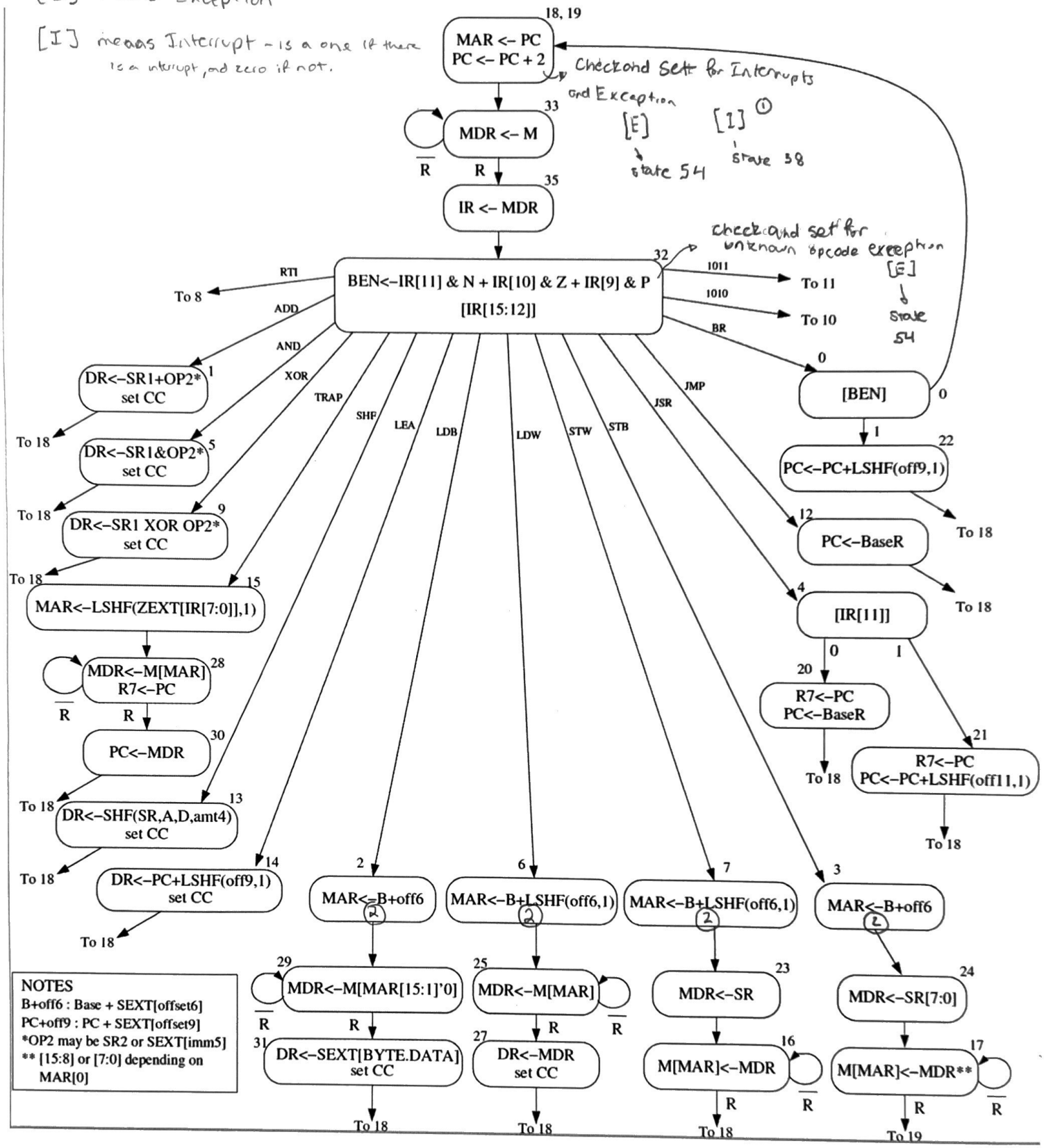
① State Path handles exceptions before interrupts - Implemented in microcontroller - happens in State 18

② At the end of the state, exceptions are checked and set. If [E] is one go to state 54.

[E] means Exception - is a one if there is an exception, and zero if not.

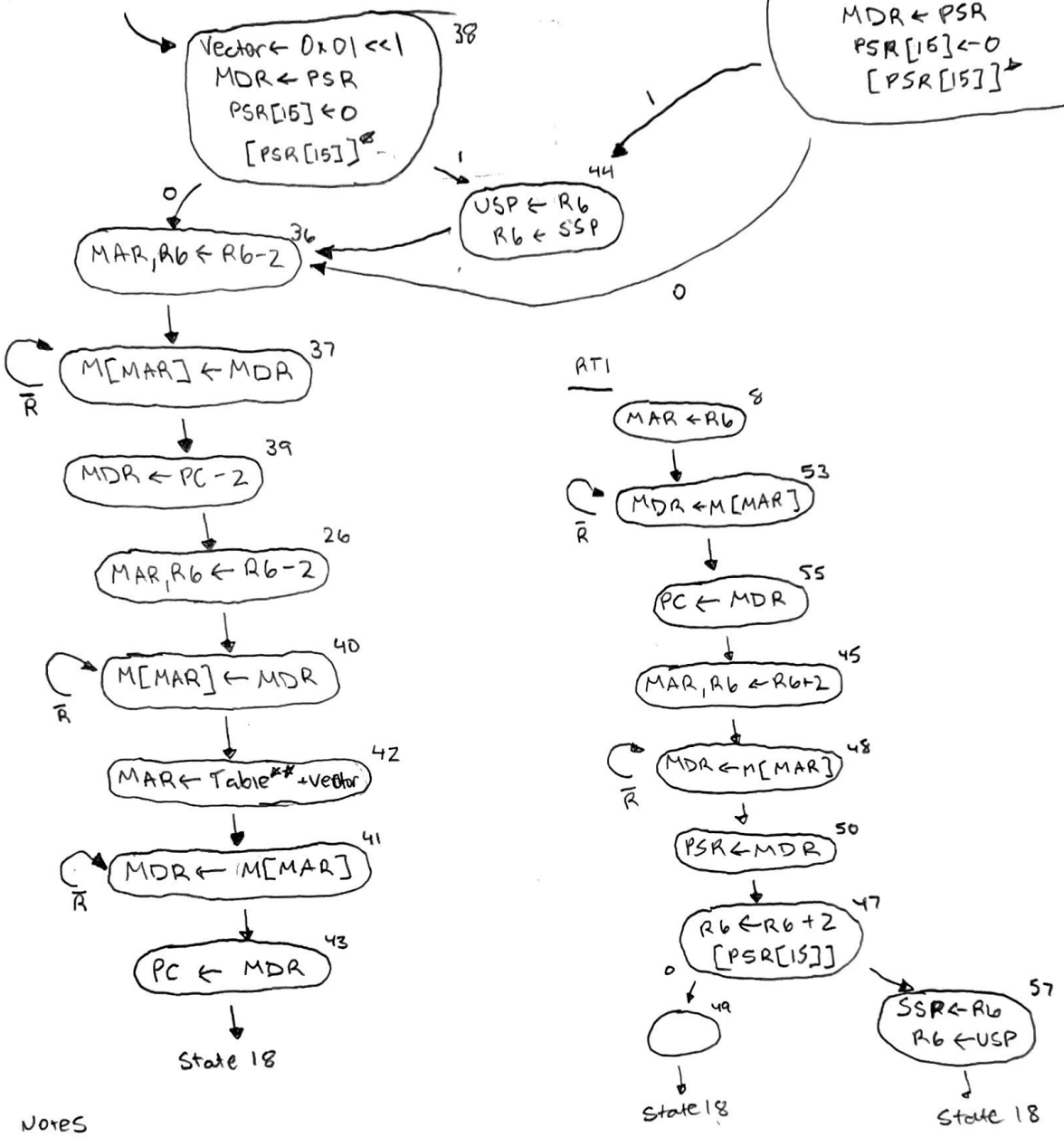
[I] means Interrupt - is a one if there is a interrupt, and zero if not.



Lab 4

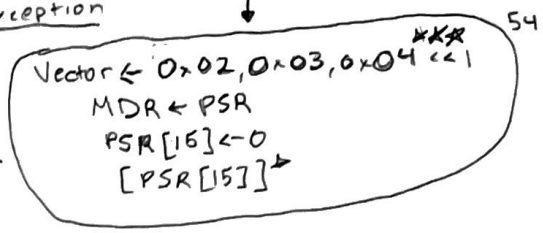
Interrupts

from State 18



Exception

from State 18, 19, 32, 2, 6, 7, or 3



RTI

Notes

* This is the old value of PSR[15] before PSR[15] gets changed to 0. PSR[15] also refers to the bit at position 15 for the register PSR
** Table is a global constant variable that is set to 0x0200. Acted as a constant value in my datapath.
*** Based on the exception that is called, different values are stored in the register vector. For protection exception the value of 0x02<<1 is stored in vector, 0x03<<1 for unaligned access exception, and 0x04<<1 for unknown opcode exception. "<<" means right shift by one.

Siddhant Pandit

State Diagram

Interrupts

State 35 - State 35 is called when interrequest == 1 and is called as the next state when you are in state 18. In state 35, the vector register is loaded with 0x01 << 1 and PSR is put on the bus for MDR to be loaded. PSR bit 15 is then changed to 0 and the next state of the machine is dependent on whether PSR the value from the bus [old PSR] is in supervisor or user mode (based on bit 15)

EXception

State 54 - State 54 is called when the [E] bit is one. This can happen either in State 18, 19, 32, 2, 6, 7, or 8. In state 54, the vector register is loaded with the offset based on the exception that occurs. PSR is loaded on the bus and MDR loads the value from the bus (PSR). PSR[15] is then changed to 0 and the next state of the machine is dependent on wether the value from the bus (old PSR) is in supervisor or user mode (bit 15).

Combined states from Interrupts and Exception

State 44

If PSR[15] was one (user mode) then this state was called. R6 is loaded into user stack pointer register and the value from the supervisor stack register is loaded into R6. This made possible because the SSP (Supervisor stack pointer) is put on the bus by GateSP and SPMUX and loaded by DrMux and at the same time SRMUX and LD.USP loads R6 into USP (user stack pointer).

State 36

This state happens either unconditionally when coming from state 44 or when PSR[15]=0 from state 35, 54. In this state R6-2 is put on the bus using SPMux and GateSP and that value is loaded into MAR and R6.

State 37

This state comes from state 36 unconditionally. In this state MDR is getting stored at address MAR. This state tkes 5 cycles. This is a memory wait state where MDR gets stored at address MAR.

State 39

This state comes from state 37. In this state PC-2 is put on the bus using the PCMux and GatePC. The value from the bus (value is PC-2.) is then loaded into MDR.

Aron

State 26 - This state comes from state 39. In this state R6-2 is put on the bus through SpMux and GateSP. The value from the bus (R6-2) is then loaded into MAR and R6. *this is similar to state 36.

State 40 - This state comes from state 26. In this state MDR is getting stored at address MAR. This state tabs 5 cycles. This is a memory write state where MDR gets stored at adress MAR.

State 42 - This state comes from state 40. The addr output from Vector Register and 0x0200 are put on the bus from gateTable. The value from the bus (vector + 0x0200) is loaded onto MAR.

State 41 - This state comes from state 42. The MDR will be getting the value at address MAR through five cycles, i.e. This is a memory read state where the value at address MAR gets stored in MDR.

State 43 - This state comes from state 41. The value from MDR will get stored on the bus using Gate MDR. The value from the bus (MDR) will get stored in PC using PC MUX and LD.PC. This state then goes to state 18

RTI

State 8 - In this state R6 gets loaded on the bus using ALU and gateALU. The value on the bus (R6) then gets loaded onto MAR using LD.Mar.

State 53 - This state comes from state 8. The MDR will be getting the value at address MAR through five cycles. This is a memory read state where the value at address MAR gets stored in MDR.

State 55 - This state coms from state 53. The value of MDR gets loaded onto the bus using GateMDR. PC then loads the value from the bus (MDR) using PCMUX and LD.PC control signals.

State 45 - This state coms from state 55. The value of R6+2 is loaded onto the bus using SPMUX and GateSP. Both MAR and R6 load the value from the bus using LD.MAR and LD.Reg respectively.

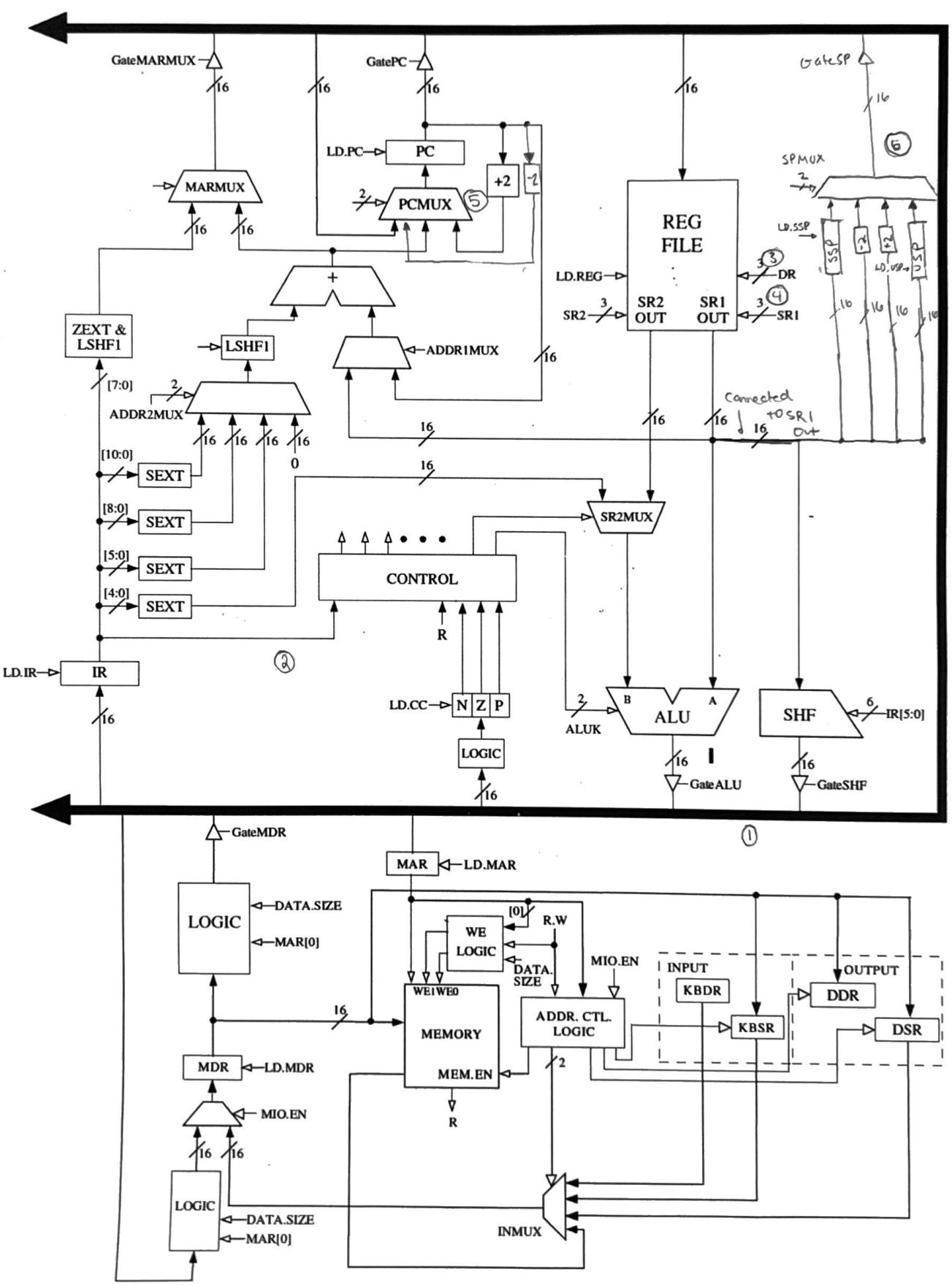
State 48 - This state comes from state 45. The MDR will be getting the value at address MAR through five cycles. This is a memory read state where the value at adress MAR gets stored in MDR.

State 50 - this state comes from state 48. The MDR value gets put on the bus and PSR loads that value from the bus using PSRMUX.

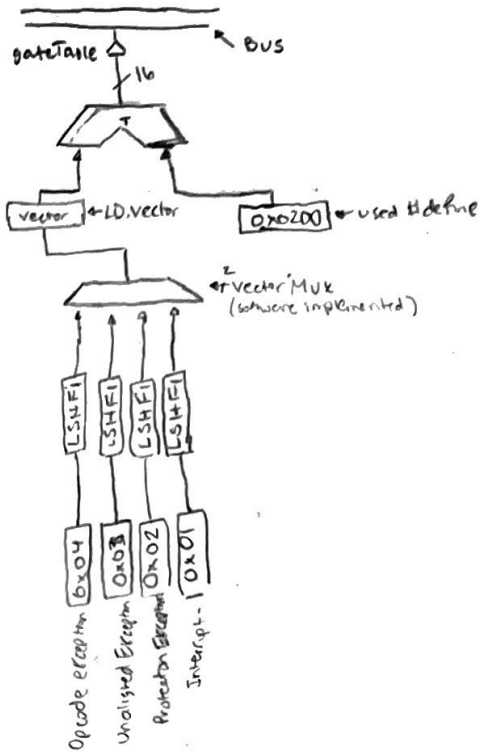
State 47 - This state comes from 50. The value R6+2 gets loaded onto the bus using SPMUX and gateSP. This value from the bus (R6+2) gets loaded into R6. The next state is then chosen by PSR[15].

State 49 - IF PSR[15] = 0 / supervisor mode and state 47 just ended, state 49 is the next state. This state is a filler state that just conditionally goes to State 18.

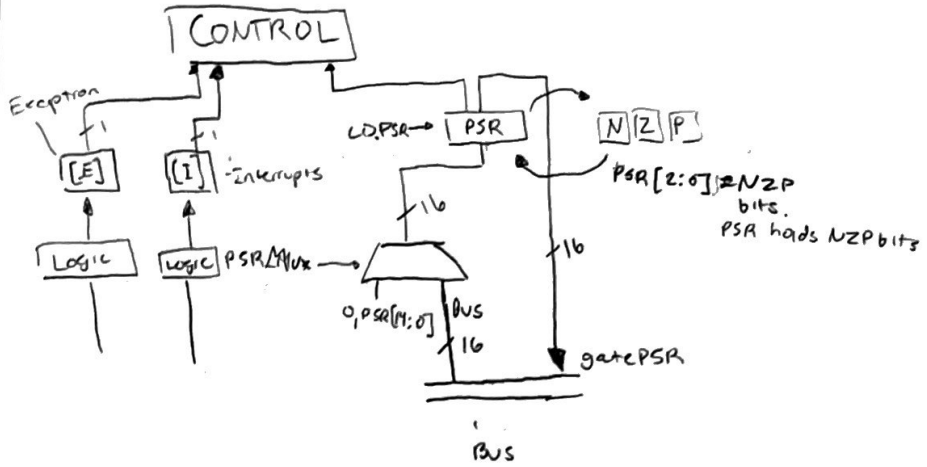
State 57 - IF PSR[15] = 1 / user mode and State 47 just ended, state 57 is the next state. USP is stored on the bus using SPMux and gateSP and the value from R6 is chosen by the SRimux. The val from the bus (USP) is then loaded into R6 while the value from SRimux is loaded into SSP using LD.SSP.


GateMARMUX
GatePC
GateSP
LD.PC
PC
+2
-2
MARMUX
PCMUX
SPMUX
REG FILE
LD.REG
DR
SR2
SR2 OUT
SR1 OUT
SR1
LD.SSP
SSP
USP
ZEXT & LSHF1
LSHF1
ADDR1MUX
ADDR2MUX
[7:0]
[10:0]
[8:0]
[5:0]
[4:0]
SEXT
SEXT
SEXT
SEXT
0
SR2MUX
CONTROL
R
IR
LD.IR
LD.CC
N
Z
P
LOGIC
ALUK
B
A
ALU
SHF
IR[5:0]
GateALU
GateSHF
GateMDR
MAR
LD.MAR
LOGIC
DATA.SIZE
MAR[0]
WE LOGIC
[0]
R.W
DATA. SIZE
MIO.EN
WE1 WE0
MEMORY
MEM.EN
ADDR. CTL. LOGIC
INPUT
KBDR
KBSR
OUTPUT
DDR
DSR
MDR
LD.MDR
MIO.EN
LOGIC
DATA.SIZE
MAR[0]
INMUX
Connected to SR1 Out

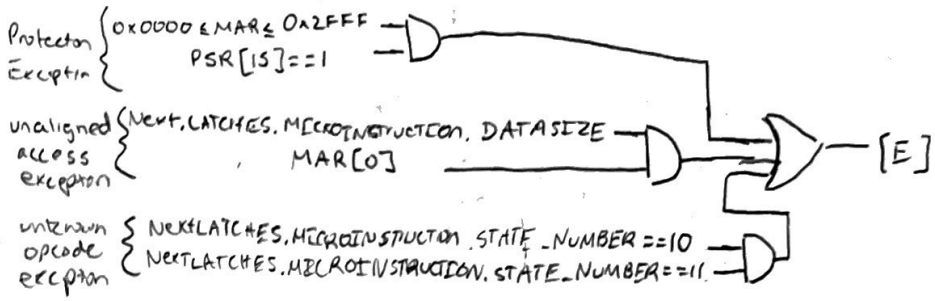
①



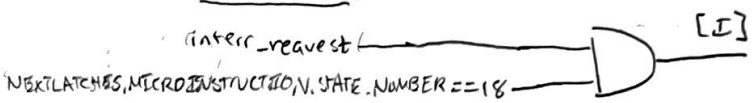
⑧



Logic for Exception of [E]

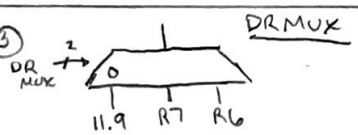


Logic for [I]

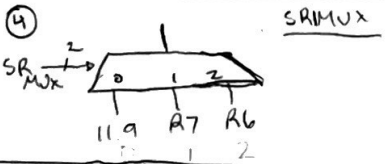


* interr_request is a global variable that becomes a one when CYCLE_COUNT becomes 300 and turns to 0 once the interrupt is called.

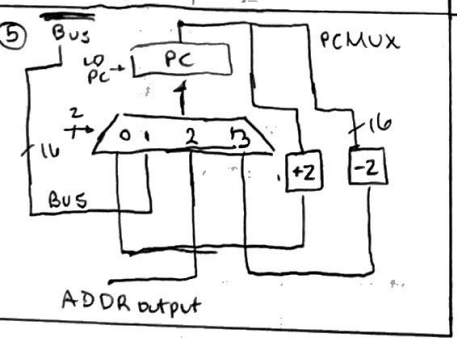
③



④



⑤



Datapath and control signals
Labeled all the structures on the datapath

My first structure from my datapath that I added is the structure that is responsible for creating the address based on an offset and a base for both interrupts and exceptions. One state that uses this structure is state 42 where MAR is populated with the adder from this structure which adds the offset based on what exception it is or interrupt plus the base value of 0x0200. The point of this structure is to yield the address of the memory location which contains the value of the starting address of the service routine that the program is trying to go to based on the exception or interrupt.

The control signals added to the first structure are VectorMux, LD. Vector and gateTable. VectorMux is used to figure out what the vector is based on if there is an interrupt or exception. There are three exceptions and one interrupt thus 4 inputs and 2 bits to choose the value of the vector. Before the values output out of the mux the value is left shifted by one. Based on the value of LD. Vector, the output from the VectorMux either gets stored in the Vector register or not. Lastly the gateTable outputs the value to the bus from the adder output of 0x0200 and the Vector register value.

My second structure from the datapath is a structure I added so that exceptions and interrupts can change the state if needed. For instance, if the Exception bit was a one then the program goes to state 54, while if the interrupt bit was a one then the program goes to state 38. The other functionality of the second structure is to add the PSR register and create a way for the PSR to both output to the PSR and change its value.

The control signals from the second structure are [E], [I], PSRMux, LD.PSR, and gatePSR. The exception and interrupt control signal are set based on the logic structure as seen in the datapath- this control signal is responsible for identifying specific exceptions and interrupt based on "logic" as seen in the datapath. The PSRMux is used to select whether you want the output of the PSR to be changing bit 15 of the PSR to 0 or to get the value from the bus. This value from the mux is loaded into PSR register based on the value of LD.PSR (if one then the value gets loaded into the register and vice versa). Lastly, gatePSR is used for outputting the PSR register value to the bus.

My third structure is not a new structure from the datapath but rather a modification of the DRMux. DRMux is used for selecting what register you want the destination be. I added Register six (R6) as a destination since R6 acts as a stack pointer and thus you need a way to store a value into R6. This modification helps the fifth structure from the datapath

The control signal DRMux changed from a one-bit selector to a two-bit selector since the number of inputs went from two to three. Thus, you need another bit selector for choosing between IR [11:9], R7, and R6.

My fourth structure is not a new structure from the datapath but rather a modification of the SRMux. SRMux is used for selecting what you want the source register to be (designated as SR1 out on the datapath). I added Register six (R6) as a source register since R6 acts as a

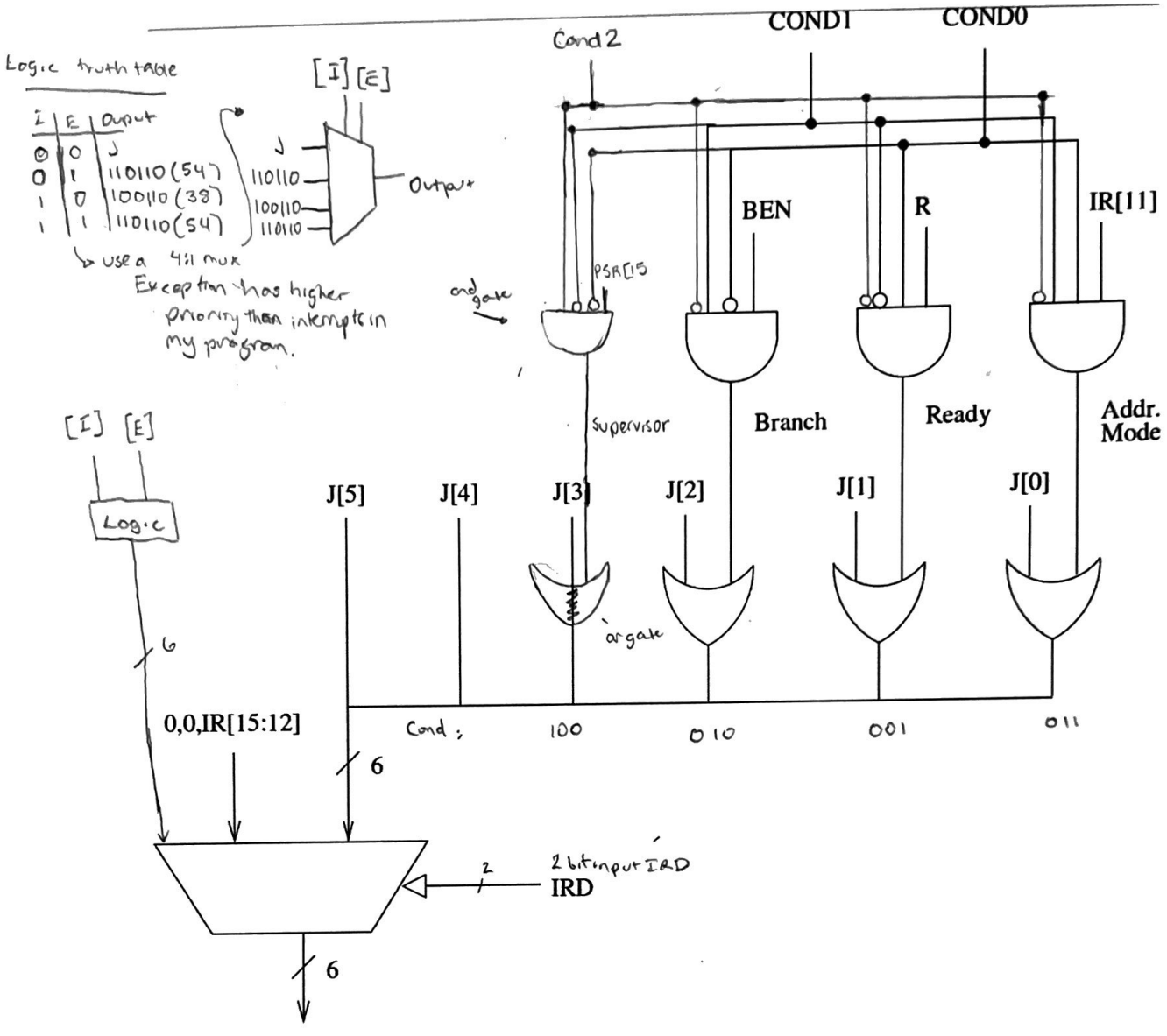
stack pointer thus you need a way to get the value from R6 at all times. This modification helps the fifth structure from the datapath.

The control signals SRMux changed from a one-bit selector to a two-bit selector since the number of inputs went from two to three. Thus, you need another bit selector for choosing between IR [11:9], IR [8:6], and now R6.

My fifth structure is not a new structure from the datapath but rather a modification of the PCMUX. PCMUX is used for selecting what value you want PC to be loaded with or you want on the bus. In this instant, I modified the PCMUX so that the value of PC-2 could be outputted to the bus. Since PCMUX already has a two-bit selector, I made PC-2 as the fourth input for the four to one mux. This modification is needed for a state such as 39.

My sixth structure from the datapath is a structure that moves the value of SR1out to USP and moves the value of SSP to SR1out or moves the value of SR1out to SSP and moves the value of USP to SR1out depending on the state. The structure is also responsible for either incrementing or decrementing the value from SR1out by 2. In the states that I added, the value of SR1out is usually R6 as that acts as the stack pointer throughout the program. This structure is made possible by the modifications made in 3 and 4 as you need the value from R6 and you need to output to R6 as well whenever this structure is in use.

The control signals used in the sixth structure is LD.SSP, LD. USP, SPMUX, and GateSP. The control signal LD.SSP and LD.USP are used to load the value from SR1Out (R6 in all my states). This can be seen in state 44 and 57. The control signal SPMUX is used to select what value you want outputted out of the mux. In this case it could either be USP register, SR1out+2, Sr1out-2, or SSP register. Lastly the value from the mux is only outputted if the value of GateSP is one.



Describe

My updated Microsequencer involves adding another input to the mux, making the 1 bit IRD to 2 bits, adding another condition bit and adding PSR[15] as a condition for changing the state. As seen in "Supervisor" I added another condition based on the bit PSR[15]. This is because if PSR[15] is one that means you are in user mode and thus you may need to change the stack pointer thus this requires changing states before returning to this can be seen in state 44 and 36 from 38 where PSR[15] causes the state to go from 38 to 44 to change the SP or R6. Since I added another condition another condition bit needs to be added as seen in Cond2. In my microsequencer, 000 as cond means J should be passed through without changing the bits. Another major change is adding the [I] and [E] mux as an input to the IRD mux. If the exception bit is one then the next state should be 54, and if the interrupt bit is one and exception bit is zero the next state should be 38 - only if IRD is 2. If both are zero then nothing should be changed and J is passed through. I added this because if there is an interruption or exception, the state path needs to go to the correct state. This part of the microsequencer is only possible in certain states such as in State 2 where IRD is 2 or state 18, 19. Lastly IRD was changed from one-bit to two bit since I added another input to the IRD MUX.