S L P - Soft Logic Processor

(An LSI Micro controller Chip (MC2))

Jerry Stolle
Bert Forbes
John Figueroa
Lam Dang

Revised October 1, 1975

INTRODUCTION

The Soft Logic Processor (SLP) is a high performance, low power consumption microcomputer primarily intended for controller applications replacing existing random logic SSI/MSI solutions. It offers a 1 usec. typical instruction cycle time to execute any of the fixed width, 16 bit, instructions which can process one, four, eight or sixteen bit fields. The instruction set has been optimized for efficient operations performed directly on I/O registers. A variety of techniques have been included for logical decision making, including direct bit testing of internal and external registers, indexed branches, priority encoder for most significant bit finder, external event synchronization, and a program status register to capture the salient characteristics of the last produced result.

Although primarily intended as a microcontroller, there are several processor provisions which should permit its adoption to a wide variety of applications. A control stack mechanism is provided for return addresses, parameters, and temporary results. A program status register collects the machine's state in one convenient place for efficient handling for interrupts. Although only two instructions move data between main store and the internal store, the addressing mechanisms provide the common data access techniques.

MICROCONTROLLER REQUIREMENTS

* ROM utilized for program storage
  - stack for return address and local temporary storage
  - literal arrays can be stored with their subroutines and share the program ROM

* High performance
  - wide instructions to decrease fetches
  - simple instruction set
  - instructions which perform operations directly on I/O registers

* Many systems do not require extensive RAM
  - on chip return address storage for first subroutine call
  - eight internal registers which can be supplemented with I/O registers which do not interface with any devices
  - limited RAM data access instructions

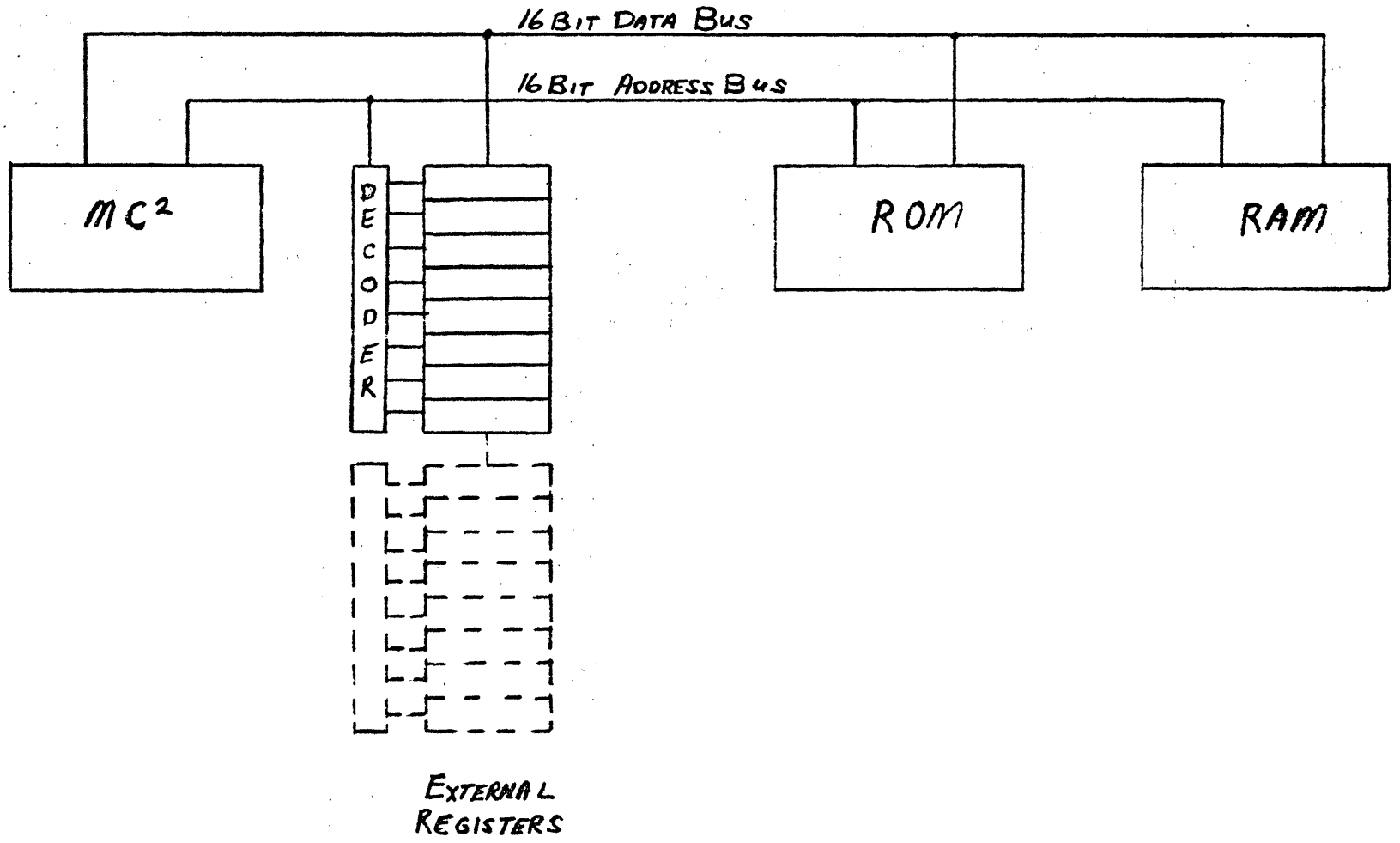
* Flexible I/O structure

* Processor synchronization with external events
  - single interrupt line with high speed identification facility
  - direct external bit testing

* Varied decision making abilities
  - indexed branch
  - non-destructive compare
  - bit test
  - program status register test

* Field oriented data access (4,8,and 16 bit widths) with provisions for multibyte arithmetic

SOFT LOGIC PROCESSOR



SYSTEM CONCEPTS

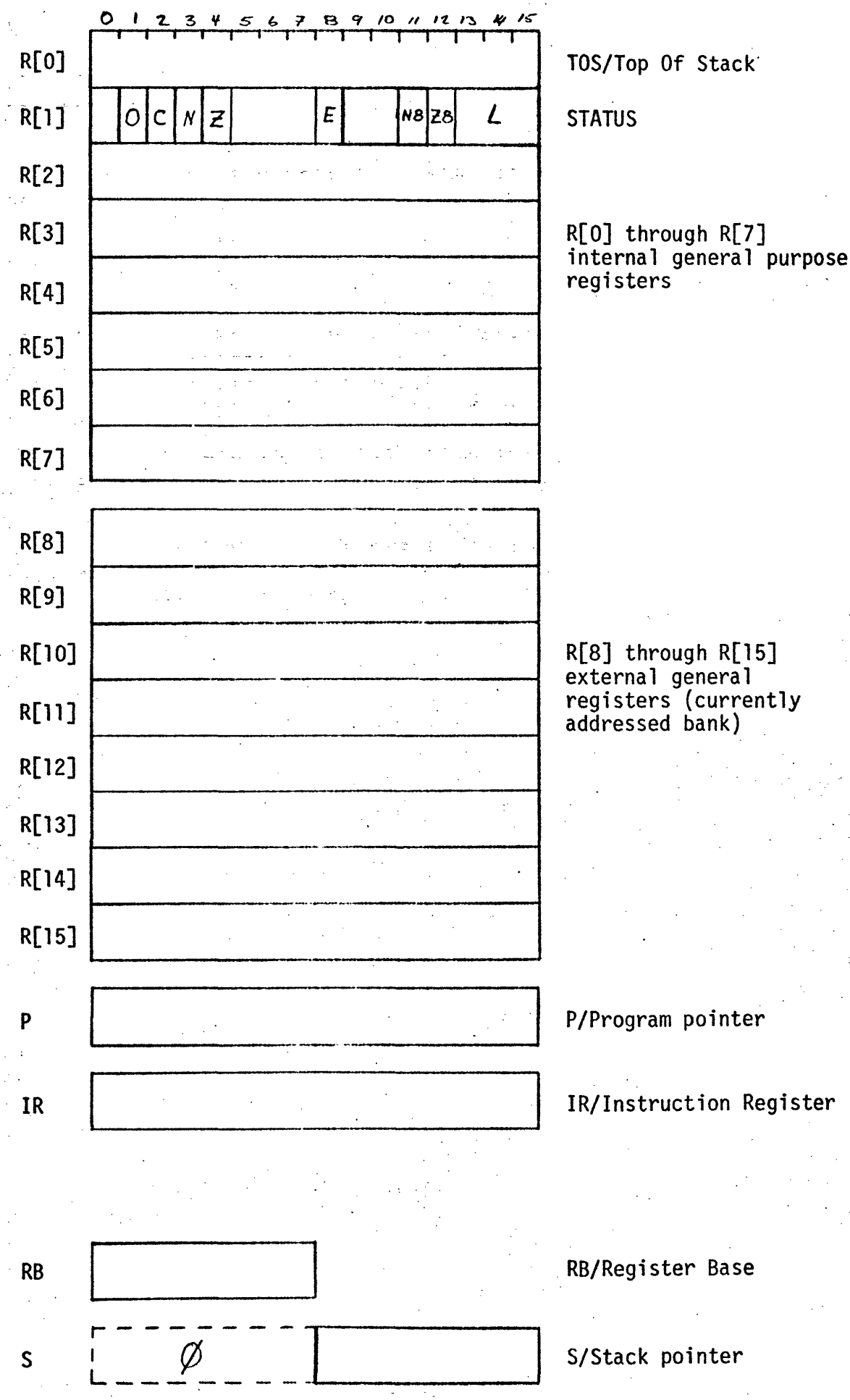
The Soft Logic Processor system consists of four major components (see figure):

1) Micro Controller Chip - MC2 which executes the instructions and is described in detail later.

2) External Registers - any speed registers may be used as all communications on the bus are asychronous. These registers are the I/O system for the SLP. They may be true registers or only static lines. They may be read-write, read only or write only as far as the MC2 is concerned. The HP-IB interface chips being developed at Data Systems will interface directly to the SLP.

3) Memory - ROM will be typically used for programs, fixed tables constants, masks, etc. RAM will be typically used for data storage, the push down stack, variables, etc. It may also be used for programs, etc. All memory may run at its own speed due to the asychronous hand shake.

4) Buses -
   - 16 bit bi-directional data bus
   - 16 bit potentially bi-directional address bus
   - GO-END handshake (I/O & memory have seperate handshakes)
   - Get Next Instruction/Idle
   - INT/INTACK
   - Write
   - Fetch Phase
   - Power On
   - System Clock


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
R[0]
R[1]
O C N Z E N8 Z8 L
R[2]
R[3]
R[4]
R[5]
R[6]
R[7]
R[8]
R[9]
R[10]
R[11]
R[12]
R[13]
R[14]
R[15]
P
IR
RB
S
Ø
TOS/Top Of Stack
STATUS
R[0] through R[7] internal general purpose registers
R[8] through R[15] external general registers (currently addressed bank)
P/Program pointer
IR/Instruction Register
RB/Register Base
S/Stack pointer

GENERAL REGISTERS

There are sixteen general purpose registers which can be explicitly referenced in register to register and literal to register instructions. Eight of these registers (R[0] through R[7]) are internal to the Micro Controller Chip (MC2). R[0] is multi-purpose. It is the top of stack for interrupts and subroutines. It will also act like a general purpose register depending on the instructions which are applied to it. R[1] is dedicated to serve as a program status register (PSR). When addressed as a g.p. destination register, status information is not set into it.

R[8] through R[15] are viewed by the processor as full 16 bit register structures which can be both read and written. These registers serve as the interfacing devices between the processor on the one side and the device on the other. Their exact characteristics (i.e., size, read only, write only, and individual field assignments) will be determined by the specific application.

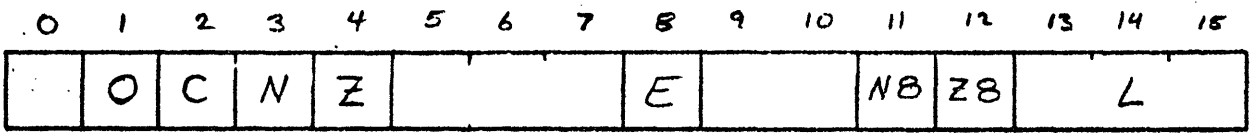
INSTRUCTION POINTER

The Instruction Pointer (P) is a sixteen bit register which indicates the next instruction to be executed after the current one is complete.

REGISTER BASE

The Register Base (RB) is an 8 bit register which specifies the current bank of external registers. Only 8 external registers can be accessed at any time, but by changing RB, up to 2048 registers may be accessed in banks of 8. RB is set to zero by Power On.

STATUS ≡ R[1]



BIT #		DESCRIPTION
0		RESERVED*
1	O	OVERFLOW FROM 16 BIT 2's COMPL. REPRESENT.
2	C	CARRY OUT OF BIT 0
3	N	NEGATIVE WORD (BIT 0 OF RESULT)
4	Z	ZERO WORD (BITS 0 THRU 15 = 0)
5, 6, 7		RESERVED*
8	E	ENABLE INTERRUPTS
9, 10		RESERVED*
11	N8	NEGATIVE (BIT 8 OF RESULT) RIGHT BYTE
12	Z8	ZERO (BITS 8 THRU 15 = 0) RIGHT BYTE
13, 14, 15	L	LEADING 1 BIT LOCATION IN LOW BYTE (PRIORITY ENCODE) 0 ⇒ BIT 8 ... 7 ⇒ BIT 15 IF Z8 = 1 THEN L = 0

* NOTE: RESERVED BITS MAY BE USED ON MC² AT THE RISK OF BEING INCOMPATIBLE WITH FUTURE DESIGNS & ENHANCEMENTS.

PROGRAM STATUS REGISTER

The Program Status Register/STATUS (i.e., R[1]) contains the salient characteristics of the most recent result produced by the processor. The following conditions are captured in status:

- Result field negative (N=1). Most significant bit equals one
- Result field zero (Z=1)
- Result field generated carry from most significant bit (C=1)
- Result generated overflow (O=1). For add, operands same sign and result is the opposite sign. For subtract, operands different sign and the result is the same sign as subtrahend (right operand).
- Position of most significant bit in byte (L = bit position labeled from left to right)
- Interrupts enabled (E=1)

STATUS modification is disabled whenever it is selected as the destination register for an operation. All non-control instructions will cause STATUS to reflect the characteristics of the resultant data created by the instruction. Although compare instructions do not create any visable new data, STATUS is modified as if a result was produced from the subtract.

The bit locator field, L, in STATUS is not found in most computers and should be amplified upon. It is a three bit, positive field containing the bit address of the most significant one in the result righthand byte. Bits are numbered from left to right within a byte and most significant bit is position zero (the least significant bit is bit number W-1, where W is the field width). If there are no set bits in a byte (i.e., Z8=1 the field is zero), L contains zero. Defaulting to this value for zero fields is desirable for indexed branches because all possible conditions (i.e., position of one bits including no ones) are covered. Because the maximum field width which can be processed is eight bits the largest value which can occur in L is 7.

Note that an all zero eight bit field makes STATUS <12:4> =1000 (8 decimal) giving a unique result.

Result				Z8	L
0 3	4 7	8 11	12 15		
XXXX	XXXX	1XXX	XXXX	0	0
XXXX	XXXX	01XX	XXXX	0	1
XXXX	XXXX	001X	XXXX	0	2
XXXX	XXXX	0001	XXXX	0	3
XXXX	XXXX	0000	1XXX	0	4
XXXX	XXXX	0000	01XX	0	5
XXXX	XXXX	0000	001X	0	6
XXXX	XXXX	0000	0001	0	7
XXXX	XXXX	0000	0000	1	0

X is don't care

STACK MECHANISM

MCP contains a simple stack mechanism for subroutine return address storage, parameter transferring, and temporary result storage. Register zero contains the top-of-stack word, while the other stack elements are stored in a RAM which is a part of the machine's address space. An eight bit, stack pointer (S) identifies the next element of the RAM stack to be used, pointing to ascending storage locations as data is pushed on to it. The stack limit is 255, based on the largest positive integer which can be held in S.

Two data transfer operations can be performed on the stack. They are:

Pushing	Popping
1. MEMORY [S]:=R[N2]	1. S := S-1
2. S := S + 1	2. R [N2]:=MEMORY [S]

The push operation implicity invoked when an interrupt or CALL occurs uses register R[0]. Otherwise, all other instructions treat R[0] like any other general purpose register and do not cause data traffic between R[0] and the RAM stack as a side effect. The POP and RETURN instructions may use any register as the top-of-stack. It is the programmers responsibility to maintain stack integrity.

INTERRUPT MECHANISM

MCP contains an interrupt port into the processor. If the interrupt system has been enabled and the interrupt is asserted, it will cause an automatic push of P+1 onto the stack and an instruction fetch from address FFFF after the current instruction is executed. Once the interrupt is acknowledged, the interrupt system is disabled and must be programmatically enabled.

A Branch will normally be executed since the interrupt return point is saved on the stack. It is the programmer's responsibility to save all registers that will be changed by the execution of the interrupt routine (esp. note STATUS). Note that TBIT, SBIT, RBIT, CBIT, hold off interrupts until after the following instruction to allow turning interrupts back on and returning to the main program before another interrupt occurs.

POWER ON

When the power on signal to the MC2 goes high, the MC2 cycles through a special power-on sequence which sets the chip in a known state:

Register Base register = 0
Stack Pointer register = 0
STATUS register = <811>=0 (interrupts off)

If the Get Next Instruction signal is high, then control passes to the address FFFD; otherwise, the MC2 goes idle until GNI goes high.

The power on signal must remain low after the VDD supply is at its minimum value for at least 8 clock periods.

IDLE STATE

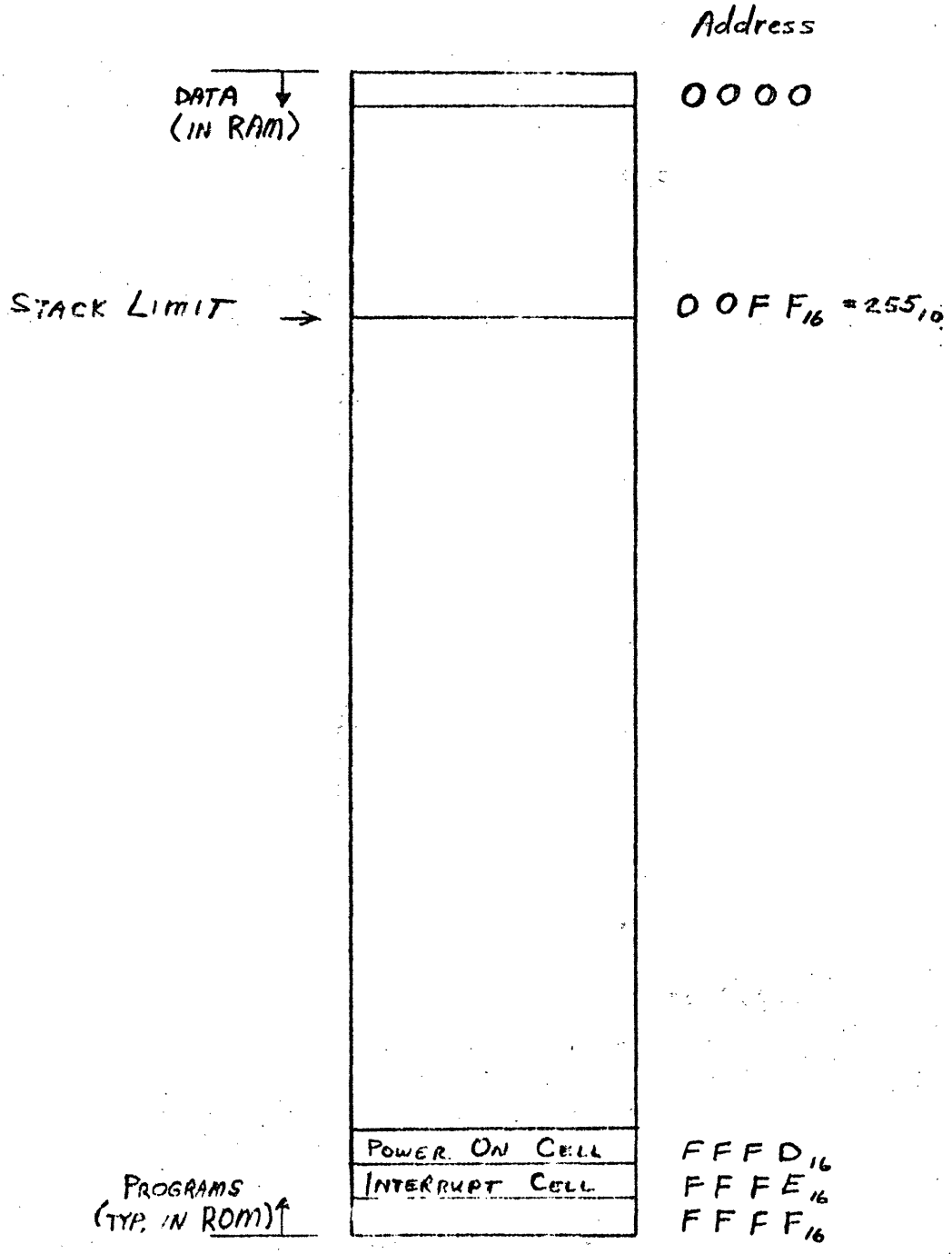
At the end of each instruction, and during the Power-on sequence, the Get Next Instruction line is tested. If it is high, execution continues. If it is low, the MC2 goes into an Idle State. While in this state, it responds as a slave module to external register reads and loads directed to bank numbers FE and FF. Bank numbers FA through FD are reserved:

Bank #	Reg #	Registers
FE		GENERAL (R[0] to R[7])
MC2		
FF	0	S
	1	P
	2	MASK (read only)
	3	IR
	#4	D
	#5	RR1
	#6	RR2
	#7	Reserved

#internal working registers

When Get Next Instruction goes high the MC2 fetches the next instruction and proceeds based on that instruction. (See Appendix C.) GNI is internally pulled high by a 13K ohm resistor.

When an instruction is being fetched and is on the Data Bus, the Fetch Phase signal is asserted. If the instruction is not a valid MC2 instruction (i.e., IR< 013 > = 1111) then the Idle state is entered and the MC2 behaves as described above.


Address
DATA
(IN RAM)
0000
STACK LIMIT
00FF16 = 25510
POWER ON CELL
FFFD16
INTERRUPT CELL
FFFE16
FFFF16
PROGRAMS
(TYP. IN ROM)

REGISTER TO REGISTER INSTRUCTIONS

The register to register instruction group constitutes the majority of MC2 instructions, in contrast to more general purpose computers which exhibit an even mixture of memory reference and register to register instructions. Controller applications tend to be more I/O and decision oriented, rather than data/RAM store oriented. These instructions either apply functions to 16 bit words or to partial fields within the word and transmit the result to a full word. The intent of these simple functions (i.e., basic arithmetic and logical functions) coupled with a uniform selection and destination mechanism is to limit the number of unique functions (to simplify LSI implementation on a chip) while providing all the facilities available in conventional instruction sets. A few examples should illustrate this concept:

- Increment/decrement register (by 1) instruction is replaced with the combination of a literal add/subtract with a value of one. As a bonus, in a single instruction time any byte literal can be added or subtracted.

- Bit set, complement, test, add and reset instructions are provided.

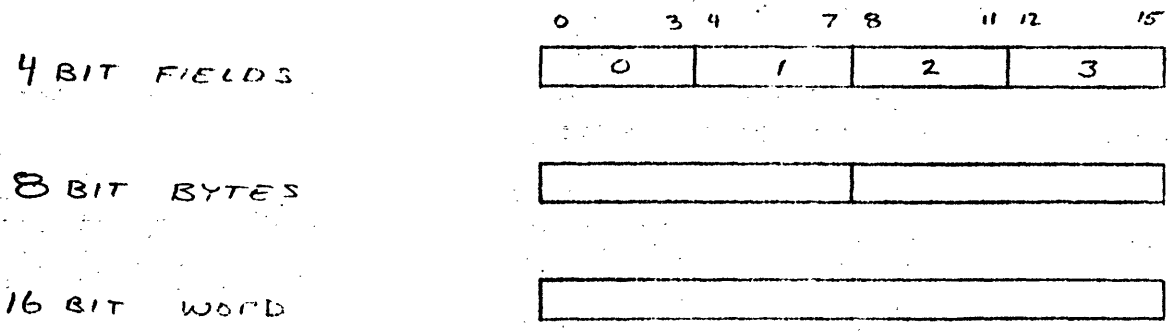
This philosophy is also used throughout the other instruction groups.

Four functions, ADD, SUBTRACT, COMPARE, and LOAD, can be applied between an eight bit literal and a full word register using the full register and literal format. Any internal register can be selected as one operand and the result destination.

In the Register format, the two operands are a full word register and a field selected out of a register. A full set of logical and arithmetic functions are available.

There are several monadic functions which can be applied to any register: one's and two's COMPLEMENT, and SHIFT. Shift operations are applied to an entire word. A shift may be either logical (zero fill), circular in either direction, LINKED (carry bit), or FILLED (one's fill).

FIELD DESCRIPTOR



FD 0 1 2 3	
0 0 1 0	PASS FULL 16 BIT WORD
0 0 1 1	PASS LEFT BYTE, ZERO RIGHT BYTE
0 1 1 0	PASS RIGHT BYTE, ZERO LEFT BYTE
0 1 0 0	MOVE LEFT BYTE TO RIGHT, ZERO LEFT BYTE
0 0 0 1	MOVE RIGHT BYTE TO LEFT, ZERO RIGHT BYTE
0 0 0 0	SWAP LEFT & RIGHT BYTE
1 1 0 0	RIGHT JUSTIFY NIBBLE 0, ZERO LEFT 12 BITS
1 1 0 1	" " " 1, " " " "
1 1 1 0	" " " 2, " " " "
1 1 1 1	" " " 3, " " " "

FOR INFORMATION ONLY
4 BIT FIELD
ZERO LEFT BYTE
DO NOT SWAP BYTES
IF NOT 4 BIT THEN ZERO RT BYTE

FD<0:4> := IR<8:4>

NOTE: THE 6 UNUSED CODES ARE RESERVED FOR FUTURE USE. THEIR ACTIONS SHOULD NOT BE RELIED UPON.

VARIABLE FIELD WIDTH OPERATIONS

To cope with a variety of control oriented, data manipulation functions MC2 is equipped with functions which can be applied to variable width fields. Four or eight bit fields can be operated upon independently of surrounding fields or the full sixteen bit word can be manipulated in parallel. Fields are aligned on binary bit boundaries and both their position and width are specified by a four bit, Field Description (FD) in those instructions which operate on variable field widths. The following illustration details the encoding scheme for field width specification and the different addressing modes for specifying field positions.

Field widths other than the basic four processed by MC2 can be conveniently manipulated through a combination of two or three instructions using the hardwired field widths as building blocks.

CONTROL TRANSFER FUNCTIONS

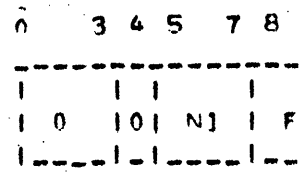
There are four instructions in the control transfer group: Conditional Branch/CBR, Indexed Branch/IBR, Subroutine Call/CALL and Subroutine Return/RTN . These instructions modify the instruction pointer register (P), thus modify the sequential nature of a program.

The CBR and CALL will transfer control to any location within the 256 word current page (IP + 1) <0:8> cat Displacement). This is the direct addressing mode. An indirect addressing mode is available which permits control transfers to any storage location. In this case the displacement selects the location in the current page which contains a full sixteen bit address to pass control to. Note that since P+1 is used to obtain page address, a CBR or CALL in the last word of a page actually uses the next sequential page. Addressing for the IBR instruction is accomplished by adding the selected field to the Instruction Pointer plus one. This instruction always specifies indirect addressing. The BR instruction transfers control to the address contained in the specified register.

STORE ACCESS INSTRUCTIONS

Because of MC2's controller orientation, the storage access instructions have been limited to two: Load and Store Register. Both instructions only access word data. The data access instructions can specify post-indexing using any of the eight internal registers. The word following the Load or Store instruction contains a full 16 bit address, if the long form is specified.

Logical AND Register & Field AND

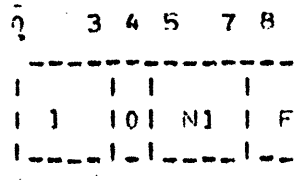


R[N1] := R[N1] ∧ R[N2]<SF>

Status Z, N, L: C, O not valid

A selected field from one register is logically ANDed with the other specified internal register N1. The results left in Register N1. Register N2 is unaffected.

Logical OR Register & Field OR

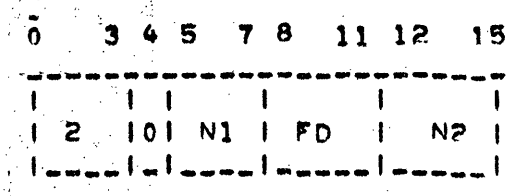


R[N1] := R[N1] V R[N2]<SF>

Status Z, N, L: C, O not valid

A selected field from one register is logically ORed with the internal register N1. The result is left in Register N1. Register N2 is unaffected.

Compare Field to Register CMPR

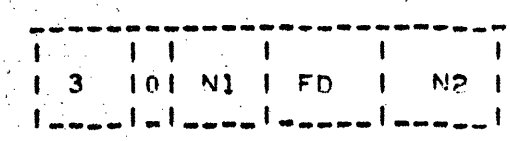


R[N2] <sf> - R[N1]

Status Z, N, L, C, O

The specified internal register (N1) is subtracted from a selected field of register N2. Z, N, L, C, O are set based on the result of the subtraction. Registers N1 and N2 are unaffected. Note: If overflow then N is complemented to maintain correct arithmetic relation of greater than and less than.

Logical Exclusive OR Register & Field XOR

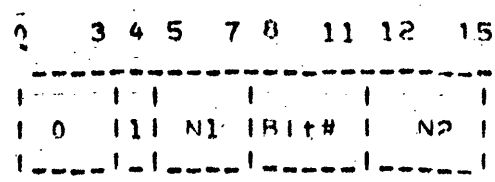


R[N1] := R[N] ⊻ R[N2] <<f>

Status Z, N, L; C, O not valid

A selected field from one register is logically exclusive ORed with the internal specified register N1. The result is left in register N1. Register N2 is unaffected.

Reset Bit RBIT

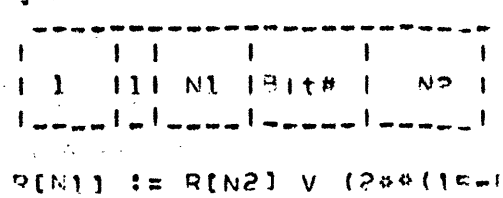


R[N1] := R[N2] ∧ Not (2**(15-Bit#))

Status Z, N, L; C, O not valid

A copy of register N2 (possibly external) is stored in internal register N1. The bit position in N1 specified by the bit number is cleared to zero. Register N2 is unaffected. Status is set based on the final value in N1.

Set Bit SBIT

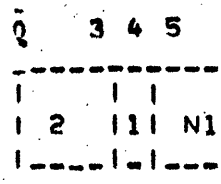


R[N1] := R[N2] ∨ (2**(15-Bit#))

Status Z, N, L; C, O not valid

A copy of register N2 (possibly external) is stored in internal register N1. The bit position in N1 specified by the bit number is set to one. Register N2 is unaffected. Status is set based on the final value in N1.

Test Bit TBIT

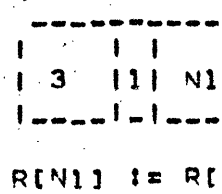


R[N1] := R[N2] ∧ (2**(15-Bit#))

Status Z, N, L; C, O not valid

A copy of register N2 (possibly external) is stored in internal register N1. All bit positions in N1 except the one specified by the BIT number are cleared to zero. Register N2 is unaffected. Status is set based on the final value in N1.

Complement Bit CBIT

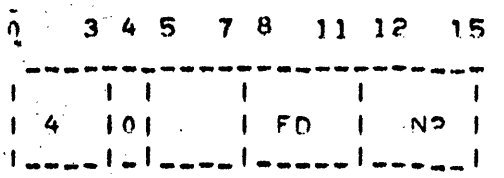


R[N1] := R[N2] ⊻ (2**(15-Bit#))

Status Z, N, L; C, O not valid

A copy of register N2 (possibly external) is stored in internal register N1. The bit position specified by the Bit number is logically complemented (toggled). Register N2 is unaffected. Status is set based on the final value in N1.

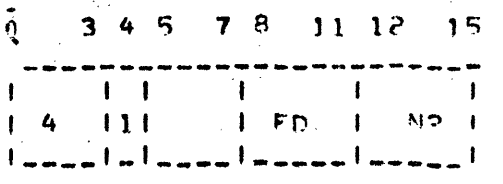
Branch BR



P := R[N2] <sf>
Status Unaffected

A selected field from (possibly external) register N2 is stored into P and execution continues at that address. Status and register N2 are are unaffected.

Indexed Branch IBR



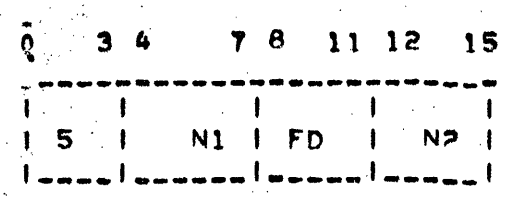
P:= MEM [P + 1 + R[N2] <sf>]

Status Unaffected

A selected field from (possibly external) register N2 is added to P register plus one. The content of this address is fetched from memory and execution continues at this indirect address. Status and register N2 are unaffected.

MOVE Field to Register

MOVE



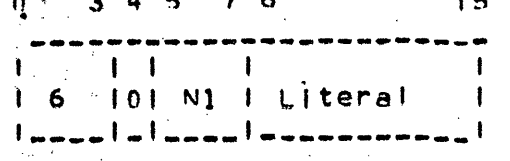
R[N1] := R[N2] <sf>

Status Z, N, L: C, O not valid

A selected field from one register is stored into the other specified register. Either or both of the registers may be external. Register N2 is unaffected. Status is set based on the final value of register N1.

ADD Literal to Register

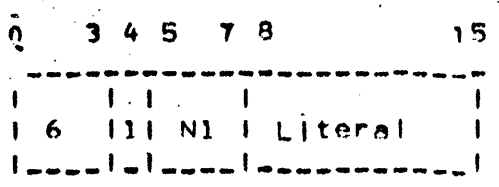
ADDI



R[N1] := R[N1] + 0 cat Literal

The eight bit literal is zero filled to the left and added to the specified register. The result is left in the register. C, O, Z, N, L are set based on the result.

Subtract Literal from Register SUBI

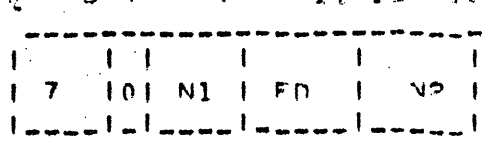


R[N1] := R[N1] - 0 cat Literal

Status C, O, Z, N, L

The eight bit literal is zero filled to the left and subtracted from the specified register. The result is left in the register. C, O, Z, N, L are set based on the result.

ADD Register to Field ADD

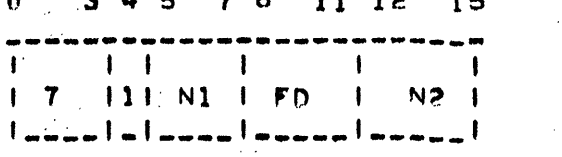


R[N1] := R[N2] <sf> + R[N1]

Status C, O, Z, N, L

A selected field from one register is added to the internal specified register N1. The result is left in register N1. Register N2 is unaffected. Status is set based on the final value in N1.

Subtract Register from field SUB

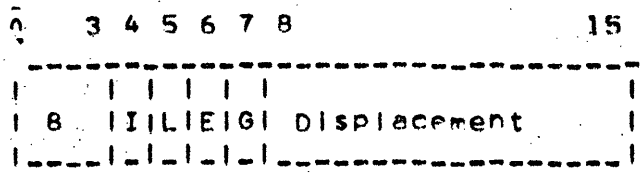


R[N1] := R[N2] <sf> - R[N1]

Status Z, N, L, C, O

The specified internal register N1 is subtracted from a selected field of register N2 (possibly external). The result is stored in register N1. Status is set based on the result. Register N2 is unaffected.

Conditional Branch CBR



condition Met := False; next

$(L \wedge N) \vee (E \wedge Z) \vee (G \wedge \bar{N} \wedge \bar{Z})$ -> CONDITION MET:=TRUE

next

Condition Met = True -> ((I = 0 -> P:=(P + 1) <0:8> cat Displacement)

(I = 1 -> P:=MEM [(P + 1) <0:8> cat Displacement)))

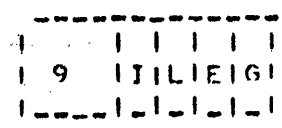
Status Unaffected

If the condition in status is met, the branch is taken by concatinating the left byte of P+1 with the displacement. For indirect addressing this address is used as an effective address to branch through.

The conditions are specified by the bits of the Instruction Register: Less than, Equal to, Greater than. The zero and negative bits in status are tested. The eight conditions are: Never, <, <=, =, >, >=, NOT=, ALWAYS.

Note that the action taken when N=7=1 should not be relied upon, as future designs may act differently.

Call Subroutine CALL



Condition Met:= False; next

(L ∧ N) ∨ (E ∧ Z) ∨ (G ∧ $\overline{N}$ ∧ $\overline{Z}$) -> CONDITION MET:=TRUE;

next

Condition met = True ->

(MEM [0 cat S] := R[0]

next S:= S + 1;

next R[0] := P + 1; next

I = 0 -> (P := (P + 1) <0:8> cat Displacement)

I = 1 -> (P := MEM [(P + 1) <0:8> cat Displacement)))

Status Unaffected

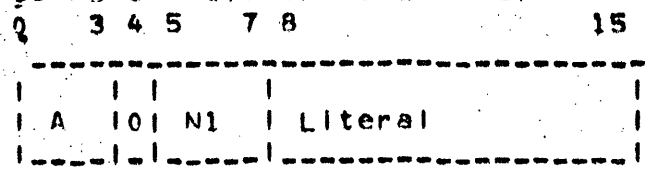
If the proper condition in status is met then the CALL is made as follows:

The address of the next sequential instruction (P+1) is pushed onto the stack. (P+1) <0:8> is concatenated with the 8 bit displacement to form an address.

For direct addressing control transfers to this address.

For indirect addressing the content of memory at this address is used as an effective address. Control transfers to this effective address.

Compare Register to Literal CMPRI

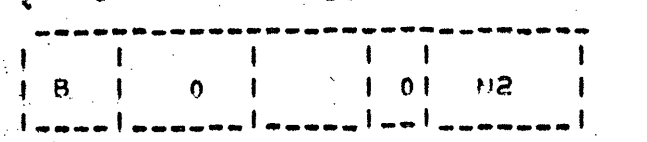


R[N1] - 0 cat Literal

Status C, O, Z, N, L

The eight bit literal is filled with zeros to the left and subtracted from the specified internal register. Z, N, L, O, C are set based on the result of the subtraction. Register N1 is unaffected. Note: If overflow then N is complemented to maintain correct arithmetic relation of greater than and less than.

PUSH REGISTER ONTO STACK

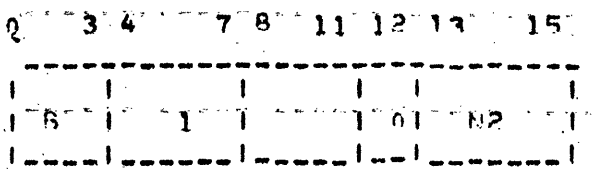


MEM[0 CAT S]:= R[N2]; NEXT
S:= S + 1;

Status Unaffected

Internal register N2 is stored in memory at the location pointed to by the S register (i.e., pushed onto stack). The S register is then incremented. Register N2 and Status are unaffected.

RETURN From Subroutine RTN



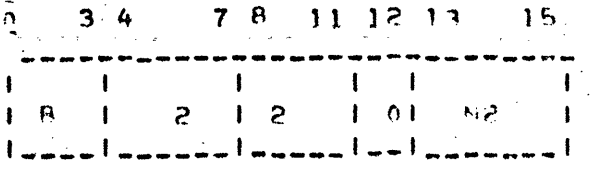

```
P := R[N2];
NEXT  S := S - 1;
NEXT  R[N2] := MEM [0 CAT S]
```

Status Unaffected

The return address on the top of the stack (R[N2]) is placed in the P register. The stack is popped.

Note that a return will normally use R[0] unless other action is explicitly desired, since both call and interrupt use R[0] as TOS.

ADD Carry to Register ADDC

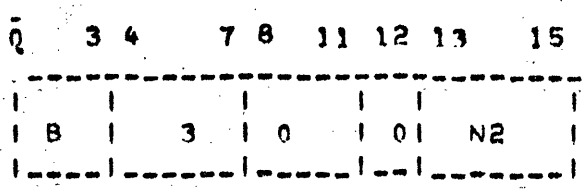



```
TEMP:= STATUS <2>; NEXT
R[N2] := R[N2] +  TEMP
```

Status Z, N, L, C, O

The value of the carry bit (status <2>) is added to internal Register N2 and the result is stored in register N2. Status is set based on the result.

One's Complement Register CMPL

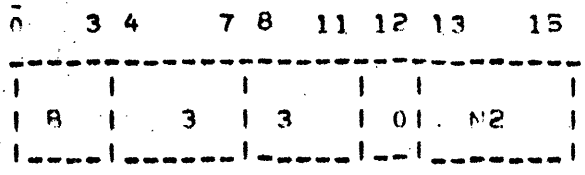


R[N2] := NOT R[N2]

Status Z, N, L; C, O not valid

The selected internal register is one's complemented (bit by bit). Status Z, N, L set on new content of N2. Carry and Overflow bits in status are INVALID.

Negate (2's Complement) Register NEG

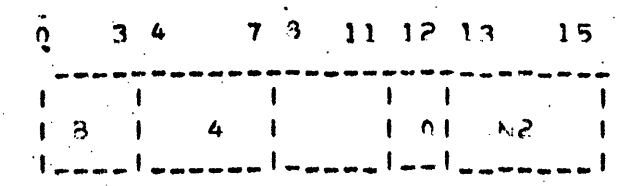


R[N2] := 0 - R[N2]

Status Z, N, L, C, O

The internal register is replaced by its two's complement. Status is set based on the result. Note that overflow is set only when attempting to take the two's complement of 8000 (most negative number) due to the asymmetric range.

STORE Register Base into Register STRB

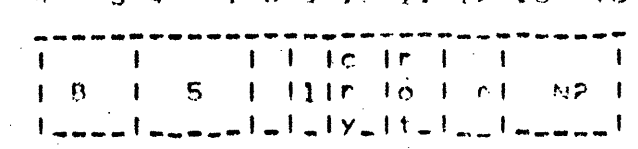


R[N2] <0:8> := RB
R[N2] <8:8> := GARBAGE

Status Unaffected

The left byte of internal register N2 is loaded with RB.

SHIFT Register Right SHFTR



TEMP := <STATUS> <2> ;next

STATUS <2> := R[N2] <15> ;next

For M = 15 Step -1 until 1 do

(R[N2] <M> := R[N2] <M-1> ;next)

IR <10:2> = 00 -> (R[N2] <0> := 0)

IR <10:2> = 01 -> (R[N2] <0> := Status <2>)<< OLD R[N2] <15> >>

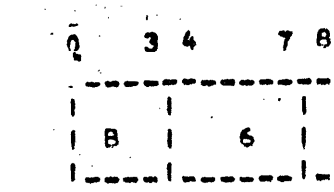
IR <10:2> = 10 -> (R[N2] <0> := TEMP) <<OLD CARRY>>

IR <10:2> = -> (R[N2] <0>: = 1)

Status Z, N, L, C; C not valid

The selected internal register is shifted right one bit, bit 15 goes into Carry. Bit 0 is filled with 0, Bit 15, OLD Carry, or 1. Status Z, N, and L are set based on new value in N2.

LOAD Register Base LDRB

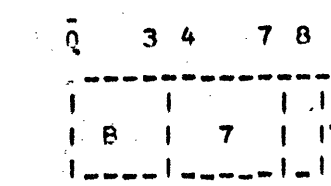


RB := [R[N2] <sf>] <0:8>

Status Unaffected

The RB register is loaded from the upper byte of A SELECTED FIELD OF internal register N2. Status and register N2 are unaffected.

SHIFT Register Left



TEMP := <STATUS> <2> ;next

Status <2> := R[N2] <0> ; next

For M = 0 step 1 until 14 do

(R[N2] <M> := R[N2] <M + 1> ; next)

IR <10:2> = 00 -> (R[N2] <15> := 0)

IR <10:2> = 01 -> (R[N2] <15> := Status <2>)

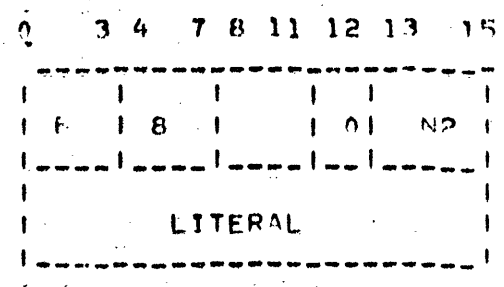
IR <10:2> = 10 -> (R[N2] <15> := Temp)

IR <10:2> = 11 -> (R[N2] <15> := 1)

Status Z, N, L, C; O not valid

The selected internal register is shifted left one bit by adding it to itself. Bit 0 goes into Carry. Bit 15 is filled with 0, Bit 0, OLD Carry, or 1. Status Z, N, L are set based on the new value in N2. Carry is loaded with most significant bit (0) of the original N2 value.

LOAD Word Immediate LDWI



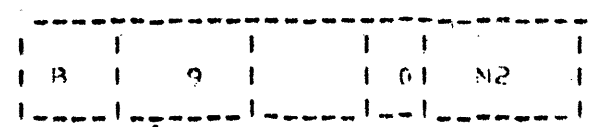
R[N2] := MEM [P + 1]; next

P := P + 1;

Status Unaffected

The word following the instruction (at P+1) is loaded into internal register N2. P is incremented an extra time to skip the literal.

POP POP



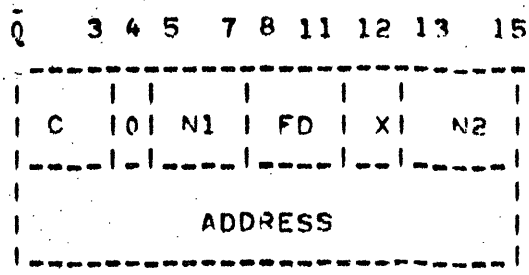
S := S - 1;

next R[N2] := MEM [0 cat S]

Status Unaffected

The stack is popped into the specified internal register.

LOAD Register from Memory (long form) LOAD



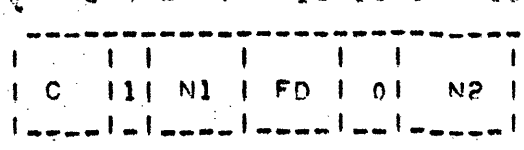
R[N2] := MEM [MEM [P +1] + (x = 0 -> 0; x = 1 -> R[N1] <sf>)]; next

P := P + 1;

Status Unaffected

The word following the instruction (at P+1) is used as an address. If indexing is specified then the selected field of internal register N1 is added to the address. The content of the effective address is fetched from memory and loaded into the selected internal register N2. N1 and status are unaffected. P is incremented an extra time to skip the address word.

LOAD Register from Memory (short form) LOADSH

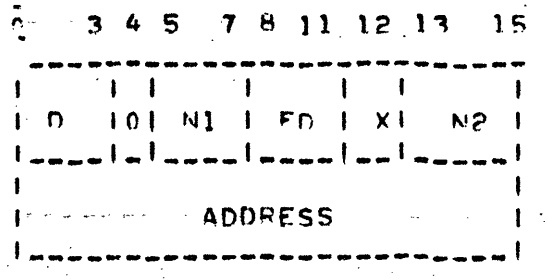


R[N2] := MEM [R[N1] <sf>]

Status Unaffected

The field extracted from internal register N1 is used as an address into memory. The content of this address is loaded into the specified internal register N2.

STORE Register into Memory (long form) STOR



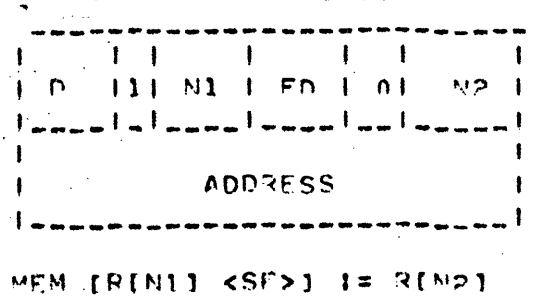
MEM [MEM [P + 1] + (x = 0 -> 0; x = 1 -> R[N1] <sf>)] := R[N2]; next

P := P +1;

Status Unaffected

The word following the instruction (at P+1) is used as an address. If indexing is specified then the selected field of internal register N1 is added to the address. This selcted internal register N2 is stored into memory at this effective address. N1 and status are unaffected. P is incremented an extra time to skip the address word.

STORE Register into Memory (short form) STORSH

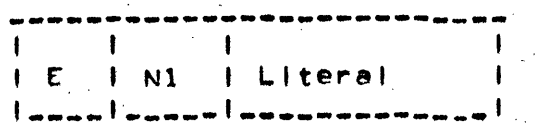


MEM [R[N1] <SF>] := R[N2]

Status Unaffected

The field extracted from internal register N1 is used as an address into memory. The content of internal register N2 is stored into this address.

LOAD Register with Byte Literal LDRI



R[N1] := 0 cat LITERAL

Status Z, N, L Invalid

The eight bit literal is filled with zeros to the left and stored into the specified register (possibly external). Status Z, N, L are set based on the literal.

APPENDIX A

Informal Description of ISP Notation

The notation used throughout this description is ISP (for Instruction Set Processor) developed by Bell and Newell[1] for the formal description of computer instruction sets. The following table identifies the symbols used and their interpretation.

Symbol	Interpretation
cat	Concatenation operator
- V Λ ⊕	Boolean operators
+-*÷	Arithmetic operators
= ≠ < ≤ > ≥	Relational operators
← :=	Data transfer
a:=(....)	Equivalence or substitution process used for name and process substitution. For every occurrence of "a" substitute "(...)"
a→b	If "a" then "b"
next	Sequencial interpretation
a[i]	Array selection. Select the ith element of the array i. (a[i:j] selects sequential elements i + j - 1)
b<n:m>	Field selection within word. Select the field starting at bit n, width = m (most significant bit is zero) b<n> selects a single bit.
a/b	Abbreviation. "a" is an abbreviation for "b"

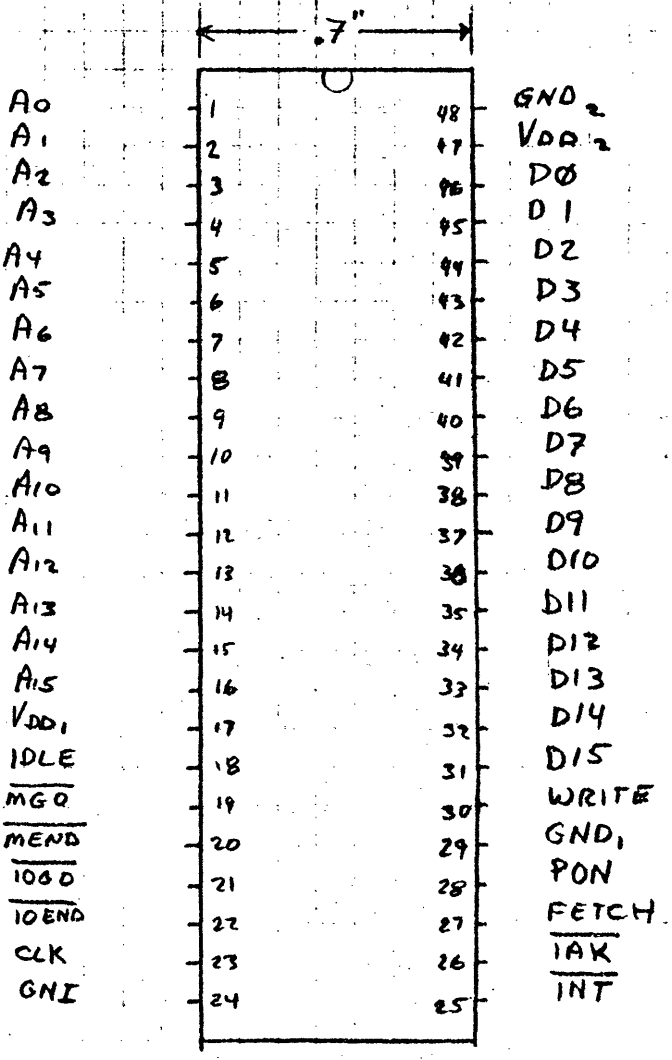
[1] Bell,C. G. and A. Newell, Computer Structures: Readings and Examples, McGraw-Hill, 1971.

[2] Barbacci, M., Bell, C. G., and Siewiorek, D., PMS: A Notation to Describe Computer Structures, Computer, March 1973, pp 19-24.

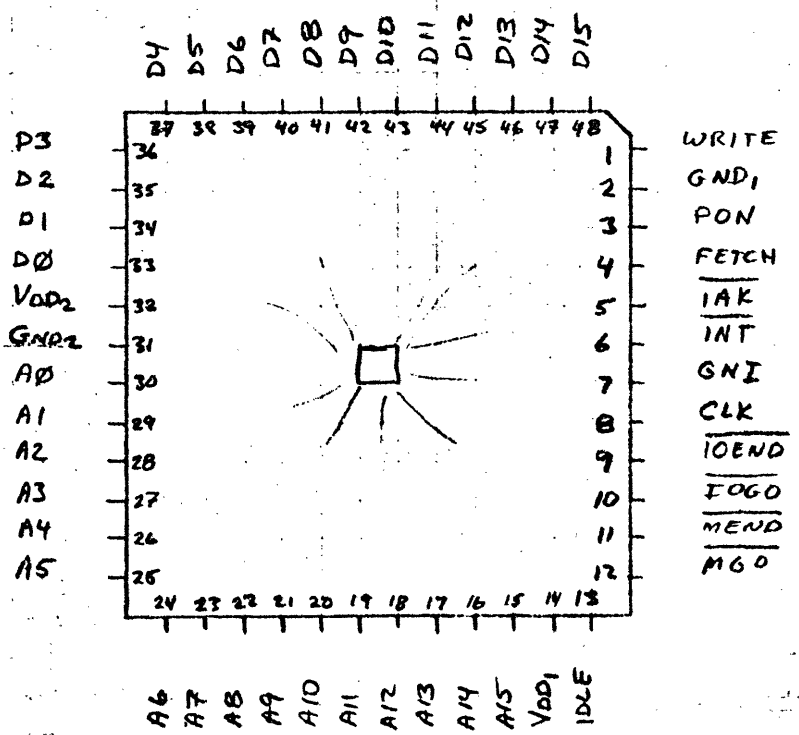
APPENDIX B
MC² PINOUTS

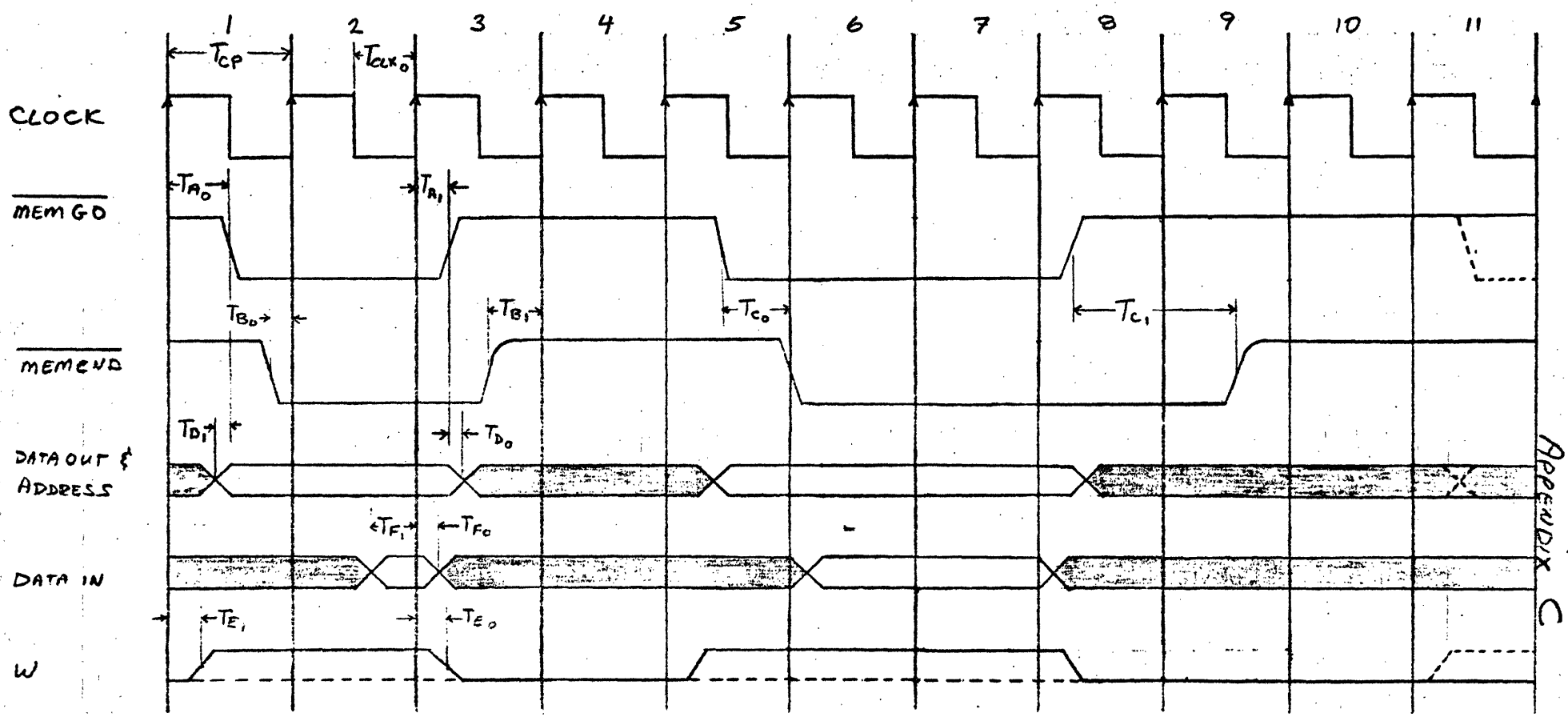
9-2-75

PROTOTYPES:



PRODUCTION:
VIEW WITH CHIP UP — MUST BE FLIPPED FOR INSERTION INTO SOCKET (i.e. this is a bottom view)

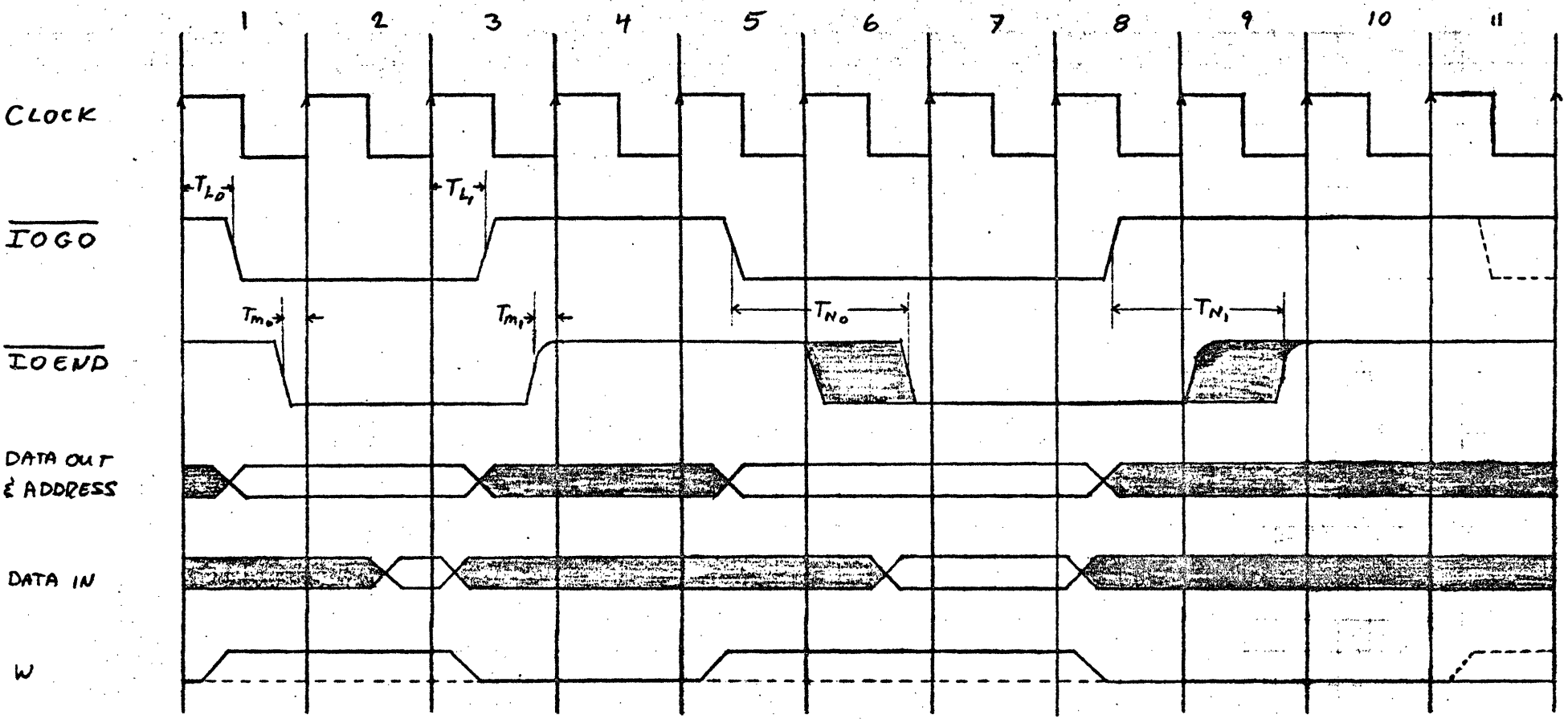




	MIN	MAX
CLOCK PERIOD T_{CP}	125	∞
CLOCK LOW PULSE WIDTH T_{CLK_0}	40	$T_{CP}-70$
MEMGO PROP DELAY TO Ø T_{A_0}		80
MEMGO " " " 1 T_{A_1}		40
MEMEND SET UP FOR Ø T_{B_0}	note 1	
MEMEND " " " 1 T_{B_1}	note 3	
GO TO END DELAY FOR Ø T_{C_0}	0	∞
GO TO END " " 1 T_{C_1}	0	∞
ABUS & DBUS OUT VALID T_{D_1}	0	10
ABUS & DBUS HOLD " T_{D_0}	10	30
DBUS IN VALID T_{F_1}	40	
DBUS IN HOLD T_{F_0}	0	
W T_{E_1}		40

CYCLE NOTE:

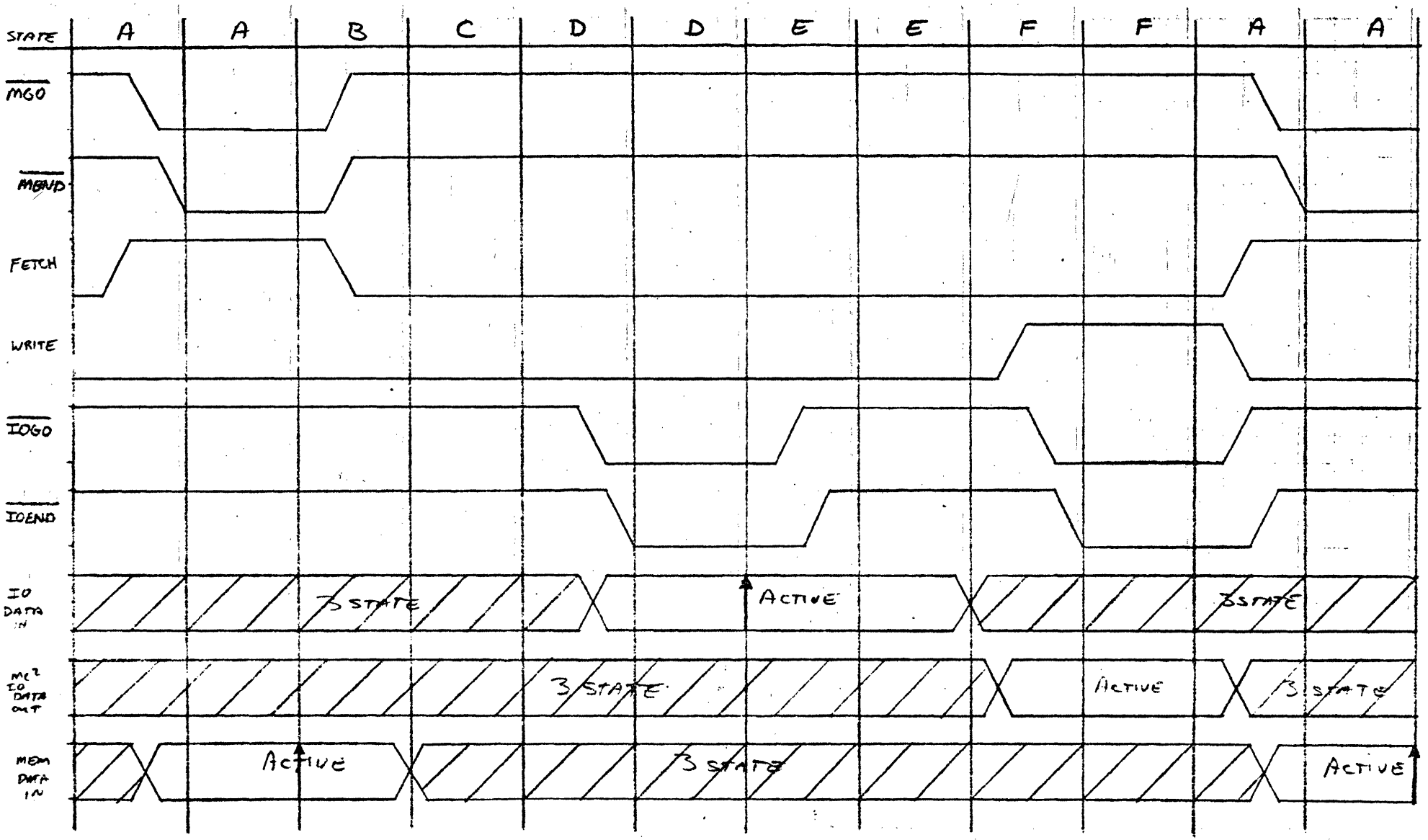
1. BOTH EDGES OF MEMEND ARE SYNCHRONIZED BY MC² — 30 NS. TO NOT MISS CLK
2. DATA IN LATCHED AT END OF THIS CYCLE, SINCE T_{B_0} SATISFIED
3. SEE NOTE 1. SYNC CAUSES 2 DEAD CYCLES ON BUS (#3 & 4)
4.
5. EXTRA CYCLE (#6) DUE TO END BEING ASYNC & MISSING SETUP (T_{B_0})
6. WHEN END IS ASYNC ON LEADING EDGE, DATA IN MUST BE VALID HERE
8. MEMGO RISES HERE DUE TO END MISSING CLK AT END OF CY. 5
9. MC² WILL WAIT ('L...) TILL 11 FOR NEXT MEM CYCLE IF T_{B_1} MET HERE
10. 2ND OF 2 DEAD CYCLES DUE TO TRAILING EDGE SYNC
11. A MEMORY ACCESS ('L....) MAY OCCUR HERE



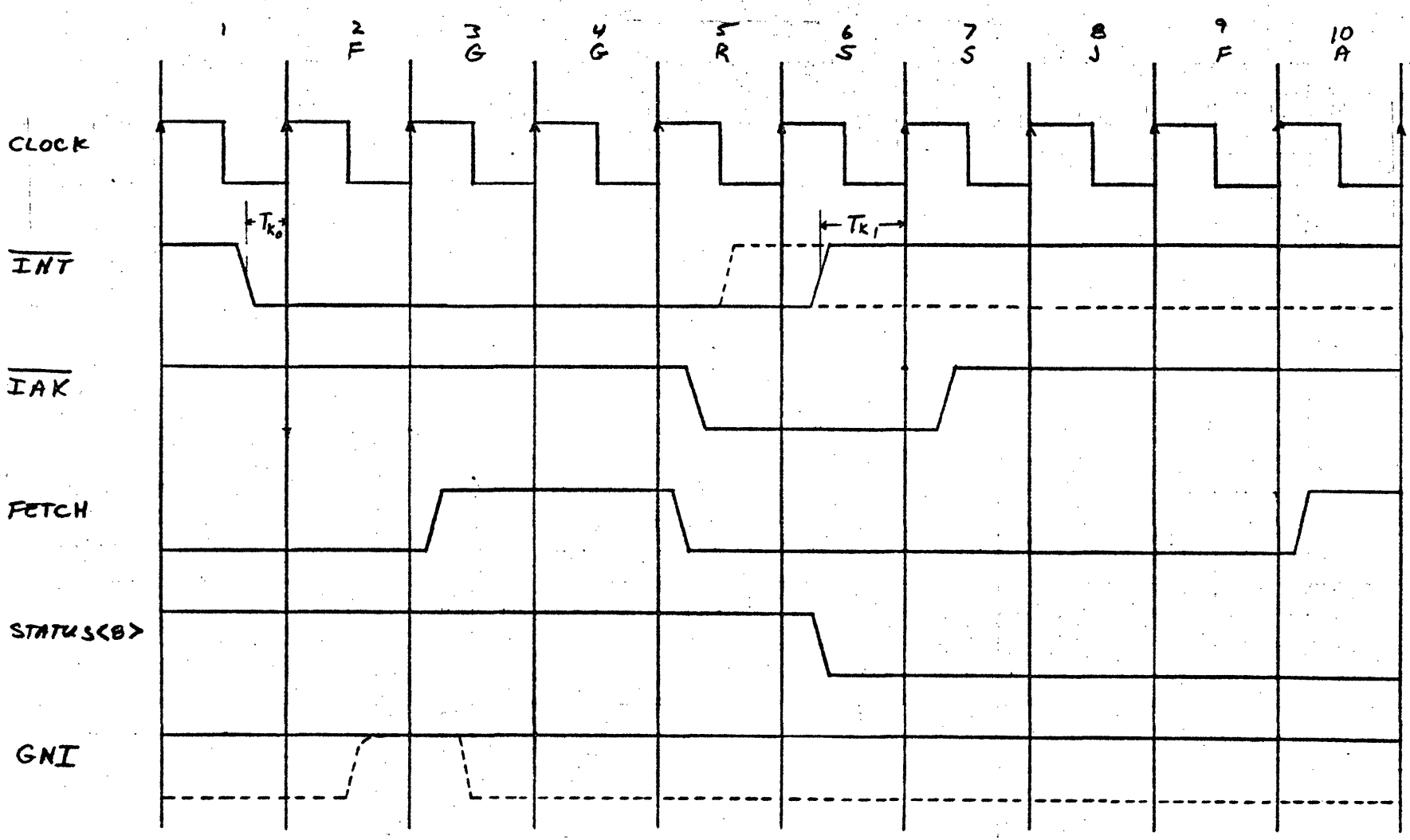
		MIN	MAX
IOGO PROP DELAY TO Ø	T_{L_0}		80
IOGO " " " 1	T_{L_1}		40
IOEND SETUP FOR Ø	T_{m_0}	note 1	
IOEND " " 1	T_{m_1}	note 3	
GO TO END DELAY " Ø	T_{N_0}	0	∞
GO TO END " " 1	T_{N_1}	0	∞

CYCLE NOTE

1. BOTH EDGES OF IOEND ARE SYNCHRONISED BY MC^2-30nS TO NOT MISCL
2. DATA LATCHED AT END OF THIS CYCLE SINCE T_{m_0} SATISFIED
3. SEE NOTE 1; FIRST OF TWO DEAD CYCLES BETWEEN I/O ACCESSES
4. SECOND DEAD CYCLE (DUE TO TRAILING EDGE SYNC FLIPFLOP)
5. IOEND TO LATE TO CATCH CLOCK – ADDS 1 CYCLE
6.
7. DATA IN MUST BE VALID AT END OF CYCLE
8. IOEND TO LATE TO CATCH CLOCK – MAY ADD 1 CYCLE*
9.
10.
11. *THIS IS THE EARLIEST A 3RD I/O ACCESS COULD OCCUR ANYWAY



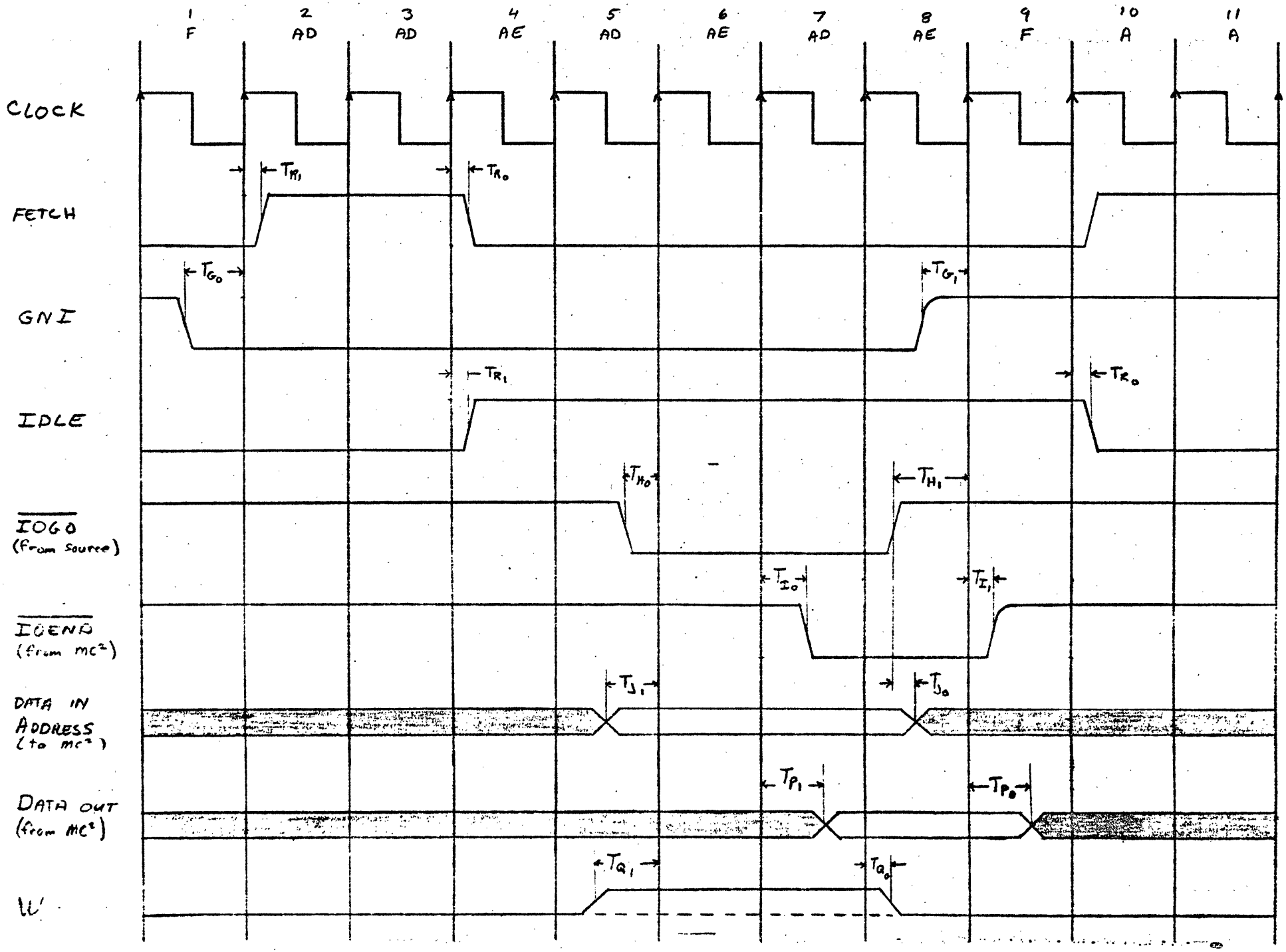
NOTE: THE SOURCE OF DATA MAY ONLY BE ACTIVE DURING TIME (CYCLES) SHOWN. THE MC² CLOCKS DATA IN ON CLOCK EDGE SHOWN ↑. THE SEQUENCE SHOWN IS FOR A MOVE FROM EXTERNAL N_2 REG. TO EXTERNAL N_1 REGISTER.

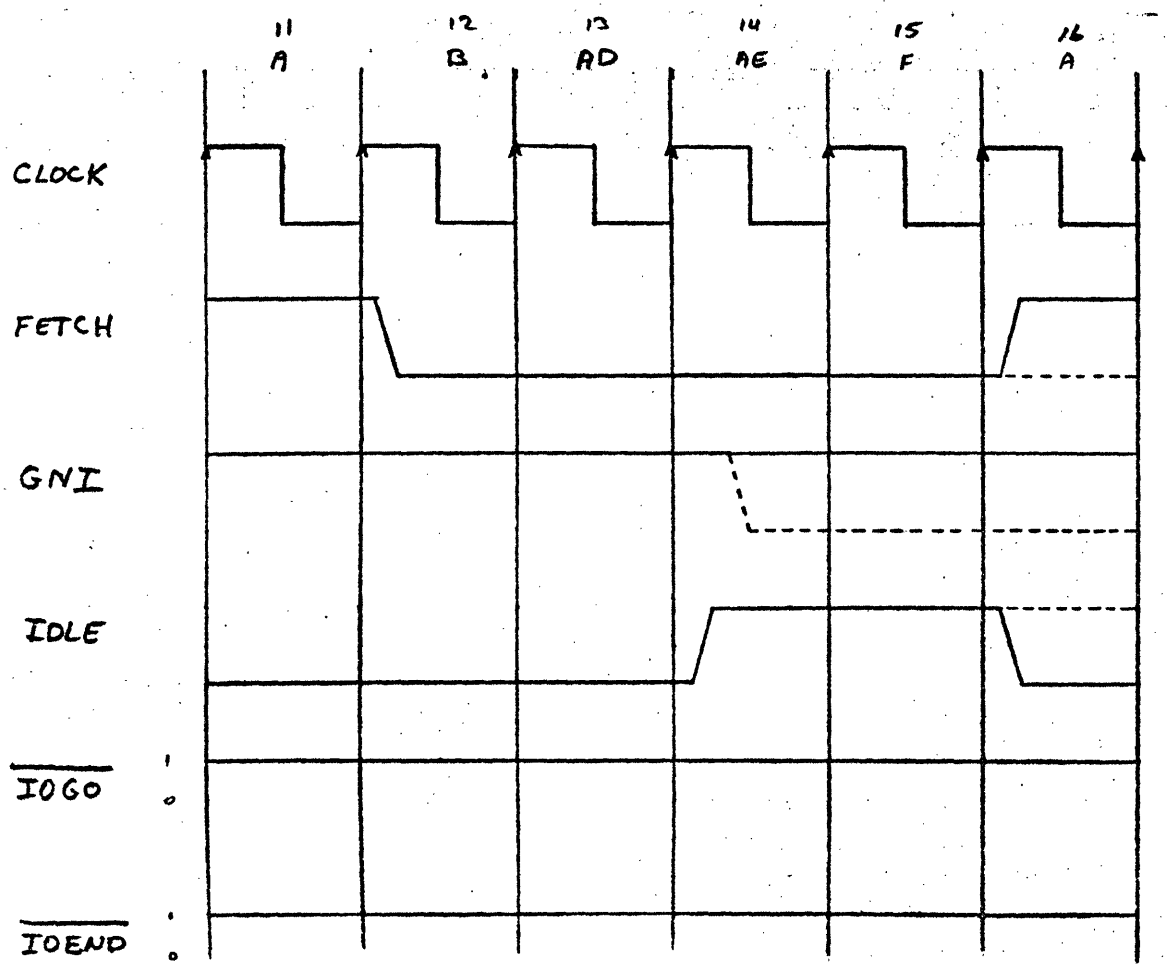


		MIN	MAX
INT SETUP TO 0	T_{K0}	note 1	—
INT " " 1	T_{K1}	note 6	—

CYCLE NOTE

1. INT IS SYNCHRONIZED BY MC². (FLIP FLOP SETUP IS 20 NS)
2. DEAD CYCLE FOR SYNC. GNI MUST BE HIGH TO GET INTERRUPT.
3&4. UNUSED FETCH OF NEXT INSTRUCTION — WILL BE REFETCHED AFTER EXIT FROM INTERRUPT ROUTINE
5. IAK ASSERTED FOR TWO CYCLES ONLY. EARLIEST INT MAY BE REMOVED
6. STATUS BIT 8 (INT. ENABLE) CLEARED
7&8. P REGISTER PUSHED ONTO STACK
10. FETCH OF 1ST INST OF INT. ROUTINE-(ADDRESS-2). IF GNI HIGH


1 F
2 AD
3 AD
4 AE
5 AD
6 AE
7 AD
8 AE
9 F
10 A
11 A
CLOCK
FETCH
GNI
IDLE
IOGO (From source)
IOGNA (from mc²)
DATA IN ADDRESS (to mc²)
DATA OUT (from MC²)
W
T_{R_1}
T_{R_0}
T_{G_0}
T_{G_1}
T_{R_1}
T_{R_0}
T_{H_0}
T_{H_1}
T_{I_0}
T_{I_1}
T_{J_1}
T_{J_0}
T_{P_1}
T_{P_0}
T_{Q_1}
T_{Q_0}



		MIN	MAX
GNI SETUP TO Ø	T_{G0}	70	
GNI " TO 1	T_{G1}	70	
$\overline{IOGO}$ " TO Ø	T_{H0}	30 note 5&6	
$\overline{IOGO}$ " TO 1	T_{H1}	80	
$\overline{IOEND}$ DELAY TO Ø	T_{I0}		30
$\overline{IOEND}$ " " 1	T_{I1}		20+RC
ABUS & DBUS IN VALID	T_{J1}	40	
ABUS & DBUS IN HOLD	T_{J0}	10	
DBUS OUT VALID	T_{P1}		70
DBUS OUT HOLD	T_{P0}	50	70
W DELAY	T_{Q1}	70	
W HOLD	T_{Q0}	0	
FETCH IDLE DELAY	T_{R1}		30

CYCLE NOTE

1. IF GNI ISN'T LOW 1 CY. BEFORE FETCH, WILL WAIT ONE INSTRUCTION BEFORE RECOGNITION
2&3. FETCH WILL BE UP A MINIMUM OF 2 CY.
4. IDLE ASSERTED
5. EARLIEST $\overline{IOGO}$ MAY BE ASSERTED, ADDRESS & WRITE MUST BE VALID. $\overline{IOGO}$ WILL BE SYN'D
6. DEAD CYCLE FOR SYNCHRONIZATION OF $\overline{IOGO}$
7. $\overline{IOEND}$ & DATA FOR READ ASSERTED
8. EARLIEST $\overline{IOGO}$, DATA & WRITE MAY BE REMOVED. THIS IS EARLIEST GNI MAY BE ASSERTED TO RESUME EXECUTION. IF GNI's ASSERTION IS DELAYED 1 OR 2 CYCLES THEN FETCH WILL BE DELAYED 2 CYCLES. GNI IS OPEN DRAIN DRIVER
9. $\overline{IOEND}$ RESET (OPEN DRAIN DRIVER)
10. FETCH ASSERTED, IDLE GOES LOW,
11. FETCH — NEW INST. WILL BE ON DBUS. THE EXAMPLE ASSUMES AN UNIMPLIMENTED INST (OPCODE F)
12. MC² INTERNAL INSTR. DECODE CYCLE. SLAVE MODULES USE THIS & 13. TO DECODE ALSO.
13. DEAD JUMP CYCLE
14. IDLE ASSERTED. IF A SLAVE MODULE-PROCESSOR WISHES TO EXECUTE THE UNIMP. (IN MC²) INSTRUCTION THIS IS LAST CYCLE TO PULL DOWN GNI. (...)
15. IDLE WILL ALWAYS BE UP A MINIMUM OF 2 CYCLES
16. IF GNI WENT LOW IN 14. THEN FETCH STAYS LOW, IDLE STAYS HIGH

INSTRUCTION TIMING

AND	7 + R2 + M	LOAD	12 + 2M + MD
OR	7 + R2 + M	LOADSH	10 + M + MD
CMPR	7 + R2 + M	LDWI	8 + 2M
XOR	7 + R2 + M	STOR	12 + 2M + MD
RBIT	7 + R2 + M	STORSH	10 + M + MD
SBIT	7 + R2 + M	PUSH	7 + M + MD
TBIT	7 + R2 + M	RTN	9 + M + MD
CBIT	7 + R2 + M	POP	8 + M + MD
RR	6 + R2 + M	NEG	6 + M
IBR	10 + R2 + 2M	CMPL	6 + M
MOVE	7 + R1 + R2 + R12 + M	STRB	5 + M
ADDI	6 + M	LDRB	6 + M
SUBI	6 + M	SHFTR	6 + M
CMPRI	6 + M	SHFTL	6 + M
LDHI	6 + M + R1	ADD	7 + R2 + M
CBR	6 + M + I if taken	SUB	7 + R2 + M
	4 + M if not taken	ADDC	6 + M
CALL	6 + M + I if taken		
	4 + M if not taken		

I = 0 if direct addressing

= 4 + m if indirect addressing

$$M = \max\left(0, \left\lceil \frac{\text{Memory Access Time} - 180}{\text{Clock Period}} \right\rceil\right)$$

M is number of cycles for Program memory
MD is number of cycles for Data Memory

R1 = 0 if R[N1] is internal

$$= \max\left(1, \left\lceil \frac{\text{Twrite} - 70}{\text{Clock Period}} \right\rceil\right)$$ if R[N1] is external

R2 = 0 if R[N2] is internal

$$= \max\left(1, \left\lceil \frac{\text{Tread} - 30}{\text{Clock Period}} \right\rceil\right)$$ if R[N2] is external

R12 = 0 if either R[N1] or R[N2] is internal

= 1 if both R[N1] and R[N2] are external

Clock Period >= 150 nanoseconds minor cycle time

$\lceil X \rceil$ = Ceiling (X) = smallest integer >= X

• LOAD
* NOT VALID

	INSTRUCTION FORMAT 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	STATUS Z_{N_L}	STATUS C/O	# CYCLES	TYPICAL DESCRIPTION
AND	0 | 0 | N_1 | FD | N_2	•	*	7	$R[N_1] \leftarrow R[N_1] \wedge R[N_2]\langle SF\rangle$
OR	1 | 0 | N_1 | FD | N_2	•	*	7	$R[N_1] \leftarrow R[N_1] \vee R[N_2]\langle SF\rangle$
CMPR	2 | 0 | N_1 | FD | N_2	•	•	7	$R[N_2]\langle SF\rangle - R[N_1]$
XOR	3 | 0 | N_1 | FD | N_2	•	*	7	$R[N_1] \leftarrow R[N_1] \veebar R[N_2]\langle SF\rangle$
ADD	7 | 0 | N_1 | FD | N_2	•	•	7	$R[N_1] \leftarrow R[N_1] + R[N_2]\langle SF\rangle$
RBIT	0 | 1 | N_1 | BIT # | N_2	•	*	7	$R[N_1] \leftarrow R[N_2] \wedge \text{NOT}[2^{**}(15-\text{BIT\#})]$
SBIT	1 | 1 | N_1 | BIT # | N_2	•	*	7	$R[N_1] \leftarrow R[N_2] \vee 2^{**}(15-\text{BIT\#})$
TBIT	2 | 1 | N_1 | BIT # | N_2	•	*	7	$R[N_1] \leftarrow R[N_2] \wedge 2^{**}(15-\text{BIT\#})$
CBIT	3 | 1 | N_1 | BIT # | N_2	•	*	7	$R[N_1] \leftarrow R[N_2] \veebar 2^{**}(15-\text{BIT\#})$
SUB	7 | 1 | N_1 | FD | N_2	•	•	7	$R[N_1] \leftarrow R[N_2]\langle SF\rangle - R[N_1]$
BR	4 | 0 | | FD | N_2			6	$P \leftarrow R[N_2]\langle SF\rangle$
IBR	4 | 1 | | FD | N_2			10	$P \leftarrow MEM[P+1 + R[N_2]\langle SF\rangle]$
MOVE	5 | | N_1 | FD | N_2	•	*	7	$R[N_1] \leftarrow R[N_2]\langle SF\rangle$
LOAD	C | 0 | N_1 | FD | X | N_2			12	$R[N_2] \leftarrow MEM[MEM[P+1] + \text{if } X \text{ then } R[N_1]\langle SF\rangle]$
LOAD SH	C | 1 | N_1 | | 0 | N_2			10	$R[N_2] \leftarrow MEM[R[N_1]]$
STOR	D | 0 | N_1 | FD | X | N_2			12	$MEM[MEM[P+1] + \text{if } X \text{ then } R[N_1]\langle SF\rangle \leftarrow R[N_2]$
STOR SH	D | 1 | N_1 | | 0 | N_2			10	$MEM[R[N_1]] \leftarrow R[N_2]$
ADDI	6 | 0 | N_1 | LITERAL	•	•	6	$R[N_1] \leftarrow R[N_1] + \emptyset \text{ cat LITERAL}$
SUBI	6 | 1 | N_1 | LITERAL	•	•	6	$R[N_1] \leftarrow R[N_1] - \emptyset \text{ cat LITERAL}$
CMPRI	A | 0 | N_1 | LITERAL	•	•	6	$R[N_1] - \emptyset \text{ cat LITERAL}$
LDBI	E | N_1 | LITERAL	•	*	6	$R[N_1] \leftarrow \emptyset \text{ cat LITERAL}$
PUSH	B | 0 | | 0 | N_2			7	$MEM[S] \leftarrow R[N_2]; \; S \leftarrow S+1$
RTN	B | 1 | | 0 | N_2			9	$P \leftarrow R[N_2]; \; S \leftarrow S-1; \; R[N_2] \leftarrow MEM[S]$
POP	B | 9 | | 0 | N_2			8	$S \leftarrow S-1; \; R[N_2] \leftarrow MEM[S]$
CMPL	B | 3 | 0 | 0 | N_2	•	*	6	$R[N_2] \leftarrow \text{NOT}[R[N_2]]$
NEG	B | 3 | 3 | 0 | N_2	•	•	6	$R[N_2] \leftarrow -R[N_2]$
STRB	B | 4 | | 0 | N_2			5	$R[N_2]\langle 0:8\rangle \leftarrow RB; \; R[N_2]\langle 8:8\rangle \leftarrow ?$
LDRB	B | 6 | FD | 0 | N_2			6	$RB \leftarrow [R[N_2]\langle SF\rangle]\langle \emptyset:8\rangle$
SHFTR	B | 5 | 1 | C_{RT} | R_{0_T} | 0 | N_2	•	*/*	6	SHIFT $R[N_2]$ RIGHT; INSERT Ø, 1, C or BIT 15
SHFTL	B | 7 | 0 | C_{RT} | R_{0_T} | 0 | N_2	•	*/*	6	SHIFT $R[N_2]$ LEFT; INSERT Ø, 1, C or BIT Ø
LDWI	B | 8 | | 0 | N_2			8	$R[N_2] \leftarrow MEM[P+1]$
CBR	8 | I | L | E | G | PAGE OFFSET			6+I/4	if CONDITION then $P \leftarrow (P+1)\langle \emptyset:8\rangle$ cat PAGE OFFSET
CALL	9 | I | L | E | G | PAGE OFFSET			6+I/4	if CONDITION then [$MEM[S] \leftarrow R[\emptyset]; \; S \leftarrow S+1;$ see]
RSRV'D	F			4	if GNI then IDLE until GNI
ADDC	B | 2 | 2 | 0 | N_2	•	•	6	$R[N_2] \leftarrow R[N_2] + \text{STATUS}\langle 2\rangle$