

Our Baby

An imaginative illustration
of a 16 bit/word stack oriented
microprocessor. with loving care.

Jim Booth

Robin Cooper

Larry M. Mixson

December 4, 1974

Motivation

The motivation for designing a processor is, of course, not of our own initiation. However, as our project progressed, a "Real World" use for a processor of our type became clear. Even though many of the early decisions were made on other considerations, a topology and control method was devised that, we believe, is very applicable to Original Equipment Manufacturers. With this market in mind, we wish to point out the advantages of the concepts and capabilities of the processor we have designed.

Objectives

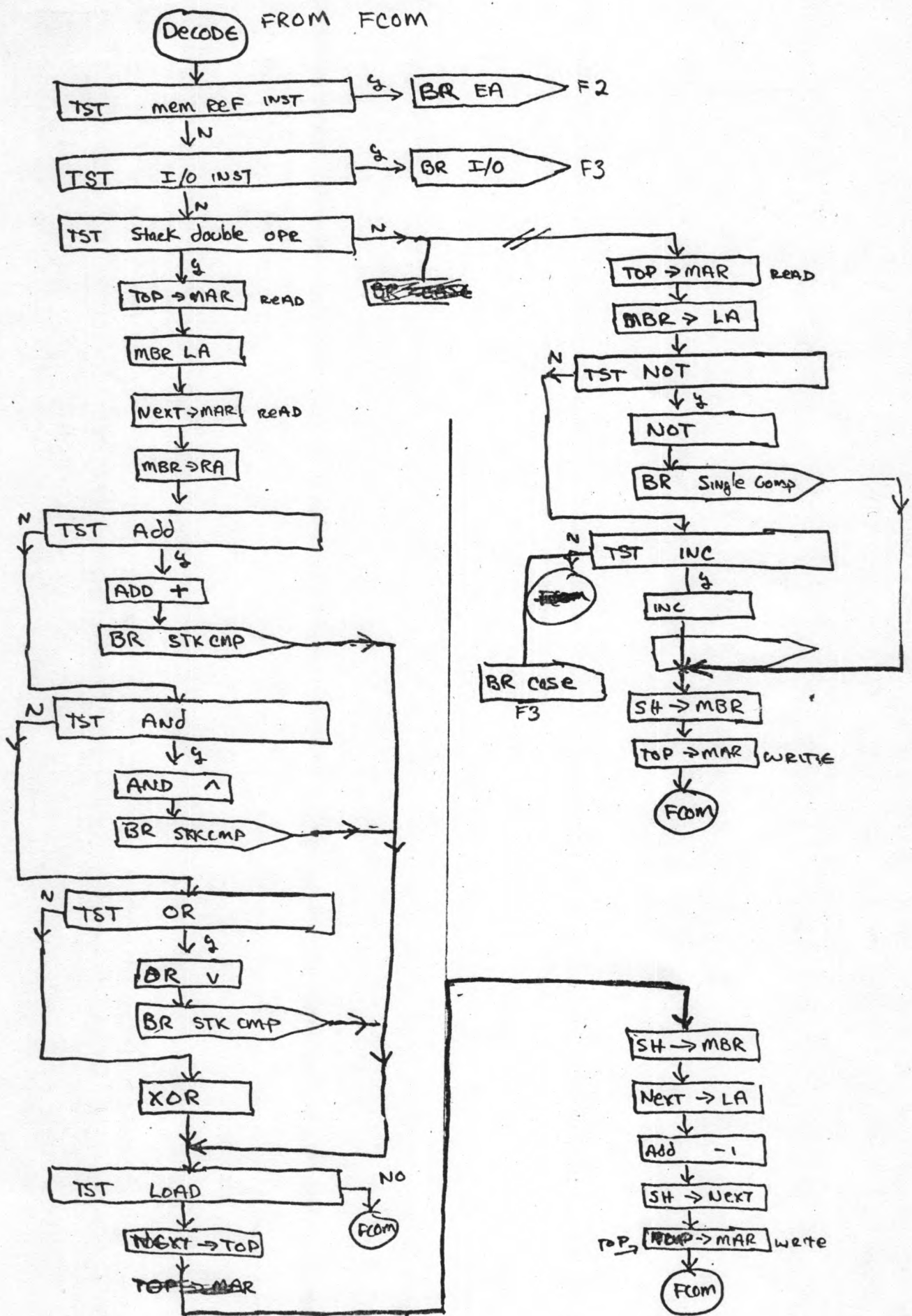
To design a simple, general, flexible mini-controller that can be incorporated easily into machinery or systems requiring resident intelligence. Simple, because of our limited experience and the short time available, and to make manufacture more profitable. General, to make the processor more modular, and more applicable to a wider group of buyers. Flexible, to allow the manufacturer of the processor to easily adapt it to specific tasks and timing the buyer requires.

Realization

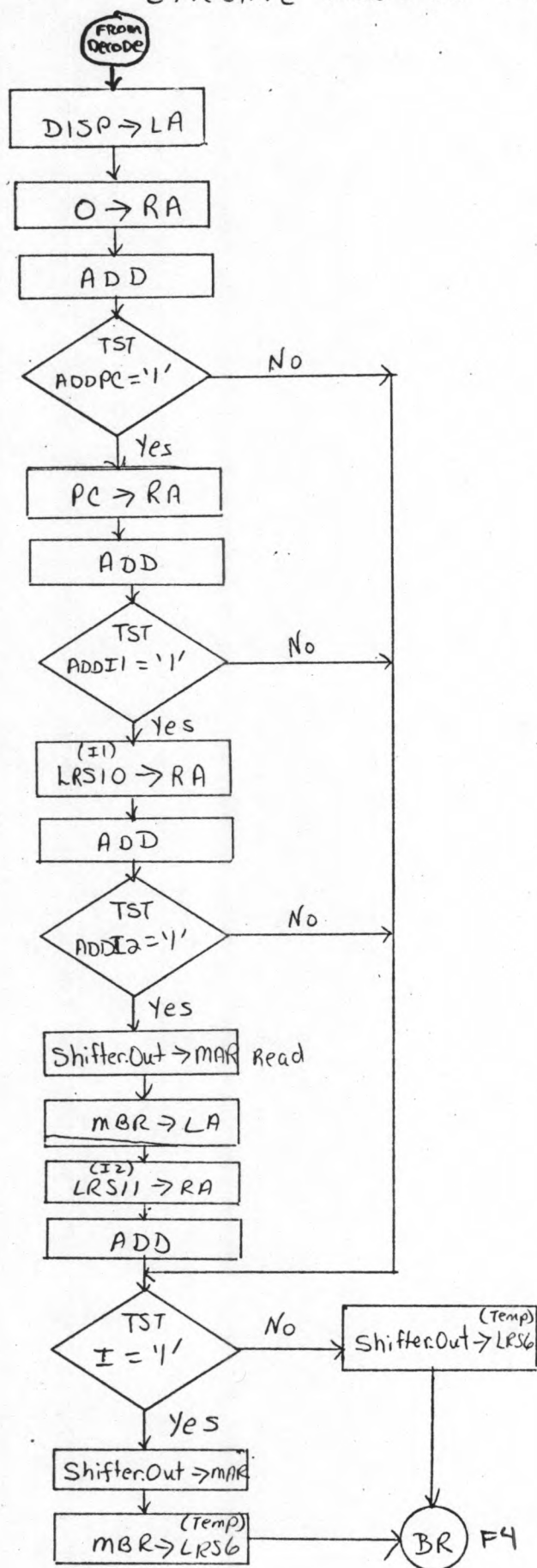
To fulfill the above objectives, we chose a single bus-drop topology. This allows for maximum modularity and minimum interfacing problems. We also chose Micro-control to insure maximum flexibility of processor characteristics and capabilities. In addition, the specified stack orientation of computation, and four level I/O structure lend very well to multiple Machine or System control. The principal drawback of our processor is in speed. Because of ROM technology and the Bus topology, speed was sacrificed for generality, flexibility and simplicity.

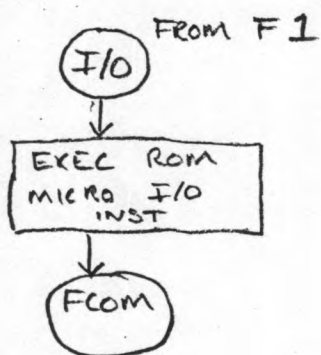
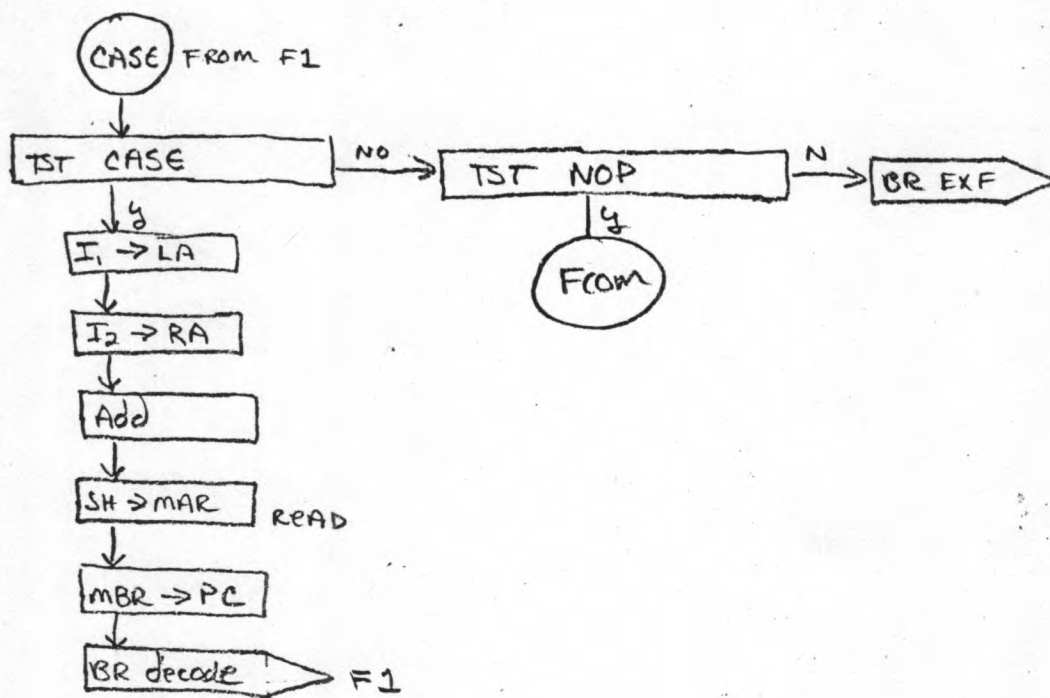
This project specifies the Software, Firmware, and Hardware included in the "General Purpose" edition of our processor, as this was the requirement of the project. However, it is conceivable that a specific buyer could configure a very wide variety of special purpose processors within our basic design.

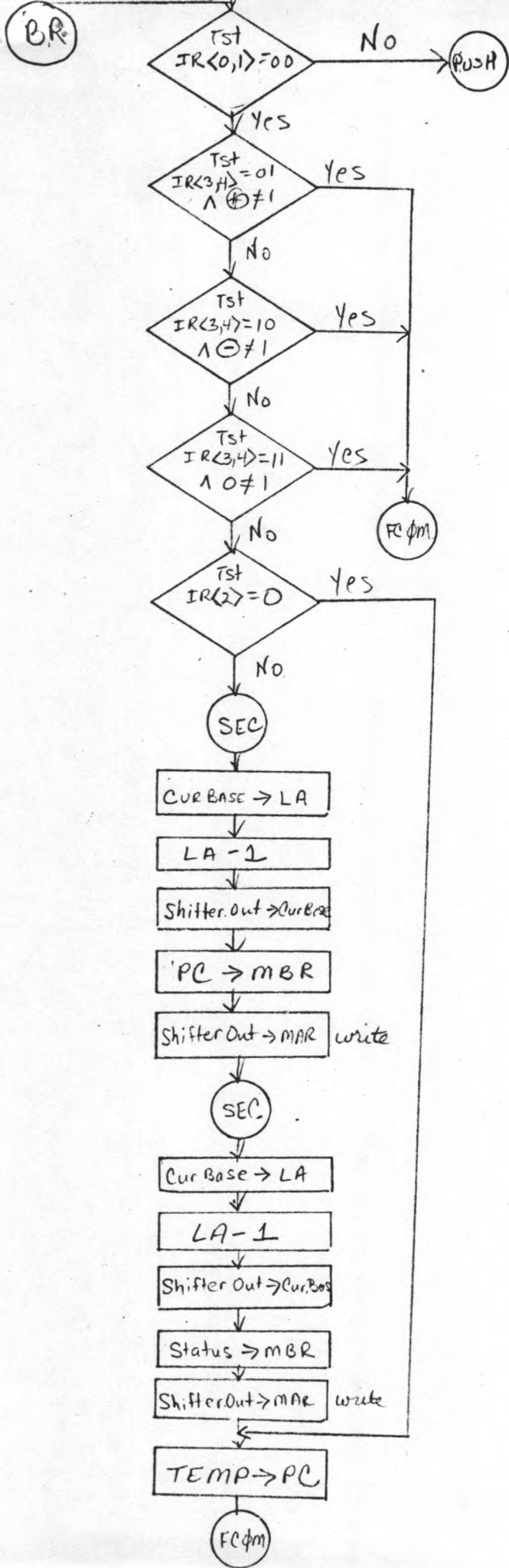
We further certify that the following project is wholly our own work, and is original in our eyes.

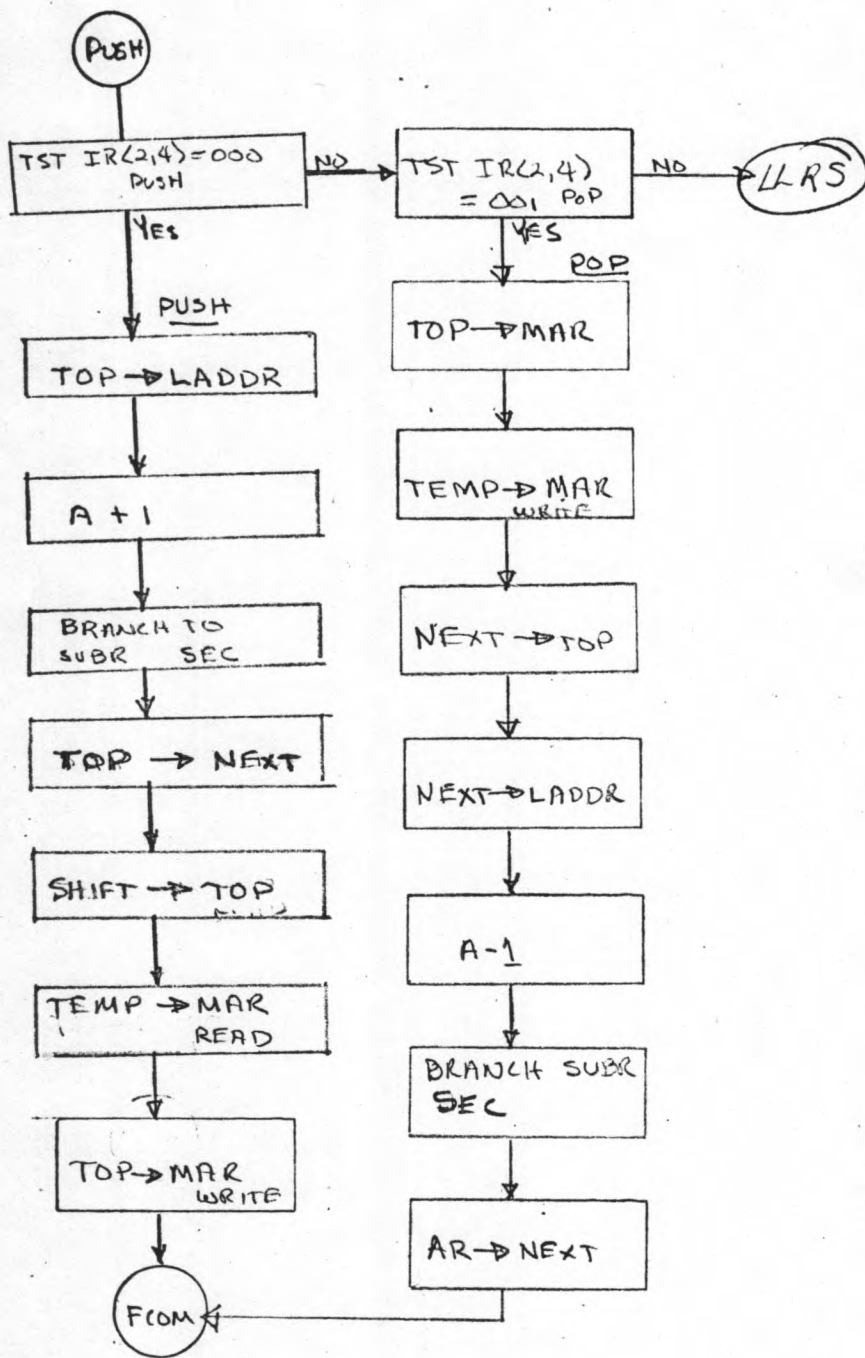


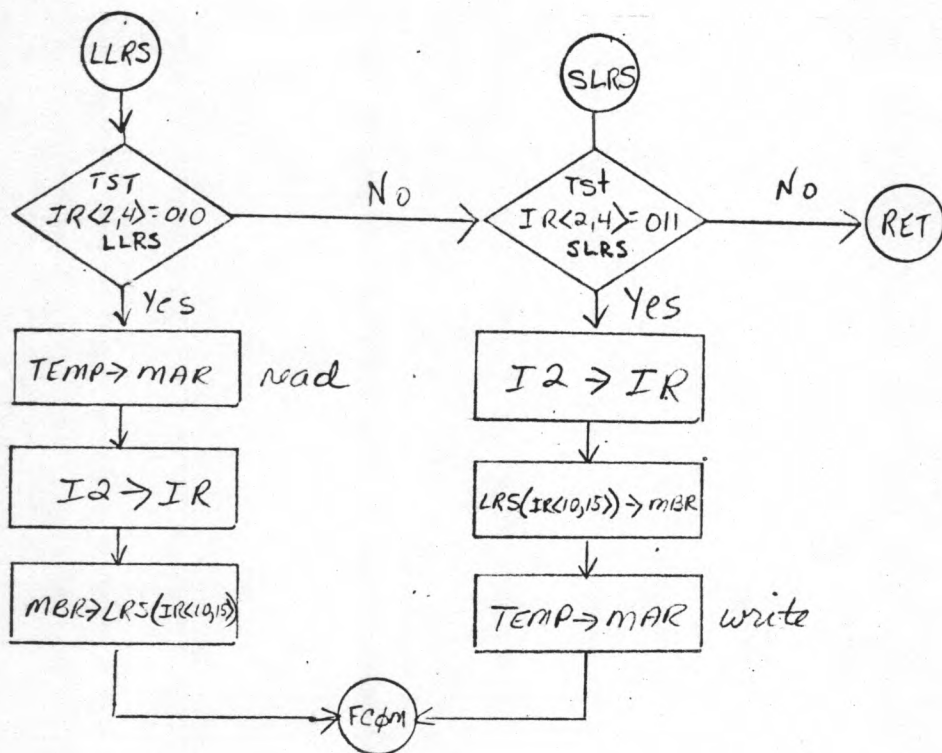
Effective Address Calculation

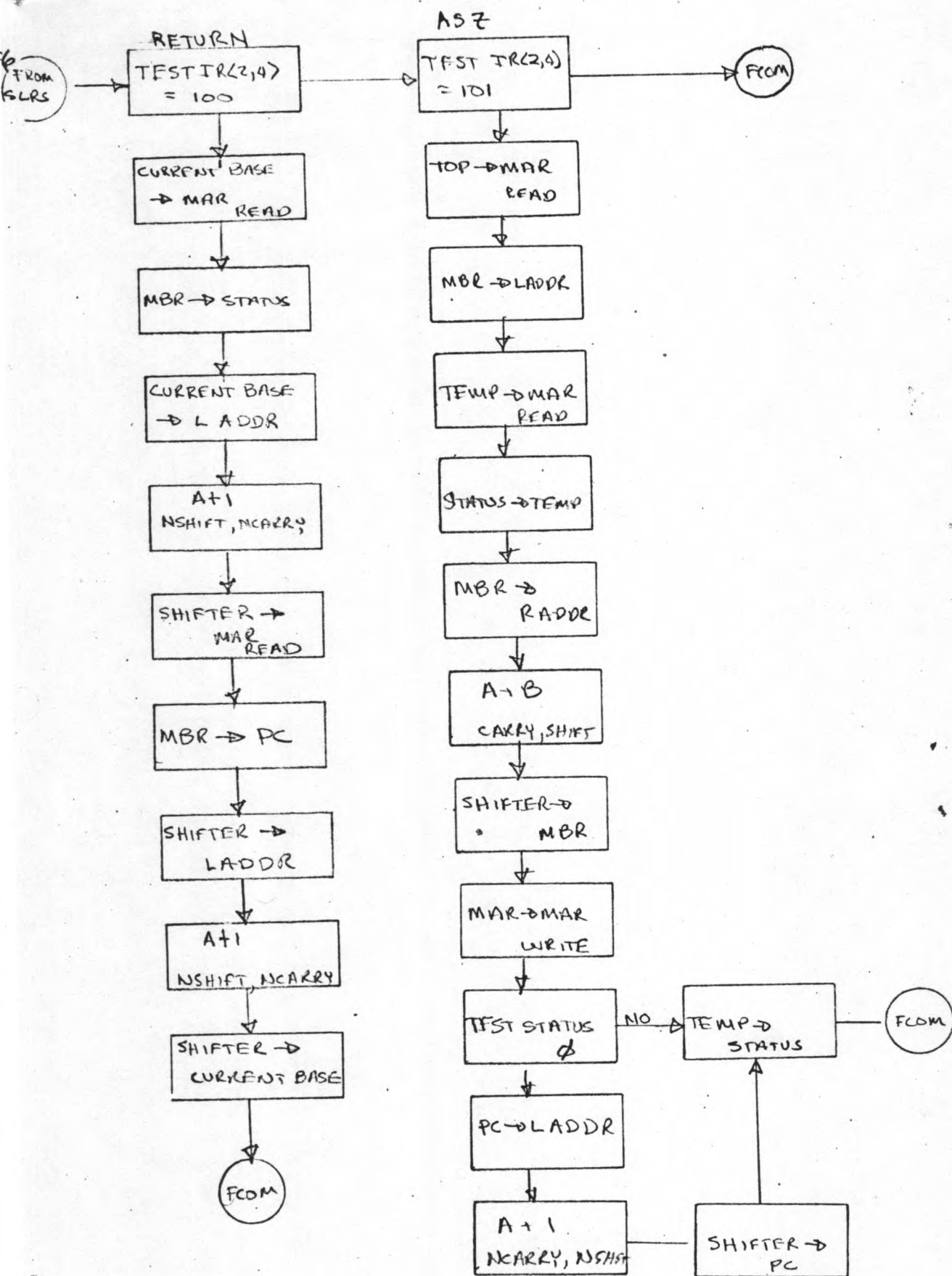












EXF

FROM
CASE
STEPS

STATUS → TEMP
TOP → MAR
READ

MBR → LADDR

I1 → IR
CONST 1

F = A
CARRY, SHIFT

SHIFTER → ADDR
L

I2 → IR
CONST 2

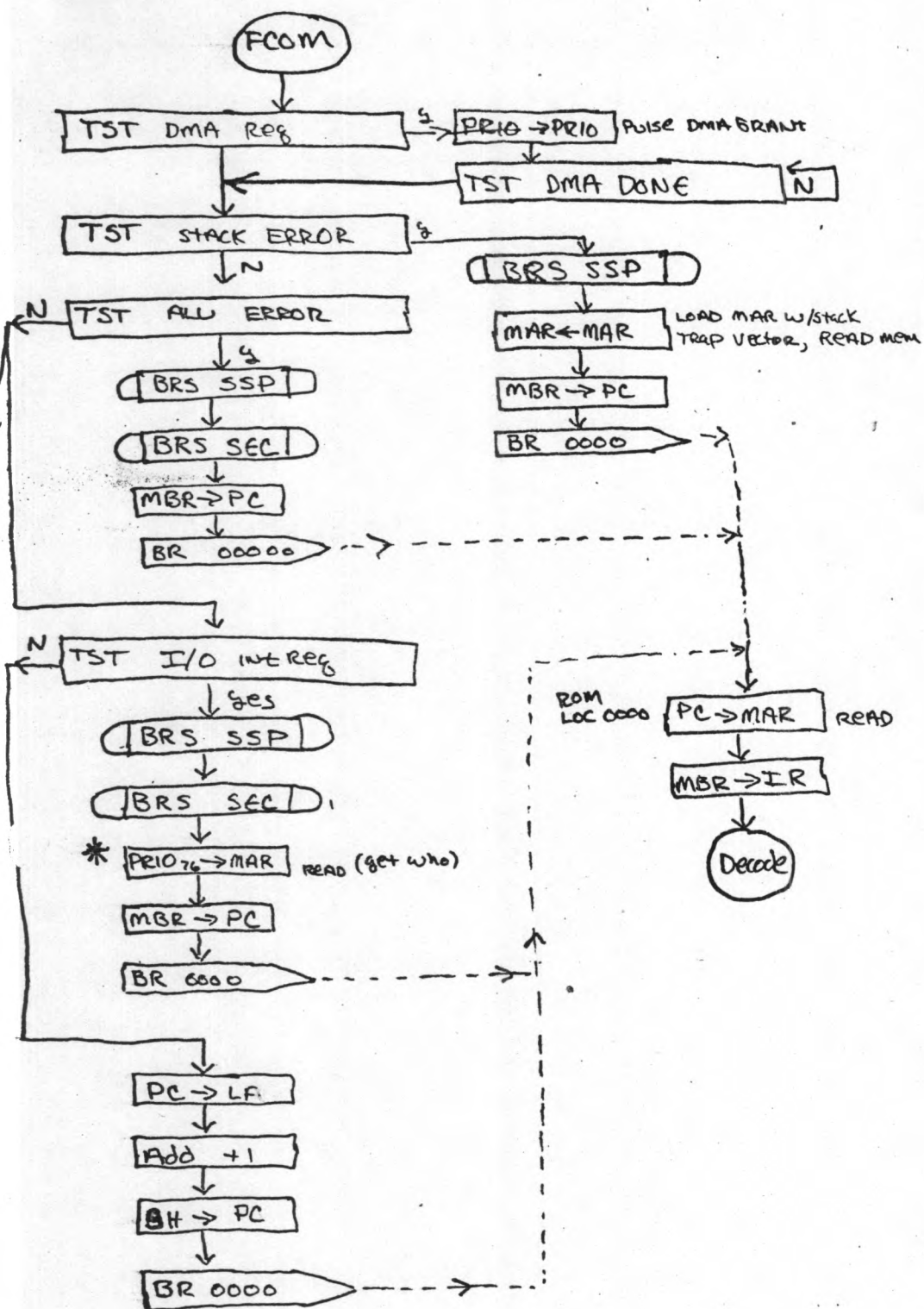
F = A
SHIFT, CARRY

SHIFTER → MBR

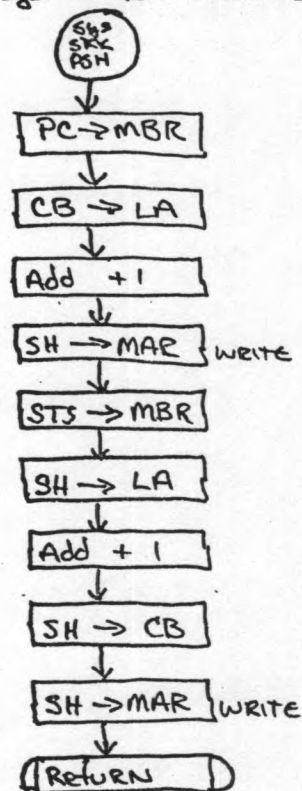
TOP → MAR
WRITE

TEMP → STATUS

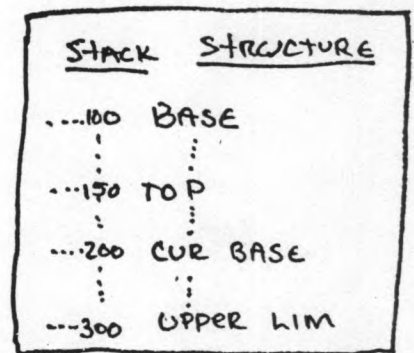
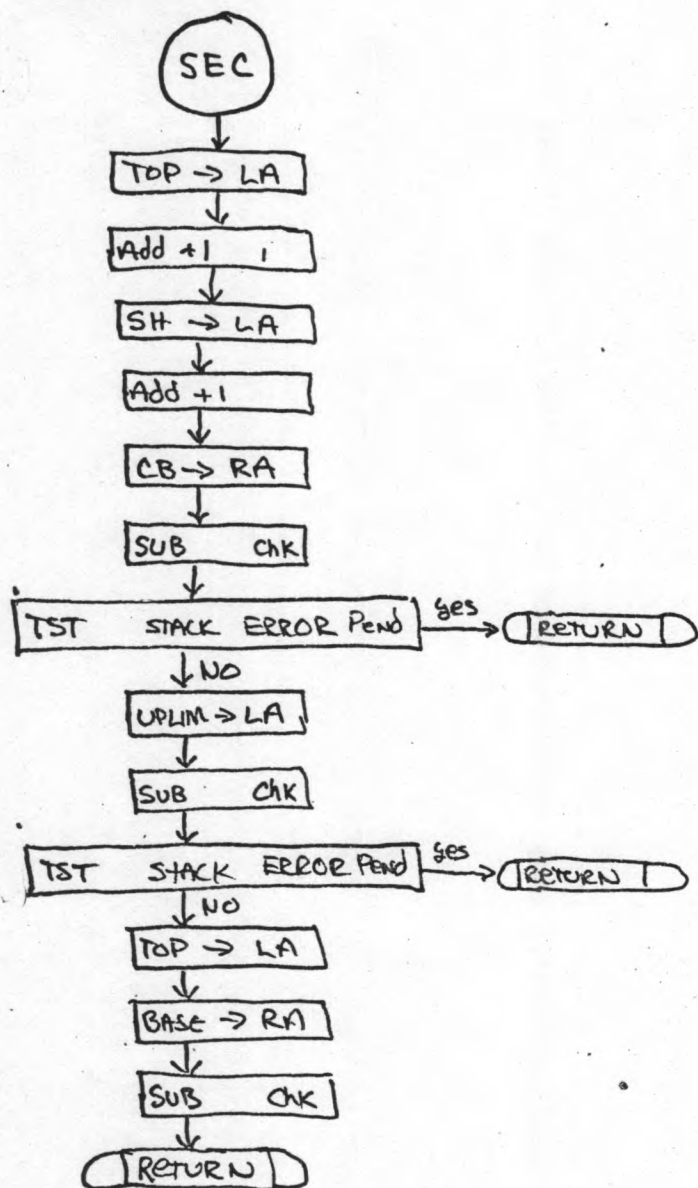
FCOM



S₃. STACK PUSH (SSP)



STACK ERROR CHECK



Processor Major Block Diagram

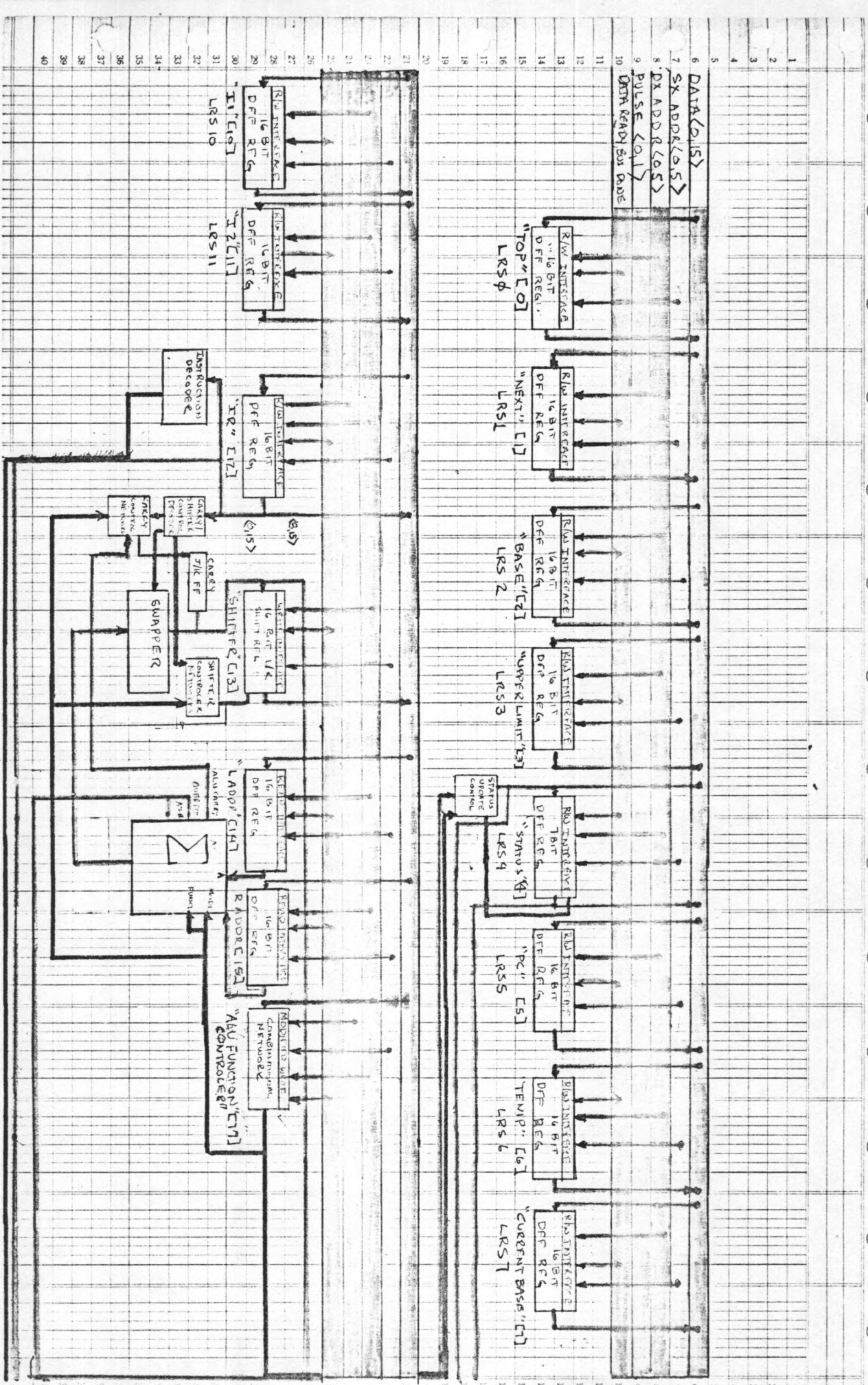
The following two drawings show schematically the modules, and their interconnections, necessary to realise a general purpose processor within our framework. Please

Note the following advantages:

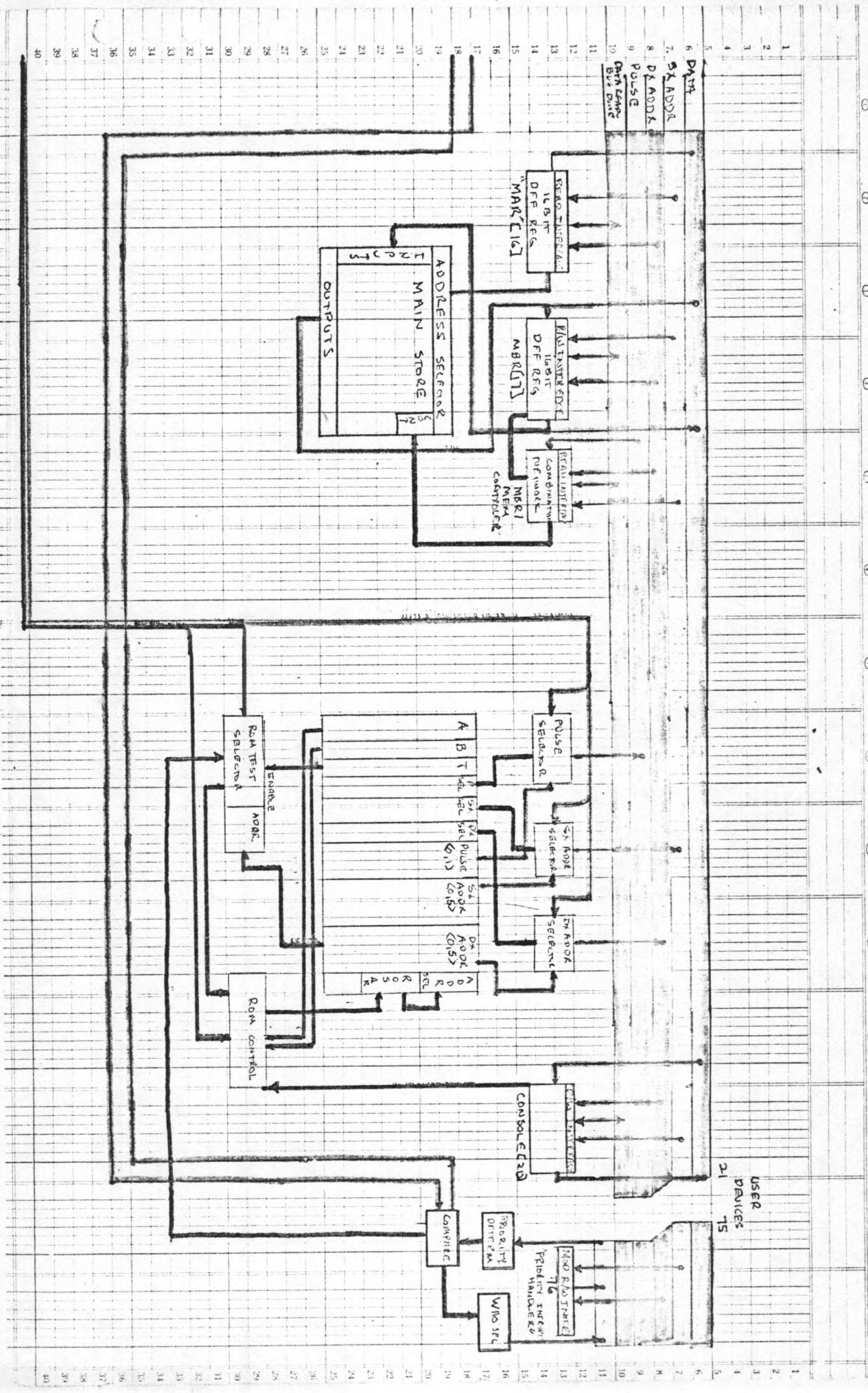
- 1) Modularity - most registers and devices are completely independent of all but Rom control. This allows the buyer to configure his system to fit his needs, and no more.
- 2) The Rom control completely supervises Bus communication. This centralises timing and control problems. In addition, because of micro control, machine characteristics may be completely defined in Micro-code-programming.

COLOR CODES

INTERNAL BUS SIGNALS	DATA <0,15>	██████████
	ST ADDR <0,5>	██████████
	DX ADDR <0,5>	██████████
	PULSE <0,1>	██████████
	DATA READY	██████████
	BUS DONE	██████████
	CONTROL LINES	██████████
	DATA PATHS	██████████
	TIMING SIGNALS	██████████
	HANDSHAKING I/O	██████████
	WHO	██████████
	PRIOR <0,3>	██████████



SYSTEM RESOURCE FLOWCHART



General Register with Read/Write Interface

This drawing shows the control and storage of registers in the processor. Registers are DFF type to satisfy cost, speed and reliability considerations.

The read interface causes parallel loading of the register from $DATA<0,15>$ by sampling the $DX ADDR<0,5>$ lines to determine when it is enabled. False enabling is prevented by checking $SX ADDR<0,5>$ for address 77_e . This rejecting is necessary when the ALU FUNCTION CONTROLLER (ADDRESS 77_e) is selected as the source and $PULSE<0,1>$ and $DX ADDR<0,5>$ are used for ALU control. (see page)

When the register is enabled as the Destination and a timing signal from the ^{SOURCE DEVICE} ~~Port control~~ (DATA READY) are present, data is loaded into the register. AFTER LOADING, BUS DONE IS GENERATED TO CLEAR DATA READY

The write interface controls the output of data from the register to $DATA<0,15>$. The register outputs appear on the $DATA<0,15>$ lines when the register is selected as the source and the timing signal (DATA ~~to~~) from the ~~Port control~~ are present.

DATA READY IS GENERATED TO CLOCK THE DATA INTO THE DESTINATION DEVICE.

STATUS REGISTER

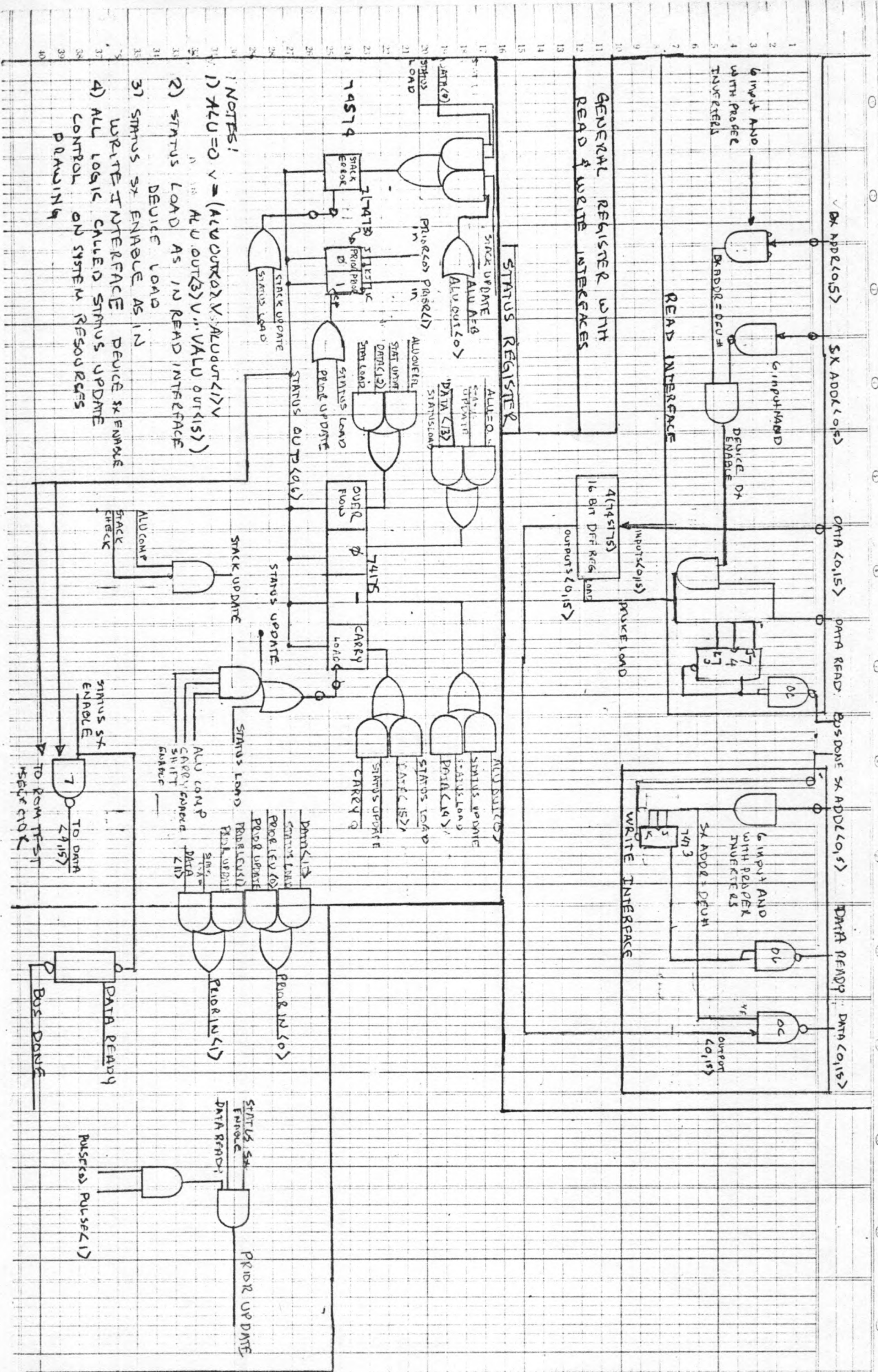
THE STATUS REGISTER STORES THE STATUS OF THE STACK, THE BASIC INFORMATION ABOUT THE PREVIOUS ARITHMETIC OPERATION DONE BY THE ALU, AND THE PRIORITY OF THE PRESENT TASK.

THE STATUS REGISTER MAY BE LOADED IN FULL VIA THE LLRS MACRO INSTRUCTION OR BY ROM CONTROL DIRECTLY. DATA<9,15> IS THE SOURCE OF DATA IN LOADING AND THE DESTINATION OF DATA WHEN WRITING THE CONTENTS OF THE REGISTER TO THE BUS.

STATUS<0> IS THE STACK ERROR BIT. IT IS SET TO LOGIC 1 WHEN EITHER STACK UNDERFLOW OR STACK OVERFLOW IS PENDING. THIS BIT IS TESTED IN FCOMP FIRMWARE ROUTINE AND AN INTERRUPT TRAP INDIRECTLY ON LOCATION 19 IS INITIATED. THIS BIT IS UPDATED VIA A FIRMWARE ROUTINE THAT USES THE ALU A-B LINE AND ALU OUT<0> GATED WITH THE ALU FUNCTION CONTROLLER SIGNAL STACK CHECK (SEE FLOWCHARTS STACK ERROR CHECK)

STATUS<0,1> HOLD THE PRIORITY OF THE PRESENT TASK. WHEN AN INTERRUPT OF HIGHER PRIORITY IS RECEIVED BY THE PRIORITY INTERRUPT HANDLER (DEVICE TG) THE PRESENT TASK IS INTERRUPTED, THE NEW PRIORITY LOADED INTO STATUS AND AN INTERRUPT TO THE INTERRUPTING DEVICE DONE.

STATUS<3,6> ARE UPDATED AFTER EVERY ARITHMETIC OPERATION (THIS IS WHEN THE ALU FUNCTION CONTROLLER SIGNALS CARRY ENABLE AND SHIFTER ENABLE SET ARE LOGIC 1)



INSTRUCTION REGISTER AND IR DECODERS

THE IR IS A 16 BIT DFF REGISTER THAT HOLDS INFORMATION FOR THE IR DECODERS. THESE DECODERS FEED CONTROL SIGNALS TO THE ROM TEST MULTIPLEXER, THE CARRY CONTROL, SWAPPER, LS INPUT CONTROL, AND SHIFT FUNCTION CONTROLLER. DATA MAY BE LOADED FROM $DATA(0,15)$ BY DIRECT ROM CONTROL OR DATA MAY BE LOADED FROM $DATA(12,15)$ AND ONE OF TWO CONSTANTS MAY BE SELECTED. THESE TWO CONSTANTS ARE USED WHEN EXECUTING THE EXTRACT FIELD INSTRUCTION TO ENABLE THE SHIFTER TO SHIFT LEFT AND PAD ϕ (CONST 1) AND SHIFT RIGHT AND PAD ϕ (CONST 2).

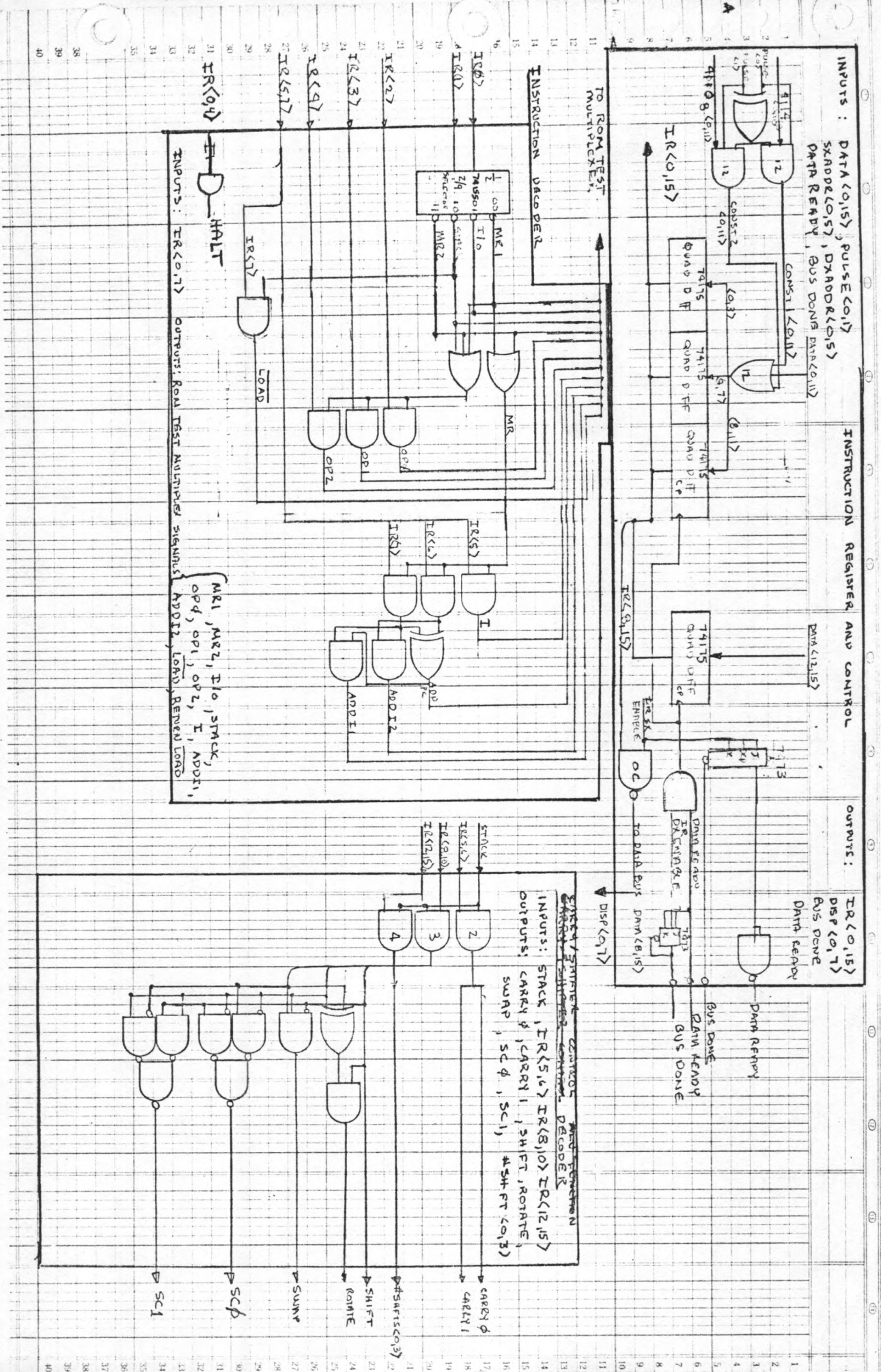
$IR(8,15)$ MAY BE WRITTEN TO THE DATA BUS AS $DISPLACEMENT(0,7)$ FOR EFFECTIVE ADDRESS CALCULATION, BUT ONLY $IR(8,15)$ CAN BE WRITTEN TO THE BUS.

THE INSTRUCTION DECODER DEVELOPS THE FOLLOWING SIGNALS TO ALLOW THE ROM FIRMWARE TO DECODE ^{MACRO} INSTRUCTIONS: $MR1, MR2, STACK, I/O, OP\phi, OP1, OP2$. TO ALLOW EFFECTIVE ADDRESS CALCULATION, THE SIGNALS $ADD PC, ADD I2, ADD I1, AND I$ ARE GENERATED. Halt inst generates signal to disable RUN FF.

THE CARRY / SHIFTER CONTROL DECODER DEVELOPES THE SIGNALS $CARRY\phi$ AND $CARRY1$. THESE SPECIFY THE VALUE OF THE CARRY LOADED INTO THE SHIFTER CARRY FLIP FLOP AFTER AN ALU FUNCTION WITH CARRY ENABLE FROM THE ALU

FUNCTION DECODER ENABLED. THE SHIFTER CARRY FLIP FLOP IS LOADED WITH EITHER THE RESULTING CARRY (00), A ZERO (00), A ONE (01), OR THE COMPLIMENT OF THE RESULTING CARRY (11). THIS VALUE IS ALSO LOADED INTO THE CARRY BIT OF THE STATUS REGISTER AT THE SAME TIME. # SHIFTS (0,3) ARE LOADED INTO THE ~~SHIFTER~~ # SHIFTS CONTROLLER AND IS A MODULO 16 COUNT OF THE NUMBER OF SHIFTS OR ROTATES. SHIFT, ROTATE, AND SWAP SPECIFY THE OPERATION TO BE PERFORMED BY THE SHIFTER. SC0 AND SC1 ARE USED TO DEVELOP THE ACTUAL SHIFT REGISTER CONTROL BITS IN THE SHIFTER FUNCTION CONTROL NETWORK

FOR



NOTES ON R ADDR AND MAR

THE RIGHT ADDR MAY BE CLEARED BY ADDRESSING THE R ADDR AS SOURCE AND DESTINATION AND SETTING $PULSE(0,1)=1$. IN HARDWARE THIS WOULD BE REALISED BY A Φ INPUT AND GATE OF R ADDR SX ENABLE, $PULSE(0)$, ~~AND~~ $PULSE(1)$ AND DATA READY DRIVING THE CLEAR INPUTS OF ALL THE DFF'S.

IN THE MAR, TWO CONSTANTS (20_8 & 19_8) ARE LOADED INTO THE MAR WHEN THE MAR IS SPECIFIED AS THE DESTINATION AND $PULSE(0,1) = 10$ or 11 for constant 1 or constant 2. THIS IS ACCOMPLISHED IN HARDWARE THE SAME WAY AS IN THE IR, WHERE CONSTANTS ARE FORCED TO ENABLE SHIFTING RIGHT / LEFT FOR THE EXTRACT FIELD MACHO INSTRUCTION. IN THE MAR, THESE TWO CONSTANTS ARE USED IN THE REALIZATION OF STACK ERROR AND ARITHMETIC OVERFLOW INTERRUPTS.

ALU / SWAPPER / SHIFTER LOOP

THE ALU FUNCTION IS APPLIED TO THE TWO OPERANDS IN L ADDR AND R ADDR WITH THE OUTPUT FINALLY STORED IN THE SHIFTER AFTER SWAPPING, SHIFTING OR ROTATING. THE DATA PATH IS DETERMINED, ONLY THE ALU FUNCTION AND SOME ENABLING OF THE DIFFERENT PARTS OF THE LOOP NEED BE SPECIFIED. THESE SPECIFICATIONS ARE MADE BY THE ALU FUNCTION CONTROLLER (DEVICE #77). When this controller is enabled as the source, THE DESTINATION BITS (D_{ADDR}(0,5)) AND PULSE BITS (PULSE(0,1)) SPECIFY THE ALU MODE (LOGIC OR ARITHMETIC) THE FUNCTION TO BE IMPLEMENTED (ALU CONTROL(0,3)) AND ^{whether} THE CARRY, SHIFTER; AND STACK ERROR FLIP FLOP IN THE STATUS REGISTER ARE ENABLED. FOR A TRUE ARITHMETIC OPERATION SPECIFIED IN A STACK INSTRUCTION, CARRY ENABLE AND SHIFTER ENABLE SET WOULD BE SET, THUS ALLOWING THE CORRECT CARRY AND SHIFTS SPECIFIED BY THE MARKO INSTRUCTION TO BE PERFORMED. HOWEVER, FOR AN OPERATION SUCH AS INCREMENTING THE PC OR COMPARING TO CHECK FOR A CONDITION THE CARRY ENABLE AND SHIFT ENABLE SET WOULD BE CLEARED, THUS PREVENTING CORRUPTION OF WANTED INFORMATION. IN CHECKING FOR STACK ERROR THE STACK CHECK SHOULD BE SET TO ENABLE COMPARISON INFORMATION FROM THE A=B LINE AND ALU OUT(0) TO BE GATED TO THE STATUS REGISTER

IT SHOULD BE NOTED THAT THE ~~ALU~~ STATUS REGISTER BITS OVERFLOW, ϕ , -, CARRY ARE UPDATED ONLY AFTER A TRUE ARITHMETIC OPERATION (CARRY ENABLE AND SHIFT ENABLE SET). ~~AND~~ ALSO the STATUS BIT FOR STACK ERROR IS UPDATED ONLY IF STACK CHECK IS SET.

THE ALU / SHIFTER LOOP TIMING IS DEFINED IN THE FOLLOWING DRAWING AND IS THE MODEL FOR THE MEMORY AND THE ROM CONTROLLER. SYNCHRONOUS DELAYS AND PULSES ARE GENERATED BY GATING DIVIDED CLOCK PULSES TO PRODUCE THE DESIRED SIGNAL. PLEASE EXAMINE THE CLOCK PULSE DIAGRAM, CLOCK GENERATOR AND th AND GATES ON THE DRAWING. ASYNCHRONOUS DELAYS ARE GENERATED BY CASCADED OR SET CLEAR FLIP/FLOPS THAT ARE SET BY SYNCHRONOUS SIGNALS, BUT CLEARED BY ASYNCHRONOUS SIGNALS. PLEASE EXAMINE THE DRAWING.

Timing Signal Description

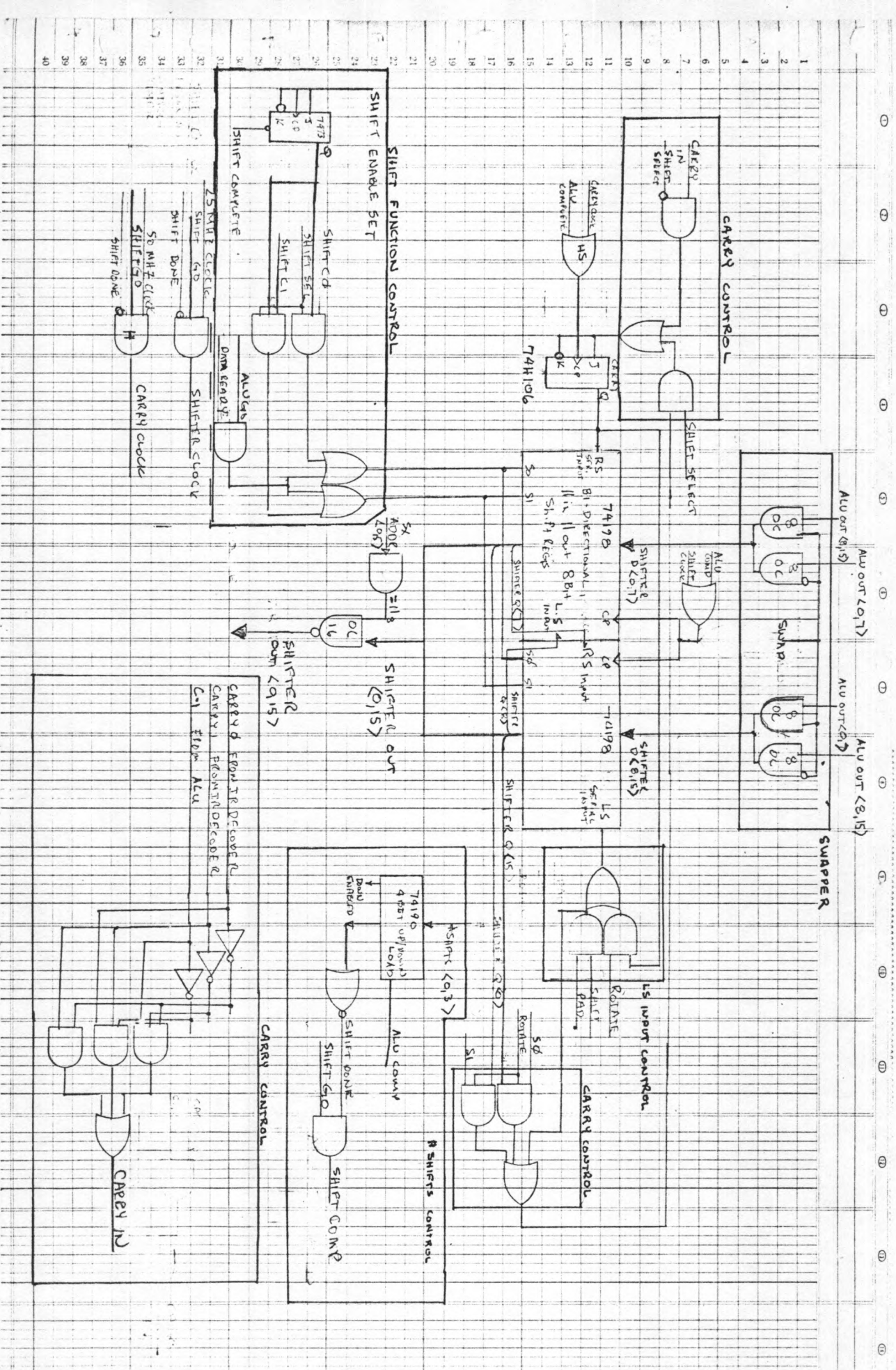
1 AT $t=0$ THE SXADDR, DXADDR AND PULSE BITS APPEAR ON THE BUS. AFTER SEVERAL GATE DELAYS AND A F/F PROPAGATION DELAY, DATA READY COMES UP FROM THE ALU FUNCTION CONTROLLER, ALONG WITH THE ALU FUNCTION CONTROL SIGNALS. THESE SIGNALS ARE ROUTED TO THE PROPER INPUTS. ALLOWING 80 ns FOR THESE ACTIONS AND FOR THE ALU ^{CONTROL} INPUTS TO STABILISE, THE SYNCHRONOUS SIGNAL ALU GO IS GENERATED. THIS SIGNIFIES THE TIME NEEDED FOR THE FUNCTION

SPECIFIED BY THE ALU MODE AND FUNCTION BITS TO BE PERFORMED, AND THE OUTPUTS (ALU OUT(0,15), ALU CARRY) TO PROPAGATE THROUGH THE SWAPPER AND THE CARRY CONTROL CIRCUITRY. THE ~~DATA~~ OUTPUTS ALU OUTPUT(0,15) AND THE MACRO DEFINED CARRY ARE LOADED INTO THE SHIFTER AND CARRY FLIP FLOP WITH ALU COMPLETE. THE STATUS REGISTER IS ALSO UPDATED WITH ALU COMPLETE IF CARRY ENABLE AND SHIFT ENABLE SET WERE LOGIC 1. NEXT THE SYNCHRONOUS PULSES SHIFT ENABLE SET AND SHIFT GO SET ARE GENERATED WHICH CAUSE THE PROPER SIGNALS TO BE FORCED TO THE SHIFT REGISTER CONTROL INPUTS, THE LS INPUT CONTROLLER, THE SHIFT FUNCTION CONTROLLER AND THE CARRY CONTROLLER, THEN ALLOW THE CLOCKS TO START THE SHIFTER SHIFTING OR ROTATING. THE # OF SHIFTS ~~UP/DOWN~~ DOWN COUNTER (WHICH IS ALSO, LOADED AT TIME ALU COMP FROM #SHIFTS(0,3)) THEN STARTS COUNTING DOWN UNTILL THE SPECIFIED # OF SHIFTS HAS BEEN COMPLETED AND THE OR OF ALL THE BITS IS ZERO. THIS SIGNALS THE SHIFT COMPLETE SIGNAL WHICH DISABLES THE CLOCKS AND THE SHIFTING STOPS. SHIFT COMPLETE ALSO SETS THE TASK DONE FF WHICH THEN WAITS FOR THE ROM CONTROL TO CLARK IT AND GO ON TO THE NEXT MICRO INSTRUCTION FETCH.

ALU 1



CHAPTER



ALL 3

RØM

The RØM microprocessor controls most timing, data transfers and general use of our machine. The RØM timing is asynchronous, dependent upon when the presently executing micro-instruction is completed. An interesting use of the RØM is to perform most of the decoding of the macro-instruction being executed. This allows a generalized flowchart which branches to more specific routines when necessary. Thus, a large amount of duplicate microprogramming is avoided. For example, all memory reference instructions will step through the same physical effective address calculation micro-instructions, and then they branch away as needed to complete their specified functions. This brings up the question of where the program begins, since the actual macro-instructions are never loaded into the RØM from the instruction register. When RESET is set, the address 000000 is loaded into the read-only storage address register (RØSAR) and this address, when executed, starts into the fetch of the next (in this case the first) macro-instruction. Another interesting feature is the RØM's ability to select the pulse, source, and destination bits from either the RØM micro-instruction or the instruction register. This gives the programmer some control of the Unibus lines via the microprocessor.

There are 5 types of micro-instructions.

- 1) JSR - saves the current RØS address in the SUB REG; the branch to a subroutine, if one is desired, will be the next micro-instruction; This micro-instruction can also be used

for looping;

- 2) RTN - loads the current address in the SUB REG into the $R\phi SAR$; then $R\phi SAR \leftarrow R\phi SAR + 1$;
- 3) BRANCH - unconditional branch to the address specified by Sx addr and Dx addr bits;
 $R\phi SAR \leftarrow Sxaddr, Dxaddr$
- 4) TEST - performs a test on a condition specified by $Dx \langle 1, 5 \rangle$; A conditional branch can be executed by combining the TEST instruction with a BRANCH instruction;
If the test is true the branch will be made;
- 5) NORMAL - this enables the PULSE, SOURCE and DESTINATION selectors and address bits; the P, Sx , and Dx selector bits determine whether the pulse, source & destination bits are selected from the $R\phi M$ or from the instruction register.

The ability to decode macro-instructions through the $R\phi M$ micro-instructions is dependent on the $R\phi M$ TEST MULTIPLEXER. This is a 32-line to 1-line multiplexer with a variety of input lines. The input lines are taken from the IR, the Status Register, and the Unibus control lines. and These many varied test lines provide the $R\phi M$ with very flexible control of the machine.

Data Input Lines to the RDM Test Multiplexer

MR1
MR2
 $\pm/\phi$
Stack
 $\phi P0$
 $\phi P1$
 $\phi P2$
 $\overline{I}$
ADDI₁
ADDI₂
 $\overline{LQAD}$
RETURN $\overline{LQAD}$

} from the instruction register

Stack error
Priority 0
Priority 1
Overflow (Arith.)
0
—
Carry

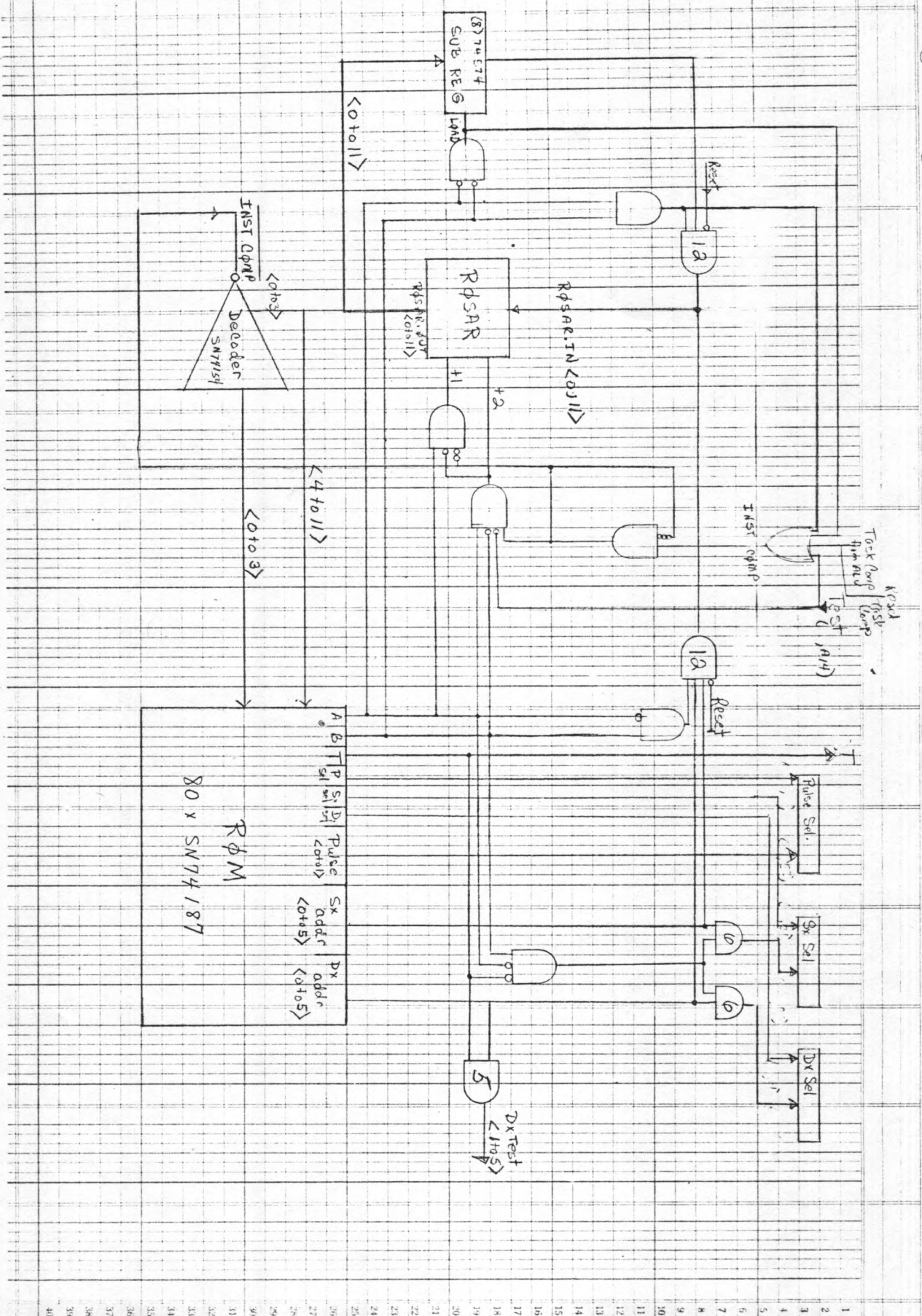
} from the Status Register

INT ACK
DMA REQ
DMA DONE

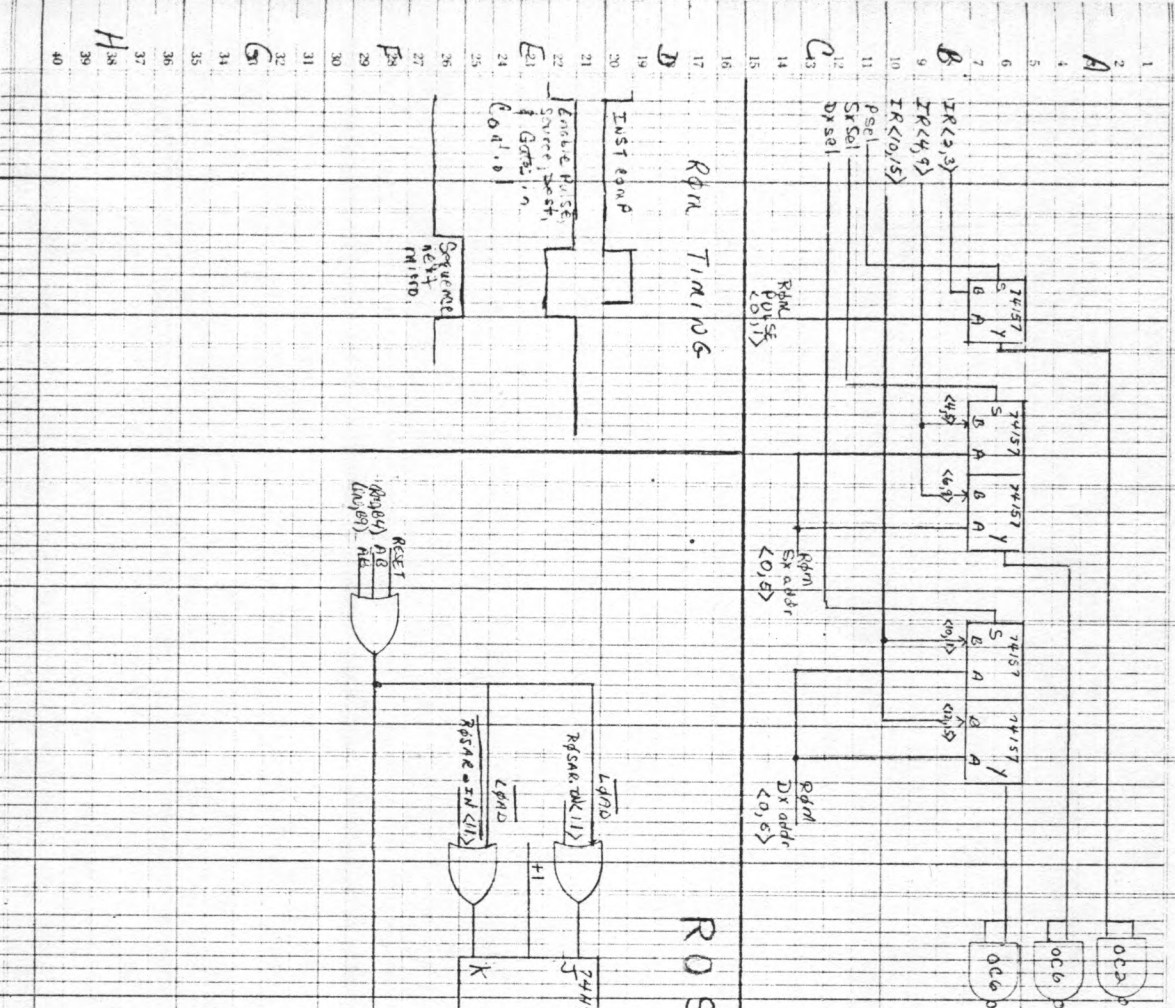
$I \langle 6 \rangle \wedge I \langle 7 \rangle$

RFM Module Diagram

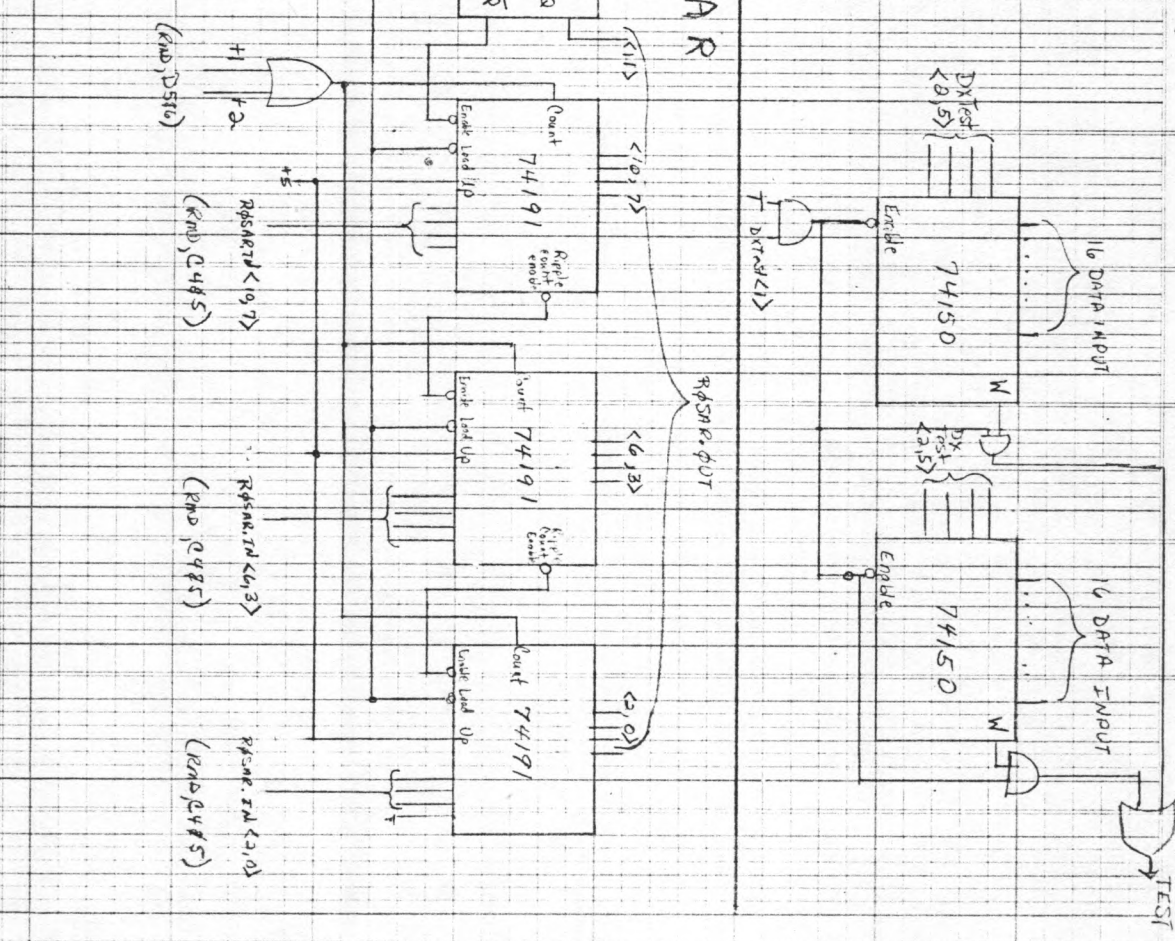
7 m D



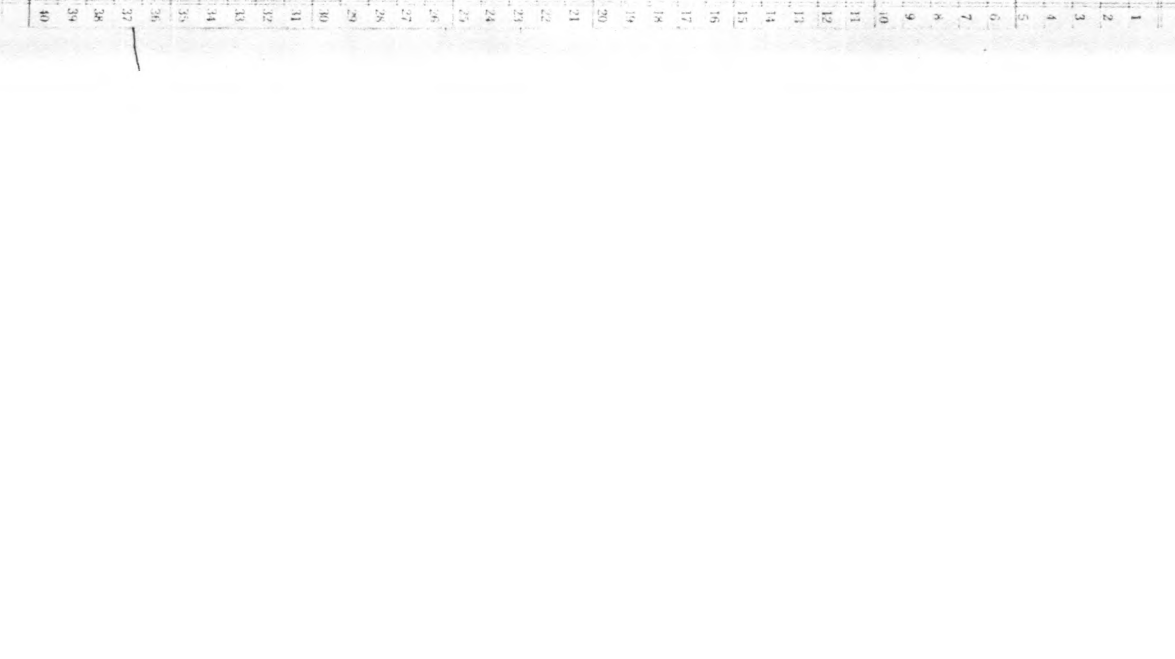
PULSE, SOURCE, AND DESTINATION SELECTORS



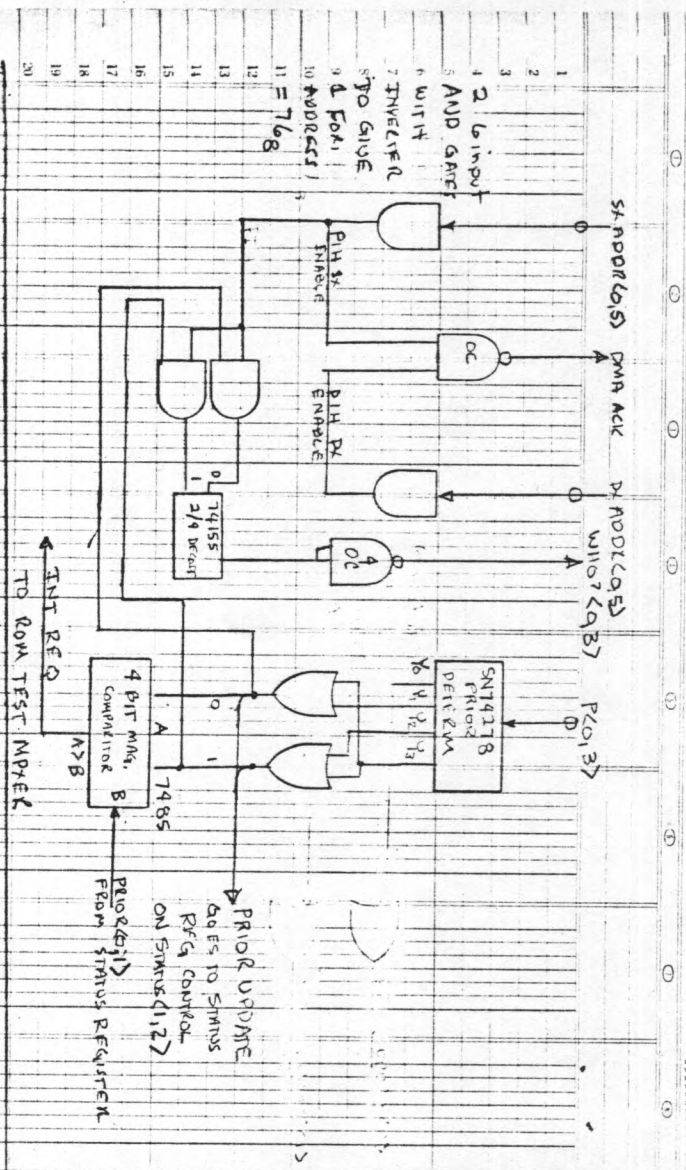
RPN TEST MULTIPLEXER

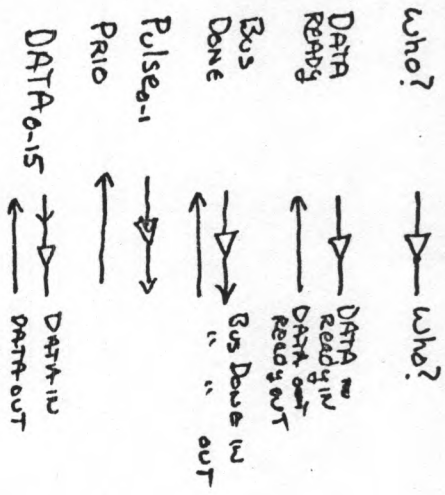
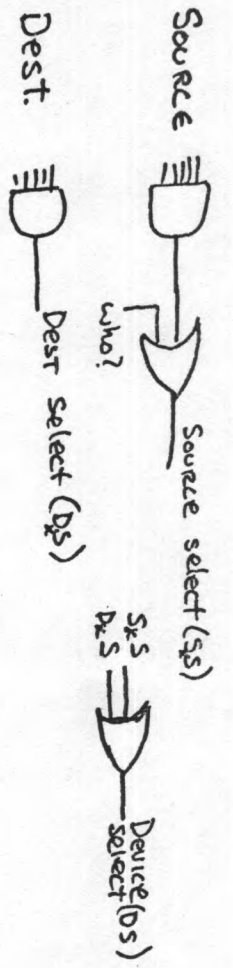


ROSAR



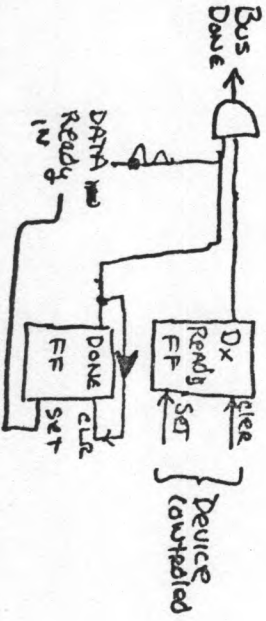
PRIORITY INTERRUPT HANDLER



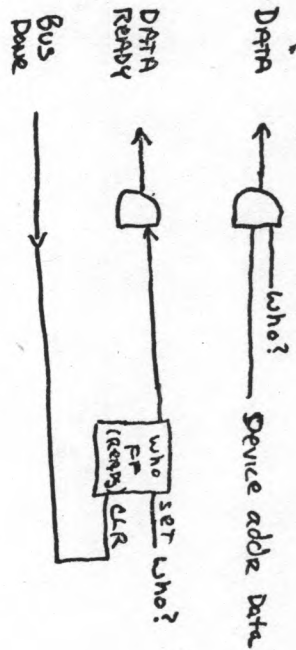
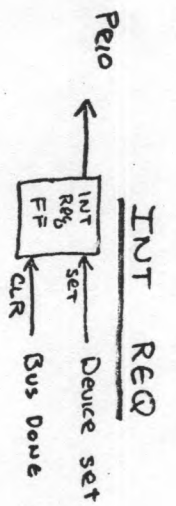


I/O BUS CONNECTIONS

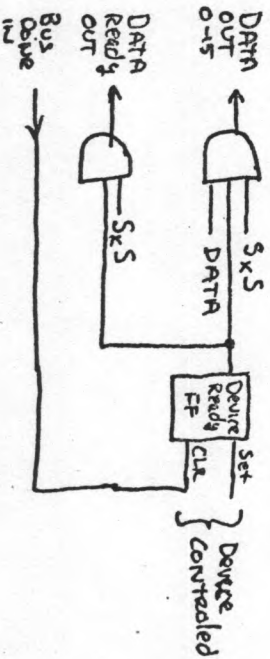
Device as Dest



EXAMPLE I/O INTERFACES



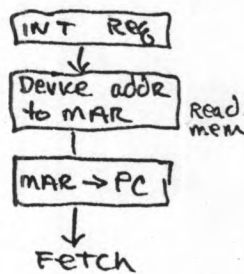
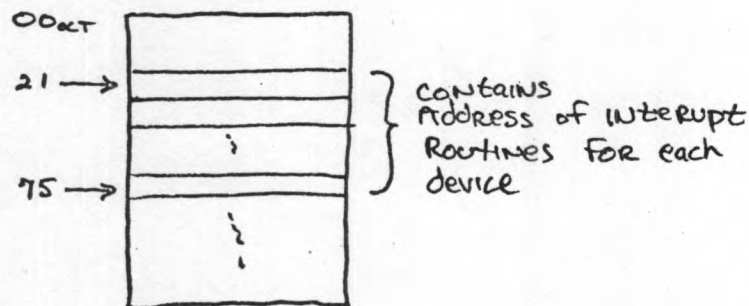
Device as Source



Priority Interrupt HANDLER (PIH) Bus Addr 76_{oct}

The four priority lines are fed into the PIH directly into a priority decoder. The priority decoder ~~selects~~ ^{selects} the priority line which has the highest priority and sends a singular 4 bit number to a 4 to 2 incoder. The incoder output is then compared to the current processor priority in the status register. If the interrupting device priority is higher the interrupt ~~acknowledge~~ ^{request} is sent to the Rom test selector. In FCOM the INT Reg is tested for and upon finding a interrupt request, saves the current PC and status upon the system stack. The ^{new} Priority is ~~then~~ loaded into the status register at this time. The ROM then sends out the PIH address as the source and the MAR as the dest. The PIH transfers its bus ^{source} control to the appropriate device by use of the who? Line. The device upon receiving a who? signal places its Bus address ~~at~~ out on the data lines, then like a normal Bus data transfer, it sends out a data ready signal, the MAR as the dest. sends back the Bus done signal. ~~reg~~ When the MAR receives the data, memory is read. getting the device interrupt vector stored at the device Bus address in main memory. FCOM then places the interrupt vector into the PC ~~and~~ then does a fetch of the next micro inst.

IF the PIH is invoked as both the source and the dest, a DMA acknowledged signal is generated.



BUS HANDSHAKING

All data transfers are done across a common BUS asynchronously.
This is done in the following manner:

- 1) Rom control sends out source and dest.
- 2) Source puts out data
- 3) Source puts out data ready
- 4) Dest reads DATA
- 5) Dest puts out Bus Done
- 6) Source and Dest "disconnect" from Bus
- 7) Rom cycles to next micro inst.

