MEMOREX 7100

CPU ARCHITECTURE AND MICRO-PROGRAMMING

L. CONWAY

7/19/72

CONTENTS

TABLES

FIGURES

1.0 INTRODUCTION

This manual documents and describes the Memorex 7100 CPU architecture, and the 7100 micro-instruction set. Also included is a description of an assembly language for the symbolic encoding of 7100 micro-programs, and some examples of techniques for micro-programming the 7100.

The Memorex 7100 is a micro-programmed special purpose processor. It executes Micro-instructions fetched from a "read-only" memory. The 7100 may be micro-programmed to function as a general purpose processor (MRX/30), in which case it would emulate the execution of (MRX/30) instructions fetched from a "read-write" memory. The 7100 may also be micro-programmed to serve other special purposes. For example, it could be micro-programmed to function as a communications adapter.

The 7100 CPU is designed to rapidly execute (by micro-code emulation) 16-bit instructions from a main memory, using simply-structured, minimum-cost hardware.

The purpose of this manual is to document the 7100 CPU design at the architectural level, and to serve as a reference manual for those who design and implement the various micro-programs for the 7100.

2.0 7100 CPU ARCHITECTURE

This section describes the 7100 CPU architecture diagrammed in Figure 1.

The 7100 CPU is organized primarily as a set of 16-bit registers on a 16-bit bus. Many micro-program functions are implemented simply by gating data from one register (SOURCE) to another (DEST), via the bus.

Additionally, a simple ALU which has two input feeder registers (A and B) may be used to perform logical or arithmetic operations on data (in A and B) prior to gating the resulting data to a register (DEST) via the bus.

Some of the registers are discrete and serve specific functions. Of these, some may only function as SOURCES or DESTINATIONS of gating operations but not as both. Figure 1 indicates symbolically which function(s) each specific register may serve.

Other registers are contained in a RAM. These are used as either SOURCES or DESTS and serve as registers for MRX/30 register emulation and MRX/30 Input-Output emulation.

The 7100 is controlled by fetching and executing micro-instructions from a "read-only" memory (UROM).

2.1 7100 CPU CONTROLS

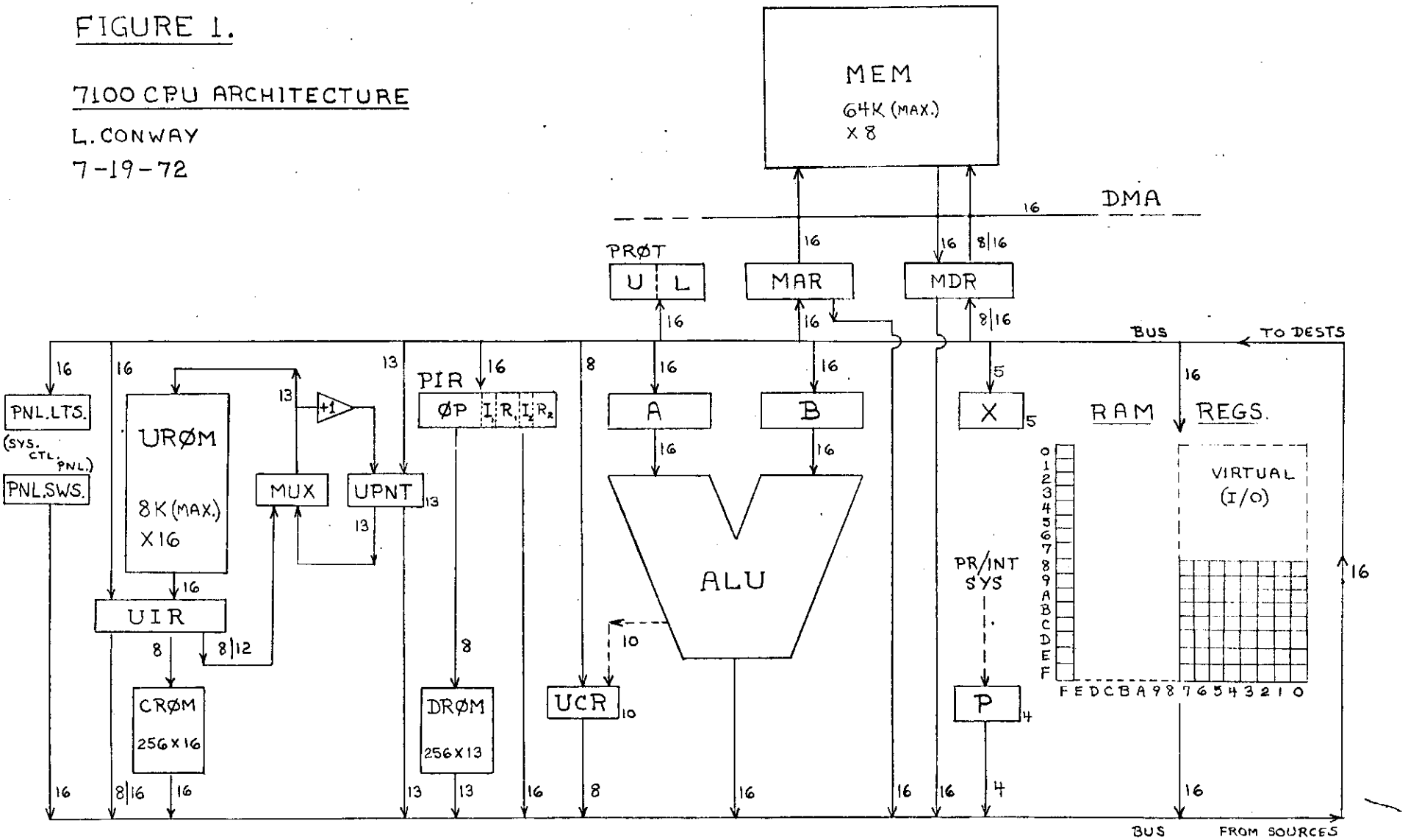
Each micro-cycle (400 ns), the 7100 fetches a micro-instruction from an address in UROM determined by the preceding instruction. The micro-instruction is fetched into the micro-instruction register (UIR). The UIR feeds the control logic which decodes and executes the micro-instruction.

FIGURE 1.

7100 CPU ARCHITECTURE

L. CONWAY

7-19-72



Encoded in the 16 bit micro-instruction are its FORMAT, and the desired SOURCE and DEST registers for gating operations or the OP and DEST for ALU operations. For gating and ALU type operations, the address of the next instruction to be fetched into UIR is formed by incrementing the current address at the time of access and holding the incremented value in the micro-instruction pointer register (UPNT). This incremented value is then used to address the next instruction fetched for execution the following micro-cycle.

Certain FORMATS, however, indicate a BRANCH instruction. These instructions may alter the sequence of instruction execution by specifying a SOURCE for the next instruction address other than the incremented value stored in UPNT.

The details of micro-instruction encoding are fully described in a later section of this manual.

2.2 7100 CPU DISCRETE REGISTERS

The 7100 CPU contains a number of discrete registers with specific, unique functions. These may be functionally separated into 3 groups: control registers, memory interface registers, and data feeders.

The following registers are control registers: UIR, UPNT, PIR, UCR, X, P, and PROT.

UIR, the micro-instruction register and UPNT, the micro-instruction pointer register have already been described (Section 2.1).

PIR, the Program Instruction Register, is used to hold MRX/30 instructions during MRX/30 emulation. An instruction decode read-only memory (DROM) is used to decode the 8-bit OP field (left 8 bits) of PIR and provide a 13-bit jump address for emulation micro-code OP decode jumps. Additionally, logic decodes the I_1, R_1, I_2, R_2, fields of the PIR so that conditional micro-code branches may be

made on these fields being zero/non-zero. I_1 = PIR(8), R_1 = PIR (9-11), I_2 = PIR (12), R_2 = PIR (13-15).

The UCR is a 10-bit register, whose bits are set/reset as the result of ALU logical or arithmetic operations. Conditional micro-code braches may be made on each specific bit of the UCR (micro-condition register) being zero/one.

Registers X and P are used in RAM register addressing which is described in Section 2.4. PROT is used for memory write protection, described below.

The Memory Address Register (MAR) and the Memory Data Register (MDR) are used as interface registers for memory operations. Encoded in certain micro-instructions are "PORT" operations which initiate/control CPU - memory data transfers.

The MAR holds the memory address for these data transfers. The 16-bit MAR can address up to 64K bytes in main memory. The MDR holds the data for writes and receives the data for reads. The PROT (protect) register participates in the control of protected writes. It holds 8-bit fields, U and L, which are the upper and lower bounds of the high order 8-bits of the write address in MAR. If a protected write is attempted out of bounds, it is not executed and an error condition is set.

The data feeder registers A and B are used to hold the input data for ALU operations.

The 'registers' PNL.LTS. and PNL.SWS. in Figure 1 represents the system control panel display lights and data-entry switches. During single-cycle operation of the CPU (under system control panel control) various CPU registers (UIR, UPNT, MAR, MDR) may be displayed in the lights or altered via the switches. These functions are described in detail in the reference manual "MEMOREX 7100 SYSTEM CONTROL PANEL", by A. Hemel. The panel switches may also be read as sources by micro-instructions.

2.3 7100 CPU ROMS

The 7100 CPU data flow contains three read-only memories (ROMS), the UROM, CROM, and DROM.

The UROM is 8192 (max) 16-bit words. This is the memory which holds the micro-instructions which control the 7100. Note that 13 bit addresses are required to address the span of the memory (8K).

The CROM is a smaller read-only memory used to hold 16-bit constants. Certain micro-instructions may specifiy an 8-bit CROM (constant ROM) address and thus select one of CROM's 256 constants for gating onto the bus to a destination.

The DROM (decode ROM) is a small 256 by 13-bit read only memory used to decode the 8-bit OP field of the PIR to obtain a 13-bit jump address.

2.4 7100 RAM REGISTERS

The 7100 CPU contains 80 registers (16-bit) in a small Random Access Memory (RAM), as indicated in Figure 1.

Micro-instructions may address these RAM registers and use them as sources or destinations of data to/from the bus. Such micro-instructions encode the 'row' of the RAM directly into their SOURCE or DEST field code. This selects one of the 16 rows of the RAM for addressing. The 'column' (one of 16) is selected by the data (4-bits) in the P or X register at the time the RAM is accessed by a micro-instruction. If the high order bit in the (5-bit) X register is 0, then P is used for RAM column selection. If the high order X bit is 1, then X is used.

Of the 256 possible RAM addresses (16 rows by 16 columns), only 80 actually contain CPU registers. Certain of the remaining addresses correspond to registers or command codes for devices external to the CPU (I/O).

This organization of the RAM register file enables the CPU to communicate with I/O devices in the same simple manner in which it uses its own internal registers. It also enables the "state" of the CPU (corresponding to the value in P) to be switched under control of a Priority-Interrupt System. These functions are described in detail in the reference manual "MEMOREX 7100 I/O SYSTEM".

3.0 7100 MICRO-INSTRUCTION SET

The 7100 CPU executes 16-bit micro-instructions. The first 3-bits of each micro-inst encode its FORMAT. Table 1 lists the various FORMATS of the micro-instruction set. Seven of the eight possible are currently defined.

The micro-instruction set logically groups into 3 basic types of instructions:

(i) ALU operations. FORMATS: ALU
(ii) GATING operations. FORMATS: GSD, GCD, GID
(iii) BRANCH operations. FORMATS: SCB, ACB, UCB

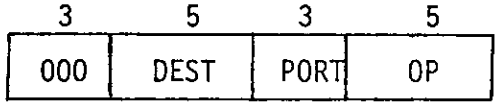
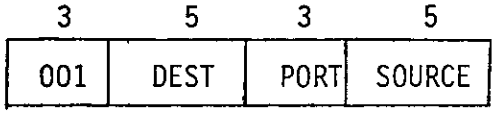
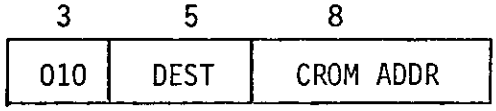
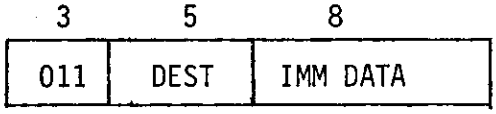
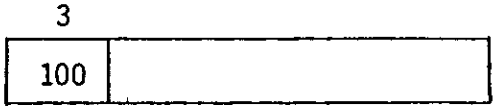
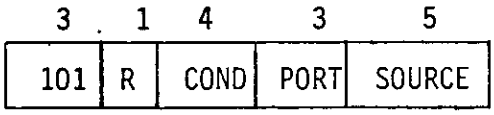
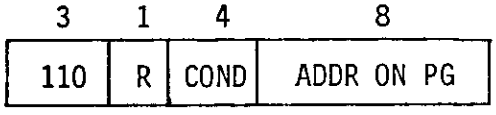
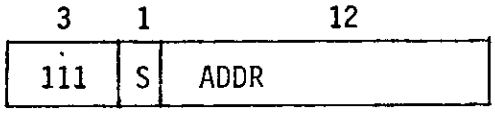
The ALU type of micro-instruction specifies that the ALU operation encoded into its OP field be performed on the data in feeder registers A and B with the result bused to a specified DEST register.

The GATING type of micro-instruction specifies a SOURCE for data and a DEST to which that data is to be bused. In the GSD FORMAT, the source is an addressed CPU register. In the GCD FORMAT, the source is an addressed CROM constant. In the GID FORMAT, the source is 8-bits of immediate data in the micro-instruction itself (R.J. on bus).

The BRANCH type of micro-instruction may alter the sequence of micro-instruction execution by specifying some address for the next instruction other than the incremented current address. In the case of the UCB FORMAT, the BRANCH is "Unconditional" and is always taken.

TABLE 1

7100 MICRO-INST SET FORMATS

MNEM	FORMAT	DESCRIPTION
ALU		ALU 'OP' performed on feeder REGS 'A' and 'B', result gated to destination.
GSD		Gate source contents to destination.
GCD		Gate the 16-bit constant at 'CROM ADDR' to destination.
GID		Gate IMM data to destination (Right justified, zero fill).
—		Unused.
SCB		Conditional branch to ADDR contained in 'SOURCE' if 'COND' true, and R = 0. BR on 'COND' false if R = 1.
ACB		Conditional branch to ADDR on current page if 'COND' true, and R = 0. BR on 'COND' false if R = 1.
UCB		Unconditional branch to ADDR in current 4K. If S = 0, then 'BRA'. If S = 1, then 'BSR' (UPNT + 1 gated to specific implied 'USRT' REG).

In that case the micro-instruction contains a 12 bit branch address for the next instruction (within the current half of the UROM as specified by the high order bit of UPNT). A special subroutine bit is also encoded in the UCB FORMAT. If it is one, then the processor saves the UPNT register in a specific register in the RAM for use later as a sub-routine return.

The ACB and SCB FORMATS are "conditional branches". For these instructions to alter micro-instruction sequencing to their specified addresses, the indicated condition must be true (if R = 0) or false (if R = 1). ACB, if taken, causes the next instruction to be fetched at the address on the current 256 word "page" in the UROM given by the 8-bit "ADDR ON PG" encoded in the ACB micro-instruction. An SCB type BRANCH, if taken, specifies as the branch address the contents of the encoded SOURCE register.

Certain of these basic ALU, GATING, and BRANCH type micro-instructions have an additional "PORT" field. This 3-bit field in these instructions encodes memory initiation and control functions. The specific functions are described in Section 3.3.

3.1 SOURCE AND DEST FIELD CODES

This section describes the encoding of the SOURCE and DEST fields of the 7100 micro-instruction set. The locations of the fields in the micro-instructions are shown in Table 1, while the CODES and MNEMONICS are listed in Table 2.

All the SOURCE and DEST field codes are encoded into 5-bits. If the high order bit is zero, then the SOURCE or DEST is in the RAM registers and the low-order four bits form the "row" address for the RAM register. RAM registers may be either SOURCE's or DEST's, although timing restrictions preclude gating from one RAM register to another RAM register in the same micro-cycle.

If the high order bit of the code is one, then one of the CPU discrete registers or some variation is indicated (See Table 2).

TABLE 2

SOURCE AND DEST FIELD CODES

CODE	SOURCE	DESTINATION
00	REGO	REGO
⋮	⋮	⋮
0F	REGF	REGF
10	X	X
11	PIR	PIR
12	MDR	MDR
13	SMDR	SMDR
14	UPNT	PROT
15	R1I	R1I
16	R2I	R2I
17	SWS	A
18	PIRD	B
19	P	UIR
1A	MAR	MAR
1B	CST	AMAR
1C	UCR	UCR
1D	———	———
1E	———	———
1F	ZRO	NOP

Certain of the CPU discrete registers may be either micro-code SOURCES or DESTS. These are:X,PIR, MAR, MDR, and UCR. Note that UCR, while it holds 10 bits, has only its left 8 bits gated to/from the bus when used as a SOURCE or DEST. In both cases the 8 bits are left justified (L. J.) on the bus. Also, when UCR is used as a SOURCE, the right-most 8 bits of the bus are zero.

Some CPU registers may only be used as a source. These are UPNT and P. UPNT is used as an implied destination for the branch address during the execution of a SCB type branch. Thus, the Figure 1 shows a data path from the bus into UPNT. However, UPNT may not be directly encoded as a DEST.

The 'SWS' source code is used to read the system control panel data entry switches.

The 'ZRO' source code places all zero's on the bus.

The mnemonic 'PIRD', indicating PIR decode, appears as a SOURCE only. This code is used to gate the DROM decode of the OP field (8 high order bits) of the PIR as a SOURCE.

The mnemonic 'CST' appears as a SOURCE only. This is used as a special function in an otherwise NOP instruction (PORT may be used) to indicate that the P register may be updated to the new value specified by the priority-interrupt system (changes CPU 'state').

Some CPU registers may only be used as destinations. These are PROT, UIR, A, B, and AMAR. AMAR is a special DEST only code which enables the bus data to be simultaneously gated to two destination registers, A and MAR. A UIR DEST OR's the bus with the next fetched micro-instruction.

Note that there is a NOP DEST code. This code disables any gating to destinations.

'SMDR' is a special use of MDR. The right-justified byte on the bus is gated to/from the MDR half indicated by MAR_{15} (0 = left, 1 = right).

R1I, R2I cause the indirect use of the R1 or R2 fields (3 bits) of the PIR as SOURCE/DEST codes. This enables indirect use of REG0 - REG7 in the RAM according to value of R1 or R2 field in the PIR.

3.2 ALU OPERATIONS (R. Stallman)

The ALU operations consist of three basic types: arithmetic functions; logic functions; and word, byte, nybl manipulations. All of these ops are performed on operands in the A and B registers. The ALU op codes are listed in table 3.

Bits 0-7 of the condition register (UCR) are dedicated to storing the status of the applicable ALU ops. The assignment of each bit location in the UCR (condition register) is diagrammed in table 2.

Arithmetic Functions

Arithmetic operations performed by the ALU are either binary or decimal. These two general categories differ from each other substantially. Binary ops: require one microcycle for their execution; involve two-16 bit operands and require binary numbers to be represented in 2's complement notation.

Decimal ops: require two micro-cycles for their execution; involve 8 bit (two decimal digits) operands and outputs; and require digits to be represented in packed BCD (8421) format.

Binary Arithmetic Ops

Signed binary values are represented by 15 bits of magnitude information with the 16th bit (MSB) representing the sign. Negative numbers have a 1 in the sign position, and positive numbers, a 0. Numbers are also in 2's complement notation. Negative numbers are formed by deriving the 1's complement of the positive representation and adding a 1 to the least significant bit:

+ 17 = 010001

- 17 = 101111

In 2's complement addition, the 16 bit numbers (including sign) are added and the carry from the most significant (sign bit) is always ignored. Subtraction is performed by adding the 1's complement of B to A, and adding a 1 to the least significant bit position. In either addition or subtraction, the resultant answer is in 2's complement form, and no correction need be made. The overflow bit in the UCR is set according to the following:

ADDITION:

(BAD+BADX) $\left[(A_0=0 \cdot B_0=0 \cdot ALU_0=1)+(A_0=1 \cdot B_0=1 \cdot ALU_0=0)\right]$

SUBTRACTION

(BSB+BSBX) $\left[(A_0=1 \cdot B_0=0 \cdot ALU_0=0)+(A_0=0 \cdot B_0=1 \cdot ALU_0=1)\right]$

A_0 = Bit zero of the A Register (Sign Bit)
B_0 = Bit zero of the B Register (Sign Bit)
ALU_0 = Bit zero of the ALU Register (Sign Bit)

ADD, SUB

ADD and SUB are ops intended for use with address arithmetic. When executed, they do not change the condition register. Carries or borrows from the condition register into the ALU are inhibited during these ops. SUB is a 16-Bit binary subtraction of B from A, producing no borrow or overflow in the condition register. ADD is a 16 bit binary addition of A and B, also producing no carry or overflow in the condition register.

BAD, BSB

BAD and BSB are ops intended for use in single precision 16-bit binary addition or subtraction. Carries or borrows into the ALU, from the condition register, are not enabled. Carries or borrows and overflow resulting from these ops are, however, stored in the condition register.

BADX, BSBX

BADX and BSBX are ops intended for use in multiple precision 16-bit binary additions or subtractions. Carries or borrows initially in the condition register are included in the addition or subtraction being performed and carries, borrows, or overflows resulting during the execution are stored in the condition register.

The following examples use 6 bits rather than 16 for simplicity:

Single-Precision ADDITION (ADD,BAD)

```
-10      110110      +10      001010
 - 3   +  111101       - 3   +  111101
-13      1 110011      + 7      1 000111
           ↑                      ↑
           └Carry-store           └Carry-store
            in UCR if BAD          in UCR if BAD
```

Single-Precision SUBTRACTION (SUB,BSB)

```
 +10        001010
-(-3)     + 000010 ←—1's complement of (-3) } Done by
 +13      +      1 ←—Forced carry-in        } ALU
            001101
            ↑
            └— No borrow - 0 in UCR if BSB

 +10        001010
-(+3)     + 111100 ←—1's complement of (+3) } Done by
 + 7      +      1 ←—Forced carry-in        } ALU
         1  000111
         ↑
         └— Borrow-store in UCR if BSB
```

Multiple-precision additions or subtractions involve binary numbers larger than the capacity of the 16-bit A and B registers. If a 32 bit number is to be added to, or subtracted from, another 32 bit number, two passes through the ALU must be made. The first pass would be a BAD or BSB as illustrated, the second pass must include the carry or borrow produced in the first pass (BADX or BSBX). The

first pass (least significant word) would not include a sign bit. The following example uses 12 bits instead of 32 for simplicity.

MULTIPLE-PRECISION ADDITION (BAD & BADX)

```
                   2nd pass | 1st pass
                            |
   1200            010010 | 110000 <---- A Reg.
 +  301           +000100 | 101101 <---- B Reg.
 ------           ----------------
   1501                     |
```

1st pass through ALU (BAD)

```
       110000
      +101101
     --------
     1 011101
     ^
     |____ Carry-stored in UCR
```

2nd pass through ALU (BADX)

```
       010010
      +000100
      +     1 <---- carry from UCR from previous pass.
      -------
       010111
     ^
     |____ No carry - 0 to UCR
```

Resultant answer: 010111011101 (1501)

MULTIPLE PRECISION SUBTRACT (BSB,BSBX)

```
                   2nd pass | 1st pass
    +301           000100 | 101101 <---- A Reg.
  -(+1200)        -010010 | 110000 <---- B. Reg.
  --------        ----------------
    -899                    |
```

1st pass through ALU (BSB)

```
       101101
      +001111 <-- complement of B  }
      +     1 <-- forced carry-in  }   Done by ALU
     --------
     0 111101
     ^
     |____ No carry - 0 stored in UCR
```

2nd pass through ALU

```
       000100
      +101101 <---- complement of B }
      +     0 <---- carry from UCR  }  Done by ALU
     --------
     0 110001
     ^
     |________ No carry, 0 stored in UCR
```

RESULTANT ANSWER: 110001111101 (2's complement of -899)

DECIMAL ARITHMETIC OPS

Decimal ops are performed on a right-justified byte in the A and B register. The data is in packed decimal form.

In the zoned decimal format, a byte is required to hold each digit, 0-9. The zone portion of each byte is unused (except for the sign in the right-most byte). These zoned decimal fields can be packed into a fewer number of bytes if the zones are removed; thus a packed or unzoned decimal format. All numeric fields must have a sign, and in the zoned decimal format the first four bits (0-3) of the right-most byte are used to hold a sign. A hexadecimal C is a plus sign; and a hexadecimal D is a minus sign. In the packed format, the 4-bit sign is moved to the last four bits of the rightmost byte.

The following illustration shows the number 18,634 in a zoned format and a packed (unzoned) format. Note the difference in placement of the sign. For easier interpretation, the actual bit patterns are not shown.

ZONED

F	1	F	8	F	6	F	3	+	4

Zone Digit Zone Digit Zone Digit Zone Digit Sign Digit

PACKED

1	8	6	3	4	+

Digit Digit Digit Digit Digit Sign

The decimal digits are represented by 4-bit BCD (8421) code, therefore each pass through the ALU adds two decimal digits to two decimal digits and a possible carry from the UCR. A carry from the high-order digit is stored in the UCR.

Subtraction of decimal representations are performed using 10's complement addition. This method of subtraction by addition is somewhat analogous to using 2's complement addition when performing binary subtraction. When using 2's complement notation, binary numbers are left in 2's complement form in memory. When decimal data is processed, however, it is stored in true form. When subtracting one number from another number, by adding the 10's complement of one to the other, the addition may or may not have produced a carry. The carry, if it occurs, is dropped and the difference is in true form. If no carry occurs, the difference is in 10's complement form and must be corrected to place it in true form. No carry indicates that a larger number was subtracted from a smaller number, and thus a negative difference has resulted. To put a BCD number in 10's complement form, the 9's complement is first derived by subtracting each digit from 9 and adding 1 to the LSB. In a similar fashion, taking the 10's complement of a number in 10's complement form, places it back in true form. Where reference to BCD is made, 8421 code is implied.

DAD,DSB

DAD and DSB are ops intended for use in 8 bit BCD addition (DAD) or subtraction (DSB). Specifically, these ops are to be used for the first pass, of a series, through the ALU. That is, when the two least significant decimal digits of a number are added. Carries or borrows from the condition register are ignored during these ops, however, if a carry or borrow is produced during the execution of the op it is stored in the UCR.

DADX, DSBX

DADX and DSBX are ops used in 8 bit BCD addition (DADX) or subtraction (DSBX). These ops are used after the first pass through the ALU, and include a carry or borrow from the UCR that the first pass may have produced. If a carry or borrow is produced during the execution of these ops, it is stored in the UCR.

The following are examples of the decimal Add ops:

(It should be noted that these are BCD additions, not Binary)

```
                        2nd pass | 1st pass
  0056                  0000 0000 | 0101 0110
+ 1234                  0001 0010 | 0011 0100
  1290
```

1st pass through ALU: (DAD)

```
                        0 <-- Carry inhibited by ALU
  56                  01010110 }
 +34                  00110100 } ALU performs BCD addition
  90                0 10010000
                      ^
                      |
                      +-- No carry, 0 stored in UCR
```

2nd pass through ALU (DADX)

```
                        0 <-- Carry from UCR resulting
  00                  00000000    from previous pass
 +12                  00010010
  12                0 00010010
                      ^
                      |
                      +-- No carry, 0 stored in UCR
```

Resulting Answer: 0001 0010 1001 0000 (1290)

```
                        2nd pass | 1st pass
 -9999                  1001 1001 | 1001 1001
 +9999                  1001 1001 | 1001 1001
```

1st pass through ALU (DAD)

```
                        0 <-- Carry from UCR inhibited
  99                 10011001 }
 +99                 10011001 } BCD add done by ALU
1 98              1 10011000
  ^                   ^
  |                   |
  +-- Carry to        +-- Carry, 1 stored in UCR
      next decade
```

2nd pass through ALU (DADX)

```
    1 <-Carry from          1 <----Carry from UCR resulting
   99  previous    10011001       from previous pass
  +99  decade      10011001
1 -99              1 10011001
  |                  |
  Carry              ----Carry, 1 stored in UCR
```

Resulting answer: 1 1001 1001 1001 1000 19998

The following are examples of the decimal subtract ops:

```
                    2nd pass | 1st pass
  1234              0001 0010| 0011 0100
 -0056             -0000 0000| 0101 0110
 ----               ---------|---------
  1178                       |
```

1st pass through ALU (DSB)

```
   34                  00110100
  -56                 +01000011 <----9's complement of 56
1 -78                 +       1 <----Forced carry-in to get
  |                     --------       10's complement
  Borrow from       0   01111000
  next decade           |
                        ----Borrow, (no carry) - 0 stored in UCR
```

Done by ALU

2nd pass through ALU (DSBX)

```
   12                   00010010
  -00                  +10011001 <----9's complement of 00
  - 1 <--Borrow from   +        0 <----borrow from previous
  ----   previous decade --------      decade (no carry)
   11                1   00010001
                         |
                         ----No borrow, (carry) - 1 stored in UCR
```

Done by ALU

Resultant answer: 1 0001 0001 0111 1000 1178

Since there was a carry (no borrow) in this answer, it is to be ignored, and the answer considered to be in true BCD form.

```
     b                  2nd pass  | 1st pass
      0056              0000 0000 | 0101 0110
     -1234             -0001 0010 | 0011 0100
   1  8822             -----------+----------
   ↑                              |
   └─borrow
```

1st pass through ALU (DSB)

```
    56           01010110
   -34          +01100101 ←—— 9's complement of 34 ⎫  Done by
   ---           +      1 ←—— Forced carry-in to   ⎬
    22         1  00100010     get 10's complement ⎭  ALU
               ↑
               └———— No borrow (carry), 1 stored in UCR
```

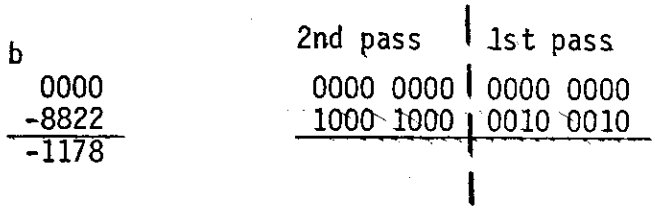
2nd pass through ALU (DSBX)

```
     b            00000000
     00          +10000111 ←—— 9's complement of 12
    -12=          +      1 ←—— Carry (no borrow) from
  1  88         0  10001000     from previous decade
  ↑             ↑
  └─borrow      └——Borrow (no carry) 0 stored in UCR
```

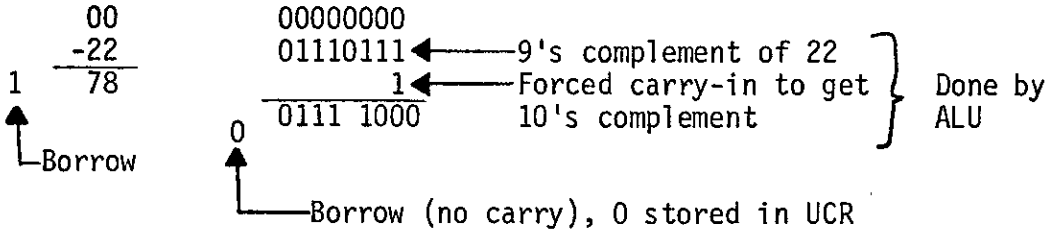
Resultant answer: 0 1000 1000 0010 0010

Since there was a borrow (no carry) in this answer, the answer is negative and is 10's complement form. It must be corrected (recomplemented). This is accomplished by taking its 10's complement, equivalent to subtracting the answer from all zeros with an implied high order borrow available.

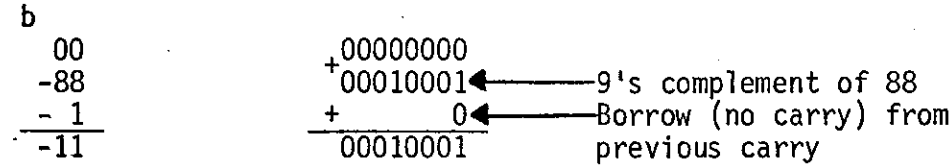
Correction (Recomplementing)



1st pass through ALU (DSB)



2nd pass through ALU (DSBX)



Resultant recomplemented answer: 0001 0001 0111 1000

It should also be remembered that this answer is negative.

Logic Functions

The logical ops include compare, right and left shifts, inclusive OR, exclusive OR, AND, and invert.

CMP

The execution of an ALU instruction with a CMP op results in a logical and an arithmetic comparison of A to B. The results of this comparison are stored in the condition register. The DEST field of the instructions is not used during this op. Bits 1-3 of the UCR reflect the results for Arithmetic compare (includes sign) of A and B. Bits 5-7 of the UCR indicate the results of a logical (maginitude of 16 bits) compare of A and B. UCR bits 0 and 4 are reset during the execution of this op.

SLO, SRO

The shift ops define 1-bit linked, end-off shifts. SLO is a 1-bit serial left shift through the 32 bits of A and B. The resulting contents of the A register are gated onto the bus. SRO is a 1-bit serial right shift through the 32 bits of A and B. The resulting contents of the B register are gated onto the bus. Bits 0-7 of the UCR are not changed during the execution of this op.

IOR

IOR is the inclusive OR (logical summation) of A and B. The results of the OR are gated onto the bus. Bits 0-7 of the UCR are not changed during the execution of this op.

XOR

XOR is the exclusive OR (modulo - 2 sum) of A and B. The results of XOR are gated onto the bus. Bits 0-7 of the UCR are not changed during the execution of this op.

AND

AND is a logical AND (logical product) of A and B. The results of AND are gated onto the bus. Bits 0-7 of the UCR are not changed during the execution of this op.

INV

INV, performs a logical complementation. This op, inverts the state of each of the 16 bits of A, onto the bus. The contents of A remain undisturbed. Bits 0-7 of the UCR do not change during the execution of this op.

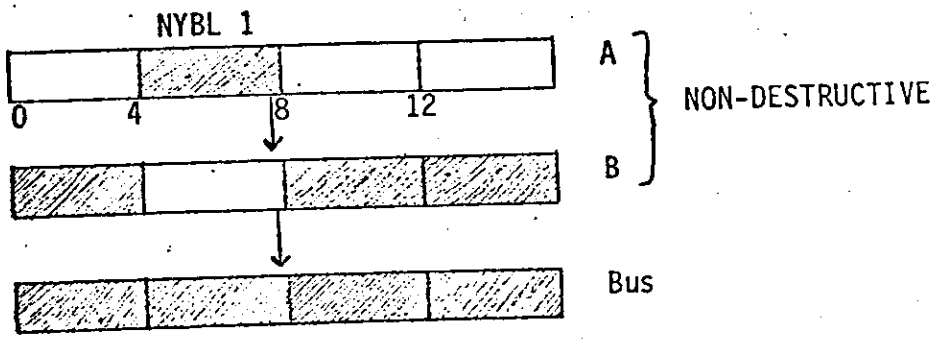
WORD, BYTE, and NYBL Manipulations:

BCB:

Gates onto the bus the combination of the right byte of A and the left byte of B. UCR bits 0-7 are not changed.

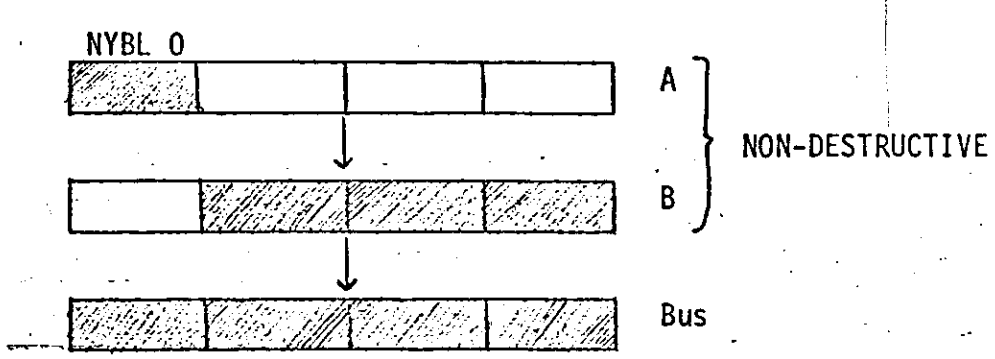
IN1:

Gates onto the bus the insertion of NYBL 1 of A into B. UCR bits 0-7 are not changed.



INZ:

Gates onto the bus the insertion of NYBL 0 of A into B. UCR bit 0-7 are not changed.



DBT

Bit test DBT is an op in which any one of the 16 bits in the A register may be tested. The location of the bit to be tested is encoded in the low-order 4 bits of the DEST field of the micro-instruction. The state of the bit tested is placed in UCR bit zero. Bits 1-7 of the UCR are reset. No data is gated on to the bus during the execution of this op.

BBT

Bit test BBT is an op identical to DBT in with the exception of the location of the address of the bit to be tested. The low-order 4 bits of the B register are encoded to hold this address. As with DBT, the results of the test are stored in Bit 0 of the UCR. Bits 1-7 of the UCR are reset. No data is gated onto the bus during the execution of this op.

AOB, BOB

The contents of the A register (AOB) or B register (BOB) are gated onto the bus. UCR bits 0-7 are not changed.

TABLE 3

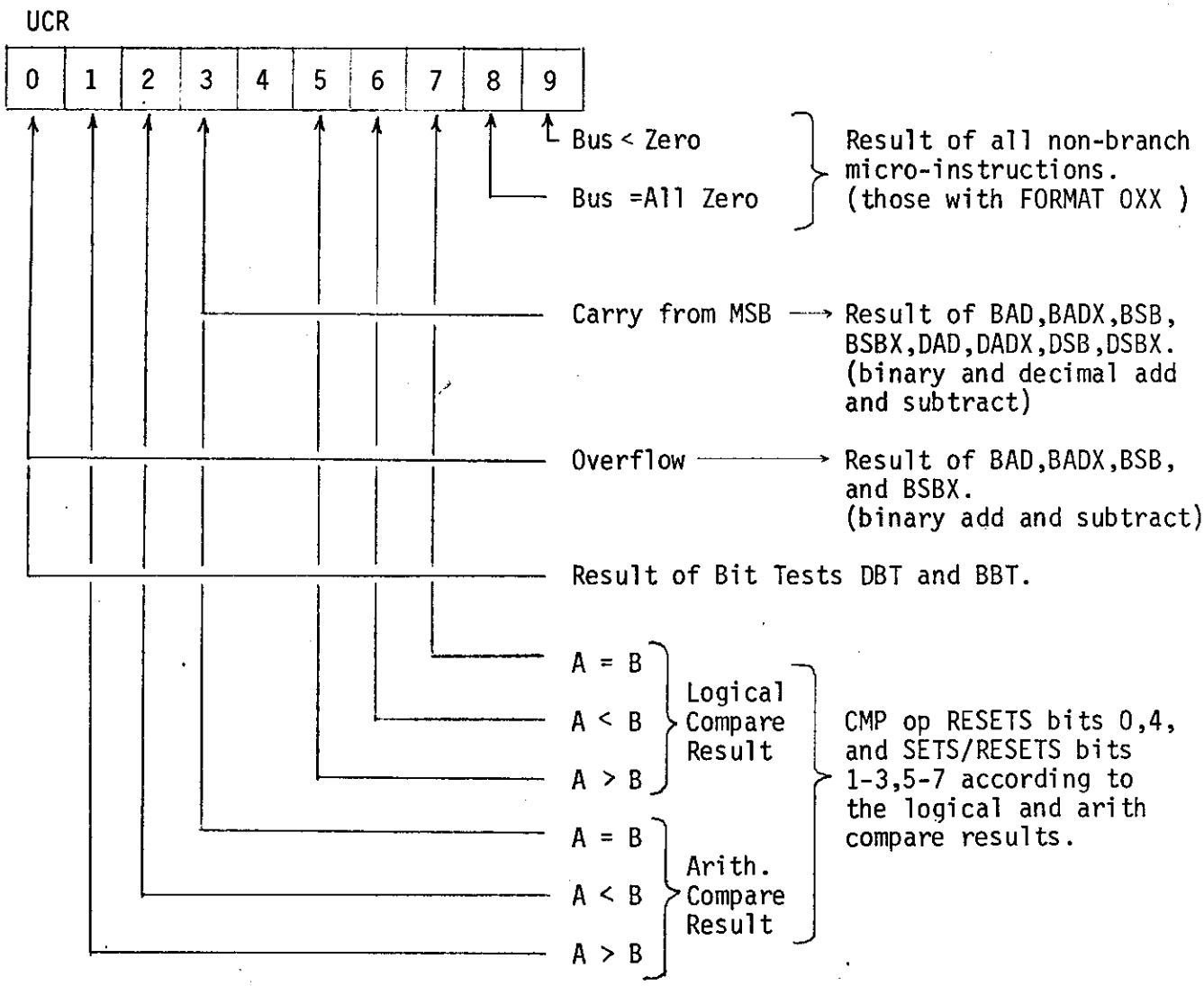
ALU OP CODES

CODE	MNEM	OPERATION
00	SLO	Shift B to A, left open, 1 bit, A onto bus.
01	DAD	BCD add, right half A to right half B, no CI, CO.
02	BADX	Add A to B, CI, CO.
03	DADX	BCD add, right half A to right half B, CI, CO.
04	CMP	Compare.
05	DSB	BCD sub, right half B from right half A, no BI, BO.
06	BSB	Subtract B from A, no BI, BO.
07	BSBX	Subtract B from A, BI, BO.
08	ADD	Add A to B, no change to UCR.
09	BAD	Add A to B, no CI, CO.
0A		
0B		
0C		
0D	DSBX	BCD sub, right half B from right half A, BI, BO.
0E	SUB	Subtract B from A, no change to UCR.
0F	BCB	Combine right byte A, left B, onto bus.
10	INV	Invert A (1's complement), onto bus.
11	SRO	Shift A to B, right open, 1 bit, B onto bus.
12		
13		
14		
15	IN1	Insert 'NYBL1' of A into B onto bus.
16	INZ	Insert 'NYBL 0' of A into B onto bus.
17		
18		
19	XOR	Exclusive OR (A, B)
1A	BOB	Gate B onto bus.
1B	IOR	Inclusive OR (A, B)
1C	DBT	Bit test. (address in DEST field)
1D	BBT	Bit test. (address in B).
1E	AND	AND (A, B)
1F	AOB	Gate A onto bus.

NOTE: CI = carry-in, CO = carry-out, BI = borrow-in, BO = borrow-out.

FIGURE 2

CONDITION REGISTER SETTING



3.3 PORT OPERATIONS

The 3-bit PORT field contained in certain types of micro-instructions is used to encode the initiation and control of CPU-memory data transfers. Table 4 lists the PORT operation codes and mnemonics.

Those PORT ops which initiate memory requests all use the MAR to hold the memory address to be read/written. The MDR holds the data for writes and receives the data during reads. During FULLWORD ops, the high-order 15 bits of MAR select the word address. During byte ops, all 16 bits of MAR are used (see below).

It should be noted that the CPU must compete for memory accesses with other devices of higher priority on the Direct Memory Access bus. Thus, a PORT memory initiation may not result in completion within a fixed number of micro-cycles. The actual completion delay time is a function of the interference on the DMA. Interlocking in the micro-code is accomplished by the use of the HDM port operation. This 'HOLD-ON-MEMORY' op is used following a memory operation to hold up micro-code execution until the memory operation is completed. Refer to section 5.0 for memory operation micro-programming techniques.

FMR, FMW are the port ops for READ and WRITE of a full 16-bit memory word. BMW is the op for a byte memory write with the left/right byte written depending on the low-order bit of MAR (0 = left, 1 = right). FPW and BPW are the same as FMW, BMW except that the protect logic is enabled, and upper(U) and lower(L) bounds are placed on MAR(0-7).

TSR (test read) is a special PORT op which performs the same function as FMR, but additionally raises a control line to the memory system which causes the CPU to have top priority during the following memory cycle. This enables the CPU to perform a READ followed immediately (locking out any interference) by a WRITE, and thus perform a "TEST and SET" programming function.

TABLE 4

PORT OPERATIONS

CODE	MNEM	DESCRIPTION
000	NOP	No operation
001	FMR	Full memory word read
010	BMW	Byte memory write. If MAR_{15} = 0, left byte. If MAR_{15} = 1, right byte.
011	FMW	Full memory word write.
100	HDM	Hold on memory.
101	TSR	Test read.
110	BPW	Protected BMW.
111	FPW	Protected FMW.

TABLE 5

BRANCH CONDITION CODES

	MNEMONICS	
CODE	R = 0	R = 1
0	TRU	NTRU
1	I1Z	NI1Z
2	I2Z	NI2Z
3	R1Z	NR1Z
4	R2Z	NR2Z
5	ZOB	NZOB
6	NOB	NNOB
7	ERR	NERR
8	UCZ	NUCZ
9	UC1	NUC1
A	UC2	NUC2
B	UC3	NUC3
C	INT	NINT
D	———	———
E	———	———
F	———	———

TABLE 6

SUMMARY OF MICRO-INSTRUCTION FIELD CODES

CODE	SOURCE	DEST	OP	PORT	COND	$\overline{\text{COND}}$
0	REG0	REG0	SL0	NOP	TRU	NTRU
1	REG1	REG1	DAD	FMR	I1Z	NI1Z
2	REG2	REG2	BADX	BMW	I2Z	NI2Z
3	REG3	REG3	DADX	FMW	R1Z	NR1Z
4	REG4	REG4	CMP	HDM	R2Z	NR2Z
5	REG5	REG5	DSB	TSR	ZOB	NZOB
6	REG6	REG6	BSB	BPW	NOB	NNOB
7	REG7	REG7	BSBX	FPW	ERR	NERR
8	REG8	REG8	ADD		UCZ	NUCZ
9	REG9	REG9	BAD		UC1	NUC1
A	REGA	REGA	——		UC2	NUC2
B	REGB	REGB	——		UC3	NUC3
C	REGC	REGC	——		INT	NINT
D	REGD	REGD	DSBX		——	——
E	REGE	REGE	SUB		——	——
F	REGF	REGF	BCB		——	——
10	X	X	INV			
11	PIR	PIR	SRO			
12	MDR	MDR	——			
13	SMDR	SMDR	——			
14	UPNT	PROT	——			
15	R1I	R1I	IN1			
16	R2I	R2I	INZ			
17	SWS	A	——			
18	PIRD	B	——			
19	P	UIR	XOR			
1A	MAR	MAR	BOB			
1B	CST	AMAR	IOR			
1C	UCR	UCR	DBT			
1D	——	——	BBT			
1E	——	——	AND			
1F	ZRO	NOP	AOB			

3.4 BRANCH CONDITION CODES

Table 5 lists the condition codes and mnemonics for the BRANCH CONDITIONS encoded in and tested by the SCB and ACB conditional-branch micro-instructions. Mnemonics are listed for both R=0 and R=1 for each condition code. If R=1 (mnem. prefix N), the inverse condition is tested.

I1Z, I2Z, R1Z, R2Z conditions are true if the corresponding field in the PIR=0.

The conditions UCZ, UC1, ---, UC3 are true if the corresponding bit in the UCR=1.

ZOB condition is true if the previous non-branch micro-instruction bused ZERO to a DEST. NOB condition is true if the previous non-branch micro-instruction bused negative data to a DEST.

The INT condition is true if the output of the PRIORITY/INTERRUPT SYSTEM is not equal to the current value of the P register.

The TRU condition always results in a branch.

The ERR condition branches if any error has been detected. For now, only a PROTECT error is detected.

4.0 MICRO-ASSEMBLY LANGUAGE

The 7100 MICRO-ASSEMBLY LANGUAGE is used to write symbolic micro-programs for the MEMOREX 7100 processor. Programs written in this language are processed by the 7100 Micro-Program Assembler to produce listing and object files.

The following sections describe the structure of the language. Many of the mnemonics and symbols used to actually construct symbolic micro-instructions are not listed in these sections. Those mnemonics which referenced actual 7100 CPU registers and operations are listed in the earlier chapters of this manual.

The input file for the Micro-Program Assembler consists of 80 character records, usually punched cards. There are four (4) types of input cards: (1) comment card, (2) assembly control statement card, (3) define constant card, and (4) micro-instruction card.

4.1 COMMENT CARDS

Comment cards have an * in column 1. These cards are listed on the output listing but have no effect on the object code produced. None of the restrictions on the remaining card types apply to comment cards. Columns 2-80 may contain any alphanumeric or special characters.

4.2 GENERAL CARD FORMAT

The following specifications apply to all card types except comment cards.

Blanks are in general ignored, unless some explicit rule is mentioned for blanks for a specific statement type. The scan and processing of a card will stop with the last card column, or with the first semicolon. Any characters on the card past the semicolon will not be processed, but will appear on the output listing. Thus comments may be inserted on any card.

The general card format is:

Columns 1-5:	Label Field
Column 6:	Blank
Columns 7-80:	Dependant on card type.

4.3 LABELS

A label consists of 1 to 5 alphanumeric characters in the label field of a card.

4.4 VALUES

A 'value' is one or more 'value elements' separated by the delimiters + or -. A value element is any of the following:

1) Label
2) X 'xxxx' a hex constant
3) D 'dddd' a decimal constant
4) * the present location counter

4.5 ASSEMBLY CONTROL STATEMENTS

There are three assembly control statement ops: (1) ORG, (2) EQU, and (3) END. For these statements card columns 7-9 must contain the op, and columns 10-80 may contain a value field.

The ORG statement controls the setting of the program counter. A label may be used. The program counter is set to the 'value' of the value field. Any labels in the value field must be pre-defined (appear before the ORG statement). A label on the ORG card will be assigned the value of the value field.

The EQU statement is used to define a symbol. A label must be used or the EQU statement will be ignored. The EQU card causes the assignment of the 'value' of its value field to its label. Any labels appearing in the value field must be pre-defined.

The END card signals the end of the input deck. The label and value fields are ignored.

4.6 DEFINE CONSTANT STATEMENTS

In addition to the UROM memory which holds micro-instructions, the 7100 CPU has a CROM memory which holds up to 256 16-bit constants which may be referred to by the GCD type micro-instructions.

The output of the assembler, in addition to micro-instruction object code, must therefore contain an object file to initialize the CROM.

This file will be constructed during the assembly process by adding a new constant on each use of a GCD type micro-instruction with a previously not encountered CROM literal value field, and on each use of a DEFINE CONSTANT (DC) statement which specifies a previously not encountered CROM value field (if previously encountered, then an equate of CROM labels is established).

The DEFINE CONSTANT statement contains the DC op in columns 7-8 and a value field in columns 10-80. The DC statement may have a label. The label field symbol, if present, causes the assignment of the assembled CROM address and DC statement value to the label field symbol.

4.7 MICRO-INSTRUCTION STATEMENTS

If the assembler finds that a card is not a comment card, assembly control card, or define constant card, it assumes that it is a micro-instruction statement card.

Each micro-instruction statement results in the generation of one micro-instruction in the output object file. The micro-instruction statement may have a label in columns 1-5, while columns 7-80 are a free form field area into which the micro-instruction is symbolically encoded.

All micro-instruction statements have the same basic free form field structure. The FORMAT type of the instruction is implied by the specific 7100 MNEMONIC symbols used by the instruction. The basic symbolic micro-instruction structure is:

```
        X = Y, modifier;

where   X = primitive
or      X = primitive.modifier
```

Following are simple examples of the encoding of micro-instructions of each format type, which will serve to approximately define the language at this time. At a later date when the 7100 architecture is more fixed, a more formal description will be constructed.

The following examples should be sufficient to provide guidelines for the construction of sample emulation programs, etc., and to indicate the level of logic required in the micro-program assembler itself.

FORMAT	EXAMPLES	FUNCTION ENCODED
ALU:	MAR = ADD;	A + B ⟶ MAR
	14 = SUB, FMR;	A - B ⟶ REGE, also FMR
GSD:	PIR = MDR;	MDR ⟶ PIR
	15 = MDR, XRA;	MDR ⟶ REGF, with X rather than P used for col. addr.
GCD:	MAR = C (X'40');	The constant X'40' from the CROM ⟶ MAR.
	MAR = C (LABEL);	The constant value (assgnd to LABEL) from the CROM ⟶ MAR.
GID:	A = I (X'F1');	X'F1' ⟶ A
	MDR = I (CHAR);	VALUE (CHAR) ⟶ MDR
SCB:	BRA.UCZ = 15, FMW;	Branch if UCR_0 = 1 to UROM ADDR in REGF. Also FMW.
ACB:	BRA.I1Z = *+D'3';	Branch if I1 = 0 to current location plus 3_{10}
UCB:	BRA = TEST;	Branch to location given by value of label 'TEST'
	BSR = TEST;	Same, but store UPNT return into assigned register in RAM.

NOTE: If PORT modifier is used alone, then prefix with a comma. For example, the following symbolic instruction assembles into a GSD format with HDM Port field and NOP DEST field:

, HDM;

5.0 MICRO-PROGRAMMING TECHNIQUES

In order to properly and efficiently micro-program the 7100, certain basic techniques, restrictions, and timing information must be understood. This chapter provides such information.

5.1 SOURCE-DESTINATION RESTRICTIONS

A number of restrictions exist on the use of sources and destinations for ALU, Gating, and Branch type operations.

An important restriction is that any one micro-instruction may make only one access (either SOURCE or DEST) to the RAM register file.

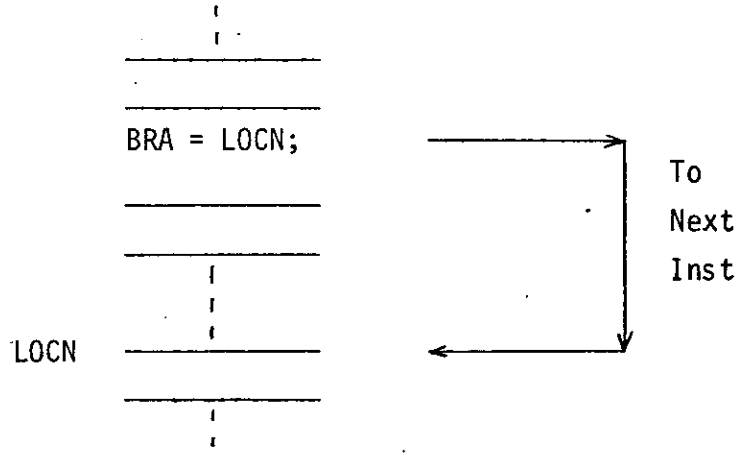
A large class of restrictions is implied by the data flow (Figure 1) and table of source and destination codes (Table 2): a number of the CPU registers may serve as only a SOURCE or DEST but not both. For example UPNT may only be a SOURCE (see Table 2) even though Figure 1. shows an input from the bus into UPNT (that path is used only by the SCB type instructions which have UPNT as an implied DEST).

Caution must be used in gatings from SOURCE's to DEST's when dissimilar width registers are involved. Justifications and unfilled bit positions must be verified (some of these have not yet been specified in this manual).

5.2 SCB BRANCH TIMING

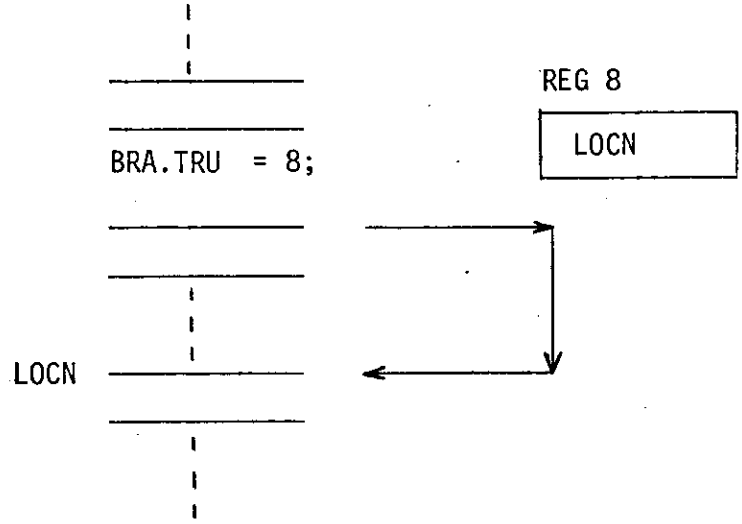
The ACB and UCB format type branch instructions function completely within one micro-instruction cycle. In other words, as expected we find the next instruction (if the branch is taken) is fetched from the specified branch address location in UROM:

EXAMPLE:



However, due to the time required to obtain the branch address from a source, the SCB format type branch actually functions one cycle after the SCB itself. In other words, the instruction following the SCB is always executed. The next instruction after that is at the branch address obtained from the source if the branch is taken:

EXAMPLE:



The SCB op enables branching across the entire 8K UROM span because a 13-bit address is loaded into UPNT from the bus to effect the branch. Note that an SCB branch followed by an ACB (both testing the same condition) enables a conditional branch to an 8-bit address specified by the ACB on the 'page' specified by the SCB.

5.3 MEMORY PORT OPERATION TIMING

Various memory port ops are described in Section 3.3, where it was noted that, due to possible interference on the DMA bus, a memory op initiated by a CPU port op does not necessarily complete within a predetermined time.

Thus the micro-code must interlock memory access initiation with "Hold-on-Memory" port ops so that completion of the access is insured before using the result of a READ or overwriting MAR or MDR for a write.

A "HOLD-ON-MEMORY" (HDM) port op in a micro-instruction causes subsequent micro-instruction execution to be inhibited if a CPU memory request is up and not yet completed. The micro-instruction containing the HDM is always completely executed before the hold takes effect.

The 7100 micro-instruction cycle is 400 ns. The 7100 main memory cycle is 1.2 μs. Thus there are three (3) micro-cycles per memory cycle.

The concept of the Hold-on-Memory (HDM) interlocking micro-instruction, and the timings which result if interference is present is displayed in Figure 3, EXAMPLE 1.

In this example, the CPU requests a memory read at the same micro-cycle as a higher priority device on the DMA. Thus, when the subsequent CPU HDM micro-instruction is executed, the CPU enters a HOLD state and micro-instruction execution is disabled while the memory request remains up.

In memory cycle 2 of example 1, another higher priority (than CPU) request is serviced and the CPU remains in HOLD.

Finally, in memory cycle 3, the CPU memory read is serviced, and the CPU leaves HOLD and its request line falls.

This example displays the ratio of three (3) micro-cycles per memory cycle.

Note, however, that the memory is not restricted to accesses which are synchronous to 1.2 μs time boundaries.

The memory may be accessed on any micro-cycle in which it is not busy. This feature is termed "zero-latency" accessing. This feature will be described in more detail later in this section.

5.3.1 WRITE OPS (see FIG.3, EX.2)

A main memory WRITE operation is requested by the CPU when any of the PORT codes FMW, BMW, FPW, FBW are decoded and executed by the CPU.

The MAR and MDR must be loaded with valid data before (or by) the micro-instruction which initiates the write.

Following the write micro-instruction and prior to and including the interlocking HDM, neither MAR nor MDR should be changed.

Following the HDM micro-instruction, the write has completed and MAR and MDR may be changed.

```
Example:    ———————, FMW;     initiates WRITE
            ———————;          any OP
            ———————, HDM;     interlocks till WRITE complete
            ———————;          MAR, MDR may now be changed
```

5.3.2 READ OPS (see FIG.3, EX.2)

A main memory READ operation is requested by the CPU when the port codes FMR or TSR are decoded and executed by the CPU.

The MAR must be loaded with valid data before (or by) the FMR or TSR micro-instruction. Both MAR and MDR must be held fixed prior to and including the interlocking HDM micro-instruction.

The new MDR value is available in the instruction following the HDM.

```
Example:    ———————, FMR;     initiates READ
            ———————;          any OP
            ———————, HDM;     interlocks till READ complete
             Y = MDR;         new MDR value available for use
```

5.4 DECIMAL ALU OP TIMING

The decimal ALU operations DAD,DSB,DADX,DSBX, require two micro-cycles for their execution (see section 3.2).

The preferred technique for micro-coding these operations to insure two micro-cycle execution timings is to consecutively execute two identical micro-instructions for each decimal ALU op to be encoded. The first micro-instruction of the identical pair will initiate the execution of the op, and the second will place valid data onto the bus to the selected destination.

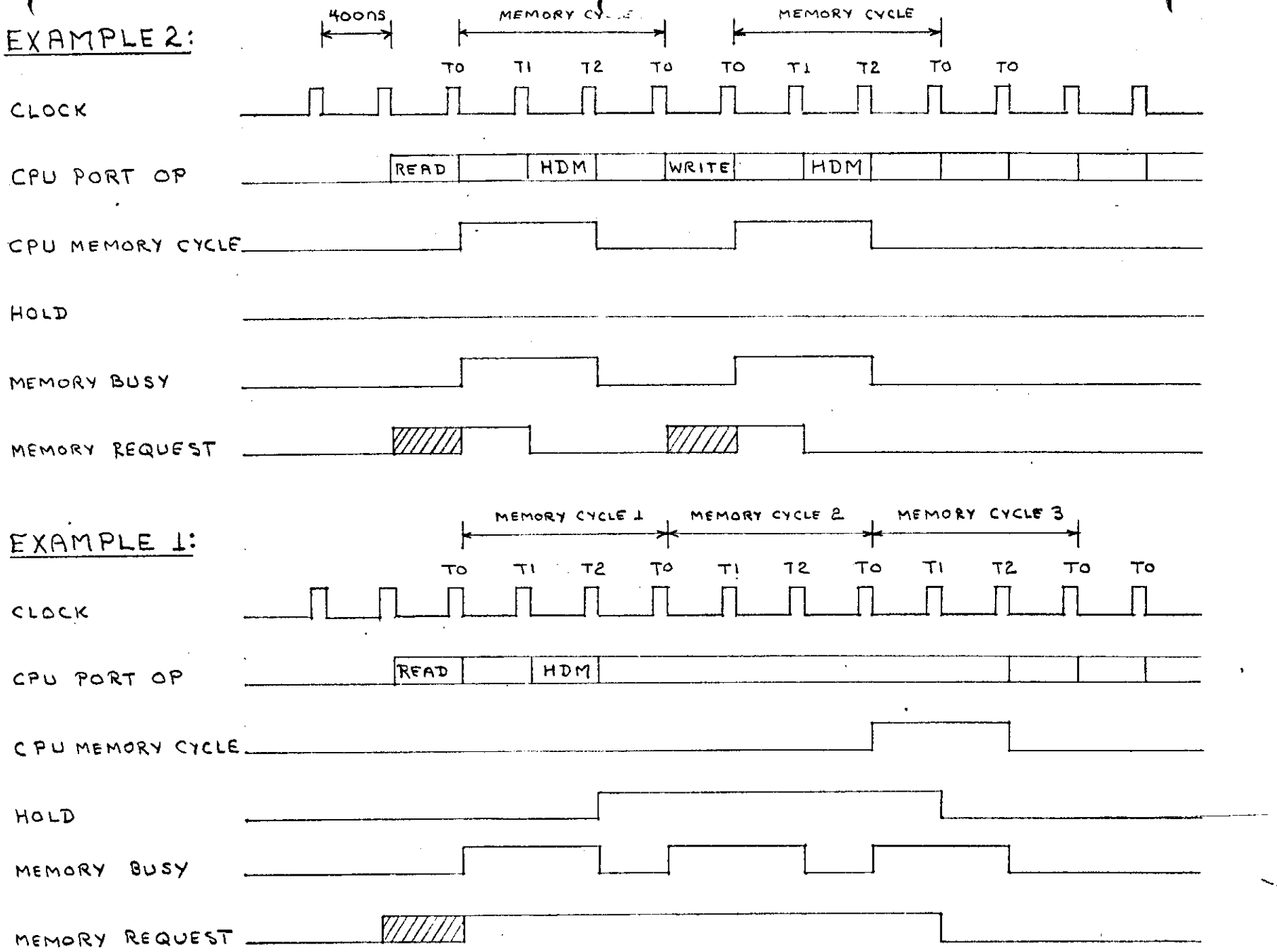


FIGURE 3. CPU-MEMORY PORT TIMING

R. W. Yee
6-15-72