Avery Atchley
Lab 4 Documentation

Datapath:

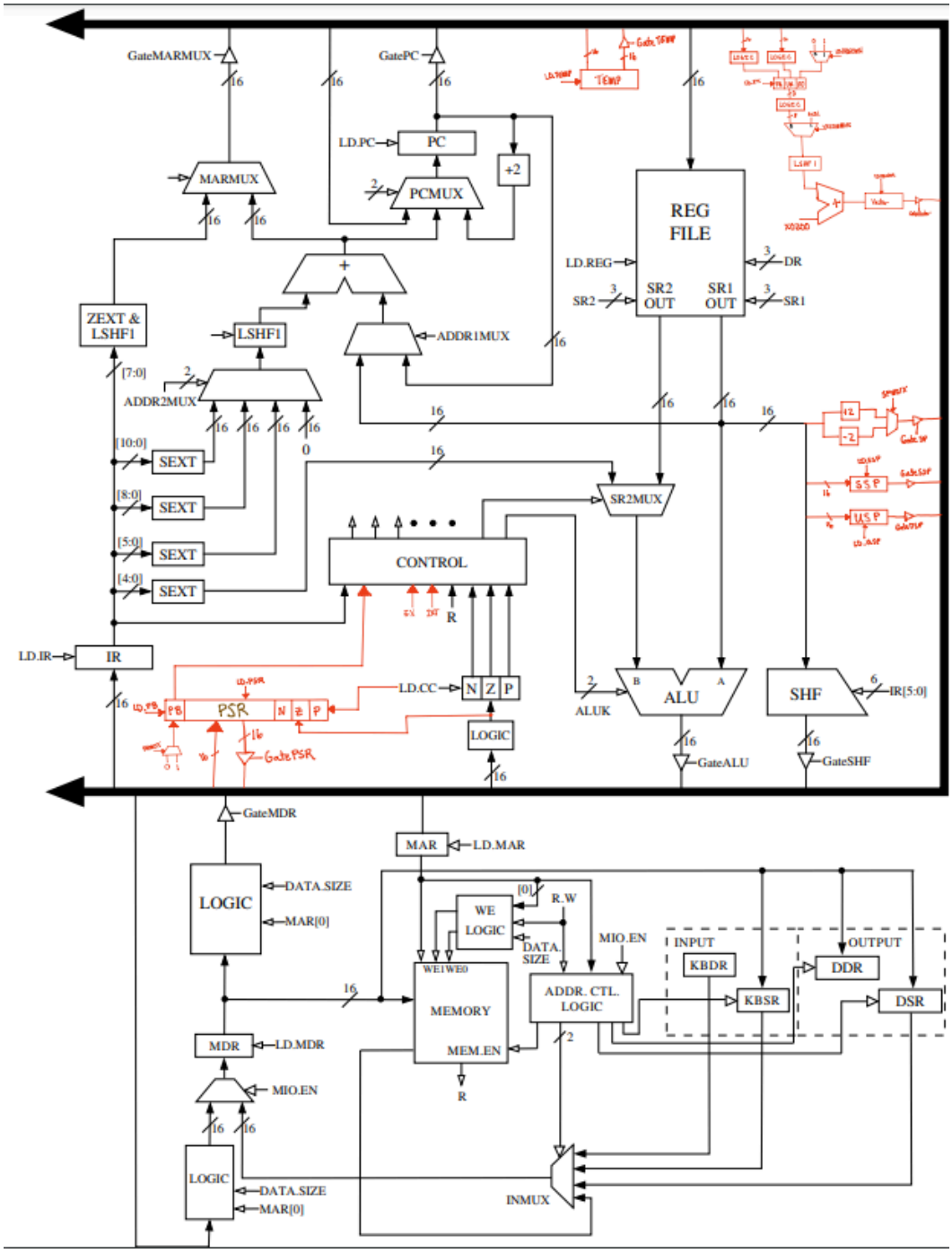


Figure 1. Updated Datapath

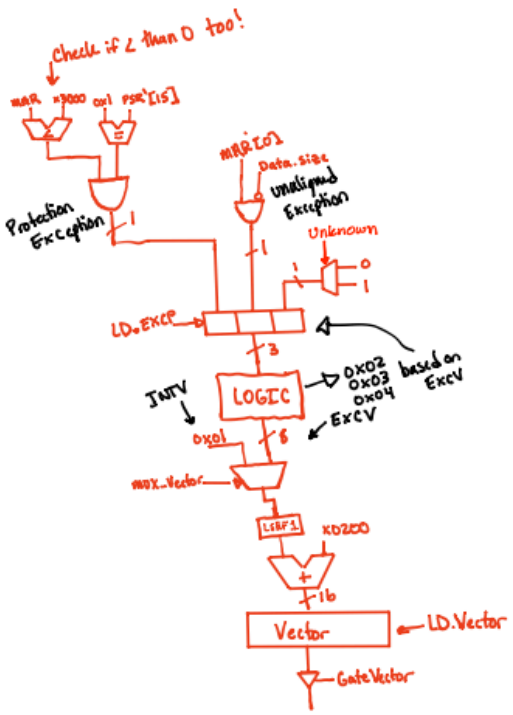


Figure 2. Logic for setting VECTOR register

For the datapath, I have added structures to accommodate interrupts and exceptions. The first structure is the PSR, storing the priority bit (user and supervisor). Control signals (GATEPSR, LD.PB, LD.PSR, and PBMUX) have been added to set this register or push it onto the bus. The second structure is the TEMP register, which holds the PC of the current instruction executing. The purpose is to ensure that when an exception or interrupt occurs, we can return to the instruction executing while the interrupt/exception occurred. Control signals (GATETEMP and LD.TEMP) were added for this functionality. The third structure added includes the ability to increment or decrement the current value in R6 by 2. Control signals (SPMUX and GATESP) were added to enable this functionality. The fourth structure added includes adding registers for SSP and USP. These registers, as the names suggest, hold the values for the user and supervisor stack pointers. These registers are loaded with the value of R6 when relevant. Control signals (LD.SSP, LD.USP, GATESSP, and GATEUSP) were added to enable this functionality. Lastly, I added EX, INT, and PSR[15] to the control set for proper logic control.

State Diagram:

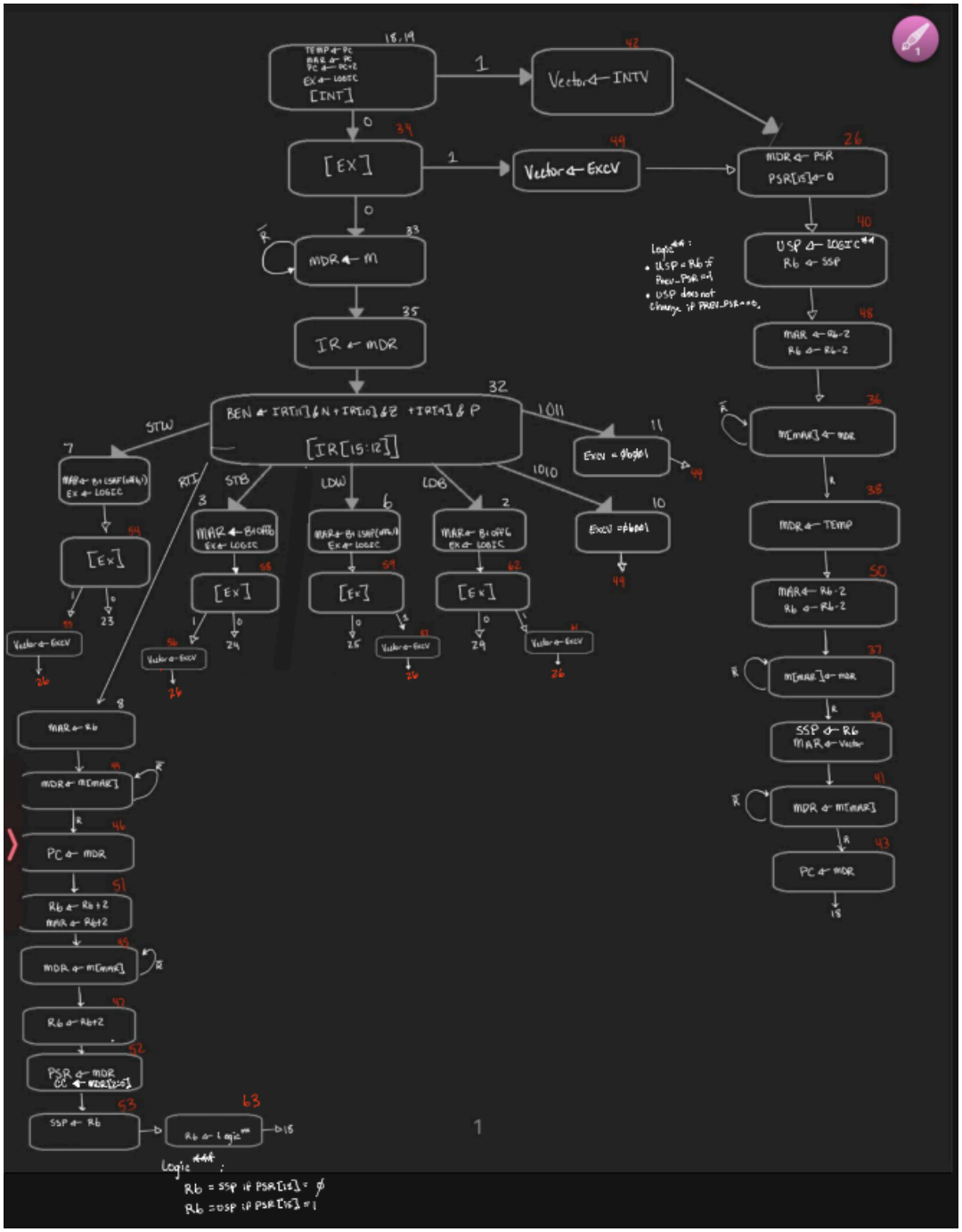


Figure 3. Updated State Diagram

The state diagram has been updated to accommodate for interrupts, exceptions, as well as adding the RTI instruction. First, we check for interrupts in state 18/19 with the interrupt flag (INT) set by software. Additionally, we set the exception flag (EX) in this state using logic located above when setting the VECTOR register. Lastly, in this state we also load the TEMP register to store the current PC of the instruction about to execute. This ensures we can return back to this

instruction after an interrupt or exception is taken. From here, an interrupt is either initiated, thus causing the ISR sequence to begin, or state 34, which checks for an exception, thus either causing the ESR sequence to begin, or transition to state 33. From here, no additions were made until the states for LDB, LDW, STW, STB, RTI, 10 or 11 were reached. The state diagram for LDB, LDW, STW, and STB were altered by incorporating an additional state before memory access occurs to ensure a protected access or unaligned access exception did not occur. This is done by setting the exception flag (EX) while the MAR is being loaded. The additional state was then input between the setting of MAR and memory access, allowing the system to either continue with the memory access, or trigger the ESR. If states 10 or 11 are reached, the exception flag (EX) is set and unconditionally triggers the ESR. Lastly, RTI was implemented by adding states to reestablish the previous PC and PSR values prior to when an interrupt or exception was taken in the program. This includes popping the values for PC and PSR off the system stack using R6, and incrementing R6.

Microsequencer:

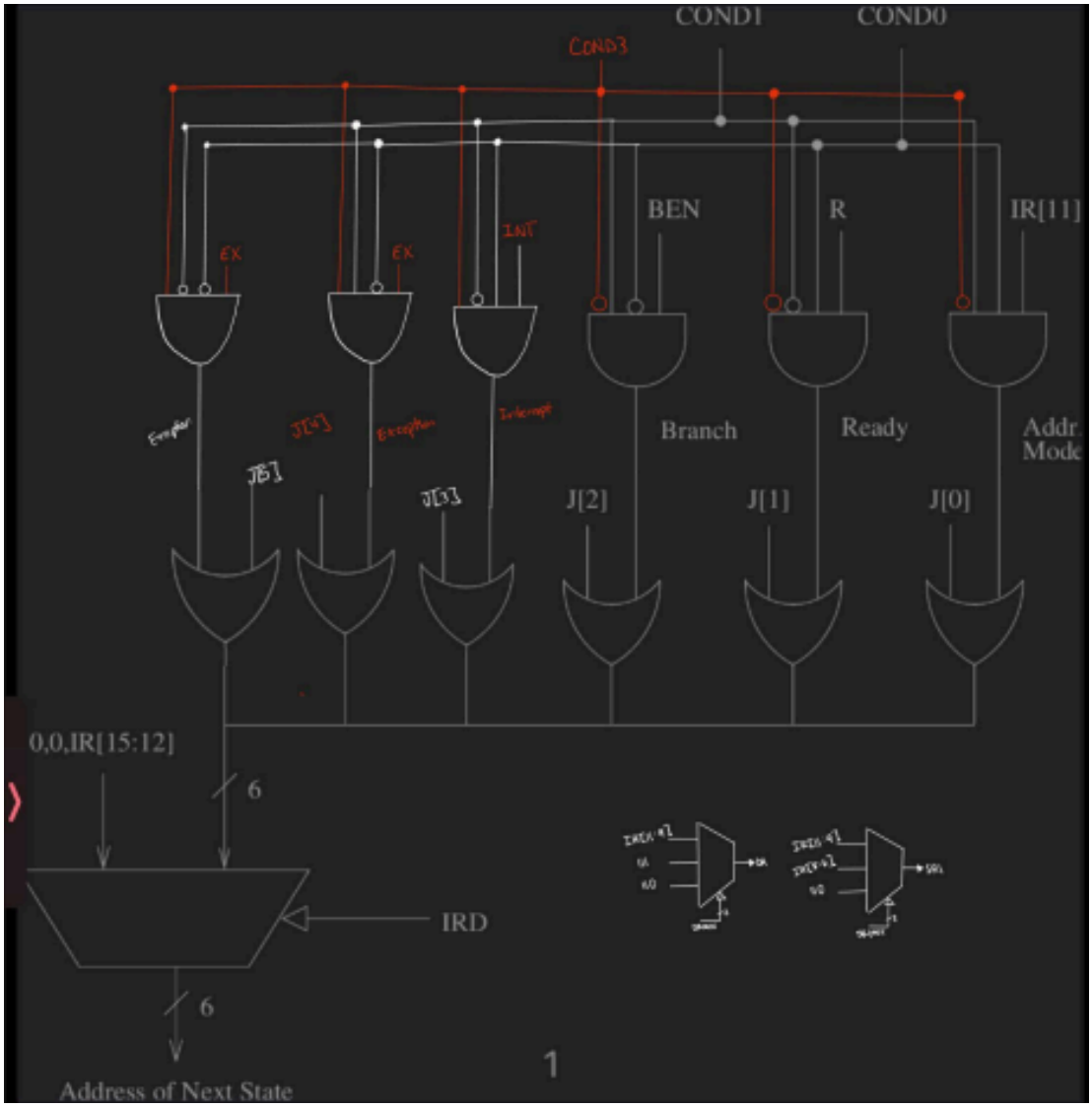


Figure 4. Microsequencer

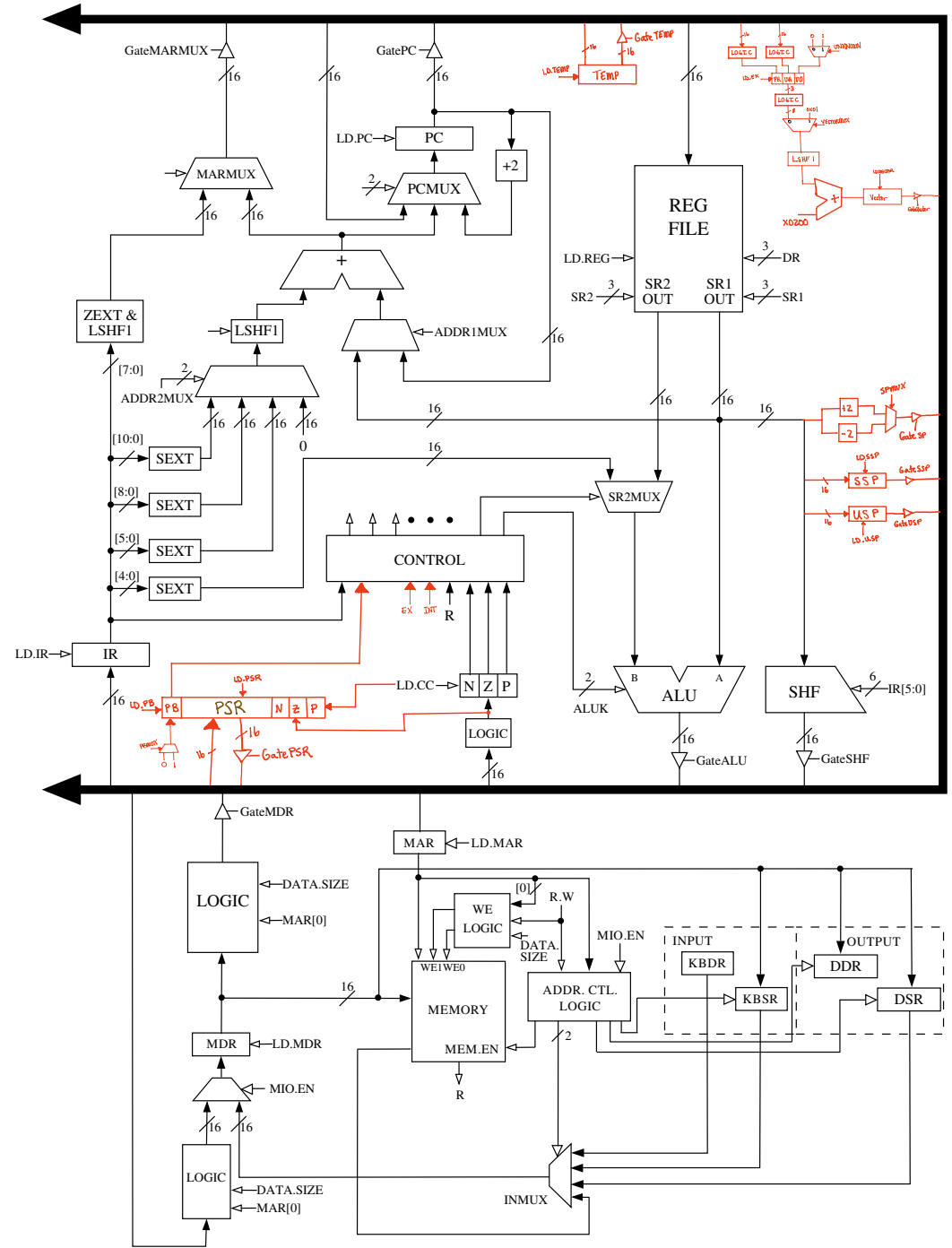
The microsequencer was also updated to accommodate for exceptions and interrupts by adding in an additional COND bit to ensure an accidental state change did not occur. This is due to the fact there are only 4 options with 2 COND bits. If we kept the initial 2 COND bits, we would have an overlap for the microsequencer. For example, if we had the same COND diagram for J5 and J2, if BEN is set at the same time as EX, the wrong state would be acquired. Thus, to prevent these issues, we add an additional COND bit, circumventing the issues of overlap. Additionally, I added a new entry to the SR1MUX and DRMUX to enable the selection of R6 for instructions, like RTI, where we would not have the correct bits to select from using the microcode. This caused me to increase the number of bits utilized for SR1MUX and DRMUX to 2 bits, as opposed to the previous 1 bit.

Microinstructions:

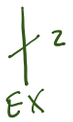
The control signals added to the microcode include the following:

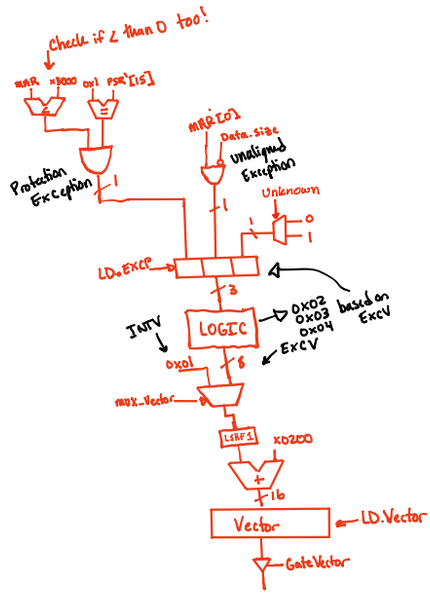
LD.EX - Load exception vector and set exception flag
LD.VECTOR - Load VECTOR register. Select between interrupt or exception
LD.USP - Load User Stack Pointer from R6
LD.SSP - Load System Stack Pointer from R6
LD.PB - Load Priority Bit in PSR (PSR[15]) from PBMUX
LD.PSR - Load PSR from BUS
LD.TEMP - Load TEMP register from BUS (holds PC of current instruction)
GATEUSP - Push USP to bus
GATESSP - Push SSP to bus
GATEVECTOR - Push VECTOR register to bus
GATETEMP - Push TEMP register to bus
GATEPSR - Push PSR register to bus
GATESPMUX - Push incremented or decremented (by 2) value of current SP in R6 to bus
PBMUX - Select between 0 or 1 for loading Priority Bit for PSR[15]
SPMUX - Choose between incrementing or decrementing current SP (R6) by 2
VECTORMUX - Choose between exception or vector → LSHF 1 and add with 0x0200
UNKNOWN - Select between 1 or 0, setting bit 0 in the EXCV for unknown opcode exception
DRMUX - Increment to 2 bits. Now selects between R7, R6, and IR[11:9]
SR1MUX - Increment to 2 bits. Now selects between IR[11:9], IR[8:6], and R6.

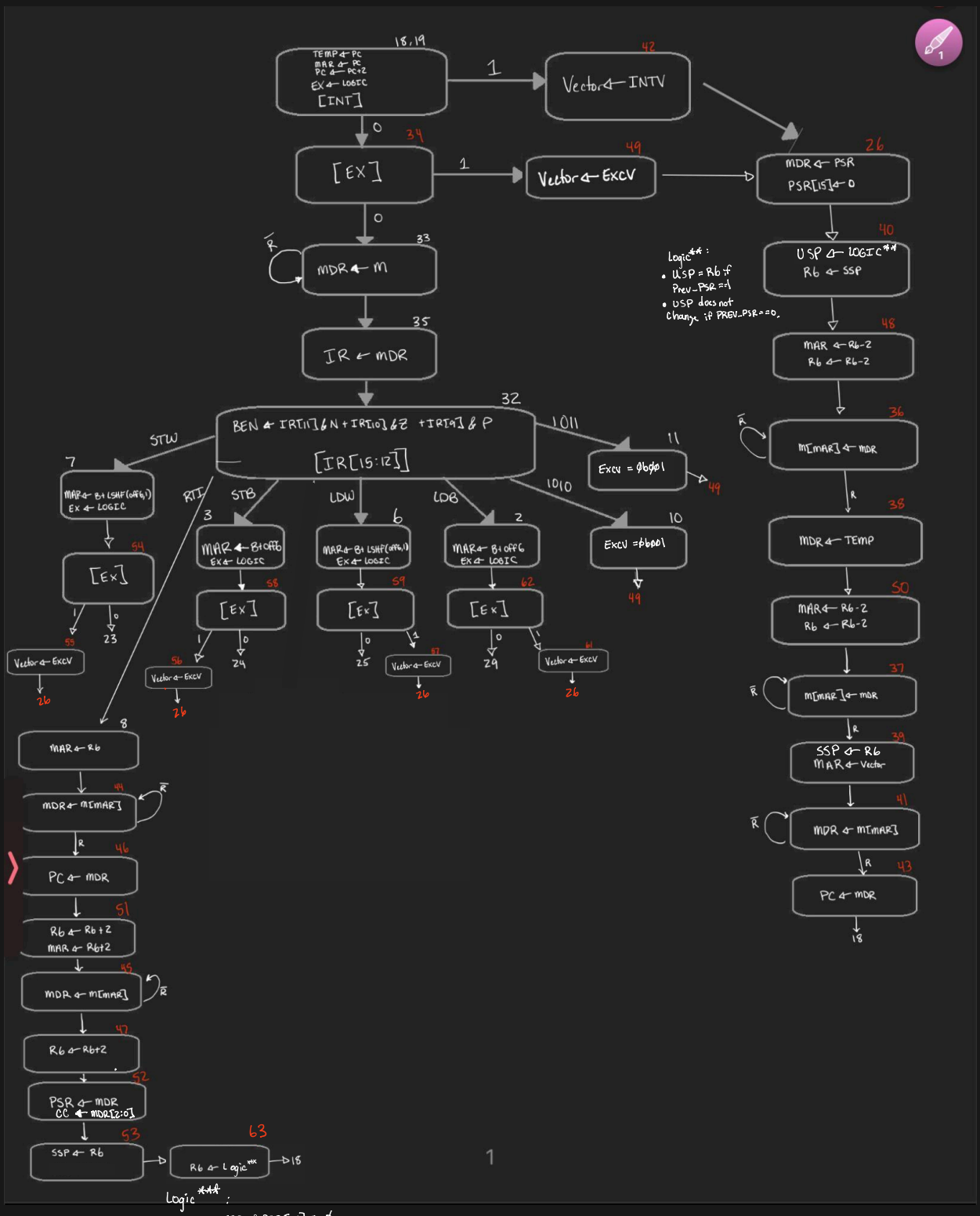
These control signals enable the functionality for interrupts and exceptions, as well as the RTI instruction. I have detailed in the above sections where each control signal is used and its functionality.

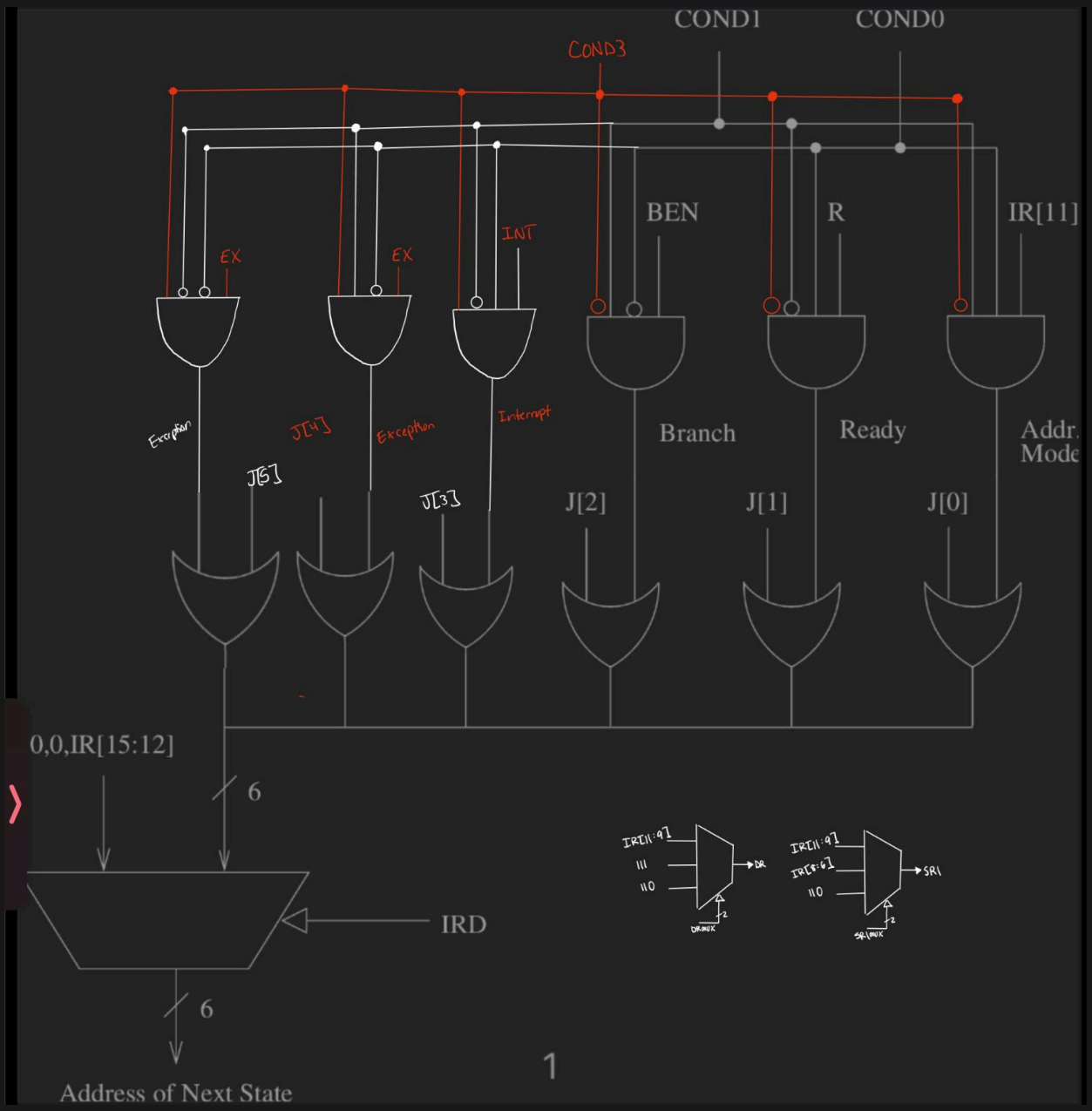

GateMARMUX
GatePC
LD.PC
PC
+2
MARMUX
PCMUX
REG FILE
LD.REG
DR
SR2
SR1
SR2 OUT
SR1 OUT
ZEXT & LSHF1
LSHF1
ADDR1MUX
ADDR2MUX
[7:0]
[10:0]
[8:0]
[5:0]
[4:0]
SEXT
SR2MUX
CONTROL
EX
INT
R
LD.IR
IR
LD.PSR
LD.PB
PSR
LD.CC
N
Z
P
LOGIC
ALUK
ALU
SHF
IR[5:0]
GateALU
GateSHF
GatePSR
GateMDR
MAR
LD.MAR
DATA.SIZE
MAR[0]
WE LOGIC
[0]
R.W
MIO.EN
DATA. SIZE
WE1WE0
MEMORY
MEM.EN
ADDR. CTL. LOGIC
INPUT
OUTPUT
KBDR
KBSR
DDR
DSR
MDR
LD.MDR
INMUX


PSR[15] UNALIGNED


Z
EX


Check if < than 0 too!
MAR
x3000
ox1
PSR[15]
MAR[0]
Data.size
Unaligned Exception
Protection Exception
Unknown
LD.EXCP
based on ExcV
INTV
LOGIC
0x02
0x03
0x04
ExcV
0x01
mux_Vector
LSHF1
x0200
16
Vector
LD.Vector
GateVector


18,19
TEMP ← PC
MAR ← PC
PC ← PC+2
EX ← LOGIC
[INT]
1
42
Vector ← INTV
0
34
[EX]
1
49
Vector ← EXCV
26
MDR ← PSR
PSR[15] ← 0
0
R
33
MDR ← M
40
USP ← LOGIC**
R6 ← SSP
Logic**:
• USP = R6 if Prev_PSR = 1
• USP does not change if PREV_PSR == 0.
35
IR ← MDR
48
MAR ← R6-2
R6 ← R6-2
32
BEN ← IR[11] & N + IR[10] & Z + IR[9] & P
[IR[15:12]]
1011
11
EXCV = #b0001
49
STW
7
MAR ← B+LSHF(off6,1)
EX ← LOGIC
RTI
STB
3
LDW
6
LDB
2
1010
10
EXCV = #b0001
49
R
36
M[MAR] ← MDR
R
38
MDR ← TEMP
54
[EX]
1
0
55
Vector ← EXCV
26
23
MAR ← B+OFF6
EX ← LOGIC
58
[EX]
1
0
56
Vector ← EXCV
26
24
MAR ← B+LSHF(off6,1)
EX ← LOGIC
59
[EX]
0
1
25
87
Vector ← EXCV
26
MAR ← B+OFF6
EX ← LOGIC
62
[EX]
0
1
29
61
Vector ← EXCV
26
50
MAR ← R6-2
R6 ← R6-2
R
37
M[MAR] ← MDR
R
39
SSP ← R6
MAR ← Vector
R
41
MDR ← M[MAR]
R
43
PC ← MDR
18
8
MAR ← R6
44
MDR ← M[MAR]
R
R
46
PC ← MDR
51
R6 ← R6+2
MAR ← R6+2
45
MDR ← M[MAR]
R
47
R6 ← R6+2
52
PSR ← MDR
CC ← MDR[2:0]
53
SSP ← R6
63
R6 ← Logic***
15
Logic***:
R6 = SSP if PSR[15] = 0
R6 = USP if PSR[15] = 1
1


COND1
COND0
COND3
BEN
R
IR[11]
EX
EX
INT
Exception
J[4]
Exception
Interrupt
Branch
Ready
Addr.
Mode
J[5]
J[3]
J[2]
J[1]
J[0]
0,0,IR[15:12]
6
IRD
6
Address of Next State
1
IR[11:9]
111
110
DR
IR[11:9]
IR[8:6]
110
SR1