0070    KCOUNT = 0
ISN 0071  20 KCOUNT = 0
ISN 0072    LACSLC(IJK) = LOC
C SCAN TO PICK UP WORD

ISN 0073  28 KCOL = 1
ISN 0074  30 I = I+1
ISN 0075  DO 35 I = 1,36
ISN 0076    IF (JN(J1) .EQ. JNB(I1)) GO TO 30

ISN 0077  35 CONTINUE
C SAVE LENGTH OF WORD

ISN 0078    KSUM = I-KCOL
ISN 0079    KCOL = I-1
C CHECK FOR BLANK OR SEMICOLON
ISN 0080    IF ( JN(J1) .EQ. JASTER ) GO TO 98

ISN 0081    IF (JN(J1) .EQ. JBLANK) GO TO 100
ISN 0082    IF (JN(J1) .EQ. JSMCLN) GO TO 40
C BRANCH TO SEARCH SYMBOL TABLE

ISN 0083  40 ASSIGN 205 TO IA
ISN 0084  GO TO 250
ISN 0085  205 GO TO (1201,18)

C LABEL WAS IN SYMBOL TABLE
C STORE CURRENT LOCATION
; ISN 0090 C KSYMAD(MX-1) = LOC
; STORE CURRENT CARD NUMBER
ISN 0091 C JACNO (MX-1) = LCARDNJJK
ISN 0092 C GO TO 215
C LABEL NOT IN SYMBOL TABLE
ISN 0093 C KSYMAD(111 - I) = LOC
; STORE CURRENT CARD NUMBER
ISN 0094 C JACNO(111 - I) = LCARDNJJK
ISN 0095 C I = I + 1
ISN 0096 C CALL BLNCKK (1,INT)
ISN 0097 C I = INT
ISN 0098 C IF (I < 80) 28,990,990
ISN 0099 C JEXITF(JJK) = 1
ISN 0100 C CHECK FOR END CARD OR OP CODE
ISN 0101 C L = 1
ISN 0102 C DD 105 T = KCOL,KCOL
ISN 0103 C IF (JN (IJ) .EQ. JEND(I)) GO TO 105
ISN 0104 C GO TO 130
ISN 0105 C END CARD
ISN 0106 C JCDTYP(IJK) = 12
ISN 0107 C WRITE (6,6) JN
ISN 0108 C MOVE CURRENT CARD TO FILE
ISN 0109 C DD 106 N = 1,80
ISN 0110 C JIN (IJK,N) = JN(N)
ISN 0111 C GO TO 699
ISN 0112 C 990 C WRITE (6,7)
ISN 0113 C GO TO 180
ISN 0114 C 993 C WRITE (6,4)
ISN 0115 C GO TO 708
C CHECK FOR OP-CODE
ISN 0116 C JASTCL (IJK) = 1
ISN 0117 C SAVE STARTING COLUMN OF OP-CODE
ISN 0118 C JCOL(JJK) = KCOL
C CLEAR JTEMP
ISN 0119 C DO 132 K=1,6
ISN 0120 C JTEMP(1) = JBLANK
ISN 0121 C L = 0
ISN 0122 C DO 135 II = KCOL,KCOL
ISN 0123 C L = L+1
ISN 0124 C JTEMP(L) = JN(II)
C TOTAL = KCOL - KCOL + 1
C SEARCH OP CODE TABLE
ISN 0125 C DO 140 II=1,256
ISN 0126 C DD 139 L=1,6
ISN 0127 C IF(JOPCODE(L,II).NE.JTEMP(L)) GO TO 140
C CONTINUE
ISN 0128 C 139 CONTINUE
ISN 0129 C GO TO 160
ISN 0131 140 CONTINUE
C UNSUCCESSFUL SEARCH
ISN 0132 WRITE(6,8)
ISN 0133 GO TO 180
ISN 0134 160 CONTINUE
ISN 0135 161 JABSOP(IJK)=II
ISN 0136 IF (II.EQ.256) JABSOP(IJK)=999
ISN 0138 JCDTYP(IJK,J}=JITYPEII)
ISN 0139 JSIDBB(IJK,J)=JSIDBB(I)
ISN 0140 IF (JSIDBB(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) = JMN(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 JBRWMH (IJK) = NN
ISN 0157 NN = NNN
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. JKPARN) 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
ISN 0178  KCOLL = 1 - 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

ISN 0183  RLPNTR(NNN) = MX - 1
ISN 0184  GO TO 303
C LABEL NOT IN POINTER

ISN 0185  410 RLPNTR(NNN) = III - 1
ISN 0186  GO TO 303
C ASTERISK

ISN 0187  320 RSFYP(NNN) = 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(NNN) = J - 27
ISN 0193  J = J + 1
ISN 0194  DO 307 J = 27, 36
ISN 0195  IF (JN(I) .EQ. JNB(I)) GO TO 351

ISN 0197  307 CONTINUE
ISN 0198  GO TO 303
C TWO DIGIT PARAMETER

ISN 0199  351 RNUNSF(NNN) = 10*RNUMSF(NNN) + 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 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) = JZERD
ISN 0209  JIREG(NI) = JZERD
ISN 0210  7069 JIREG(NI) = JZERD
ISN 0211  DO 1719 L = 1, 5
ISN 0212  1719 JHFDL(L) = JZERD
C CHECK FOR END CARD

ISN 0213  IF (JCDTYP(K) - 12) 701, 715, 715
ISN 0214  701 KK = JCDTYP(K)
0270  DO 176  L = 27,36
0271  IF (JN(I) .NE. JNB(L)) GO TO 174
0272  176 CONTINUE

0274  J = 5

0275  IF (J = K) GO TO 177

0276  DO 177 LL = 1, K

0277  JFLD(J) = JN(JA+KK)

0278  J = J - 1

0279  177 KK = KK-1

0280  GO TO 708

0281  I = I + 1

0282  IF (JN(I) .NE. JCOMMA) GO TO 993

0284  GO TO 151

0285  IF (JN(I) .NE. JBLANK) GO TO 708

0287  GO TO 993

0288  I = I + 1

0289  GO TO 155

C  EXIT INSTRUCTION

0290  705 IF (KBRSTT = 1 ) 707,720,720

ISN 0291  707 KN = KN + 1

0292  GO TO 708

C  BRANCH STATE IS ACTIVE

0293  720 KN = JACDNDK(KX)

0294  N = 2

0295  JNXTLC = KSYMAD(KX)

0296  GO TO 770

0297  708 CONTINUE

0298  IF(JSIDBB(E(K)-1) 760, 760, 765

0299  760 JNXTLC = LACSLC(K) + 1

0300  GO TO 770

0301  765 JNXTLC = LACSLC(K) + 2

0302  770 LIN = JCOLP(K)

0303  LNI = JASTC(K) -1

0304  DO 785 J = 1,7

0305  785 JTYP4(J) = JBLANK

0306  JJ = 0

0307  DO 786 J = LIN, LNI

0308  JJ = JJ + 1

0309  786 JTYP4(JJ) = JIN(K,J)

C  PRINT-PUNCH CARD OF TRACE

0310  IF(JDECK,.NE.1) GO TO 2900

C  PRINT-OUTPUT

0312  797 WRITE (6,950) LACSLC(K), (JTEMP(J), J = 1,7)

C  PRINT-OUTPUT

6  *(JREG(I), I = 1,2), (JREG(I), I = 1,2), (JREG(I), I = 1,2),

* (JHFLD(I), I = 1,5), KBRSTT,

* KSKPS, JEXITF(K), (JHFLD(I), I = 1,5), JNXTLC,

* JABSOP(K), JSIDBB(K)

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(JJREG(1).EQ.JKNUM(LK)) JIR=JIR+10*(LK-1)
ISN 0320 IF(JJREG(2).EQ.JKNUM(LK)) JIR=JIR+LK-1
ISN 0321 IF(JJREG(3).EQ.JKNUM(LK)) JJR=JJR+10*(LK-1)
ISN 0322 IF(JJREG(4).EQ.JKNUM(LK)) JJR=JJR+LK-1
ISN 0323 IF(JJREG(5).EQ.JKNUM(LK)) JKR=JKR+10*(LK-1)
ISN 0324 IF(JJREG(6).EQ.JKNUM(LK)) JKR=JKR+LK-1
ISN 0325 JHTEMP=0
ISN 0326 DO 2961 LK=1,10
ISN 0327 IF(JHFLD(5).EQ.JKNUM(LK)) JHTEMP=JHTEMP+LK-1
ISN 0328 IF(JHFLD(4).EQ.JKNUM(LK)) JHTEMP=JHTEMP+10*(LK-1)
ISN 0329 IF(JHFLD(3).EQ.JKNUM(LK)) JHTEMP=JHTEMP+100*(LK-1)
ISN 0330 IF(JHFLD(2).EQ.JKNUM(LK)) JHTEMP=JHTEMP+1000*(LK-1)
ISN 0331 IF(JHFLD(1).EQ.JKNUM(LK)) JHTEMP=JHTEMP+10000*(LK-1)
ISN 0332 JTRACE(INTR,5)=JIR
ISN 0333 JTRACE(INTR,6)=JJR
ISN 0334 JTRACE(INTR,7)=JKR
ISN 0335 2961 CONTINUE
ISN 0336 JHTEMP
ISN 0337 JTRACE(INTR,8)=JHTEMP
ISN 0338 JTRACE(INTR,9)=KBRSTT
ISN 0339 JTRACE(INTR,10)=KSKPST
ISN 0340 JTRACE(INTR,11)=JXITF(K)
ISN 0341 JTRACE(INTR,12)=JHTEMP
ISN 0342 JTRACE(INTR,13)=JXITL
ISN 0343 JTRACE(INTR,14)=JABSP(K)
ISN 0344 JTRACE(INTR,15)=JSIDBR(K)
ISN 0345 DO 2977 LK=1,20
ISN 0346 JTRACE(INTR,LK+20)=JTEMP(LK)
ISN 0347 2977 CONTINUE
C C INCR. TRACE INPUT POINTER

ISN 0348 INTR=INTR+1
ISN 0349 IF(INTR.LE.1000) GO TO 2950
ISN 0350 WRITE(6,2910)
ISN 0351 RETURN
ISN 0352 2950 CONTINUE
ISN 0353 2977 CONTINUE
ISN 0354 GO TO (781,780), M
ISN 0355 M = 1
ISN 0356 KBRSTT = 0
ISN 0357 781 GO TO (796, 793, 795), MM
ISN 0358 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) = JABSP(K)
ISN 0378     JTRACE(INTR,15) = JSDBB(K)
ISN 0379     DO 2978 LK = 1,6
ISN 0380     JTRACE(INTR,1K+20) = JSSTOP(LK)
ISN 0381     2978 CONTINUE
ISN 0382     WRITE(6,3002)
ISN 0383     RETURN

C
C ANALYZE BRANCH OR SKIP ENTRY IN TABLE
C
ISN 0384     800 GO TO (802,803), JTYPE
C
ISN 0385     802 IF (KBRSTT = 1) 803, 725, 725
ISN 0386     803 IF (ROWCRT(N) - ROWTGT(N)) 801, 801, 725
C PROCESS NEXT ROW
ISN 0387     801 KL = ROWCRT(N)
ISN 0388     KL = KL + N - 1
ISN 0389     L = RSFFYPP(KL)
ISN 0390     GO TO (819,830), JTYPE
ISN 0391     819 GO TO (820,830), L
C SUCCESS
ISN 0392     820 KA = KLNPTR(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 (1860,8801), L
ISN 0397     860 GO TO (870,8901), 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     KKCN = 0
ISN 0404     DO 260 MX = 1,III
ISN 0405     IF (KKCN = KSUM) 255,225,255
ISN 0406     255 KKCN = 0
ISN 0407     J = 0
ISN 0408     DO 260 L = KCOL,KCOLL
ISN 0409     J = J + 1
ISN 0410     IF (JSMTAB1(MX,J)) .EQ. JNIL)
         *KKCN = KKCN + 1
ISN 0412 260 CONTINUE
C LABEL NOT IN SYMBOL TABLE
C
C STORE LABEL
ISN 0413  L = 1
ISN 0414  DO 265 II = KCOL,KCOL
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 266 IIJ = 1,8
ISN 0419  266 JSMTAB(III,IIJ) = JBLANK
ISN 0420  GO TO IA,(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,II) *.NE. JBLANK) GO TO 255
ISN 0423  230 IB = 2
ISN 0424  GO TO IA,(205,405)
ISN 0425  1 FORMAT (BOA1)
ISN 0426  2 FORMAT (' ', I10, 80A1)
ISN 0427  3 FORMAT (' *TOO MANY INPUT CARDS*')
ISN 0428  4 FORMAT (' ', 'OPERAND FIELD ERROR*')
ISN 0429  6 FORMAT (' ', 'ERROR ON FOLLOWING CARD*')
ISN 0430  7 FORMAT (' ', 'ERROR ON NEXT CARD NOT IMPLEMENTED*')
ISN 0431  8 FORMAT (' ', 'OPER CODE ON NEXT CARD NOT IMPLEMENTED*')
ISN 0432  950 FORMAT (' ', 'TERMS', UNROLL')
ISN 0433  2910 FORMAT (' ', 'TRACE EXCEEDS 1000 INSTRUCTIONS -- TERM, UNROLL')
ISN 0434  3000 FORMAT (' ', 'UNROLLER INPUT PROGRAM -- TERM, UNROLL')
ISN 0435  3001 FORMAT (IHO)
ISN 0436  3002 FORMAT (IHI)
ISN 0437  END
SUBROUTINE BLANKC (N, INN)

INTEGER*2 N, INN

DATA JBLANK = 0
COMMON/AREA/JN(80), IH, IJ

DO 10 I = 1, 80
  IF (JN(I) .EQ. JBLANK) GO TO 10
  INN = I
10 CONTINUE

RETURN

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

COMPILED OPTIONS - NAME = MAIN, OPT=00, LINECNT=56, SOURCE, EBODIC, 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),
          *JTYPE (256), JEXITF (300)
ISN 0007  COMMON/AREA*/JN(80), I, IJ
ISN 0008  DIMENSION JANS(2)
ISN 0009  IJ = 2
ISN 0010  DO 10 L = 1, 26
ISN 0011  IF (LJN(I)) .EQ. JNB(L)) GO TO 40
ISN 0012  10 CONTINUE
ISN 0013  10 CONTINUE
ISN 0014  10 CONTINUE
ISN 0015  IF (LJN(I)) .EQ. JNB(L)) GO TO 25
ISN 0016  20 CONTINUE
ISN 0017  1 J = 1
ISN 0018  RETURN
ISN 0019  RETURN
ISN 0020  25 I = I + 1
ISN 0021  DO 30 L = 27, 36
ISN 0022  IF (LJN(I)) .EQ. JNB(L)) GO TO 35
ISN 0023  30 CONTINUE
ISN 0024  30 CONTINUE
ISN 0025  JANS(I) = JZERO
ISN 0026  JANS(I) = JN(I-1)
ISN 0027  RETURN
ISN 0028  35 JANS(I) = JN(I-1)
ISN 0029  JANS(I) = JN(I)
ISN 0030  I = I + 1
ISN 0031  RETURN
ISN 0032  40 I = I + 1
ISN 0033  GO TO 15
ISN 0034  END
LEVEL 5 DEC 66

DATE YS/360 FORTRAN M

DATE 67.236/21.68.08

COMPIlER OPTIONS - NAME= MAIN,OPT=00, LINESCT=50, SOURCE, FACCIF, NOLIST, DEC, LOAD, MAP, NOEDIT, NOID

ISN 0002 BLOCK DATA

C

C FORM THE DECODE TABLE HERE WITH DATA STATEMENTS

C

COMMON/TAGS/

X  
T01(256), T02(256), T03(256), T04(256), T05(256), T06(256),  
T07(256), T08(256), T09(256), T10(256), T11(256), T12(256), T13(256),  
T14(256), T15(256), T16(256), T17(256), T18(256), T19(256), T20(256),  
T21(256), T22(256), T23(256), T24(256), T25(256), T26(256), T27(256),  
T28(256), T29(256), T30(256), T31(256), T32(256), T33(256), T34(256),  
T35(256), T36(256), T37(256), T38(256), T39(256), T40(256), T41(256),  
T42(256), T43(256), T44(256), T45(256), T46(256), T47(256), T48(256),  
T49(256), T50(256), T51(256), T52(256), T53(256), T54(256), T55(256),  
T56(256), T57(256), T58(256), T59(256), T60(256), T61(256), T62(256),  
T63(256), T64(256), T65(256), T66(256), T67(256), T68(256), T69(256),  
T70(256),  

DATA T01/1,1,0,1,1,3*0,1,1,0,1,5*0,8*0,11,0,1,0,3*1,1,4*0,4*1,  
0,1,0,1,3*0,1,1,3*0,1,1,3*0,8*1,0,1,3*0,1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,3*0,1,3*0,8*1,0,1,
UNROLLED DATA

ISN 0075
DIMENSION JN8(16), J1TYPE (256)

ISN 0076
DIMENSION JOPCDE (6,256), JS1DB (256)

ISN 0077
DIMENSION J1(96), J2(96), J3(96), J4(96), J5(96), J6(96), J7(96), J8(96),

ISN 0078
EQUIVALENCE (J1,JOPCDE(1,1)),
X (J2,JOPCDE(1,17)),
X (J3,JOPCDE(1,33)),
X (J4,JOPCDE(1,63)),
X (J5,JOPCDE(1,81)),
X (J6,JOPCDE(1,97)),
X (J7,JOPCDE(1,113)),
X (J8,JOPCDE(1,129)),
X (J9,JOPCDE(1,145)),
X (J10,JOPCDE(1,161)),
X (J11,JOPCDE(1,177)),
X (J12,JOPCDE(1,193)),
X (J13,JOPCDE(1,209)),
X (J14,JOPCDE(1,225)),
X (J15,JOPCDE(1,241))

ISN 0079
DATA J1/192HL X H L X L X A S T H S T X

ISN 0080
DATA J2/192HS T D H S T D L A T L H S T A

ISN 0081
DATA J3/192HS T M Z S T M / A

ISN 0082
DATA J4/192HS L C M R C M X P M K P

ISN 0083
DATA J5/192HS S R S U M N M N

257

L. Conway
Archives
| ISN 0097   | DATA JTYPE/201*4,9*1,4*3,8*2,33*4,12/ |
| ISN 0098   | COMMON /AKEAZ/ JNB,JOPCDE,JS108,JITYPE,JEX1FF |
| ISN 0099   | END |

L. Conway Archives
<table>
<thead>
<tr>
<th>ISN 0002</th>
<th>SUBROUTINE INIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISN 0003</td>
<td>IMPLICIT INTEGER*2(A-Z)</td>
</tr>
<tr>
<td>ISN 0004</td>
<td>DIMENSION COM(300)</td>
</tr>
<tr>
<td>ISN 0005</td>
<td>DIMENSION SAV(200000)</td>
</tr>
<tr>
<td></td>
<td>COMMON TIME, IPAR1, IPAR2, IPAR3,</td>
</tr>
<tr>
<td></td>
<td>A AINPT, NABUF, ABUS(50), XINPT, NBUF,</td>
</tr>
<tr>
<td></td>
<td>B XBUS(50), IFRAD, IPRAD, IBUS,</td>
</tr>
<tr>
<td></td>
<td>C BRAP, E(8), BE(8), ETT(8), NBUF,</td>
</tr>
<tr>
<td></td>
<td>D AHOLDT, XHOLDT, AFRC, XRCT, BOSC,</td>
</tr>
<tr>
<td></td>
<td>F BNOP, XEP, AEP, FI(100), PRINT,</td>
</tr>
<tr>
<td></td>
<td>G ESTADD, NDODT, NDPSC, NDBUS, NADSP,</td>
</tr>
<tr>
<td></td>
<td>C NXOSP</td>
</tr>
<tr>
<td></td>
<td>COMMON/RLS/ FIRST, NAREGS, NXREGS, NABUS,</td>
</tr>
<tr>
<td></td>
<td>A NBUS, STAS, ACON, XCON, AEMP,</td>
</tr>
<tr>
<td></td>
<td>B XEMP, NEX, AFULL(12), XFULL(12), AGO(12),</td>
</tr>
<tr>
<td></td>
<td>C XGO(12), NAAP, NAGO, AGO, NTEST, NTEST,</td>
</tr>
<tr>
<td></td>
<td>D NAFAG, NX Feng, ABUSY, ABUSY(200), XBUSY,</td>
</tr>
<tr>
<td></td>
<td>F XBUSY(200), ABUFF(12,100), XBUFF(12,100), ASOR(12,200),</td>
</tr>
<tr>
<td></td>
<td>G XSR(12,200), ADEST(12,200), XDEST(12,200),</td>
</tr>
<tr>
<td></td>
<td>H AFIGC(12,15), AFAGC(14,15,20), ARET, XFIGC(14,15,20), XRET,</td>
</tr>
<tr>
<td></td>
<td>I ABBUSC(14,15,20), XBBUSC(14,15,20), XBBUSC(14,15,20), XFIGC(14,15,20),</td>
</tr>
<tr>
<td></td>
<td>J AFOLY(12), XFDYL(15), AFIGC(14,15), XFIGC(14,15), XFIGC(14,15),</td>
</tr>
<tr>
<td></td>
<td>K AFIGC(200), XFIGC(200), ABUFF(200), XBUFF(200),</td>
</tr>
<tr>
<td></td>
<td>L QA(16,16), SRDL(32,2), QSBUS, MNBUS, MFBUS, MPRE,</td>
</tr>
<tr>
<td></td>
<td>N QUAD, MEMDLY, MEMORY(16), NBOX, NBOX, NBOX,</td>
</tr>
<tr>
<td></td>
<td>O MXTIME, OUTVL, ITU(4,16), RTN, LONGBR,</td>
</tr>
<tr>
<td></td>
<td>P SR(8), ST(8), SXSP, SKAP, NSBUF,</td>
</tr>
<tr>
<td></td>
<td>Q APA(200), XPA(200), OUT(2), JOS(6), SSTOP,</td>
</tr>
<tr>
<td></td>
<td>R MCP(16), ABUS(15), ARBUS(10), XBUS(15), XBUS(10), XBUS(10),</td>
</tr>
<tr>
<td>ISN 0009</td>
<td>COMMON/RLS/ LAST</td>
</tr>
</tbody>
</table>

| ISN 0009 | INTEGER OUT |
| ISN 0010 | REAL WMDLY, MXTIME |
| ISN 0011 | REAL TIME |
| ISN 0012 | COMMON/CALNDY | ISL, ITL, LINK(200), |
|          | A CTIME(200), NEVENT(200), KOL(200), KOL(200), KOL(200), |
| ISN 0013 | REAL CTIME |
| ISN 0014 | REAL X |
| ISN 0015 | INTEGER I |
| ISN 0016 | COMMON/TAGS(1256,70) |
| ISN 0017 | EQUIVALENCE(CM(1),TIME), (X,CTIME(1)), |
| ISN 0018 | EQUIVALENCE(SAV(1),FIRST) |

| ISN 0019 | DD 520 I=1,200 |
| ISN 0020 | DD 520 COM(1)=0 |
| ISN 0021 | DD 525 I=1,20000 |

---

L. Conway
Archives
ISN 0024  525 SAV(1)=0
ISN 0025  525 CONTINUE

C

C INITIALIZE THE CALENDAR
C
ISN 0026  DO 92 ITL=2,199
ISN 0025  92 LINK(ITL)=ITL+1
ISN 0026  ISL=2
ISN 0027  ITL=1
ISN 0024  X=1.0E30
ISN 0029  T1M=0.0

C

C INITIALIZE THE EVENT NUMBERS
C
ISN 0030  STATS=1
ISN 0031  MXD=2
ISN 0032  ACON=3
ISN 0033  XCON=4
ISN 0034  REMP=5
ISN 0035  XEMP=6
ISN 0036  ARET=7
ISN 0037  XRET=8
ISN 0038  EAV=9
ISN 0039  OCON=10
ISN 0040  OEMP=11
ISN 0041  MAUS=12
ISN 0042  MFREE=13
ISN 0043  LOAD=14
ISN 0044  RTN=15

C

C SET UP STARTING EVENTS
C
ISN 0045  CALL CAUSE(STATS,TIMEx0.0,0,0,0)
ISN 0046  CALL CAUSE(ACON,TIMEx0.1,0,0,0)
ISN 0047  CALL CAUSE(XCON,TIMEx0.1,0,0,0)
ISN 0048  CALL CAUSE(OCON,TIMEx0.1,0,0,0)
ISN 0049  CALL CAUSE(MXD,TIMEx0.6,0,0,0)

C

C INITIALIZE THE MACHINE PARAMETERS
C
ISN 0050  RXP=1
ISN 0051  RAP=1
ISN 0052  SKXP=1
ISN 0053  SKAP=1
ISN 0054  NAREGS=90
ISN 0055  NAREGS=90
ISN 0056  AINPT=1
ISN 0057  QINPT=1
ISN 0058  XINPT=1
ISN 0059  DOT=0.1=1.32

L. Conway
Archives
<table>
<thead>
<tr>
<th>ISN</th>
<th>0060</th>
<th>ABUPS(1)=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISN</td>
<td>0061</td>
<td>50 XBUPS(1)=0</td>
</tr>
<tr>
<td>ISN</td>
<td>0062</td>
<td>DO 51 I=33,89</td>
</tr>
<tr>
<td>ISN</td>
<td>0063</td>
<td>ABUPS(1)=0</td>
</tr>
<tr>
<td>ISN</td>
<td>0064</td>
<td>51 XBUPS(1)=1</td>
</tr>
<tr>
<td>ISN</td>
<td>0065</td>
<td>NSLOT=15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INITIALIZE AFAC TABLES</td>
</tr>
<tr>
<td>ISN</td>
<td>0066</td>
<td>MBAUS=6</td>
</tr>
<tr>
<td>ISN</td>
<td>0067</td>
<td>NAFA=10</td>
</tr>
<tr>
<td>ISN</td>
<td>0068</td>
<td>DO 10 J=1,10</td>
</tr>
<tr>
<td>ISN</td>
<td>0069</td>
<td>10 AFSLOT(1+J)=1</td>
</tr>
<tr>
<td>ISN</td>
<td>0070</td>
<td>DO 9 J=4,9</td>
</tr>
<tr>
<td>ISN</td>
<td>0071</td>
<td>9 AFSLOT(4,J)=1</td>
</tr>
<tr>
<td>ISN</td>
<td>0072</td>
<td>AFSL(16,J)=1</td>
</tr>
<tr>
<td>ISN</td>
<td>0073</td>
<td>DO 8 J=4,12</td>
</tr>
<tr>
<td>ISN</td>
<td>0074</td>
<td>8 AFSLOT(7,J)=1</td>
</tr>
<tr>
<td>ISN</td>
<td>0075</td>
<td>AFDLY(1)=3</td>
</tr>
<tr>
<td>ISN</td>
<td>0076</td>
<td>AFDLY(2)=4</td>
</tr>
<tr>
<td>ISN</td>
<td>0077</td>
<td>AFDLY(3)=3</td>
</tr>
<tr>
<td>ISN</td>
<td>0078</td>
<td>AFDLY(4)=9</td>
</tr>
<tr>
<td>ISN</td>
<td>0079</td>
<td>AFDLY(5)=2</td>
</tr>
<tr>
<td>ISN</td>
<td>0080</td>
<td>AFDLY(6)=5</td>
</tr>
<tr>
<td>ISN</td>
<td>0081</td>
<td>AFDLY(7)=15</td>
</tr>
<tr>
<td>ISN</td>
<td>0082</td>
<td>AFDLY(8)=1</td>
</tr>
<tr>
<td>ISN</td>
<td>0083</td>
<td>AFDLY(9)=1</td>
</tr>
<tr>
<td>ISN</td>
<td>0084</td>
<td>AFDLY(10)=1</td>
</tr>
<tr>
<td>ISN</td>
<td>0085</td>
<td>AFIBUS(1)=2</td>
</tr>
<tr>
<td>ISN</td>
<td>0086</td>
<td>AFIBUS(2)=1</td>
</tr>
<tr>
<td>ISN</td>
<td>0087</td>
<td>AFIBUS(3)=3</td>
</tr>
<tr>
<td>ISN</td>
<td>0088</td>
<td>AFIBUS(4)=1</td>
</tr>
<tr>
<td>ISN</td>
<td>0089</td>
<td>AFIBUS(5)=1</td>
</tr>
<tr>
<td>ISN</td>
<td>0090</td>
<td>AFIBUS(6)=2</td>
</tr>
<tr>
<td>ISN</td>
<td>0091</td>
<td>AFIBUS(7)=2</td>
</tr>
<tr>
<td>ISN</td>
<td>0092</td>
<td>AFIBUS(8)=1</td>
</tr>
<tr>
<td>ISN</td>
<td>0093</td>
<td>AFIBUS(9)=2</td>
</tr>
<tr>
<td>ISN</td>
<td>0094</td>
<td>AFIBUS(10)=3</td>
</tr>
<tr>
<td>ISN</td>
<td>0095</td>
<td>AFIBUS(1)=2</td>
</tr>
<tr>
<td>ISN</td>
<td>0096</td>
<td>AFIBUS(2)=1</td>
</tr>
<tr>
<td>ISN</td>
<td>0097</td>
<td>AFIBUS(3)=4</td>
</tr>
<tr>
<td>ISN</td>
<td>0098</td>
<td>AFIBUS(4)=9</td>
</tr>
<tr>
<td>ISN</td>
<td>0099</td>
<td>AFIBUS(5)=2</td>
</tr>
<tr>
<td>ISN</td>
<td>0100</td>
<td>AFIBUS(6)=4</td>
</tr>
<tr>
<td>ISN</td>
<td>0101</td>
<td>AFIBUS(7)=4</td>
</tr>
<tr>
<td>ISN</td>
<td>0102</td>
<td>AFIBUS(8)=6</td>
</tr>
<tr>
<td>ISN</td>
<td>0103</td>
<td>AFIBUS(9)=1</td>
</tr>
<tr>
<td>ISN</td>
<td>0104</td>
<td>AFIBUS(10)=3</td>
</tr>
<tr>
<td>ISN</td>
<td>0105</td>
<td>ABOX(1)=1</td>
</tr>
<tr>
<td>ISN</td>
<td>0106</td>
<td>ABOX(2)=2</td>
</tr>
<tr>
<td>ISN</td>
<td>0107</td>
<td>ABOX(3)=3</td>
</tr>
<tr>
<td>ISN</td>
<td>0108</td>
<td>ABOX(4)=4</td>
</tr>
</tbody>
</table>
INITIALIZE XFAC TABLES

- NFXUS = 10
- NFXAC = 9
- XFSLOT(1,2) = 1
- XFSLOT(5,3) = 1
- XFDLY(1) = 1
- XFDLY(2) = 1
- XFDLY(3) = 1
- XFDLY(4) = 1
- XFDLY(5) = 6
- XFDLY(6) = 9
- XFDLY(7) = 1
- XFDLY(8) = 1
- XFDLY(9) = 1
- XFDLY(10) = 10
- XFDLY(11) = 1
- XFDLY(12) = 2
- XFDLY(13) = 3
- XFDLY(14) = 4
- XFDLY(15) = 5
- XFDLY(16) = 6
- XFDLY(17) = 7
- XFDLY(18) = 8

The fix to allow MAX of

L. Conway
Archives
ISN 0030  DO 81 K=1,NAREGS
ISN 0031  ASK(R1,K)=0
ISN 0032  81 ADEST(R1,K)=0
ISN 0033  DO 82 K=1,NAFAC
ISN 0034  AFAK(R1,K)=0
ISN 0035  82 AGRUS(R1,K)=0
ISN 0036  ABUFF(R1,2)=0
ISN 0037  IN=H3 K=9,15
ISN 0038  83 ABUFF(R1,K)=0
ISN 0039  80 CONTINUE
ISN 0040  84 CONTINUE

C THIS EVENT CAN PASS ABUFF FOR INST WHICH CAN GO
C SCAN FOR NAGO OUT OF NAESE
ISN 0041  DO 10 KEG=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 0046  IF(INS.EQ.1) GO TO 21
ISN 0047  DO 11 REG=1,NAREGS
ISN 0048  11 SORBSY(REG)=SORBSY(REG)+ASOR(INS-1,REG)
ISN 0049  DESBSY(REG)=DEBSY(REG)+ADEST(INS-1,REG)
ISN 0050  INS=INS+1
ISN 0051  105 CONTINUE

C PREV EXIT INTILKS ALL CODE BELOW
ISN 0052  IF(ABUFF(R1,14,INS)).EQ.1) GO TO 100
C PREV SKIP INTILKS ALL STARRED CODE BELOW
C AND ALL SKIPS BELOW
ISN 0053  IF(ABUFF(R1,13,INS)).EQ.0) GO TO 20
ISN 0054  IF(ABUFF(IN[13,INS,1]).OR.ABUFF(IN[9,INS,1]) GO TO 100
ISN 0055  20 CONTINUE
ISN 0056  21 CONTINUE
C IF EXIT INTILK AGAINST ER
ISN 0057  IF(ABUFF(IN[14,INS,1]).NE.1) GO TO 28
ISN 0058  IFERER(IN[14,INS,1]) GO TO 100
C EXIT PART OF CP GOES, MARK GO EXIT.
ISN 0059  ABUFF(R1,5,INS)=1
ISN 0060  28 CONTINUE
ISN 0061  IN=22 REG=1,NAREGS
ISN 0062  IF(ASK(R1,INS,REG).EQ.1).AND.(DEBSY(REG,INS,NE,0)) GO TO 100
ISN 0063  IF(ADEST(INS,REG,INS).EQ.1).AND.(SORBSY(REG,INS).EQ.0) GO TO 100
ISN 0064  IF(ADEST(INS REG,INS,INS).EQ.0).AND.(DEBSY(REG,INS,NE,0)) GO TO 100
ISN 0065  22 CONTINUE
C FIND FAC USED
ISN 0066  DO 25 FAC=1,NAFAC
ISN 0067  IF(AFAK(IN[INS,FAC]).NE.0) GO TO 26
ISN 0068  25 CONTINUE
C NO FAC USED. ISSUE OP
ISN 0082  FAC=0
ISN 0083  26 CONTINUE
ISN 0084  SPEC=0
C --- --- --- --- TEST FOR SPECIAL OPS HERE --- --- --- --- --- ---
C
ISN 0085  IF(ADEST(IN$89),NE.1) GO TO 27
ISN 0087  SPEC=1
C PREV NOGU STORE INLKS
ISN 0088  IF(IN$,EQ.1) GO TO 10
ISN 0090  DU 16 l=1;INSM$
ISN 0091  IF(ADEST(I,89),EQ.1).AND.((AGO(I),EQ.0)) GO TO 100
ISN 0093  16 CONTINUE
ISN 0094  19 CONTINUE
ISN 0095  IF(TABRSY(I),EQ.1) GO TO 17
ISN 0097  A1RSY(I)=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, INLKL AGAINST PREV NOGU STARRED OPS, SMT RESOLVED.
ISN 0104  IF(ABUFF(IN$,13),NE.1) GO TO 132
ISN 0106  IF((sk(SKAP),NE.1) GO TO 100
ISN 0108  IF(IN$,EQ.1) GO TO 131
ISN 0110  DU 130 l=1;INSM$
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 --- --- --- --- IF NORMAL OR SPEC OP AND NGO = NGO, DO NOT ISSUE --- --- --- --- --- ---
C IF REPLACE OR NOP, CAN ISSUE ANYWAY.
ISN 0116  IF((FACT,NE.0).OR.(SPEC,NE.0)).AND.(INGO,EQ.NAGO)) GO TO 100
ISN 0117  IF((FACT,EQ.0)) GO TO 95
ISN 0118  IF((FACT,NE.0)) GO TO 95
C IF MULT IDENT FAC, GO TO SPEC HANDLING.
ISN 0120  IF((FACT,IN$),FACT),ST.1) GO TO 49
C CHECK INBUS=FACT SLOT=OUTBUS INLKS
ISN 0122  INBUS=FACT(SLOT) (FAC)
ISN 0123  IF((FACT(INBUS),EQ.1)) GO TO 100
ISN 0125  FACT=FACT(IN$)
ISN 0126  IF((FACT(OUTBUS),EQ.1)) GO TO 100
ISN 0128  DU 100 T=1;WSDL
ISN 0129  IF((FACT(SLOT),(FAC),T),EQ.1)) GO TO 100
ISN 0131  30 CONTINUE
ISN 0132  OBUS=FACT(HUS(FAC)
ISN 0133  DELAY=FACT(FAC)
ISN 0134  IF((ABUS(IN$,OBUS),NE.0)).AND.(ABUSC1,OBUS,DELAY),NE.0))
X GO TO 100
C SUCCESS. MARK GO AND SET SHIFT CELLS

C

ISN 0136 IF(ABUSINS,OBUS+1),NE.0).AND.(ABUSSC(1,OBUS+1,DELAY),NE.0).AND. X (OBUS+1),LE.NABUS)) GO TO 100

C

ISN 0138 31 CONTINUE

C

ISN 0139 AIBBSY(INBUS)=1

ISN 0140 ARBXSY(BOXI)=1

ISN 0141 ABUFF1=ABUFFINS(1)

ISN 0142 DO 31 I=1,NSLOT

ISN 0143 IF(AFSLITO(FAC,T),EQ.0) GO TO 32

ISN 0145 AFACSC(1,FAC,I)=1

ISN 0146 AFACSC(2,FAC,I)=ABUFF1

ISN 0147 32 CONTINUE

C

ISN 0148 ABUSSC(1,OBUS,DELAY)=ABUSINS,OBUS)

ISN 0149 ABUSSC(2,OBUS,DELAY)=ABUFFINS(1)

ISN 0150 ABUSSC(3,OBUS,DELAY)=ABUFFINS(2)

ISN 0151 IF(AIOBUSINS,OBUS+1),EQ.0) GO TO 95

C

ISN 0153 IF(OBUS+1),GT,NABUS) GO TO 95

ISN 0155 ABUSSSC(1,OBUS+1,DELAY)=ABUSINS,OBUS+1)

C

ISN 0156 ABUSSSC(2,OBUS+1,DELAY)=ABUFFINS(1)

ISN 0157 ABUSSSC(3,OBUS+1,DELAY)=ABUFFINS(2)

ISN 0158 GO TO 95

C SPEC ROUTINE TO HANDLE MULT IDENT FAC INTLK

ISN 0159 49 CONTINUE

C

ISN 0160 INBUS=AIBBSY(FAC)

ISN 0161 IF(AIBBSY(INBUS),EQ.1) GO TO 60

ISN 0163 UOX=ARBXSY(FAC)

ISN 0164 IF(ARBXSY(BOXI),EQ.1) GO TO 60

ISN 0166 DO 50 I=1,NSLOT

ISN 0167 IF(AFSLITO(FAC,T),EQ.1),AND,(AFACSC(1,FAC,T),EQ.1)) GO TO 60

C

ISN 0169 50 CONTINUE

C

ISN 0170 OBUS=AIOBUS(FAC)

ISN 0171 DELAY=AIOFLY(FAC)

ISN 0172 IF((AIOBUSINS,OBUS),NE.0).AND.(AIOUSSC(1,OBUS,DELAY),NE.0)) X GO TO 60

C SUCCESS

C

ISN 0174 DO 51 OBUS=1,NABUS

ISN 0175 51 IF(BUS,NE,OBUS) AIOBUSINS,BUS)=0

ISN 0177 GO TO 31

C

ISN 0179 60 CONTINUE

C

ISN 0180 IF((FAC,GT,NAFAC),OR,(AFACINS,FAC),LE.1)) GO TO 100

C

ISN 0182 GO TO 49

C

ISN 0183 49 CONTINUE

C IF OP USES NO FACILITIES, AND IS NOT SPECIAL OP THEN IT IS A REPLACE OP, AND GOES WITHOUT INCREMENTING NGO.

C

ISN 0186 IF((FAC,NE.0),OR,(SPEC,NE.0)) NGO=NGO+1

C

ISN 0187 100 CONTINUE

C ------- ------- ------- EXIT EXECUTION ------- -------

C CHECK FOR NGO EXITS TO SET AHOLDT

C

208

L. Conway
Archives
ISN 0188 AHOLO=0
ISN 0189 DO 200 I=1,NABUF
ISN 0190 IF((ABUFF(I,14).EQ.1).AND.(ABUFF(I,15).NE.1)) AHOLO=1
ISN 0192 C 200 CONTINUE
ISN 0193 C  CHECK FOR GO EXIT,ET
ISN 0194 C  AFRCT=0
ISN 0195 DG 201 I=1,NABUF
ISN 0196 IF((ABUFF(I,15).NE.1) GO TO 201
ISN 0197 IF((ABUFF(I,14).NE.1) GO TO 201
ISN 0198 C  ABUFF(I,14)=0
ISN 0200 ABUFF(I,15)=0
ISN 0201 IF((ABUFF(I,10).EQ.1) GO TO 202
ISN 0203 C 201 CONTINUE
ISN 0204 DG TO 300
C 201 CONTINUE
C  FOUNJ GO EXIT,ET. NOP AND MARK GO ALL CODE BELOW IT.
C  ALSO SET AFRCT.
ISN 0205 202 AFRCT=1
ISN 0206 IF((I.EQ.NABUF) GO TO 300
ISN 0207 C  I=I+1
ISN 0208 DO 209 I=1,NABUF
ISN 0209 C  AGF(J)=I
ISN 0210 DG TO 204 K=1,NAREGS
ISN 0211 C  ASK(J,K)=0
ISN 0212 DE 204 ADEST(J,K)=0
ISN 0213 C  204 ADEST(J,K)=0
ISN 0214 C  ADEST(J,K)=0
ISN 0215 C  205 ADEST(J,K)=0
ISN 0216 C  205 ADEST(J,K)=0
ISN 0217 DG TO 206 K=9,15
ISN 0218 C  206 ADEST(J,K)=0
ISN 0219 DG TO 207 K=1,NAFAC
ISN 0220 C  AFACE(J,K)=0
ISN 0221 207 CONTINUE
ISN 0222 203 CONTINUE
ISN 0223 300 CONTINUE
269
C  ------ IF SKIP NOT TAKEN, REMOVE FLAGS FROM ALL OPS THRU 1ST SKIP
ISN 0224 DO 225 I=1,NABUF
ISN 0225 IF((ABUFF(I,11).EQ.0).ABUFF(I,9)=0
ISN 0226 IF((ABUFF(I,13).EQ.1) GO TO 86
C  ------ 85 CONTINUE
ISN 0229 85 CONTINUE
ISN 0230 C  ------ 86 CONTINUE
ISN 0231 RETURN
ISN 0232 END
ISN 0077 9 ABUSY(RED)=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 0082 DO 30 I=INS, M
ISN 0083 AGU(I)=AGU(I+1)
ISN 0084 AFULL(I)=AFULL(I+1)

ISN 0086 DO 25 J=1, 25
ISN 0087 25 ABUFF(I, J)=ABUFF(I+1, J)

ISN 0088 UU 26 J=1, NAREGS

ISN 0089 ASU(I, J)=ASU(I+1, J)

ISN 0090 ADEST(I, J)=ADEST(I+1, J)

ISN 0091 UU 27 FAC=1, NAFAC

ISN 0092 27 FAC(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 AGU(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 ADEST(NABUF, J)=0

ISN 0103 126 ADEST(NABUF, J)=0

ISN 0104 DD 127 FAC=1, NAFAC

ISN 0105 127 FAC(NABUF, FAC)=0

ISN 0106 DD 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
SUBROUTINE XARET

COMMON TIME, IPAR1, IPAR2, IPAR3, XINFO
A AINPT, NABUF, ABUS(50), XINPT, NXBUF
B XBUP(50), IFADD, IFDST, IFRTN, BUXP
C BRAP, ER(18), BE(18), ET(18), NDBUF
D AQDELETE, XREAD, AREAD, XREAD, BADD
E BNOPT, XEP, AEP, PH1(100), PRINT
F FSTADD, NODOT, NODPS, NDBUS, NADSP
G NDBUS

COMMON/KLS/ FIRST, NAREGS, NXREGS, NABUS
A NXBUS, TXTED, TXTE, ACON, XCON, AEMP
B XEMP, MXO, AFULL(12), XFULL(12), AGO(12)
C XGUI(12), NAGD, NXGD, NATTEST, NXTEST
D NAFAC, NXFAC, ABUSY, ABUSY(20), XBUZY
E AXBUSY(20), ABUSY(20), XBUFY(100), ABUSY(20), ASQR(12,200)
F AXSQR(12,200), ADEST(12,200), XDEST(12,200), AFAC(12,15)
G XFAC(12,15), AIRFAC(4,15,20), ARET, XFAC(4,15,20), XARET
H ABUSC(4,10,20), ABBBBY(10), XBUSSC(10,10,20), XBBBBY(10), XBBBBY(15)
I ADBUSY(20), XDBUSY(10), AFSL(15,20), XFSL(15,20), XFBUSY(15)
J AFDLY(15), XFDLY(15), AFSL(15), XFSLOT(15,20), XFBUSY(15)
K ABUPSY(20), XBUFSY(20), ABUFUL(20), XBUFSY(20)
L Q16166, SQBA(32,2), NQBUF, NGDD, NGQ
M QINPT, QCON, GEMP, MBUSY, MFREE
N LOAD, MEMDY, MEMORY(16), NOBX, EAV
O MXTIME, OUTVL, IQ4(16), RTN, LONGBR
P SR1B, ST(1), SKXP, SKAP, NSBUF
Q APASS(200), XPASS(200), OUT(2), JOB(6), SSTP
R MEMCNT(16), ABOX(15), ABXY(10), XBOX(15), XBOXSY(10)

COMMON/KLS/ LAST

REAL MEMDY, MXTIME

USES OBUS SHIFT CELLS TO BUS TO DESTINATIONS
PERFORMS ANY EXEC ACTIVITY ASSIG WITH RETURN
FIRST RESET THE WAITING VECTOR

DU 10 BUS=1:NABUS
DEST=ABUSC(4,10,20)
IF DEST NOT XBU GO HANDLE NORMALLY
IF(XBUFSY(DEST).NE.1) GO TO 9
IF DEST IS XBU, SEE IF CORRESP XREG IS BUSY
IF SO, RETURN DEST TO IT. ELSE SET XBU BUSY.
IF(XPASS(DEST,NE.0) GO TO 8
XBUFUL(DEST)=1
GO TO 10
XBUFSY(DEST)=0
XPASS(DEST)=0
9 ABUSY(DEST)=0
10 CONTINUE
C
C
SHIFTS THE SHIFT CELLS

ISN 0022  DO 99 I=1,10
ISN 0023  ABBBSY(I)=0
ISN 0024  99
ISN 0025  SLOTM1=NSLOT-1
ISN 0026  DO 101 J=1,10
ISN 0027  DO 100 SLOT=1,SLOTM1
ISN 0028  ABUSSC(I,J,SLOT)=ABUSSC(I+1,J,SLOT+1)
ISN 0029  ABUSSC(I+1,J,SLOT)=ABUSSC(I,J,SLOT+1)
ISN 0030  ABUSSC(I,J,SLOT)=ABUSSC(3,J,SLOT+1)
ISN 0031  100 CONTINUE
ISN 0032  ABUSSC(I,J,NSLOT)=0
ISN 0033  ABUSSC(I+1,J,NSLOT)=0
ISN 0034  ABUSSC(I,J,NSLOT)=0
ISN 0035  101 CONTINUE
ISN 0036  DO 103 J=1,NAFAC
ISN 0037  DO 102 SLOT=1,SLOTM1
ISN 0038  AFACS(1,J,SLOT)=AFACS(1,J,SLOT+1)
ISN 0039  AFACS(2,J,SLOT)=AFACS(2,J,SLOT+1)
ISN 0040  102 CONTINUE
ISN 0041  AFACS(1,J,NSLOT)=0
ISN 0042  AFACS(2,J,NSLOT)=0
ISN 0043  103 CONTINUE
ISN 0044  RETURN
ISN 0045  END
PROGRAM DECUS

COMMON /RIS/ FIRST, NAREGS, NXREGS, NABUS,
A RXBUS, STATS, ACON, XCON, AEMP,
B XEMP, MXD, AFULL(12), XFULL(12), AGO(12),
C XGOL(12), NAG0, NX0, NAXTEST, NAXTEST,
D NAAC, NXAC, ABUSY, ABUSY(200), XBUSY,
E XBUSY(200), ABUFF(12,100), XBUFF(12,100), ASR0(12,200),
F XSOR(12,200), GDAFLST(12,200), XDEST(12,200), AFACT(12,15),
G XFACT(12,15), AFACTS(4,15,20), ARE, XFACTS(4,15,20), XRE,
H ABUS(14,10,20), ABUS (14,10,20), XBUFFS(4,10,20), XBUFFS(15,10), XBUFFS(15),
I ABUS(12,10), XBUFFS(12,10), ASR0(15,20), XBUFFS(15,20), AFACTS(15),
J ADFL(15), ADFLX(15), ADFBUS(15), ADFBUS(15), XSLOT,
K ABUS(200), ABUS(200), ABUS(200), ABUS(200),
L ABUS(200), ABUS(200), ABUS(200), ABUS(200),
M ABUS(200), ABUS(200), ABUS(200), ABUS(200),
N ABUS(200), ABUS(200), ABUS(200), ABUS(200),
O ABUS(200), ABUS(200), ABUS(200), ABUS(200),
Q ABUS(200), ABUS(200), ABUS(200), ABUS(200),
R ABUS(200), ABUS(200), ABUS(200), ABUS(200),
S ABUS(200), ABUS(200), ABUS(200), ABUS(200),
T ABUS(200), ABUS(200), ABUS(200), ABUS(200),
U ABUS(200), ABUS(200), ABUS(200), ABUS(200),
V ABUS(200), ABUS(200), ABUS(200), ABUS(200),
W ABUS(200), ABUS(200), ABUS(200), ABUS(200),
X ABUS(200), ABUS(200), ABUS(200), ABUS(200),
Y ABUS(200), ABUS(200), ABUS(200), ABUS(200),
Z ABUS(200), ABUS(200), ABUS(200), ABUS(200),

ENTRY BUSIOA

MOVE UP FROM ABUS TO ABUFF(AINPT)

DU 10 I=1,125

10 ABUFF(AINPT,1)=ABUS(I)

ABUFF(AINPT)=1

PERFORM COMPLETE UP DECODE HERE

FIRST DECODE SOURCE-DEST INTERLOCK TAGS

I=AIXPT

UP=ABUFF(1,2)

I=ABUFF(1,3)

I=ABUFF(1,4)

I=ABUFF(1,5)
TEST VALID OP TAG TO SEE IF OP VALID
IF(DIOP,30).EQ.0) GO TO 11

INVALID OP. ISSUE ERROR MESSAGE, INCR AINPT, AND RETURN THUS MAKING OP INTO NOP.

WRITE(6,998)
WRITE(6,999) OP,ABUS(1)
AINPT=AINPT+1
RETURN
II CONTINUE

SET A(I) SOURCE
IF(DIOP, 4).EQ.1) ASORI(I,11+1)=1

SET A(I) DEST
IF(DIOP, 5).EQ.1) ADEST(I,11+1)=1

SET A(I+1) SOURCE
IF(DIOP, 6).EQ.1) ASORI(I+1,MOD(II+1,32)+11)=1

SET A(I+1) DEST
IF(DIOP, 7).EQ.1) ADEST(I+1,MOD(II+1,32)+11)=1

SET A(J) SOURCE
IF(DIOP, 8).EQ.1) ASORI(I1,J+1)=1

SET A(J) DEST
IF(DIOP, 9).EQ.1) ADEST(I1,J+1)=1

SET A(J+1) SOURCE
IF(DIOP,10).EQ.1) ASORI(11,J1+1,MOD(II+1,32)+11)=1

SET A(J+1) DEST
IF(DIOP,11).EQ.1) ADEST(I1,J1+1,MOD(II+1,32)+11)=1

SET XB(I) DEST
IF(DIOP,13).EQ.1) ADEST(I,II+33)=1

SET XB(J) DEST
IF(DIOP,14).EQ.1) ADEST(I,J+33)=1

SET CO(I) DEST
IF(DIOP,15).EQ.1) ADEST(I,II+65)=1

SET STORAGE DEST
IF(DIOP,28).EQ.1) ADEST(I,89)=1

REMOVE ANY SOURCE-DEST TAGS ON A0,XBU(0)
ASDIR(I,1)=0
ASDIR(I,33)=0
ADEST(I,1)=0
ADEST(I,33)=0

C SET FACILITY USE TAGS, BUS DEST TAGS

DO 20 FAC=1,NAFAC
20 CONTINUE

C CHECK FOR DOUBLE DEST. IF SO, PLACE ON ADJ. BUS

DESTP1=DEST+1

IF(DESTP1.GT.NAREGS) GO TO 20

DO 27 DEST2=DESTP1,NAREGS

IF(ADEST(I,DEST2).NE.0) GO TO 28

27 CONTINUE

DEST2=DEST+1

IF(ADEST(I,DEST2).NE.0) GO TO 20

28 CONTINUE

C INCREMENT AINPT

AINPT=AINPT+1

RETURN

C ENTRY BUSIUX

MOVE OP FROM XBUS TO XBUFF(XINPT)

DO 110 I=1,125

110 XBUFF(XINPT,I)=XBUS(I)

XFULL(XINPT)=1

C PERFORM COMPLETE OP DECODE HERE

C FIRST DECODE SOURCE-DEST INTERLOCK TAGS

I=XINPT

OP=XBUFF(I,2)

II=XBUFF(I,3)

IJ=XBUFF(I,4)

IX=XBUFF(I,5)

C TEST VALID OP TAG TO SEE IF OP VALID

IF(D(OP,30).EQ.0) GO TO 111

INVALID OP. ISSUE ERROR MESSAGE, INCR XINPT.

C AND RETURN IN CASE MAKING UP INTO NOP.

WRITE(6,998)

WRITE(6,999) OP,XBUS(I)

WRITE(6,998)

XINPT=XINPT+1

RETURN
ISN 0097 111 CONTINUE

ISN 0098
SET X(I) SOURCE
IF(DIOP,161,EQ,1) XSOR(I,I+33)=1

ISN 0100
SET X(I) DEST
IF(DIOP,171,EQ,1)XDEST(I,I+33)=1

ISN 0102
SET X(I+1) SOURCE
IF(DIOP,181,EQ,1) XSOR(I,MOD(I+1,32)+33)=1

ISN 0104
SET X(I+1) DEST
IF(DIOP,191,EQ,1)XDEST(I,MOD(I+1,32)+33)=1

ISN 0106
SET X(IJ) SOURCE
IF(DIOP,201,EQ,1) XSOR(IJ,I+33)=1

ISN 0108
SET X(IJ) DEST
IF(DIOP,211,EQ,1)XDEST(IJ,I+33)=1

ISN 0110
SET X(IJ) SOURCE
IF(DIOP,221,EQ,1) XSOR(IJ,IK+33)=1

ISN 0112
SET AB(I) DEST
IF(DIOP,231,EQ,1)XDEST(I,I+1)=1

ISN 0114
SET C(I) SOURCE
IF(DIOP,241,EQ,1) XSOR(I,IK+65)=1

ISN 0116
SET C(I) DEST
IF(DIOP,251,EQ,1)XDEST(I,IK+65)=1

ISN 0118
SET C(IJ) SOURCE
IF(DIOP,261,EQ,1) XSOR(IJ,IK+65)=1

ISN 0120
SET C(K) SOURCE
IF(DIOP,341,EQ,1) XSOR(IK+65)=1

ISN 0122
SET STORAGE SOURCE
IF(DIOP,271,EQ,1) XSOR(I,89)=1

ISN 0124
SET STORAGE DEST
IF(DIOP,281,EQ,1)XDEST(I,89)=1

REMOVE ANY SOURCE-DEST TAGS ON X(I),ABU(I)

ISN 0126
XSOR(I,1)=0

ISN 0127
XDEST(I,1)=0

ISN 0128
XSOR(I,33)=0

ISN 0129
XDEST(I,33)=0

SPECIAL DECODE FOR BRANCH OPS

PLACE NO FACILITY IF K FIELD = O FOR BRANCH OP
SUBROUTINE JSTART(ENDRUN)

COMMON TIME, IPAR1, IPAR2, IPAR3,
A AINPT, NABUF, ABUS(501), XINPT, NXBUF,
B XBUS(501), IFAUD, IFUST, IFKTN, BRRKP,
C BRAP, ER(8), BR(8), ET(8), NBBUF,
D ABOL, XMEDT, AFRCT, XFRCT, BOSL,
E BNDP, XEP, AEP, PH1(100), PRINT,
F FSTADD, NODUT, NODSP, NDBUS, NADSP,
G NV

COMMON/RLS/ FIRST, MAREGS, NXREGS, NABUS,
A NXBUS, STATS, ACIN, XCON, AEMP,
B XEMP, NXDS, AFULL(121), XFULL(121), AGLT(121),
C XG(121), NAGO, NXGO, NATEST, XNTEST,
D NAFAC, NAFAC, ABUSY, ABUSY(200), XBUSY,
E XBUSY(200), ABUSY, XBUSY(12,100), XBUSY(12,100), ASOR(12,100),
F FSO(12,200), ADEST(12,200), XDEST(12,200), ADEPT(12,15),
G XFC(12,15), AFASC(4,15,20), ARET, XFASC(4,15,20), XRET,
H XBUS(14,10,20), ABUSY(11,10), XBUSC(4,14,10,20), XBBXY(10), XFBUS(15),
I ADBUSY(14,10), XBUSY(12,10), AFSLOT(15,20), XFSLOT(15,20), AFIBUS(15),
J AFBUSY(15), XFBUSY(15), AFBUS(15), AFBUS(15), XFBUS(15),
K ADBUSY(200), ABUS(200), ABUS(200), XABUS(200),
L XABUS(200), SBAT(127,2), NOUT, NOTEST, NQGO,
M QINT, QBON, QEMP, MBUS, MFREE,
N LOAD, MENDY, MEMORY(16), NBOX, EAV,
D XTIME, OUTLVL, IQ(4,16), RTN, LNDGBR,
F XSKAP, XSKAP, NDBUF, NSDUP, NDBUS,
Q APASS(200), XPASS(200), OUT(2), JOB(6), SSTOP,
R PMEM(16), ABUSX(15), ABUSX(10), XABUSY(10), XABUSY(10)

COMMON/RLS/ LAST

INTEGER OUT
REAL MEMDLY, XTIME
DIMENSION AREPT(10), XREPT(10)
INTEGER*2 ENDRUN
READ PARAM CARD FOR JOB
READ(5,100) END=107, JOB=I(6),
W NABUF, NTEST, NAGO, NXBUF, NXTEST, NXGO,
X NQGO, NOTEST, NQGO, NBOX, NBUF, NSBUF, NDBUS, NODSP, NADSP, NXDSP,
Y XTIME, MEMDLY, OUTLVL, FSTADD
PRINT=OUTLVL
ENDRUN=0
WRITE(6,202)
WRITE(6,200)
WRITE(6,300)
WRITE(6,200)
WRITE(6,200)
WRITE(6,400)
(1)J(107), I(6),

L. Conway
Archives
ISN 0021  WRITE(6,201)
ISN 0022  CALL TMTUICOUT(11)
ISN 0023  WRITE(6,3333) OUT(1),OUT(2)
ISN 0024  WRITE(6,201)
ISN 0025  WRITE(6,201)
ISN 0026  WRITE(6,5001)
ISN 0027  WRITE(6,201)
ISN 0028  WRITE(6,6001) NABUF,NXBUF,NQBUF
ISN 0029  WRITE(6,201)
ISN 0030  WRITE(6,7001) NTEST,NXTEST,NQTEST
ISN 0031  WRITE(6,201)
ISN 0032  WRITE(6,8001) NACG,NXGO,NQGO
ISN 0033  WRITE(6,201)
ISN 0034  WRITE(6,9001) REMDLY
ISN 0035  WRITE(6,201)
ISN 0036  WRITE(6,9011) NDBX
ISN 0037  WRITE(6,201)
ISN 0038  WRITE(6,9101) NDBUF,NXBUF,NQDOY
ISN 0039  WRITE(6,201)
ISN 0040  WRITE(6,9201) NOPSC
ISN 0041  WRITE(6,201)
ISN 0042  WRITE(6,9301) NDBUS
ISN 0043  WRITE(6,201)
ISN 0044  WRITE(6,9401) NADSP,NXDSP
ISN 0045  WRITE(6,201)
ISN 0046  WRITE(6,2001)
ISN 0047  WRITE(6,10001)
ISN 0048  WRITE(6,201)
C      CALC REP TIMES
ISN 0049  DO 20 I=1,NAFAC
      AREPT(I)=0
ISN 0050  AREPT(I)=AREPT(I)+AFSLOT(I,J)
ISN 0051  DO 20 J=1,NSLT
ISN 0052  20 AREPT(I)=AREPT(I)+AFSLOT(I,J)
ISN 0053  WRITE(6,10011)AREPT(I),I=1,NAFAC
ISN 0054  WRITE(6,10021)AFDLY(I),I=1,NAFAC
ISN 0055  WRITE(6,10031)AFIBUS(I),I=1,NAFAC
ISN 0056  WRITE(6,10051)ABOX(I),I=1,NAFAC
ISN 0057  WRITE(6,10041)AFOBUS(I),I=1,NAFAC
ISN 0058  WRITE(6,2001)
ISN 0059  WRITE(6,20001)
C      CALC REP TIMES
ISN 0060  DO 30 I=1,NXFAC
      XREPT(I)=0
ISN 0061  DO 30 J=1,NSLT
ISN 0062  30 XREPT(I)=XREPT(I)+XFSLOT(I,J)
ISN 0063  WRITE(6,10011)XREPT(I),I=1,NXFAC
ISN 0064  WRITE(6,10021)XFIRLY(I),I=1,NXFAC
ISN 0065  WRITE(6,10031)XFIBUS(I),I=1,NXFAC
ISN 0066  WRITE(6,10041)XFOBUS(I),I=1,NXFAC
ISN 0067  WRITE(6,10051)XFOB(1),I=1,NXFAC
ISN 0068  WRITE(6,10001)

281
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,F4.1,1X,I2,1X)
ISN 0076 101 FORMAT(IHI,6AI)
ISN 0077 200 FORMAT(IHO)
ISN 0078 201 FORMAT(IHI)
ISN 0079 202 FORMAT(IHI)
ISN 0080 300 FORMAT(25H SIMULATION PROGRAM -- ALGOL MPN --)
ISN 0081 400 FORMAT(30H INPUT PROGRAM FOR THIS RUN = S,6AI)
ISN 0082 500 FORMAT(30H MACHINE PARAMETERS FOR THIS RUN -- --)
ISN 0083 600 FORMAT(22H NUMBER OF A BUFFERS =,12X,5X,21H NUMBER OF X BUFFERS =,12X)
(ISN 0084 700 FORMAT(22H NUMBER OF A UPS TESTED =,12X,5X,21H NUMBER OF A UPS TESTED =,12X)
(ISN 0085 800 FORMAT(22H MAX A UPS ISS/CYCLE =,12X,5X,21H MAX A UPS ISS/CYCLE =,12X)
(ISN 0086 900 FORMAT(22H MINIMUM Q-MEM DELAY =,F4.1)
(ISN 0087 901 FORMAT(22H NUMBER OF BOSM =,12X)
(ISN 0088 910 FORMAT(22H NUMBER OF BRANCH REGS =,12X,5X,21H NUMBER OF BRANCH REGS =,12X)
(ISN 0089 920 FORMAT(22H NUMBER OF PSC REGS =,12X)
(ISN 0090 930 FORMAT(22H NUMBER OF DISP BUSES =,12X)
(ISN 0091 940 FORMAT(22H MAX A UPS DSP/CYCLE =,12X,5X,21H MAX A UPS DSP/CYCLE =,12X)
(ISN 0092 1000 FORMAT(66H A FACILITIES -- FAI FA2 FM FD IA IM ID IC
(ISN 0093 1001 FORMAT(16H REP TIME =,15(3X,I2))
(ISN 0094 1002 FORMAT(16H DELAY TIME =,15(3X,I2))
(ISN 0095 1003 FORMAT(16H INBUS =,15(3X,I2))
(ISN 0096 1004 FORMAT(16H OUTFUS =,15(3X,I2))
(ISN 0097 1005 FORMAT(16H BOX =,15(3X,I2))
(ISN 0098 2000 FORMAT(66H X FACILITIES -- EA1 EA2 L S M D XA C
(ISN 0099 3333 FORMAT(19H TIME/DATE OF RUN =,Z(1X,Z8)))
ISN 0100 END
SUBROUTINE XGCON

IMPLICIT INTEGER*(A-Z)

COMMON TIME, IPAR1, IPAR2, IPAR3,
A AINPT, NABUF, ABUS(50), XINPT, NXBUF,
B XBUS(50), IFADD, IFDST, IFRIN, BRXP,
C BRAP, ERI(8), BE(8), E1(8), NBUF,
D XMRL, XHKD(12), AFRCT, XFKC, BRSC,
E BNP, XEP, AEP, PH(100), PRINT,
F FSTADD, NNDT, NPSC, NDBUS, NADSP,
G NXDSP
COMMON/RLS/ FIRST, NAREGS, NXREGS, NABUS,
A NABUS, STATS, ACON, XCON, AEMP,
B XEMP, MXG, AFULL(12), XFULL(12), AGO(12),
C XGO(12), NAGO, NEXG, NATEST, NXTEST,
D NAFAC, NXFAC, ABUSYZ, ABUSY(200), XBUSY,
E XBUSY(200), ABUFF(12,100), XBUFF(12,100), ASOR(12,200),
F XSOR(12,200), ADEST(12,200), XDEST(12,200), AFACT(12,15),
G XFACT(12,15), AFACTC(4,15,20), ARET, XFACTC(4,15,20), XRET,
H XBUSC(4,10,20), AIBS(10), XBUSC(4,10,20), XIBS(10), XIBUS(15),
I XBUS(12,10), XBUS(12,10), AFSLOT(12,20), XFSLOT(12,20), AFSLOT(15),
J AFSLOT(15), XFSLOT(15), XFSLOT(12,20), AFSLOT(15),
K AFSLOT(15), XFSLOT(12,20), AFSLOT(15),
L Q(16,16), SDAB(12,2), NBQBF, NQTEST, NQCC,
M QINPT, QCON, QEMP, MBUSY, MFREE,
N LOAD, MEMLY, MEMORY(16), NBUS, EAV,
O OUT(16), OUTLY, IQ(4,16), RTN, LONGBR,
P SRT(8), STR(8), SKXP, SINS,
Q APASS(200), XPASS(200), OUT(2), JOB(6), SSTUP,
R REMCNT(16), ABUSY(15), ABUSY(10), XBUSY(15), XBUSY(10)

COMMON/RLS/ LAST

INTEGER OUT
REAL MEMLY, MXTIME
REAL ALGORITHM...LOADS OUT OF ORDER WITH BOM INTLK.

CALL QSUM(QCON, TIME, 1,0,0,0)
CALL QSUM(QEMP, TIME, 0,0,0,0)
NG=0
DO 11 I=1,NQBUF
Q=1
DO 10 INS=1,NQTEST
IF(INS.EQ.0) GO TO 100
IF(INS.EQ.1) GO TO 11
INSI=INS-1
DO 10 1=1,INSI
10
C IF PREV GO INS TO SAME BUM, NGGO
C IF PREV INS TO SAME WORD, NGGO
C IF PREV INS TO SAME WORD, NGGO
1210
IF LOAD, CAUSE DEST LOAD IN MEMDLY CYCLES

ISN 0025 IFQ(INS,2).EQ.1 CALL CAUSE(LOAD,TIME+MEMDLY-1.0,DEST,A,X)
REMOVE INS FROM QUEUE

ISN 0027 LDQO
ISN 0029IAQ(INS,2).EQ.0 GO TO 31
ISN 0031 DO 30 I=INS,\nISN 0032 DO 30 J=1,16
ISN 0033 Q(I,J)=Q(I+1,J)
ISN 0034 30 CONTINUE
ISN 0035 31 CONTINUE
ISN 0036 DO 32 J=1,16
ISN 0037 QINQBUF,J=0
ISN 0038 32 CONTINUE
ISN 0039 GO TO 5
ISN 0040 100 CONTINUE
C ISSUE GO INS FETCH REQ TEST
ISN 0041 DO 200 11=1,4
ISN 0042 IF(IQ(11,1).EQ.0) GO TO 200
ISN 0044 IF(IQ(11,16).EQ.0) GO TO 200
C ISSUE REQ
ISN 0046 BOM=IQ(11,6)
ISN 0047 DEST=IQ(11,15)
ISN 0048 L=IQ(11,1)
ISN 0049 MEMCNT(BOM)=MEMCNT(BOM)+1
ISN 0050 CALL CAUSE(HABOSY,TIME+MEMDLY-1.0,BOM,L,0)
ISN 0051 CALL CAUSE(IFFREE+TIME+MEMDLY-2.1,BOM,0,0)
C ZERO POSN IN IQ
ISN 0052 DO 150 1=1,16
ISN 0053 150 IQ(11,1)=0
C IF LAST OF 4 INS FETCH REQSTS, CAUSE RTN
ISN 0054 DO 160 1=1,4
ISN 0055 IF(IQ(11,1).NE.0) GO TO 200
ISN 0057 160 CONTINUE
ISN 0058 CALL CAUSE(RTN,TIME+MEMDLY-1.0,DEST,0,0)
ISN 0059 200 CONTINUE
C C FILL IQ IF EMPTY AND INS REQ PRESENT ON INTERFACE
C FIRST TEST IF IQ EMPTY
ISN 0060 DO 250 1=1,4
ISN 0061 IF(IQ(11,1).NE.0) GO TO 400
ISN 0063 250 CONTINUE
C TEST IF REQ PRESENT
ISN 0064 IF(IFDST.EQ.0) GO TO 400
C FILL IQ
ISN 0066 BOM=MOD(IFADD,NBOX)+1
'BLCU'  call every cycle
'BLENTL' to get bus control card

3 x 64 bits requests

L: Conway
Archives
Current MFM-MA interface:

C Variables:
- Q(16,16)
- SD&A(22,2)
- NQFWF
- NWTEST
- NWGO
- QINPT
- MEMORY
- MEMORY(16)
- N80A
- IQ(4,16)
- MEMENT(16) (see comment)

EVENTS-CAUTION

L. 8. 9

ACON  MFREE  QEMP  RTN  MKSY  LOAD  EAV

XCON test for availability of D space as one of the interlocks.

XEMP if go up, place on the interface

AGRAP store A - keep in order, inhibit bus

PLACE data on interface

I. Conway
Archives
289
INS Q:

INTERFACE VARIABLES: IFDSI, IFADD, IFRTN

IF (IFDSI NEQ 0) FILL IQ (or PLACE IFDSI)

CODE(IFDSI) is partition literal

ZERO & Interface IFDSI, IFADD

RTN:

IFRTN = DEFSI
ISN 0026        IF(LPASS(DEST).NE.0) GO TO 8
ISN 0027        ABUFUL(DEST)=1
ISN 0028        RETURN
ISN 0029        9 ABUSY(DEST)=0
ISN 0030        9 XBUSY(DEST)=0
ISN 0031        RETURN
    C
ISN 0032        ENTRY XAIN
ISN 0033        DEST=1PAR1
ISN 0034        IFRIN=DEST
ISN 0035        RETURN
    C
ISN 0036        ENTRY XEAV
ISN 0037        DO 10 I=1,NOBUF
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
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.) GO TO 60  
ISN 0024  DO 50 INDEX=1,35800  
ISN 0025  50 B(BINDEX)=BLNK  
ISN 0026  60 CONTINUE  
ISN 0027  CALL CAUSE(STATS,TIME,1,0,0,0,0)  

C ----- PLACE PER CYCLE OUTPUT HERE -----  
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  II TIME=TIME  
ISN 0033  JT=MOD(ITIME,100)+1  
ISN 0034  IF(JT.NE.1) GO TO 2050  
ISN 0036  IF(TIME.EQ.0.) GO TO 2050  

C ----- OUTPUT 100 CYCLE OUTPUT -----  

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-1  
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 PHI -----  

ISN 0049  WRITE(6,2630)  
ISN 0050  WRITE(6,2640)(SPHI(I),I=T=1,100)  
ISN 0051  WRITE(6,2641)(SPHI(I),I=T=1,100)  
ISN 0052  DO 90 I=1,8  
ISN 0053  J=I+2  
ISN 0054  90 WRITE(6,2622)(SPHI(I),I=T=1,100),I  
ISN 0055  WRITE(6,2630)  
ISN 0056  WRITE(6,2643)(SPHI(I),I=T=1,100)  
ISN 0057  WRITE(6,2641)(SPHI(I),I=T=1,100)  
ISN 0058  DO 91 I=1,8  
ISN 0059  J=I+2  
ISN 0060  91 WRITE(6,2622)(SPHI(I),I=T=1,100),I  
ISN 0061  WRITE(6,2630)  
ISN 0062  WRITE(6,2644)(SPHI(I),I=T=1,100)
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(6,2641)(SPH1(13,T),T=1,100)
ISN 0069       WRITE(6,2641)(SPH1(32,T),T=1,100)
ISN 0070       DO 93 1=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(1,T),T=1,100)
ISN 0077       888 CONTINUE
                     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,NBUF
ISN 0083       104 CONTINUE
ISN 0084       IF(I.EQ.1)  GO TO 104
ISN 0085       WRITE(6,2638)BSYM(2*J-1),BSYM(2*J),(SB(1,T),T=1,100)
ISN 0086       104 CONTINUE
ISN 0087       DU 103 I=25,34
ISN 0088       103 WRITE(6,2638)BSYM(2*J-1),BSYM(2*J),(SB(1,T),T=1,100)
                     OUTPUT SKIP CONTROLS
ISN 0089       WRITE(6,2630)
ISN 0090       WRITE(6,2647)SYM(1),SYM(2),(SS(1,T),T=1,100)
ISN 0091       DO 102 K=1,NBUF
ISN 0092       102 WRITE(6,2648)SYM(2*K-1),SYM(2*K),(SSK,T),T=1,100
ISN 0093       DO 101 K=9,10
ISN 0094       101 WRITE(6,2648)SYM(2*K-1),SYM(2*K),(SSK,T),T=1,100
ISN 0095       101 CONTINUE
ISN 0096       WRITE(6,2630)
ISN 0097       I=1
ISN 0098       WRITE(6,2623)I,(SABUFF(I,T),T=1,100),I
ISN 0099       DO 110 T=2,NBUF
ISN 0100       110 WRITE(6,2622)I,(SABUFF(I,T),T=1,100),I
ISN 0101       110 CONTINUE
ISN 0102       WRITE(6,2630)
ISN 0103       I=1
ISN 0104       WRITE(6,2624)I,(SXBBUFF(I,T),T=1,100),I
ISN 0105       DO 111 T=2,NBUF
ISN 0106       111 WRITE(6,2622)I,(SXBBUFF(I,T),T=1,100),I
ISN 0107       111 CONTINUE
ISN 0108       WRITE(6,2630)

295

L. Conway
Archives
ISN 0109  DO 108 I=2,NAPAC
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)AMN(I),I,(SXFAC(I,T),T=1,100),I
ISN 0114  DO 109 I=2,NAPAC
ISN 0115  109 WRITE(6,2635)AMN(I),I,(SXFAC(I,T),T=1,100),I
          C  OUTPUT Q BUSY
ISN 0116  WRITE(6,2630)
ISN 0117  IF(OUTYLE.EQ.3) GO TO 889
ISN 0118  I=1
ISN 0119  WRITE(6,2633)I,(SQ(I,T),T=1,100),I
ISN 0120  DO 106 I=2,NFILE
ISN 0121  106 WRITE(6,2622)I,(SQ(I,T),T=1,100),I
          C  OUTPUT IQ BUSY
ISN 0122  WRITE(6,2630)
ISN 0123  I=1
ISN 0124  WRITE(6,2634)I,(SIQ(I,T),T=1,100),I
ISN 0125  DO 105 I=2,NFILE
ISN 0126  105 WRITE(6,2622)I,(SIQ(I,T),T=1,100),I
          C  OUTPUT MEM BUSY
ISN 0127  WRITE(6,2630)
ISN 0128  I=1
ISN 0129  WRITE(6,2634)I,(SMEM(I,T),T=1,100),I
ISN 0130  DO 107 I=2,NBOX
ISN 0131  107 WRITE(6,2622)I,(SMEM(I,T),T=1,100),I
          C  OUTPUT AREGS BUSY
ISN 0132  WRITE(6,2630)
ISN 0133  J=0
ISN 0134  I=1
ISN 0135  WRITE(6,2625)J,(SAREG(I,T),T=1,100),J
ISN 0136  DU 112 I=2,32
ISN 0137  J=1
ISN 0138  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,(SABU(I,T),T=1,100),J
ISN 0144  DO 113 I=2,32
ISN 0145  J=1
ISN 0146  113 WRITE(6,2622)J,(SABU(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=1

296

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=1-1
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,(SCBIT(I,T),T=1,100),J
ISN 0165  DO 116 I=2,24
ISN 0166      J=1-1
ISN 0167  116 WRITE(6,2622)J,(SCBIT(I,T),T=1,100),J

ISN 0168  WRITE(6,2630)
ISN 0169  889 CONTINUE
ISN 0170  DO 2550 INDEX=1,35800
ISN 0171  2550 BBI(INDEX)=BLNK
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 ENTRY FINIS
ISN 0179     ABNORM=1
ISN 0180       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=JO
ISN 0187          CALL STORUF(CYCLE)
C TEST FOR STOP CONDITION
ISN 0188     IF(PHL(100).EQ.0) RETURN
ISN 0190      DO 200 J=1,NABUF
ISN 0191     IF(ABUFF(I,J).NE.0) RETURN
ISN 0193      200 CONTINUE
ISN 0194      DO 201 J=1,NXBUF
ISN 0195     IF(XBUFF(I,J).NE.0) RETURN

297

L. Conway
Archives
| ISN 0252 | 2633 FORMAT(18H MEMORY QUEUE (D) 12,1X,100A1,12) |
| ISN 0253 | 2634 FORMAT(18H MEMORY 12,1X,100A1,12) |
| ISN 0254 | 2635 FORMAT(15H 12,1X,100A1,12) |
| ISN 0255 | 2636 FORMAT(18H MEMORY QUEUE (1) 12,1X,100A1,12) |
| 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 1B,1X,100A1,2H *) |
| ISN 0259 | 2641 FORMAT(20H 00,1X,100A1,2H *) |
| ISN 0260 | 2642 FORMAT(20H 1X,100A1,2H *) |
| ISN 0261 | 2643 FORMAT(20H DSPX2 1B,1X,100A1,2H *) |
| ISN 0262 | 2644 FORMAT(20H DSPA1 1B,1X,100A1,2H *) |
| ISN 0263 | 2645 FORMAT(20H DSPA2 1B,1X,100A1,2H *) |
| ISN 0264 | 2646 FORMAT(20H PHI 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 |
SUBROUTINE STOBUF(CYCLE)

COMMON TIME, IPAR1, IPAR2, IPAR3,
      AINNT, NABUF, ABUS(50), XINPT, NXBUF,
      BXBUS(50), IFADD, IFOST, FRINT, BRXP,
      CRAP, ER(8), BE(8), ET(8), NBUF,
      DAHOLDT, XHOLDT, AFAC1, XFAC1, BSRC,
      EBNOP, XEP, AEP, PH(100), PRINT,
      FSTADD, NDODT, NCPSC, NDBUS, NADSP,
      GNSXDP

COMMON/RLE/ FIRST, NAREGS, NxREGS, NABUS,
      AXBUS, STATS, ACON, XCON, AEMP,
      BXMMA, MXMMA, AFULL(12), XFULL(12), AGC(12),
      CXG(12), NAGO, NGO, NATEST, NXTEST,
      DNXFAC, NXFA(12), ABUSY, ABUSY(200), XBUSY,
      EFBUS(1200), ABUFF(12,100), XBUFF(12,100), ASDRT(12,200),
      FSORT(12,1200), ADEST(12,1200), XDEST(12,1200), AFAC(12,15),
      GXFAC(12,15), AFAC(12,15), ARET, XFAC(15,20), XRET,
      HABUSC(4,10,20), AIBBY(10), XBUSC(4,4,10,20), XIBBY(10), XIBUSC(15),
      IAABBUSC(12,10), ABUS(12,10), ASORT(15,20), XSORT(15,20), AFIBUS(15),
      JAFDL(15), XFDL(15), AFDBUS(15), XFBUS(15), NSLOT,
      KABUSP, ABUSP(200), XBUSP(200), ABUSP(200), XFBUSP(200),
      LBS(16,16), SDBA(32,2), NDBUF, NTEST, NADSP,
      MGINP, QCON, GEMP, MBUSY, MFREE,
      NLOAD, MEMLY, MEMORY(16), NBOX, EAV,
      OMXM, OUTLV, IQ(14,16), RIN, LONGR,
      PNR(8), ST(10), SKXP, SKAP, NBUF,
      QPASS(200), XPASS(200), OUT(2), JDB(6), SSTOP,
      REMCNYT(16), ABOX(15), ABUSY(10), XBOX(15), XBOX(10)

COMMON/RLE/ LAST

ISN 0006
ISN 0007  INTEGER OUT
ISN 0008  REAL MEMLY, WXTIME
ISN 0009  REAL TIME
ISN 0010  COMMON/BUF/ SPPH(180,100), SABUFF(12,100), SXBUFF(12,100),
      SAREC(32,100), SXREC(32,100), SABREC(32,100), SBCBIT(24,100),
      SCBIT(24,100), SABSC(15,100), SXFAC(15,100), SBFULL(16,100),
      SHM(16,100), S1Q4(14,100), SBIT(16,100), S5T(10,100)

ISN 0011  INTEGER#2 CYCLE
      JT=CYCLE

ISN 0012  C
      FILL CYCLE POSITION IN OUTPUT BUFFER
      JT=CYCLE

ISN 0013  C
      FILL DISP REG, PHL, OUTPUT BUFFER

ISN 0014  DO 5 I=1,80
      5 SPHI(I, JT)=PHI(I)

ISN 0015  C
      FILL BRANCH CONTROL OUTPUT BUFFER

ISN 0016  DO 6 I=1,8
      6 SBI(I, JT)=0

ISN 0017  SBI(I+8, JT)=0

ISN 0018  SBI(I+16, JT)=0

L. Conway Archives
ISN 0010    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(AFRCI,NE.0) SB(29,JT)=(AFRCI+240)*256
ISN 0038    IF(AFRC2,NE.0) SB(30,JT)=(AFRC2+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
ISN 0048    C    FILL SKIP CONTROL OUTPUT BUFFER
ISN 0049    DO 4 I=1,8
ISN 0050    4 SS(I,JT)=0
ISN 0051    IF(SR(I),NE.0) SS(I,JT)=(SR(I)+240)*256
ISN 0052    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
ISN 0059    C    FILL ABUFF OUTPUT BUFFER
ISN 0060    DO 10 I=1,NABUF
ISN 0061    10 SABUFF(I,JT)=ABUFF(I,1)
ISN 0062    C    FILL XBUFF OUTPUT BUFFER
ISN 0063    DO 11 I=1,NABUF
ISN 0064    11 SBUFF(I,JT)=XBUFF(I,1)
ISN 0065    C    FILL A AND X FACILITY OUTPUT BUFFER
ISN 0066    DO 9 I=1,NXFAC
ISN 0067    9 SXFAC(I,JT)=XSFAC(I,2)
ISN 0068    DO 19 I=1,NAFAC
ISN 0069    19 CONTINUE
ISN 0070    C    FILL Q AND MEM OUTPUT BUFFER
ISN 0071    DO 8 I=1,16
ISN 0072    8 SQ(I,JT)=Q(I,1)
ISN 0073    C    FILL IQ OUTPUT BUFFER
ISN 0074    DO 7 I=1,4
ISN 0075    7 SIQ(I,JT)=IQ(I,1)
ISN 0076    C    FILL REG BUSY OUTPUT BUFFERS
ISN 0077    C    FILL REG BUSY OUTPUT BUFFERS
ISN 0078    DO 12 I=1,32
ISN 0079    12 SAREG(I,JT)=ABUSY(I)
ISN 0080    SABREG(I,JT)=XBUSY(I)
ISN 0081    12 SREG(I,JT)=XBUSY(I+32)
ISN 0082    12 CONTINUE
ISN 0078  DD 13 I=L+24
ISN 0079  SCBIT(I,JT)=XBUSY(I+64)
ISN 0080  SCBIT(I,JT)=#BUSY(I+64)
ISN 0081  13 CONTINUE
ISN 0082  RETURN
ISN 0083  END
ISN 0027  XFAC(I,K)=0
ISN 0028  XOBUS(1,K)=0
ISN 0029  XBUFF(1,2)=0
ISN 0030  XBUFF(1,12)=0
ISN 0031  4 CONTINUE
ISN 0032  5 CONTINUE

C-- IF SKIP TAKEN NOP ALL STARRED OPS UP TO 1ST EXEC SKIP --

ISN 0033  00 80 1=1,NXBUF
ISN 0034  IF(XBUFF(1,1),EQ,0) GO TO 79
ISN 0036  IF(XBUFF(1,9),EQ,0) GO TO 84
ISN 0038  IF(XBUFF(1,11),EQ,0) GO TO 84
ISN 0040  79 CONTINUE
ISN 0041  IF(XBUFF(1,11),EQ,0) GO TO 80
ISN 0043  IF(XBUFF(1,9),EQ,0) GO TO 90
ISN 0045  80 81 K=1,NXREGS
ISN 0046  XSORT(1,K)=0
ISN 0047  81 XOEST(1,K)=0

C APPROX WAY TO HANDLE SKIPPED BRANCHES
C LET GO AS UNSUC Branch - NO S/D INFLKS

ISN 0048  IF(XBUFF(1,12),NE,-1) GO TO 1081
ISN 0050  XBUFF(1,10)=0
ISN 0051  1081 CONTINUE
ISN 0053  00 82 K=1,NXFAC
ISN 0054  XFAC(I,K)=0
ISN 0055  82 XOBUS(1,K)=0
ISN 0056  XBUFF(1,2)=0
ISN 0057  83 K=4,5
ISN 0058  83 XBUFF(1,K)=0
ISN 0059  83 CONTINUE
ISN 0060  84 CONTINUE

C-- THIS EVENT SCANS XBUFF FOR INST WHICH CAN GO
C SCAN FOR XBUF OUT OF XTEST

ISN 0061  00 10 REG=1,NXREGS
ISN 0062  SORASV(REG)=0
ISN 0063  10 DESASV(REG)=XBUSY(REG)
ISN 0064  00 I=10 INS=1,XTEST
ISN 0065  IF(INC(I+15),EQ,0) GO TO 100
ISN 0067  IF(INC(I+11),EQ,0) GO TO 21
ISN 0069  00 11 K=1,NXREGS
ISN 0070  SORASV(REG)=SORASV(REG)+XSOR(INS-1,REG)
ISN 0071  11 DESASV(REG)=DESASV(REG)+XOEST(INS-1,REG)
ISN 0072  INS=INS-1
ISN 0073  00 20 I=1,INS

C PREV EXIT INFLKS ALL CODE BELOW
C PREV SKIP INFLKS ALL STARRED CODE BELOW
C AND ALL SKIPS BELOW
ISN 0076 IF(XBUFF(I,13).EQ.0) GO TO 20
ISN 0078 IF((XBUFF(IN,13),.EQ.11).OR.(XBUFF(IN,9),.EQ.1)) GO TO 100
ISN 0080 20 CONTINUE
ISN 0081 21 CONTINUE
ISN 0082 C IF EXIT, INTLK AGAINST PREV BRANCHES AND ER
ISN 0084 IF(E(BXRP),.NE.1) GO TO 28
ISN 0086 IF(IN,.EQ.1) GO TO 129
ISN 0088 DD I=1,1,INSM1
ISN 0089 IF(XBUFF(I,12),.EQ.1) GO TO 100
ISN 0091 128 CONTINUE
ISN 0092 C EXIT PART OF OP GOES, MARK GO EXIT.
ISN 0093 129 CONTINUE
ISN 0094 XBUFF(IN,15)=1
ISN 0095 28 CONTINUE
ISN 0096 DD 22 REG=1,XXREGS
ISN 0097 IF((XSOR(IN,REG),.EQ.11).AND.(DESBY(REG),.NE.00)) GO TO 100
ISN 0098 IF((XDESI(IN,REG),.EQ.11).AND.(DESBY(REG),.NE.00)) GO TO 100
ISN 0100 IF((XDESI(IN,REG),.EQ.11).AND.(DESBY(REG),.NE.00)) GO TO 100
ISN 0102 22 CONTINUE
ISN 0103 C FIND FAC USED
ISN 0104 DO 25 FAC=1,XXFAC
ISN 0106 25 CONTINUE
ISN 0107 C NO FAC USED, ISSUE OP
ISN 0108 FAC=0
ISN 0109 26 CONTINUE
ISN 0109 C ---- TEST FOR SPECIAL OPS HERE ----
ISN 0109 SPEC=0
ISN 0112 SPEC=1
ISN 0114 QPT=QINPT+QGO
ISN 0115 IF(QPT,GT,NOBUFF) GO TO 100
ISN 0116 27 CONTINUE
ISN 0117 C IF BRANCH, INTLK AGAINST PREV BRANCHES AND EH T AVAIL
ISN 0118 IF(XBUFF(IN,12),.NE.1) GO TO 29
ISN 0119 C IF XBPRI(BXRP),.EQ.1) GO TO 100
ISN 0120 C IF SHORT BRANCH, TEST LONGBR INTLK
ISN 0121 IF(IN,.EQ.1) GO TO 231
ISN 0124 DD I=1,1,INSM1
ISN 0126 IF(XBUFF(I,12),.EQ.1) GO TO 100
ISN 0128 230 CONTINUE
ISN 0129 231 SPEC=1
ISN 0130 C IF SKIP, INTLK AGAINST PREV NOGO STARRIED OPS, SHT AVAILABLE
ISN 0131 IF(XBUFF(IN,13),.NE.1) GO TO 132
ISN 0133 IF(SR(SKXP),.EQ.1) GO TO 100
ISN 0135 IF(IN,.EQ.1) GO TO 131
ISN 0137  DO 130 I=1,INSH1
ISN 0138   IF((XBUFF(1),EQ.1).AND.(XGO(1),NE.1)) GO TO 100
ISN 0140   130 CONTINUE
ISN 0141   131 SPEC=1
ISN 0142   132 CONTINUE
C       ------------------------------------------
C        IF NORMAL OR SPEC UP AND NGO =NXGO, DO NOT ISSUE
C        IF REPLACE OR NOD, CAN ISSUE ANYWAY.
C ISN 0143   IF(((FAC,NE.0).OR.(SPEC,NE.0)).AND.(NGO,EQ.NXGO)) GO TO 100
C ISN 0144   IF NO FACS USED GO DIRECTLY TO 95
C ISN 0145   IF(FAC,EQ.0) GO TO 95
C        IF MULT IDENT FAC, GO TO SPEC HANDLING
C ISN 0146   IF(XFACT(INSH,FAC),GT.1) GO TO 49
C        CHECK INBUS,FAC SLOT,OUTBUS INLKS
C ISN 0147   C        NO INBUS CONFLICTS IN X
C ISN 0148   BOX=XB0X(FAC)
ISN 0149   IF(XXIRSY(BOX),EQ.1) GO TO 100
ISN 0150   DO 30 T=1,N200
ISN 0152   30 CONTINUE
ISN 0153   IF((XFSLOT(FAC,T),EQ.1).AND.(XFACSC(1,FAC,T),EQ.1)) GO TO 100
ISN 0155   30 CONTINUE
ISN 0156   OBS=XB0US(FAC)
ISN 0157   DELAY=XXFDLY(FAC)
ISN 0158   IF((XBUSU(INSH,BUS1),NE.0).AND.(XXBUSU11(INSH,BUS,DELAY),NE.0))
C X GO TO 100
ISN 0160   IF((XBUSU(INSH,BUS),NE.0).AND.(XXBUSU1(INSH,BUS+1),DELAY),NE.0).AND.
C X ((OBUS+1),LE.NBUX) GO TO 100
C       SUCCESS. MARK GO AND SET SHIFT CELLS
ISN 0162   C        31 CONTINUE
ISN 0163   XIRSSY(INBUS)=1
ISN 0164   X00SYY(0,00)=1
ISN 0165   XB0US1=XBUFF(INSH,1)
ISN 0166   DO 32 T=1,N200
ISN 0167   IF((XFSLOT(FAC,T),EQ.0) GO TO 32
ISN 0169   XFACSC(1,FAC,T)=1
ISN 0170   XFACSC(2,FAC,T)=XBUFF
ISN 0171   32 CONTINUE
ISN 0172   XB0USC1(INSH,BUS,DELAY)=XOBUSU(INSH,BUS)
ISN 0173   XB0USC2(INSH,BUS,DELAY)=XBUFF(INSH,1)
ISN 0174   XB0USC3(INSH,BUS,DELAY)=XBUFF(INSH,2)
ISN 0175   IF(XBUSU1(INSH,BUS+1),EQ.0) GO TO 95
ISN 0176   IF((INBUS1),T.NXBUSU) GO TO 95
ISN 0177   XB0USC1(INSH,BUS+1),DELAY)=XOBUSU(INSH,BUS+1)
ISN 0178   XB0USC2(INSH,BUS+1),DELAY)=XBUFF(INSH,1)
ISN 0181   XB0USC3(INSH,BUS+1),DELAY)=XBUFF(INSH,2)
ISN 0182   49 CONTINUE
C        SPEER ROUTINE TO HANDLE MULT IDENT FAC INLKS
ISN 0183   49 CONTINUE
C        NO INBUS CONFLICTS IN X
ISN 0184   BOX=XB0X(FAC)
ISN 0185   IF(X00SYY(BOX),EQ.1) GO TO 60

306
L. Conway
Archives
ISN 0187  DO 50 T=1,NSLOT
ISN 0188  IF((XFSLFT(FAC,T).EQ.1).AND.(XFACSC1(FAC,T).EQ.1)) GO TO 60

ISN 0190  50 CONTINUE
ISN 0191  OBUS=XBUS(FAC)
ISN 0192  DELAY=XFDLY(FAC)
ISN 0193  IF((XOBUS(INS,OBUS).NE.0).AND.(XBUSSC1(INS,OBUS,DELAY).NE.0))
           XGO=1
           GO TO 60

C SUCCESS

ISN 0195  DO 51 BUS=0,INS
ISN 0196  51 IF((BUS.NE.INS).AND.(INS.NE.INS)) XBUS(INS,BUS)=0.

ISN 0198  GO TO 11
ISN 0199  60 CONTINUE

ISN 0200  FAC=FAC+1
ISN 0201  IF((FAC.GT.XFAC).OR.(XFAC(INS,FAC).EQ.1)) GO TO 100

ISN 0203  GO TO 49
ISN 0204  95 CONTINUE

ISN 0205  XGO(INS)=1
ISN 0206  IF((XSRC(INS,89).EQ.1).OR.(XDEST(INS,89).EQ.1)) QGO=QGO+1

C IF QGO USES NO FACILITIES, AND IS NOT SPECIAL QP THEN IT
C IS A REPLACE QP, AND GOES WITHOUT INCREMENTING QGO.

ISN 0208  IF((FAC.NE.0).OR.(SPEC.NE.0)) QGO=QGO+1

ISN 0210  100 CONTINUE

C -- -- -- -- -- -- EXIT EXECUTION -- -- -- -- -- --
C CHECK FOR QGO EXITS TO SET XHOLDT

ISN 0211  XHOLDT=0
ISN 0212  XHOLDT=0
ISN 0213  IF((XBUFFER(1,14).EQ.1).AND.(XBUFFER(1,15).EQ.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((XBUFFER(I,15).NE.1)) GO TO 201

ISN 0220  IF((XBUFFER(I,14).NE.1)) GO TO 201
ISN 0222  XBUFFER(I,14)=0
ISN 0223  XBUFFER(I,15)=0
ISN 0224  IF((XBUFFER(I,10).EQ.1)) GO TO 202

ISN 0226  201 CONTINUE
ISN 0227  GO TO 300

C FOUND GO EXIT,ET. NOP AND MARK QGO ALL CODE BENEATH IT.
C ALSO SET XFRCT.

ISN 0228  239 XFRCT=1
ISN 0229  IF(1.EQ.NXBUF) GO TO 300

ISN 0231  I=I+1
ISN 0232  DO 203 J=1,NXBUF
ISN 0233  XGO(J)=1
ISN 0234  DO 204 K=1,NXREGS
ISN 0235  XSOR(J,K)=1
ISN 0236  204 XDEST(I,K)=0
ISN 0237  DO 205 K=1,10
ISN 0238  205 XBUFFER(I,K)=0

307

L. Conway
Archives
ISN 0239   XBUFF(1,2)=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--------------------- CALL BOSEX TO SAVE .1 BUS 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 0250   IF(XBUFF(I,13).EQ.1) GO TO 86
ISN 0251   85 CONTINUE
ISN 0252   86 CONTINUE

ISN 0253   RETURN
ISN 0254   END
SUBROUTINE XXEMP

IMPLICIT INTEGER(2-1Z)

COMMON TIME, IPAR1, IPAR2, IPAR3

A AINPT, NABUF, ABUS(50), XINPT, NBUF,
B XBUS(50), (FADD, IFST, IFRTN, XRXP,
C BRAP, ER(8), RE(8), ET(8), NABUF,
D AHOLDT, XRHT, AFRCT, XFRCT, BOSC,
E BNOP, XEP, AEP, PHIL(100), PRINT,
F FNAGD, XNODT, NODSC, NODUS, NADSP,
G NQSP

COMMON/KLS/ FIRST, NAREGS, NXREGS, NABUS,

A NAXUS, STA5, ACUN, XCON, AEMP,
B XEMP, XMAG, AFR(12), XFR(12), AG0(12),
C AG0(12), NAG0, NXG0, NATEST, NTEXT,
E XBUSY(2001), ABUFF(12,100), XBUFF(12,100), AOD(12,2001),
F XBUS(12,2001), AODST(12,2001), XODST(12,2001), AODC(12,15),
G XFRAC(12,15), AFACSC(2,15), ARF, XFRACSC(4,15,20), XRKT,
H ABUSC(4,15,20), ARP, ABUS(12,10), ABUSC(4,15,20), XRKT,
I ABOUS(1,12,10), XBOUS(12,10), AFSLUT(15,20), XFSLUT(15,20), AFIBUS(15),
J AFJULY(15), XJULY(15), AFJULY(15), XFSLUT(15,20), AFIBUS(15),
K AUBPSZ(15), XJULY(15), AFJULY(15), XFSLUT(15,20), AFIBUS(15),
L JU(16,15), XJULY(15), AFSLUT(15,20), NBUF, NTEST, NQCS,
M GINPT, QCON, QEMP, MBUSY, MPREE,
N LOAD, MEMAD, MEMADY(16), NCHO, EAV,
O MTIME, OUTVLX, IQI(4,16), KTM, LONTBR,
P SR(18), XSSRKP, NSBUF, NSBUF,
Q APASS(2001), XPASS(2001), XOUT(2), JDB(6), SSSTOP,
R MENGNT(16), AOM(15), ABUSXY(10), XBUS(15), XBUSXY(10)

COMMON/KLS/ LAST

INTEGER OUT

COMMON/FAS/(DOB155,70)

REAL MENDY, MXTIME

REAL TIME

DO 100 IN=1,NKBUF

IF(XOR(XIN))E=0 GO TO 100

IF(XOR(XIN))E=0 GO TO 100

C ISSUE INS
C --- TEST FOR SPECIAL OPS HERE -----
C IFSH SP TO QUEUE

IF((XSCM(IN,59),NE,1).AND.(XDEST(IN,89),NE,1)) GO TO 7

CALL CAUSESTEV, TIME+1, 0, 0, 0

IN=GRTK

C QINPT=GMPNTI

SET a LETTER

C Q(IN,1)=XBUFF(IN,1)

SET a

C Q(IN,4)=XBUFF(IN,7)

SET a

C Q(IN,4)=XBUFF(IN,7)
IF(Q(IN,4).NE.1) Q(IN,5)=1
        SET Q EFF ADD

Q(IN,7)=XBUFF(INS,6)
        SET Q BOM

Q(IN,6)=MUD(Q(IN,7),ABOX)+1
        SET Q LOAD

IF(XSROR(INS,9).EQ.11) Q(IN,2)=1
        SET Q STORE

IF(XDEST(INS,9).EQ.11) Q(IN,3)=1
        SET Q DATA VALID FOR X STORE

IF((Q(IN,5).EQ.1).AND.(Q(IN,3).EQ.1)) Q(IN,9)=1
IF(Q(IN,2).NE.1) GO TO 88

SET Q DEST FOR LOADS

DU d REG=1, NXXREGS

IF(XDEST(INS,REG).NE.0) Q(IN,15)=REG

88 CONTINUE

IF STORE A, GET DATA OR SET WAIT

IF(Q(IN,3).NE.1).OR.(Q(IN,4).NE.1)) GO TO 7

IF(ISUBA(1,1).NE.11) GO TO 6

DATA AVAIL

Q(IN,9)=1

GO 50 I=1, J1

SUBA(1,1)=SUBA(1+1,1)

50 SUBA(1,2)=SUBA(1+1,2)

SUBA(32,2)=0

SUBA(32,1)=0

GO TO 7

6 CONTINUE

DATA NOT AVAIL. SET FIRST FREE WAIT BIT

NE 4 I=1, J1

IF(SUBA(1,2).EQ.1) GO TO 4

SUBA(1,2)=1

GO TO 7

4 CONTINUE

A=1

A=20000

C=102

CALL TROJAL(A,A,C)

7 CONTINUE

ISSUE BRANCH UP

IF(XBUFF(INS,12).NE.1) GO TO 200

IF long branch, set LONGBR=1 to INTLK SHORT BRANCH NXTCYC

LONGBR=1

IF(XBUFF(INS,5).NE.0).AND.(XBUFF(INS,2).NE.(30)) LONGBR=1

IF succ long branch, set LONGBR=2

LONGBR=2

200 CONTINUE

ISSUE SKIP-INC & SKIP POINTER, SET SR

IF(XBUFF(INS,13).NE.1) GO TO 60
ISR 0073
SR(SKXP)=1
ISR 0074
SKXP=SKXP+1
ISR 0075
IF(SKXP.GT.NXBUF) SKXP=1
ISR 0077
60 CONTINUE

C                   
ISR 0078
OP = XBUFF(INS,2)
ISR 0079
REPL=D10(OP,32)

C                   
ISR 0080
DD 10 REG=1,NXREGS
C                   
ISR 0081
IF(XDEST(INS,REG).NE.1) GO TO 10
C                   
ISR 0083
IF(REG.EQ.89) GO TO 10
C                   
ISR 0085
IF(XBUPX(REG).NE.1) GO TO 9
C                   
ISR 0087
IF(REPL.EQ.0) GO TO 9
C                   
ISR 0089
IF(XBUFF(REG).NE.1) GO TO 99
ISR 0091
XBUFUL(REG)=0
ISR 0092
ABUSY(REG)=0
ISR 0093
G0 TO 10
ISR 0094
99 XBUFF(REG)=XBUFF(INS,1)
ISR 0095
9 XBUFF(REG)=XBUFF(INS,1)
ISR 0096
10 CONTINUE

C                   
ISR 0097
XINPT=XINPT+1
ISR 0098
M=NXBUF=I
ISR 0099
IF(INS=0,NXBUF) GO TO 31
ISR 0101
DD 40 I=TNS,M
ISR 0102
XBUF(I)=XBUF(I+1)
ISR 0103
XFULL(I)=XFULL(I+1)
ISR 0104
DU 25 J=1,25
ISR 0105
25 XBUF(I,J)=XBUFF(I+1,J)
ISR 0106
DD 25 J=1,NXREGS
ISR 0107
XSUK(I,J)=XSUK(I+1,J)
ISR 0108
26 XDEST(I,J)=XDEST(I+1,J)
ISR 0109
DD 27 FAC=1,NXFAC
ISR 0110
27 XFAC(I,FAC)=XFAC(I+1,FAC)
ISR 0111
DD 28 BUS=1,NXHUS
ISR 0112
28 XBUFF(I,BUS)=XBUFF(I+1,BUS)
ISR 0113
30 CONTINUE
ISR 0114
31 CONTINUE
ISR 0116
AGC(NXBUF)=0
ISR 0117
60 125 J=1,25
ISR 0118
125 XBUFF(NXBUF)=0
ISR 0119
DU 126 J=1,NXREGS
ISR 0120
XBUF(NXBUF)=0
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
L. Conway
Archives
ISN 0022
9. BUSY(DEST) = 0

ISN 0023
10 CONTINUE

PLACE ANY EXEC ACTIVITY HERE -- -- -- --

ISN 0024

C CHECK FOR MOVE OF HLT POSITION

MOVE BXPR

ISN 0025
BRXP = BRXP + 1

ISN 0027
IF (BRXP, GT, NBUF) BRXP = 1

ISN 0029
200 CONTINUE

ISN 0030
IF (IDERA, NE, 11, OR (DAP, NE, 11)) GO TO 201

ISN 0032
BE (BRAPI) = 0

ISN 0033
IF (BRAPI) = 0

ISN 0034
ET (BRAPI) = 0

ISN 0035
IF (BRAPI, GT, NBUF) BRAPI = 1

ISN 0036
201 CONTINUE

ISN 0038

C CHECK FOR RESETTING OF HUP

ISN 0039
IF (HUNP, EQ, 11) AND (HUP, EQ, 11) HUP = 0

ISN 0041
MAIN BRANCH EXECUTION ROUTINE

ISN 0043
IF (ETXPS, NE, 11) GT, TO, 250

ISN 0045
IF (SUSSCU, EQ, 11) GO TO 240

ISN 0047
IF (UBRP, NE, 11) GO TO 290

ISN 0049
ER (UBRP) = 1

ISN 0050
ET (UBRP) = 0

ISN 0051
GT, TO, 290

ISN 0052
230 CONTINUE

ISN 0053
ER (UBRP) = 1

ISN 0054
ET (UBRP) = 0

ISN 0055
GT, TO, 290

ISN 0056
240 CONTINUE

ISN 0057
ER (UBRP) = 1

ISN 0058
ET (UBRP) = 1

ISN 0059
BRAPI = 1

ISN 0060
GT, TO, 290

ISN 0061
250 CONTINUE

ISN 0062
IF (HUP, NE, 11) GO TO 290

ISN 0064
IF (HUP, EQ, 11) GO TO 290

ISN 0066
IF (SUSSCU, EQ, 11) GO TO 290

ISN 0068
ER (BRAPI) = 1

ISN 0069
ET (BRAPI) = 0

ISN 0070
290 CONTINUE

C SHIFTS THE SHIFT CELLS

ISN 0071
DU, 99, I = 1, 10

L. Conway
Archives
L. Conway
Archives
ISN 0002  SUBROUTINE TSTEP(IEVENT)
ISN 0003  IMPLICIT INTEGER*2(A-Z)
ISN 0004   IPAR1, IPAR2
ISN 0006   ICOUNT, ICALENDAR, ICYCLE
ISN 0007   A CTIME(200), NEVENT(200), KOL1(200), KOL2(200), KOL3(200)
ISN 0008   REAL CTIME
ISN 0009   INTEGER IEVENT
ISN 0010   ID=ITL
ISN 0011   ITL=LINK(ID)
ISN 0012   ISIL(IDC)=ILS
ISN 0013   ICOL=KOL1(ID)
ISN 0014   KOL1(IDC)=KOL2(ID)
ISN 0015   KOL2(IDC)=KOL3(IDC)
ISN 0016   IEVENT=NEVENT(IDC)
ISN 0017   ICOUNT=ICOUNT+1
ISN 0018   RETURN
ISN 0019   END
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 I8, 8(2X I8, 4X ))
ISN 0038  3333 FORMAT(19H TIME/DATE OF RUN =,2(1XZ8))
ISN 0039  END
SUBROUTINE CAUSE(IEV,T,IP1,IP2,IP3)
IMPLICIT INTEGER*2(A-Z)
C CAUSE ENTERS EVENTS ONTO CALENDAR
C ITL IS LOCATION OF FIRST EVENT IN CALENDAR
C IPL IS LOCATION OF FIRST AVAILABLE IN CALENDAR
NEXT=1FL
GO TO 20
LOOP UNTIL GIVEN TIME IS LESS THAN NEXT ENTRY IN CALENDAR
10 LAST=NEXT
NEXT=LINK(NEXT)
20 IF ITL.GT. (TIME(NEXT)) GO TO 10
30 IF ITL.LT. IPL GO TO 10
35 IF IPL.GT. (TIME(NEXT)) GO TO 10
40 IF (NEXT.EQ. ITL) GO TO 40
C SEE IF THIS EVENT WILL BE THE FIRST IN THE LIST
50 LINK(LAST)=ID
55 ID=EVENT(ID)+T
60 KOL1(ID)=IP1
65 KOL2(ID)=IP2
70 KOL3(ID)=IP3
RETURN
END
INPUT PROGRAM FOR THIS RUN = MM-MS

TIME/DATE OF RUN = 5A63CE76 0067271F

MACHINE PARAMETERS FOR THIS RUN ---

1) NUMBER OF A BUFFERS = 8
2) NUMBER OF X BUFFERS = 2
3) NUMBER OF Q BUFFERS = 8
4) NUMBER OF A OPS TESTED = 6
5) NUMBER X OPS TESTED = 2
6) MAX A OPS ISS/CYCLE = 2
7) MAX X OPS ISS/CYCLE = 2
8) MAX Q OPS ISS/CYCLE = 2
9) MINIMUM Q-MEM DELAY = 5.0
10) NUMBER OF BOMS = 8
11) NUMBER BRANCH REGS = 3
12) NUMBER OF SKIP REGS = 4
13) SIZE OF DD TABLE = 6
14) NUMBER OF PSC REGS = 8
15) NUMBER DISP BUSES = 1
16) MAX A OPS DSP/CYCLE = X
17) MAX X OPS DSP/CYCLE = X

A FACILITIES -- FA1 FA2 FM PD IA IM ID C L S
REP TIME = 1 1 X2 X8 1 2 10 1 1 1 1
DELAY TIME = 3 4 X4 X10 2 5 15 1 X2 X2 X2
INBUS = 2 3 1 1 2 2 1 2 3 1
BOX = 1 3 X3 2 X3 X3 X3 X3 X3
OUTBUS = 2 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 X5 X5 X5 X5 X3 X3
OUTBUS = 5 6 1 3 2 2 7 10 8

L. Conway Archives
<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>MIN</th>
<th>TYP</th>
<th>MAX</th>
<th>COLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOBNAME</td>
<td>1</td>
<td>8</td>
<td>12</td>
<td>1-6</td>
</tr>
<tr>
<td>NABUF</td>
<td>1</td>
<td>8</td>
<td>NABUF</td>
<td>9-10</td>
</tr>
<tr>
<td>NATEST</td>
<td>1</td>
<td>8</td>
<td>NABUF</td>
<td>11-12</td>
</tr>
<tr>
<td>NAGØ</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>13-14</td>
</tr>
<tr>
<td>NXBUF</td>
<td>1</td>
<td>3</td>
<td>12</td>
<td>15-16</td>
</tr>
<tr>
<td>NTEST</td>
<td>1</td>
<td>3</td>
<td>NXBUF</td>
<td>17-18</td>
</tr>
<tr>
<td>NXGØ</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>19-20</td>
</tr>
<tr>
<td>NQBUF</td>
<td>1</td>
<td>8</td>
<td>16</td>
<td>21-22</td>
</tr>
<tr>
<td>NQTEST</td>
<td>1</td>
<td>8</td>
<td>16</td>
<td>23-24</td>
</tr>
<tr>
<td>NQGØ</td>
<td>1</td>
<td>2</td>
<td>NBØX</td>
<td>25-26</td>
</tr>
<tr>
<td>NBØX</td>
<td>1</td>
<td>8</td>
<td>16</td>
<td>27-28</td>
</tr>
<tr>
<td>NBBUF</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>29-30</td>
</tr>
<tr>
<td>NSBUF</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>31-32</td>
</tr>
<tr>
<td>NØDØT</td>
<td>1</td>
<td>4</td>
<td>16</td>
<td>33-34</td>
</tr>
<tr>
<td>NØPSC</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>35-36</td>
</tr>
<tr>
<td>NDBUS</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>37-38</td>
</tr>
<tr>
<td>NADSP</td>
<td>1</td>
<td>4</td>
<td>NABUF</td>
<td>39-40</td>
</tr>
<tr>
<td>NXDSP</td>
<td>1</td>
<td>3</td>
<td>NXBUF</td>
<td>41-42</td>
</tr>
<tr>
<td>MXTIME</td>
<td></td>
<td>300.0</td>
<td></td>
<td>60-66 (F7.1)</td>
</tr>
<tr>
<td>MEMDLY</td>
<td>2.0</td>
<td>5.0</td>
<td></td>
<td>68-71 (F4.1)</td>
</tr>
<tr>
<td>QUTLVL</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>73-74</td>
</tr>
<tr>
<td>FSTADD</td>
<td>0</td>
<td>0</td>
<td></td>
<td>76-80</td>
</tr>
</tbody>
</table>
INPUT PROGRAM FOR THIS RUN = MM-MS
TIME/DATE OF RUN = 5A63CE76 0067271F

MACHINE PARAMETERS FOR THIS RUN ----

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

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

MAX A OPS ISS/CYCLE = 8
MAX X OPS ISS/CYCLE = 8
MAX Q OPS ISS/CYCLE = 8

MINIMUM Q-MEM DELAY = 5.0

NUMBER OF BOMS = 8

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

NUMBER OF PSC REGS = 8

NUMBER DISP BUSES = 2

MAX A OPS DSP/CYCLE = 3
MAX X OPS DSP/CYCLE = 3

--- A FACILITIES --- FA1 FA2 FM FD IA IM ID C L S
REP TIME = 1 1 8 1 2 10 1 1 1 1
DELAY TIME = 3 1 10 2 5 15 1 1 1 1
INBUS = 2 3 1 2 2 1 1 2 3 1
OUTBUS = 2 3 2 2 4 6 1 3 1

--- X FACILITIES --- EA1 EA2 L S M D XA C SP
REP TIME = 1 1 1 2 8 1 1 1
DELAY TIME = 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

L. Conway
Archives
EXAMPLE OF INIT
MODIFICATIONS TO
CHANGE FACILITY
STRUCTURE:

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,DE30
ISN 0029 TIME=0.0

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 QCUN=10
ISN 0040 QEMP=11
ISN 0041 MBUSY=12
ISN 0042 MPREE=13
ISN 0043 LOAD=14
ISN 0044 RIN=15

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(QCUN,TIME+0.1,0,0,0)
ISN 0049 CALL CAUSE(MXO ,TIME+0.6,0,0,0)

C C INITIALIZE THE MACHINE PARAMETERS
ISN 0050 BRXP=1
ISN 0051 BRAP=1
ISN 0052 SKXP=1
ISN 0053 SKAP=1
ISN 0054 MXREGS=90
ISN 0055 NXREGS=90
ISN 0056 AINPT=1
ISN 0057 QINPT=1
ISN 0058 XINPT=1
ISN 0059 DO 50 I=1,13,32
ISN 0060 50 XBUPS(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 C INITIALIZE APAC TABLES
ISN 0066 NABUS=6
ISN 0067 NAFAC=10
A FACILITIES:
- REP TIME \rightarrow \text{AFSLOT}(3,3)
- DELAY TIME \rightarrow \text{AFDLY}(3)
- INBUS \rightarrow \text{AFSBUS}(3)
- BOX \rightarrow \text{ABOX}(3)
- OUTBUS \rightarrow \text{AFSBUS}(3)

X FACILITIES:
- REP TIME \rightarrow \text{XFSLOT}(3,3)
- DELAY TIME \rightarrow \text{XFSLY}(1)
- BOX \rightarrow \text{XBXX}(3)
- OUTBUS \rightarrow \text{XFBSI}(3)

\text{AFSLAT}(3,4) = 1
\text{AFSLAT}(4,10) = 1
\text{AFDLY}(3) = 4
\text{AFDLY}(4) = 10
\text{AFDLY}(9) = 2
\text{AFDLY}(10) = 2
\text{ABOX}(4) = 3
\text{ABOX}(5) = 3
\text{ABOX}(6) = 3
\text{ABOX}(7) = 3
\text{ABOX}(8) = 3
\text{ABOX}(9) = 3
\text{ABOX}(10) = 2
\text{ABOX}(11) = 2
\text{XBOX}(4) = 3
\text{XBOX}(5) = 3
\text{XBOX}(6) = 3
\text{XBOX}(7) = 3
\text{XBOX}(8) = 3

\text{DP} 991 I = 56, 66
D(I, 56) = 1
D(I, 57) = 0
991 CONTINUE

\text{to get rid of FAZ}
\text{had to change OS DECODE TAGS}
November 29, 1967
Advanced Computing Systems
Menlo Park, California
988/031

Subject: Cover Letter for Preliminary Distribution of Logical Design Memorandum

Reference:

To: Mr. S. F. Anderson  Mr. R. J. Robelen
    Mr. B. O. Beebe      Dr. H. Schorr
    Dr. C. V. Freiman    Dr. E. H. Sussenguth
    Mr. M. E. Homan      Mr. W. P. Wissick
    Mr. B. J. Mooney

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
November 29, 1967
Advanced Computing Systems
Menlo Park, California
988/031

Subject: ACS Logical Design Conventions: A Guide for the Novice

References:

To: FILE

L. Conway

329
L. Conway Archives
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1 - 1</td>
</tr>
<tr>
<td>The ACS Logical Circuits</td>
<td>2 - 1</td>
</tr>
<tr>
<td>Logic Equation Conventions</td>
<td>3 - 1</td>
</tr>
<tr>
<td>Logic Circuit Diagram Conventions</td>
<td>4 - 1</td>
</tr>
<tr>
<td>Elementary Logic Design</td>
<td>5 - 1</td>
</tr>
</tbody>
</table>
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.
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:

\[
\begin{array}{cccc}
A & B & X & Y \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 \\
\end{array}
\]

\[X = \overline{A} \cdot \overline{B}\]

\[Y = A + B\]

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 \(Y = \overline{X}\).

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:

![Graph of Max. Current Switch Fan-in vs. Max. Input e.f. Dot Fan-in]
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:

```
A

B

C

OCD

X
```

\[ X = A \cdot B \cdot C \]

Orthogonality Restriction:

No two inputs may be 0 (negative)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N.A.</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>N.A.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>N.A.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>N.A.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(N.A. = not allowed)
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: $\leq 5$

Fan-out: See fan-out for current switch. Same description applies here.

**The Emitter Follower Dot:**

```
A
\( \rightarrow \)
X
```

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$X = A + B$

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: $\leq 5$

Fan-out: $\leq 12$ (try for $\leq 8$)
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 \( Y = B + C \):

![Diagram](attachment:image.png)

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:

![Diagram](attachment:image.png)
Logic Equation Conventions

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:

\[
\begin{align*}
\text{AND}(A, B) & : \quad A \cdot B = AB = A \overline{B} \\
\text{OR}(A, B) & : \quad A + B = A \lor B \\
\text{NOT}(A) & : \quad \overline{A} = A' = -A
\end{align*}
\]

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 \( \overline{A} \) and \(-A\), or perhaps even \(-\overline{A}\) 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).
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 \(A, B, C\):

\[
\begin{align*}
\bar{X} &= A \cdot B \cdot \bar{C} \\
-X &= A \cdot B \cdot \bar{C} \\
+X &= -(A \cdot B \cdot \bar{C})
\end{align*}
\]

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 \(+\) 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.
Logic Circuit Diagram Conventions

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: \( \Diamond \)) in order to avoid confusing O.C.D. with emitter follower dot.

![Method I Diagram](image1)

![Method II Diagram](image2)
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.
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.
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 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:

![Wedge symbol]

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).
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 \( X = \frac{1}{Y} \), 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.
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.
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 \( \text{NOR} (a, b) = (a + b) \). 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.

![Logic Circuit Diagram](attachment:image.png)

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.
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:
\[ f = ab + bc + ad \]

Step 2: Transform the equation to NOR-OR form by complementing each variable and replacing the AND operators with NOR operators:
\[ f = \text{NOR} (\overline{a}, \overline{b}) + \text{NOR} (\overline{b}, \overline{c}) + \text{NOR} (\overline{a}, d) \]

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.
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 \( f = ab + bc + ad \).

Using minus (-) inputs we can use "MINUS AND INVERT" circuits to obtain the terms \( ab, bc, \) and \( ad \). 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.
October 31, 1967
Advanced Computing Systems
Menlo Park, California
988/031
252

Subject: A Proposed ACS Logic Simulation System (LSS)

Reference:
1. Specifications for Input and Output of ACS/TALES Simulator,
2. TALES - ACS Simulation Capability, A. G. Auch, Dept. E24,
   SDD Poughkeepsie, August 15, 1967.
3. ACS AP #67-115, MPM Timing Simulation, L. Conway,
4. ACS AP #66-022, ACS Simulation Technique, D. P. Rozenberg,

To: File

L. Conway

L. Conway

LCaw
CONTENTS

Introduction 1 - 1
The LSS Programs 2 - 1
Possible Procedures for Use 3 - 1
Requirements for Development 4 - 1
Additional Benefits of LSS 5 - 1
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.
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.
FIG 1. 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.

INPUT

SIMULATION OF SYSTEM LEVEL DESIGN → OUTPUTS

SIMULATION OF LOGIC CIRCUITS IMPLEMENTING THE SYSTEM LEVEL DESIGN → OUTPUT

(COMPAR)
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 flowcharting the relations between the designer's input, the various LSS programs, the DRKS files, and the various LSS internal files.
FIG 3. THE ACS LOGIC SIMULATION SYSTEM

I. SELECT PARTITION OF MPM:

II. GENERATE PARTITION INTERFACE SIGNALS:

III. SIMULATE LOGIC OF SELECTED PARTITION:

DESIGNER'S INPUT CARDS, SPECIFY LOGIC TO BE SELECTED

DRKS FILES
DRKS RULES

SIP: SIMULATION INTERFACE PROGRAM.
(To be written: POK). EXTRACTS SELECTED LOGIC FROM DRKS FILES AND FORMS INTO INPUT FOR LOGIC SIMULATOR

SELECTED LOGIC FILE

INPUT PROGRAM FOR TIMING SIMULATOR

TIMING SIMULATOR: (ALREADY WRITTEN: ACS)
PERFORMS SYSTEM LEVEL SIMULATION OF MPM EXECUTING THE INPUT PROGRAM

INTERFACE SIGNAL FILE GENERATOR:
(To be written: ACS). EXTRACTS INPUT/OUTPUT SIGNALS AT THE DESIGNER'S INTERFACE WITHIN THE TIMING SIMULATOR DURING EACH CYCLE OF SIMULATED EXECUTION. FORMS FILE OF THESE SIGNALS TO INPUT THE LOGIC SIMULATOR

INTERFACE SIGNAL FILE

TANLP PROGRAM: (TO BE WRITTEN: POK).
OPERATES ON THE INTERFACE SIGNAL FILE, REFORMATS FILE TO FORM TAILS INPUT

TAILS PROGRAM: THE LOGIC SIMULATOR: (TO BE WRITTEN: POK).
SIMULATES THE SELECTED LOGIC CIRCUITS OPERATING UPON THE GENERATED INTERFACE INPUT SIGNALS. COMPARES CIRCUIT OUTPUT SIGNALS WITH THOSE GENERATED BY THE TIMING SIMULATOR. ANY MISTAKES ARE PRINTED OUT WITH ADDITIONAL DEBUGGING INFORMATION

OUTPUT
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)
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.
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 participate 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 "programming" 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 SIT program which actually extracts the logic design of a partition and forms the file to input the logic simulator.
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:

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td>D</td>
</tr>
</tbody>
</table>
```

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.
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.

```
A
Timing simulator dummy for rest of machine
```

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.
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
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.
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.
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 $L = \# \text{ of nets to be simulated (in thousands)}$

Also the fan-out from a block (macro, U.L. or dot) is

$$= \sum_{i=1}^{n} (\text{source}_i \cdot \text{load}_i) \leq 31$$

Thus

```
+----------+---------+---------+
<table>
<thead>
<tr>
<th></th>
<th>load</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 loads</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>macro</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
```

fan-out = 30

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.
(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.

```
  net 1
  net 2  net 4
  net 3
  4 nets
```

```
  net 1
  macro
  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 $= \left(\frac{0.8X}{5}\right)^2$

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. $= \left(\frac{0.2X}{4}\right)^5$

\[
\frac{1.6X}{5} + \frac{X}{4} = 32,000
\]

or $X = \frac{32,000}{0.57} = 56K$ circuits
(iv). \textbf{DRKS} does not handle macros made up of \textit{U. L.} blocks from different portions of the same chip, let alone different chips. So if a high number of \textit{U. L.} blocks is being dotted externally, the above capability will be desirable to keep the net count down.
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.
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:

Memorandum to: File

L. Conway

LC:aw
August 6, 1968

The Computer Design Process: A Proposed Plan for ACS

by: L. Conway

<table>
<thead>
<tr>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mr. J. G. Adler</td>
</tr>
<tr>
<td>Dr. G. M. Amdahl</td>
</tr>
<tr>
<td>Mr. S. F. Anderson</td>
</tr>
<tr>
<td>Dr. R. F. Arnold</td>
</tr>
<tr>
<td>Mr. A. M. Baptiste</td>
</tr>
<tr>
<td>Mr. B. O. Beebe</td>
</tr>
<tr>
<td>Mr. R. T. Blosk</td>
</tr>
<tr>
<td>Mr. J. Earle</td>
</tr>
<tr>
<td>Mr. A. F. Fitch--B73/959 Pok.</td>
</tr>
<tr>
<td>Dr. H. Fleisher--C14/704 Pok.</td>
</tr>
<tr>
<td>Dr. H. Freitag--12/234 Yorktown</td>
</tr>
<tr>
<td>Mr. R. R. Hanko</td>
</tr>
<tr>
<td>Mr. L. J. Hasbrouck</td>
</tr>
<tr>
<td>Mr. F. B. Jones</td>
</tr>
<tr>
<td>Dr. D. G. Keenh</td>
</tr>
<tr>
<td>Mr. L. E. King--B57/979 Pok.</td>
</tr>
</tbody>
</table>
CONTENTS

Introduction 1-1
The Overall Design Process 2-1
System Architecture 3-1
Logic Design and Engineering 4-1
Design and Process Automation 5-1
Maintenance 6-1
Conclusions 7-1
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
(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
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.
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
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 independently.

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:
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.
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.
(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,
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.
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.
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.
FIG. 3. POSSIBLE PLACEMENT TECHNIQUES:

(1) EXISTING NOW:

LOGIC DESIGN

DESIGNER PARTITIONS AND MAPS LOGIC INTO CHIPS

ALMS PROGRAM PRODUCES LIST OF THESE CHIPS FOR EACH MCM OF BOARD

L. WILLIAMS' PROGRAM PRODUCES PLACEMENT OF THESE CHIPS ON EACH MCM

(11) IN DEVELOPMENT:

LOGIC DESIGN

DESIGNER PARTITIONS LOGIC CIRCUITRY AMONG MCM'S OF A BOARD

PROGRAMS IMPLEMENTING UNO KODRES' METHOD WILL:

(1) PRODUCE LIST OF CHIPS TO USE
(11) ASSIGN LOGIC PORTIONS TO THESE CHIPS
(111) PERFORM PLACEMENT OF CHIPS ON MCM'S

(111) ULTIMATELY:

LOGIC DESIGN

PROGRAM IMPLEMENTING PARTITIONING ALGORITHM (IN DEVELOPMENT BY R. GOLDBERG) WILL PARTITION LOGIC AMONG MCM'S
**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...
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.
FIG. 4. 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.

INPUT

SIMULATION OF SYSTEM LEVEL DESIGN

OUTPUTS

(compare)

OUTPUT

SIMULATION OF LOGIC CIRCUITS IMPLEMENTING
THE SYSTEM LEVEL DESIGN

FIG. 5. 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.

SYSTEM LEVEL SIMULATOR FOR COMPLETE MACHINE

INPUT

SYSTEM LEVEL SIMULATOR
FOR PART OF MACHINE

OUTPUTS

(compare)

OUTPUT

LOGIC SIMULATOR FOR
LOGIC CIRCUITS IMPLEMENTED
SELECTED PART OF MACHINE
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.

(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:
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.
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.
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.
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.
(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.
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.