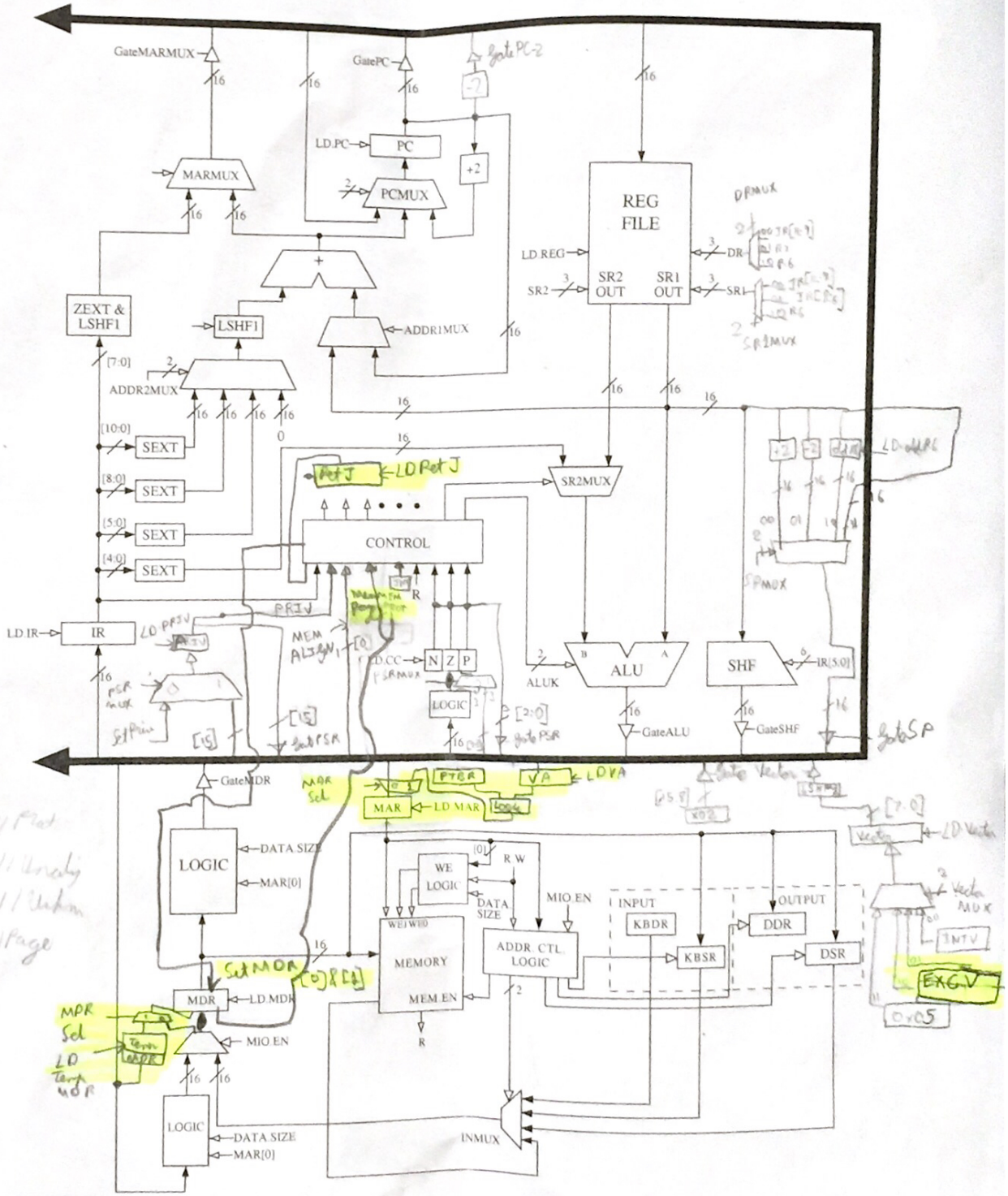
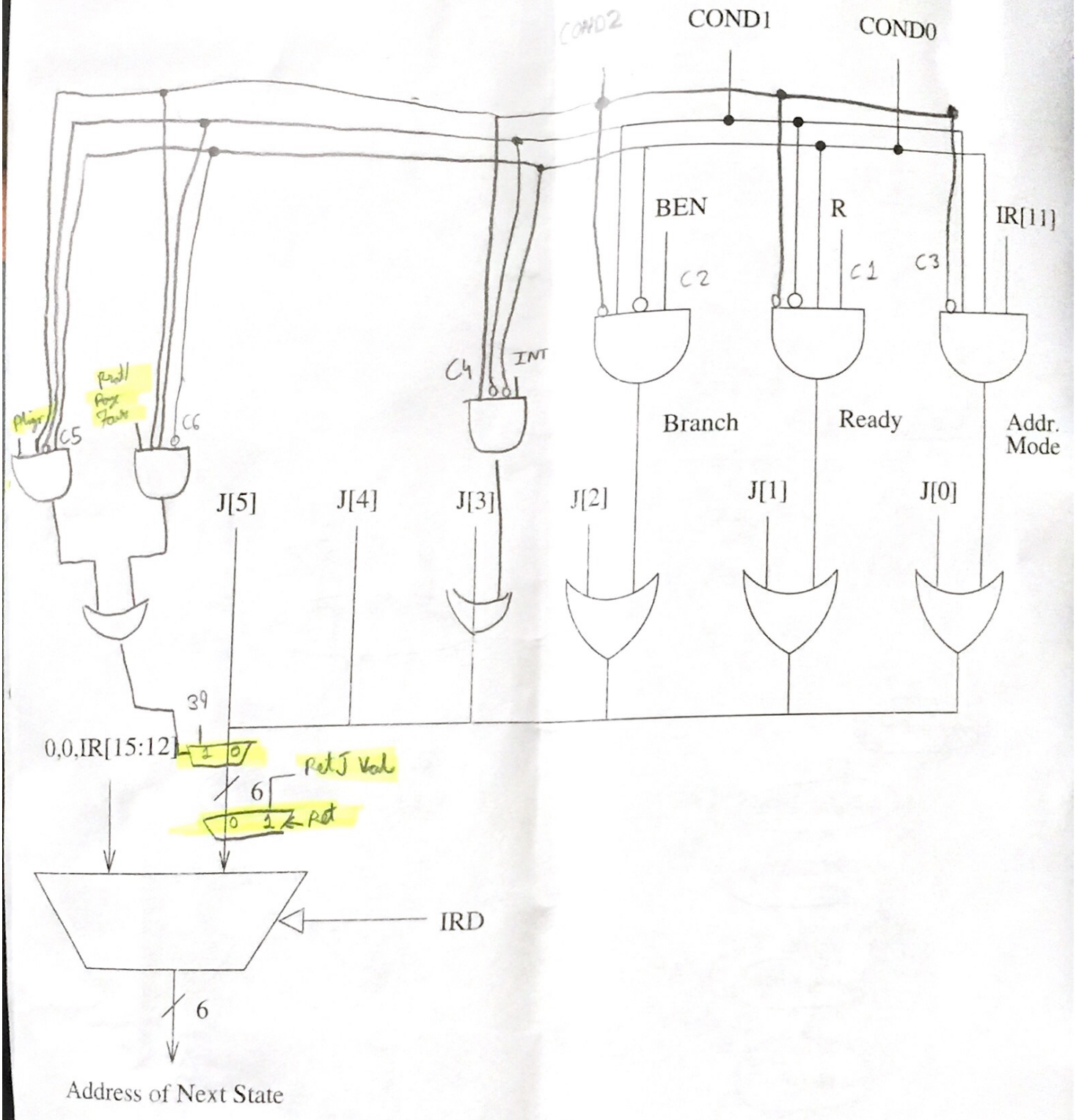
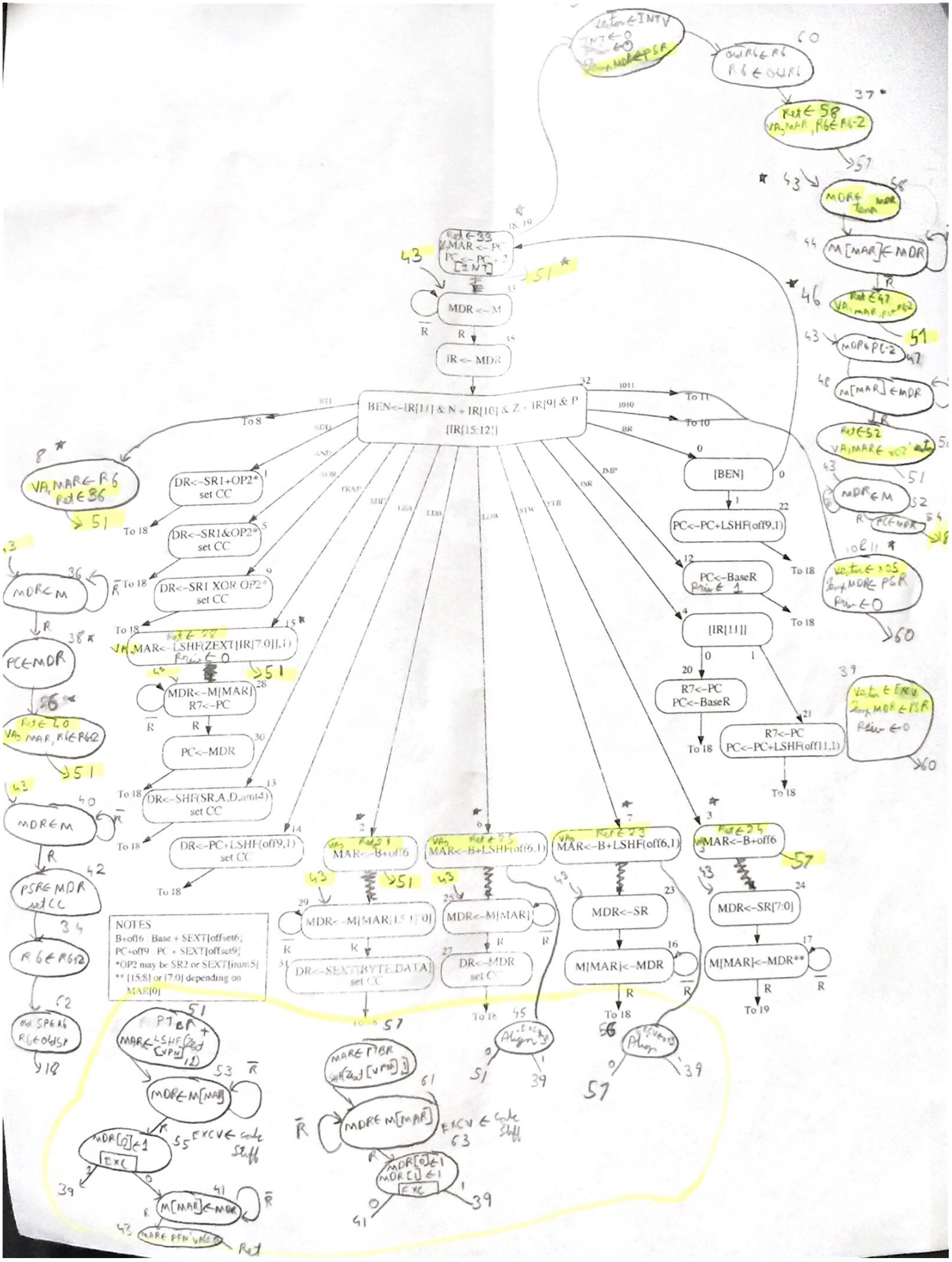

GateMARMUX
GatePC
LD.PC
PC
PCMUX
MARMUX
+2
REG FILE
LD.REG
SR2 OUT
SR1 OUT
DR
SR1
SR2
ZEXT & LSHF1
LSHF1
ADDR1MUX
ADDR2MUX
[7:0]
[10:0]
[8:0]
[5:0]
[4:0]
SEXT
SR2MUX
CONTROL
PRIV
LD.IR
IR
N
Z
P
LD.CC
ALU
SHF
ALUK
GateALU
GateSHF
IR[5:0]
LOGIC
GateMDR
MAR
LD.MAR
DATA.SIZE
MAR[0]
WE
R.W
MIO.EN
MEMORY
MEM.EN
ADDR_CTL LOGIC
INPUT
OUTPUT
KBDR
KBSR
DDR
DSR
MDR
LD.MDR
INMUX


COND2
COND1
COND0
BEN
R
IR[11]
C2
C1
C3
C4
INT
C5
C6
Branch
Ready
Addr. Mode
J[5]
J[4]
J[3]
J[2]
J[1]
J[0]
39
0,0,IR[15:12]
6
IRD
6
Address of Next State


BEN<-IR[11] & N + IR[10] & Z + IR[9] & P
[IR[15:12]]
MDR<-M
IR<-MDR
MDR<-M
PC<-MDR
MDR<-M[MAR]
R7<-PC
PC<-MDR
MDR<-M
PSR<-MDR setCC
R6<-R6+2
M[MAR]<-MDR
M[MAR]<-MDR
M[MAR]<-MDR
MDR<-M[MAR]
MDR<-M[MAR]
MDR<-SR
MDR<-SR[7:0]
M[MAR]<-MDR**
[BEN]
[IR[11]]
DR<-SR1+OP2* set CC
DR<-SR1&OP2* set CC
DR<-SR1 XOR OP2* set CC
DR<-SHF(SR,A,D,amt4) set CC
DR<-PC+LSHF(off9,1) set CC
DR<-SEXT[BYTE.DATA] set CC
DR<-MDR set CC
PC<-PC+LSHF(off9,1)
PC<-BaseR
R7<-PC
PC<-BaseR
PC<-PC+LSHF(off11,1)
MAR<-B+off6
MAR<-B+LSHF(off6,1)
MAR<-B+LSHF(off6,1)
MAR<-B+off6
MAR<-LSHF(ZEXT[IR[7:0]],1)
VA,MAR<-R6 Ret<-36
VA,MAR,Ret<-58 R6<-R6-2
Ret<-33 VA,MAR<-PC PC<-PC+2
Ret<-47 VA,MAR,R6<-R6-2
Ret<-52 VA,MAR<-R6 R6<-R6+2
Ret<-28 VA,MAR<-LSHF(ZEXT[IR[7:0]],1) Privin<-0
Ret<-20 VA,MAR,R6<-R6-2
Ret<-23 VA,MAR<-B+LSHF(off6,1)
Ret<-24 VA,MAR<-B+off6
Vector<-INTV INT<-0 Temp<-MDR PSR
Vector<-x05 Temp<-MDR PSR Privin<-0
Vector<-EXCV Temp<-MDR PSR Privin<-0
Align
Align
PTBR + MAR<-LSHF(ZEXT[VPN],12)
MDR<-M[MAR]
MDR[0]<-1
EXC
M[MAR]<-MDR
MAR<-PTBR + LSHF(ZEXT[VPN],3)
MDR<-M[MAR]
MDR[0]<-1
MDR[1]<-1
EXC
EXCV<-code stuff
EXCV<-code stuff
Ret
NOTES
B+off6 : Base + SEXT[offset6]
PC+off9 : PC + SEXT[offset9]
*OP2 may be SR2 or SEXT[imm5]
** [15:8] or [7:0] depending on MAR[0]

Lab 4 Readme Lab Report
Dhruv Sandesara (djs3967)

1. Changes you made to the state diagram. Include a picture similar to the state machine that shows the new states you added (only show your changes or mark your changes in a new state diagram). This picture should include the encodings of new states you added. Clearly show where each state fits in the current state diagram. Describe what happens in each new state.

2. Changes you made to the data-path. Clearly show the new structures added, along with the control signals controlling those structures. Describe the purpose of each structure.

3. New control signals you added to each microinstruction. Briefly explain what each control signal is used for.

4. Changes you made to the micro-sequencer. Draw a logic diagram of your new micro-sequencer and describe why you made the changes.

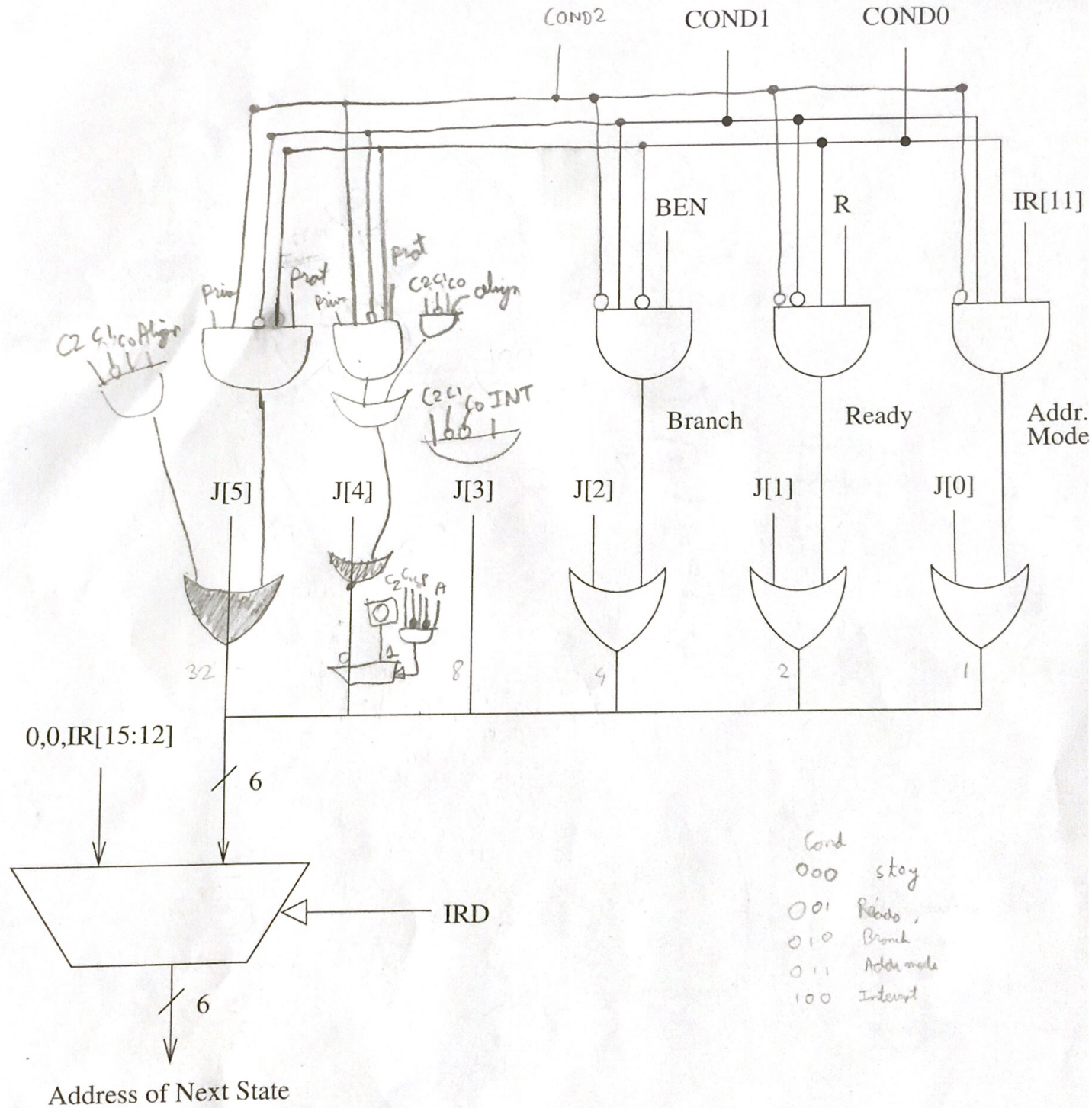
1) Ans: The picture of the state diagram is included above. All the changes made are made in pencil. The changes are as follows:
    1) Made a change in the TRAP 15 state to set PRIV to 0 so the next access to restricted mem is allowed
    2) Made a change to JMP State 12 to set priv to 1 so that execution can return from trap to user mode
    3) Changed State 2 and 3 to go to state 43 and 58 which do the same thing of checking MEmPROT and either continuing normal execution or trigger MemProt exception named by a common state name of 49 55 56 57 and 61
    4) Changed state 6 and 7 to go to state 45 and 53 which basically check for both mem align and mem prot exception. If non occur go to base normal execution in state 25 and 23 or go to common mem align case if only align error occurred in state number 39 and 41. If Mem Pro occurs go the the common state labeled in bullet point 3
    5) Also changed state 18 and 19 to check if an interrupt has occurred. If so then go to state 59 which is common interrupt loader case or go to state 51 which checks if mem protection error has occurred similar to bullet point 3
    6) In State 51 check if mem prot occurred if not continue normal execution to state 33 or go to common mem prot case of state 49 and all similar equivalent values
    7) These conclude the changes we made to existing state. Now we show the new states.
    8) When RTI occurs go to state 8 which loads the mar with r6 or SP
    9) Then load the top value in state 36 which when ready goes to state 38
    10) State 38 takes popping the top stack value and puts it into PC.
    11) Then in state 63 increment R6 by 2 to finish popping off value and persist the changes in MAR and R6.
    12) Now read the value from mem again in state 40 and when done go to state 42.
    13) In state 42 take the MDR and load up the value of the PSR to complete the context switch and return the execution to the interrupted programs context.
    14) We also set CC in this state to make the changes persist
    15) Next go to state 34 to decrement r6 by 2 to make sure the sp points to the post popped off state
    16) The we swap values of R6 with Old R6 which basically swaps the user and supervisor stack pointer and finished context switch to return to state 18
    17) Now let's talk about the common exception states.
    18) For state 59 which handles the interrupt case. We load the vector with the Interrupt Vector. It also puts the PSR in the MDR so we can be ready for context switching. Also

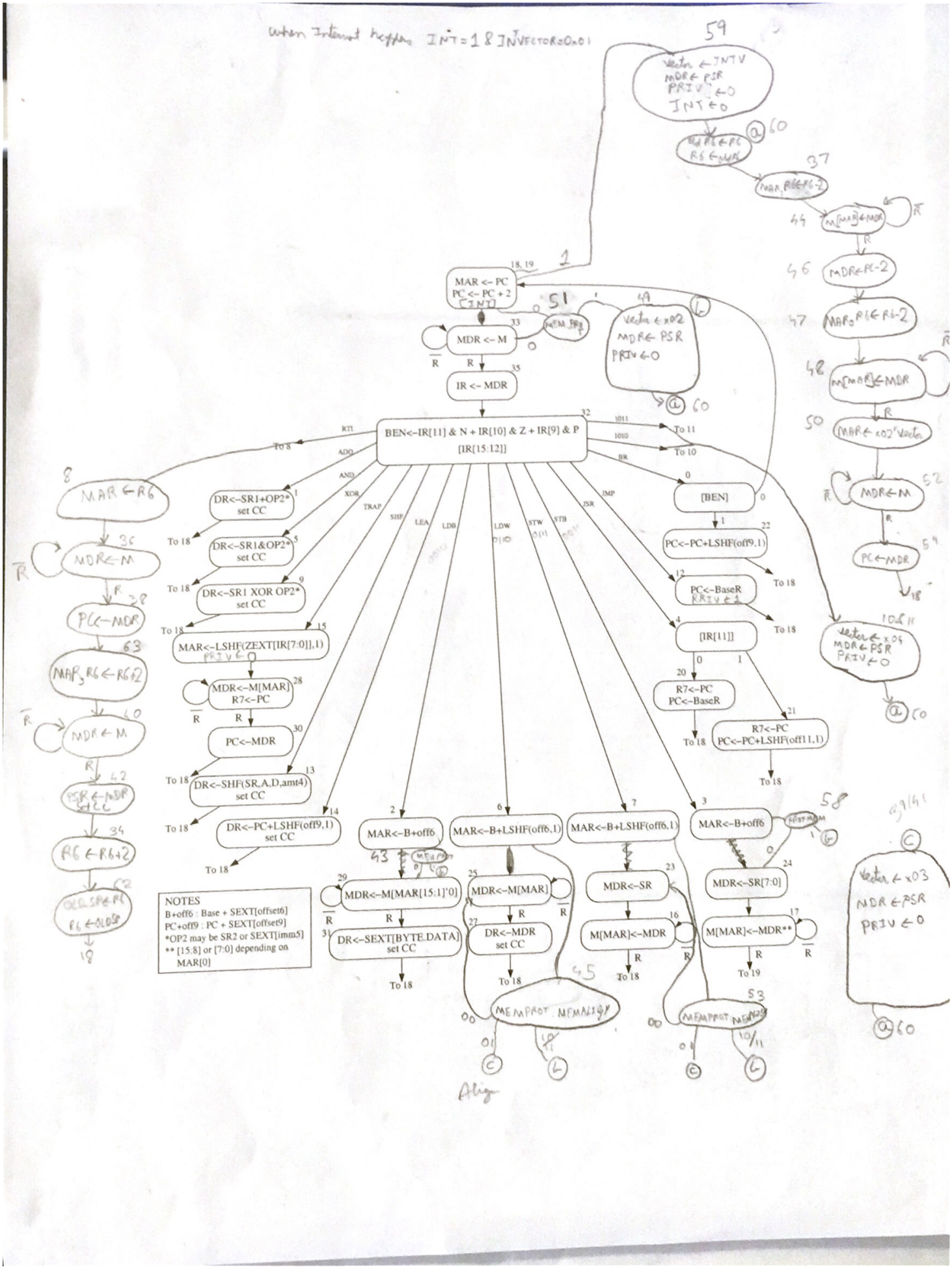
we set the Priv to 0 to make our next executions occur in supervisor mode. And we also clear the interrupt bit to signify that we have handled the interrupt.

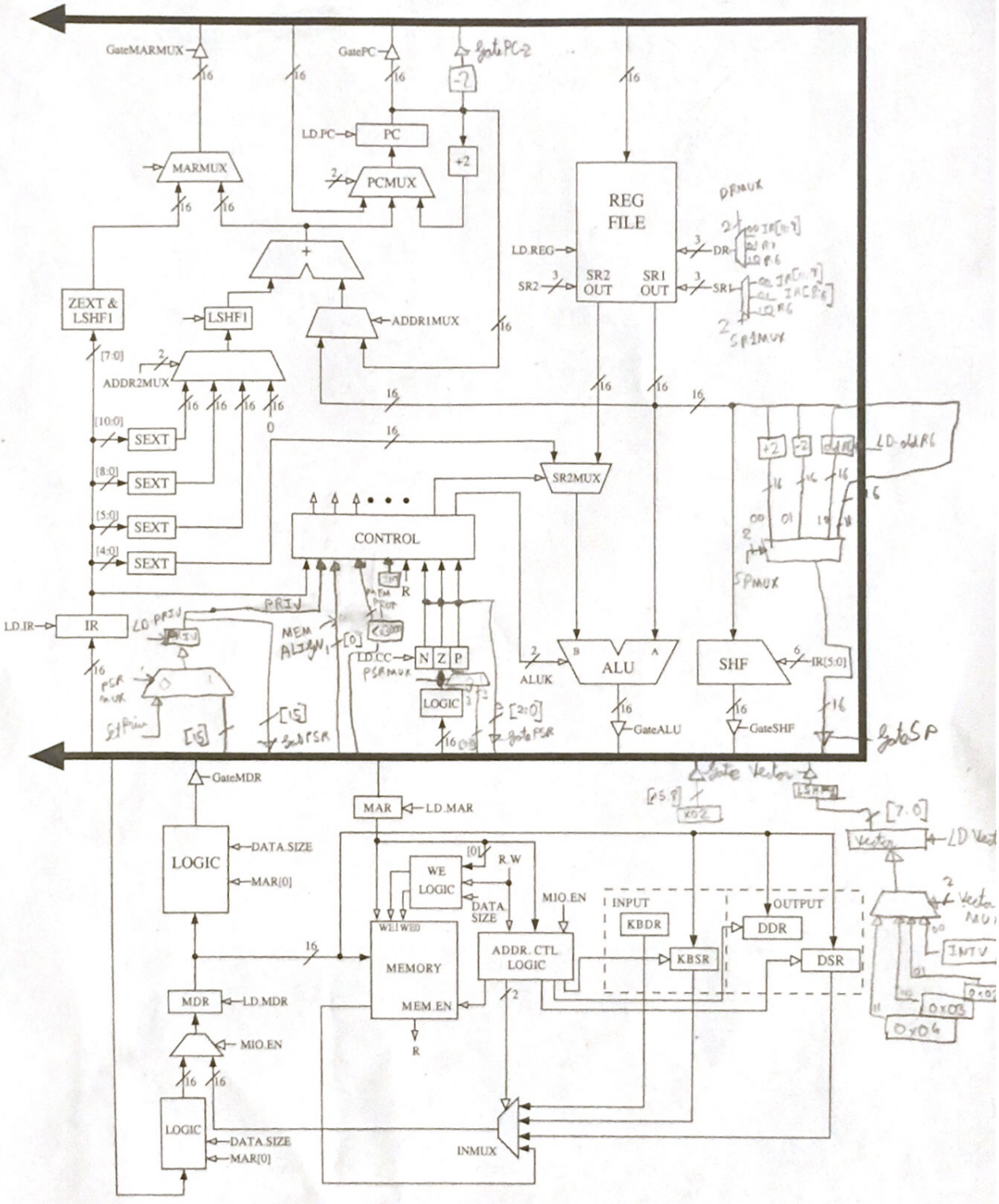
19) Int the common mem prot case we do the exact same thing but don't make changes to the interrupt bit and load the vector with 0x02 to signify that the exception that occurred is the mem protect one
20) Similar for alignment we load the vector with 0x03 to signify the correct exception
21) This is also done in state 10 and 11 to signify the unknown opcode exception occurred with vector loaded with 0x04
22) All these state go the the common state of 60 which continues the context switch
23) State 60 swaps the R6 and Old R6 to swap user stack and supervisor stack to signify that we are now in supervisor mode
24) This then goes to state 37 which decrements the r6 by 2 and puts it in R6 and mar to get ready to push a value on the supervisor stack.
25) Then we put the value in memory in state 44 and when ready go to state 46 to load MDR with pc-2 so when return from RTI the instruction can be reexecuted when.
26) Then we go to state 47 to again decrement r6 by 2 and put it in mad and r6 to get ready to push a value on stack
27) We do this in state 48 and when ready load mar with the vector to read the exception/ interrupt handler location with the beginning value of the handler.
28) Thus we read from mem in state 52 and when ready move to state 54
29) In state 54 we put the MDR value into pc so in the next instruction cycle we can start executing from the top of the handler. Thus next state is state 18 to start the handler

2) Ans: The picture of the data-path is included above and the changes are made in pencil. The changes are:
    1) Added a GatePC-2 to put PC-2 on the bus
    2) Augmented the DRMUX and SR1MUX to select r6 directly
    3) Added a SPMUX to select OldR6(which is the SP not currently used), Current R6, Current R6-2 and CurrentR6 +2
    4) And added a gateSP to put the selected value on the bus
    5) Added A vector MUX to chose between INTV, 0x02, 0x03, and 0x04. And added a reg to add this to A reg named Vector with a signal LdVector to input.
    6) Added LSH on this vector to agree with the LC3b mem alignment. Added a gate vector to put this vector on the bus with low 8 bits from vector and high 8 bits as 0x02 which is vector base
    7) Added a 1 bit reg called Priv to signify which privilege mode the program is running in with 1 for user and 0 for supervisor. The value is inputted from a PRS MUX which selects either the 15th bit from the bus or the setPriv bit from the microcode. Also added a gate PSR which puts this value on the bus. These 2 PSRMUX and GatePSR also work on the conditon codes to either select value of conditon code from the logic block or the low 3 bits on the bus and also to put conditon codes as the low 3 bits on the bus
    8) Added a Int signal to the control to signify if a interrupt has taken place
    9) Added a control signal to check for memalign to see If the low bit on bus is 0 or 1
    10) Added a control signal called mem prot which checks if value on bus is less than x3000 or not
3) Ans: The new signals added are:
    1) Priv: Determines which privilege mode the process is running in
    2) Mem Align: Checks if there is a mem align problem
    3) Mem Prot: Checks if a address less than 0x3000 is accessed
    4) INT: Checks if a interrupt flag is set
    5) Cond2: Added extra condition code
4) Ans: Changes made to the micro-sequencer are included in the picture above with changes made in pencil. Even the Logic diagram is set in there. The major changes were:

1) Added another Condition code to chose to act on control signals of alignment, protection and interrupt.
2) For Condition 100 the microcode checks if an interrupt has occurred and if so then toggle the J3 bit.
3) For Condition 101 Check if an exception has occurred. If so they fall in two categories; a protection or an alignment. IF a protection has occurred turn on both J5 and J4. If only an alignment has occurred then turn on J5 but turn off J4. The logic for this is drawn and are the only changes to the micro-sequencer.


Cond 101
Prot 0
Align 1
COND2
COND1
COND0
BEN
R
IR[11]
Branch
Ready
Addr. Mode
J[5]
J[4]
J[3]
J[2]
J[1]
J[0]
32
8
4
2
1
0,0,IR[15:12]
6
IRD
6
Address of Next State
Cond
000 stay
001 Reado
010 Branch
011 Addr mode
100 Interrupt


when Interrupt happens INT=1 & INVECTOR=0x01
59
Vector ← INTV
MDR ← PSR
PRIV ← 0
INT ← 0
60
MAR ← PC
PC ← PC + 2
[INT]
18, 19
1
MDR ← M
33
MEM.PRP
51
49
Vector ← x02
MDR ← PSR
PRIV ← 0
IR ← MDR
35
BEN ← IR[11] & N + IR[10] & Z + IR[9] & P
[IR[15:12]]
32
RTI
To 8
ADD
AND
XOR
TRAP
SHF
LEA
LDB
LDW
STW
STB
JSR
JMP
BR
1011
To 11
1010
To 10
0
[BEN]
1
22
PC ← PC+LSHF(off9,1)
To 18
12
PC ← BaseR
To 18
4
[IR[11]]
0
1
20
R7 ← PC
PC ← BaseR
To 18
21
R7 ← PC
PC ← PC+LSHF(off11,1)
To 18
8
MAR ← R6
36
MDR ← M
38
PC ← MDR
63
MAR, R6 ← R6+2
40
MDR ← M
42
PSR ← MDR
SET CC
34
R6 ← R6+2
62
OLD.SP ← R6
R6 ← OLD.SP
18
1
DR ← SR1+OP2*
set CC
To 18
5
DR ← SR1&OP2*
set CC
To 18
9
DR ← SR1 XOR OP2*
set CC
To 18
15
MAR ← LSHF(ZEXT[IR[7:0]],1)
PRIV ← 0
28
MDR ← M[MAR]
R7 ← PC
30
PC ← MDR
13
DR ← SHF(SR,A,D,amt4)
set CC
To 18
14
DR ← PC+LSHF(off9,1)
set CC
To 18
2
MAR ← B+off6
43
MEM.PRP
29
MDR ← M[MAR[15:1]'0']
31
DR ← SEXT[BYTE.DATA]
set CC
To 18
6
MAR ← B+LSHF(off6,1)
25
MDR ← M[MAR]
27
DR ← MDR
set CC
To 18
45
MEMPROT MEMALIGN
00
01
10/11
Align
7
MAR ← B+LSHF(off6,1)
23
MDR ← SR
16
M[MAR] ← MDR
To 18
53
MEMPROT MEMALIGN
00
01
10/11
3
MAR ← B+off6
58
SPEC.MEM
1
0
24
MDR ← SR[7:0]
17
M[MAR] ← MDR**
To 19
NOTES
B+off6 : Base + SEXT[offset6]
PC+off9 : PC + SEXT[offset9]
*OP2 may be SR2 or SEXT[imm5]
** [15:8] or [7:0] depending on MAR[0]
10&11
Vector ← x05
MDR ← PSR
PRIV ← 0
60
a) 41
C
Vector ← x03
MDR ← PSR
PRIV ← 0
60


GateMARMUX
GatePC
gatePC-2
LD.PC
PC
+2
MARMUX
PCMUX
REG FILE
LD.REG
SR2 OUT
SR1 OUT
DR
SR1
SR2
DRMUX
SR1MUX
ZEXT & LSHF1
LSHF1
ADDR1MUX
ADDR2MUX
SEXT
SR2MUX
CONTROL
LD.IR
IR
LD.PRIV
PRIV
MEM ALIGN
LD.CC
N Z P
PSR MUX
LOGIC
ALUK
ALU
SHF
GateALU
GateSHF
gatePSR
gateSP
SPMUX
LD.SSP
LD.USP
GateMDR
MAR
LD.MAR
DATA.SIZE
MAR[0]
WE LOGIC
R.W
MIO.EN
INPUT
OUTPUT
KBDR
DDR
KBSR
DSR
ADDR. CTL. LOGIC
MEMORY
MEM.EN
WE1 WE0
MDR
LD.MDR
INMUX
gateVector
LD.Vector
INTV
0x02
0x00
0x04