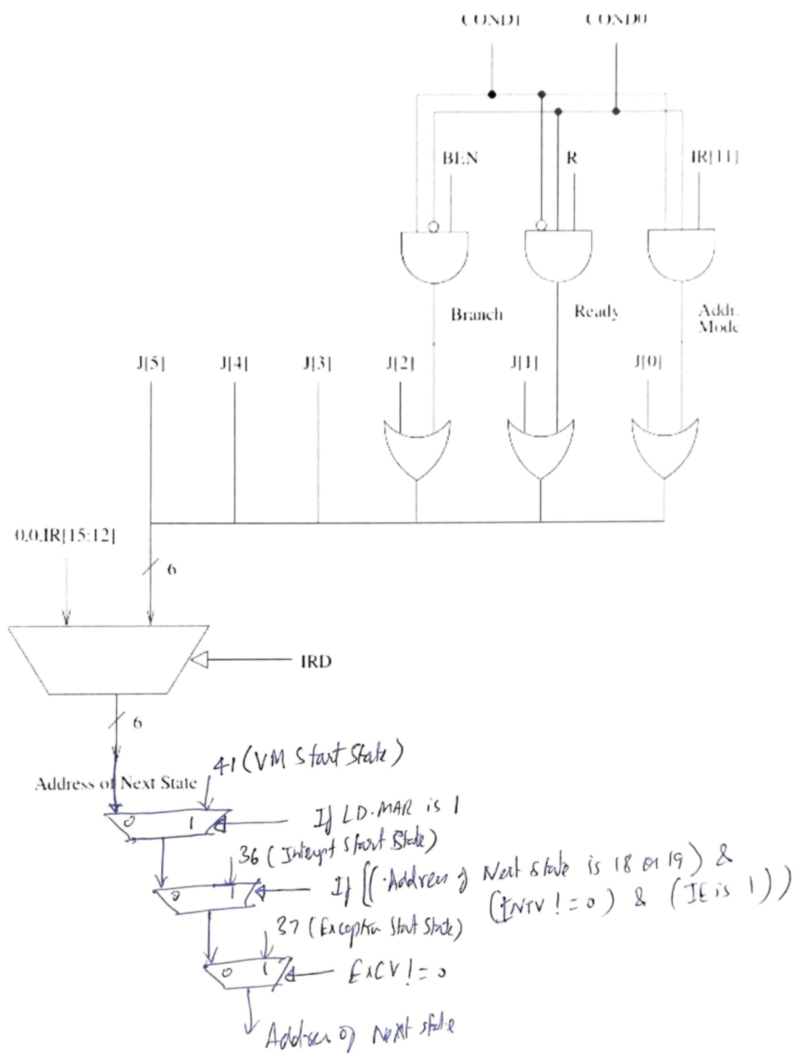
Lab5 Report

Unused States: **10, 11, 26, 34, 58, 61, 63** (A total of 7 unused states)

Total number of States used: **57/64**

Micro sequencer Changes



The starting state for the interrupt is allocated as State 36 and starting state for exception is allocated as 37. The starting state for Virtual to physical address translation is allocated as 41.

From the above diagram, it is clear that the exceptions are executed in the next cycle after they occur or detected. When the exception is detected, the EXCV register is set to the corresponding exception vector and is generally non-zero. If it is non-zero, it will choose the next state as 37 according to the lowest MUX logic. If there is no exception, then there is a check for any pending interrupts.

As opposed to the exceptions, interrupts are serviced only after the current execution is completed. Hence, the condition to check for the interrupt is as follows – If the next state is either 18/19 (indicating the start of a new instruction) and if there is a active pending interrupt (indicated by INTV register) and if the IE register is enabled. If the above condition is satisfied, then the next state address is calculated as State 36.

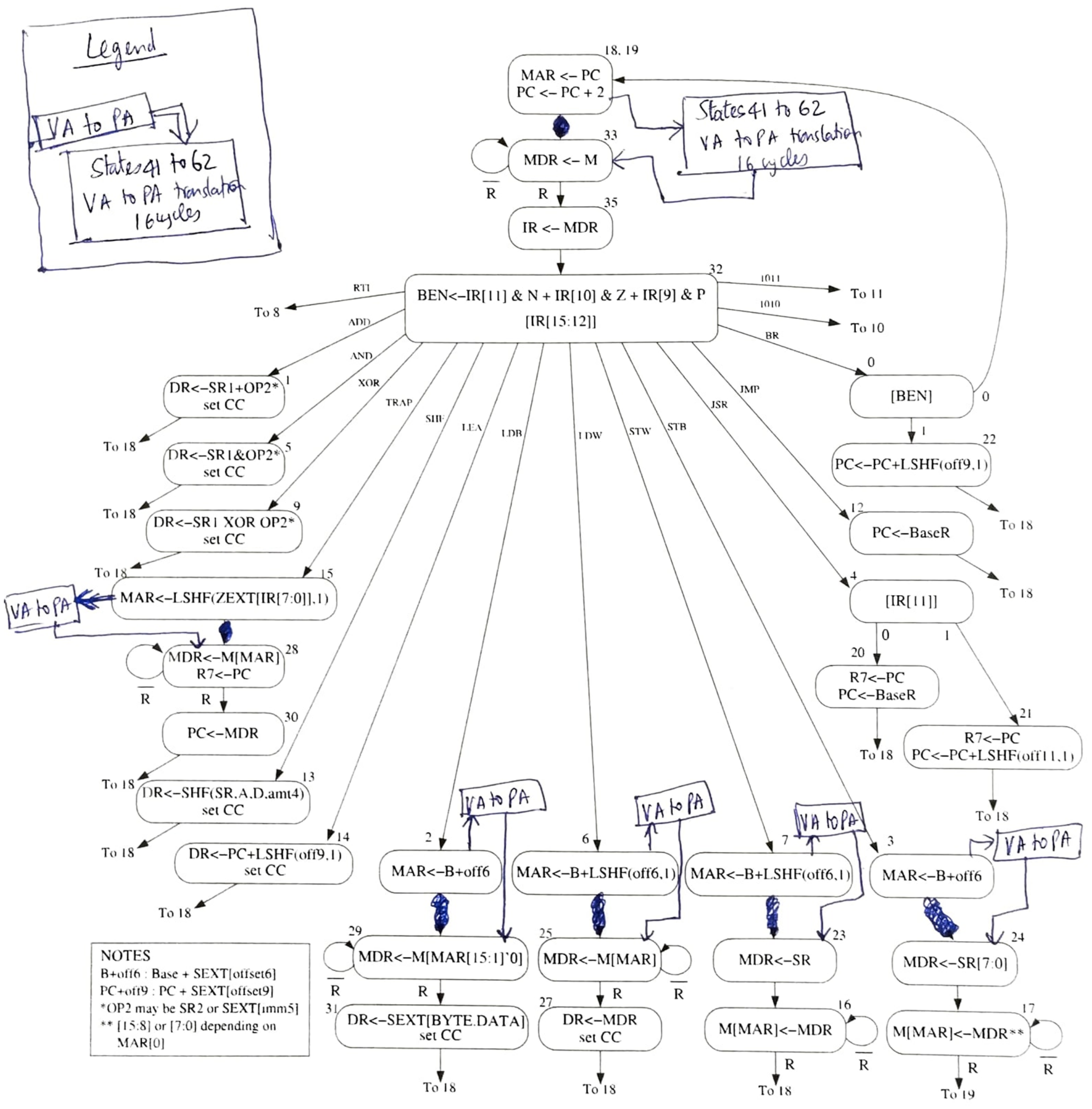
To support the Virtual to Physical address translation, the LD.MAR signal is used as the input to the MUX as shown in the figure. If the MAR is loaded in the current cycle, then the next state is taken as 41. (VM start state).

If there is no pending exceptions or interrupt, then the above micro sequencer would work as the legacy one.

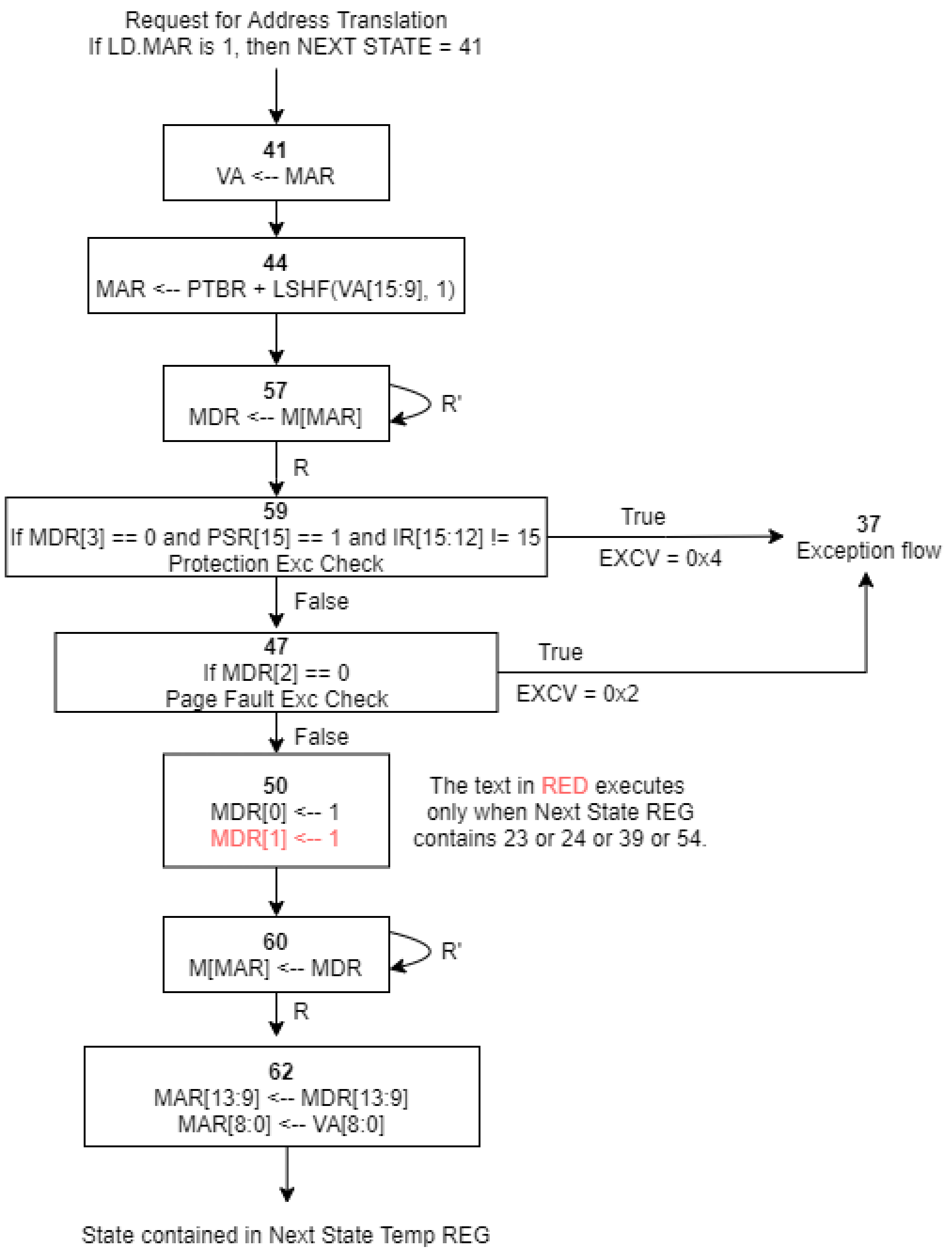
State Diagram Changes

a. Generic Instructions Flow

Wherever MAR is being loaded, we need to trigger the Virtual to Physical address translation flow.

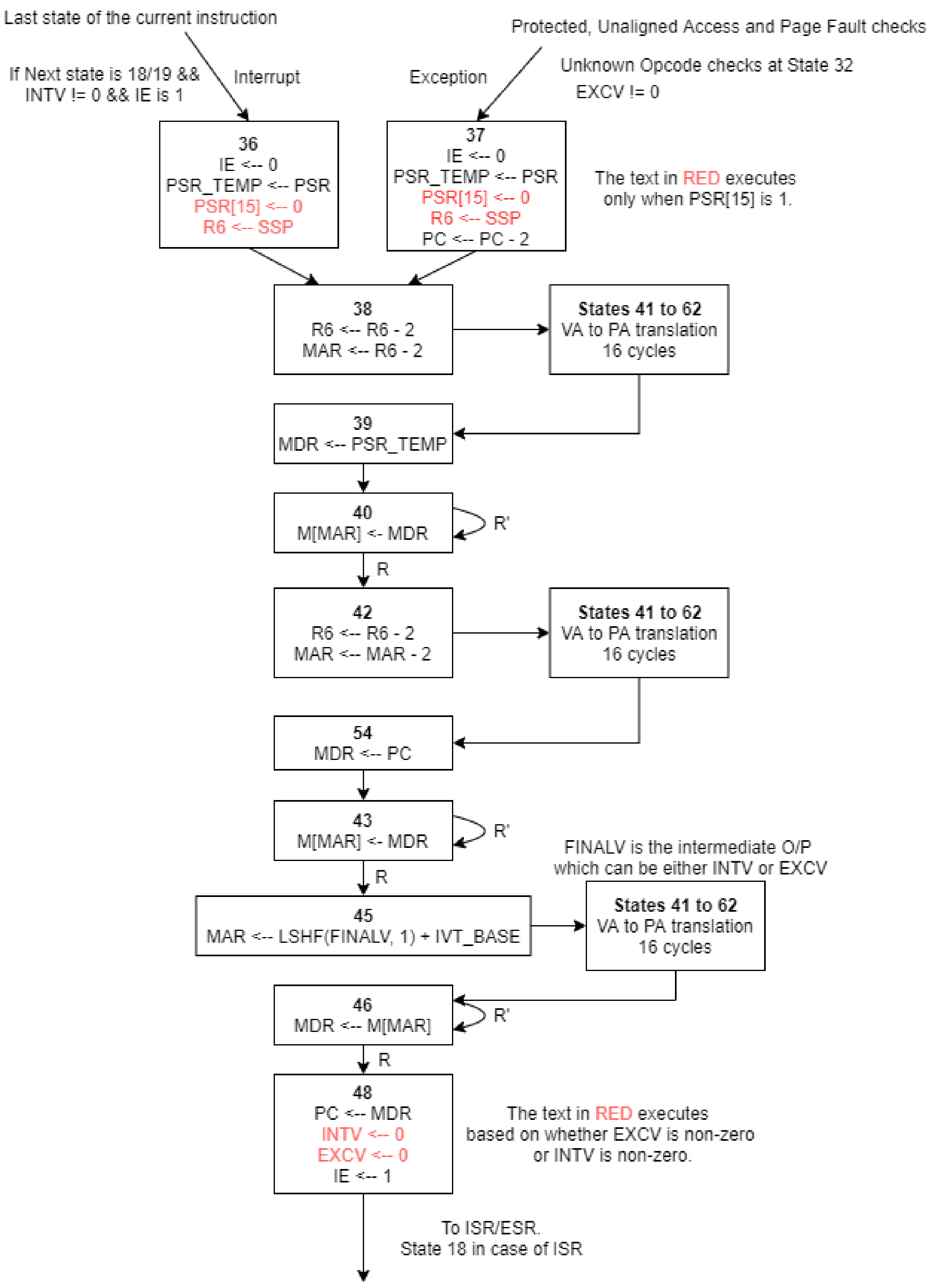


b. Virtual to Physical Address Translation Flow



Request for Address Translation
If LD.MAR is 1, then NEXT STATE = 41
41
VA <-- MAR
44
MAR <-- PTBR + LSHF(VA[15:9], 1)
57
MDR <-- M[MAR]
R'
R
59
If MDR[3] == 0 and PSR[15] == 1 and IR[15:12] != 15
Protection Exc Check
True
EXCV = 0x4
37
Exception flow
False
47
If MDR[2] == 0
Page Fault Exc Check
True
EXCV = 0x2
False
50
MDR[0] <-- 1
MDR[1] <-- 1
The text in RED executes only when Next State REG contains 23 or 24 or 39 or 54.
60
M[MAR] <-- MDR
R'
R
62
MAR[13:9] <-- MDR[13:9]
MAR[8:0] <-- VA[8:0]
State contained in Next State Temp REG

c. Interrupt/Exception Flow



Last state of the current instruction
If Next state is 18/19 && INTV != 0 && IE is 1
Interrupt
Exception
Protected, Unaligned Access and Page Fault checks
Unknown Opcode checks at State 32
EXCV != 0
36 IE <-- 0 PSR_TEMP <-- PSR PSR[15] <-- 0 R6 <-- SSP
37 IE <-- 0 PSR_TEMP <-- PSR PSR[15] <-- 0 R6 <-- SSP PC <-- PC - 2
The text in RED executes only when PSR[15] is 1.
38 R6 <-- R6 - 2 MAR <-- R6 - 2
States 41 to 62 VA to PA translation 16 cycles
39 MDR <-- PSR_TEMP
40 M[MAR] <- MDR
R'
R
42 R6 <-- R6 - 2 MAR <-- MAR - 2
States 41 to 62 VA to PA translation 16 cycles
54 MDR <-- PC
43 M[MAR] <- MDR
R'
R
FINALV is the intermediate O/P which can be either INTV or EXCV
45 MAR <-- LSHF(FINALV, 1) + IVT_BASE
States 41 to 62 VA to PA translation 16 cycles
46 MDR <-- M[MAR]
R'
R
48 PC <-- MDR INTV <-- 0 EXCV <-- 0 IE <-- 1
The text in RED executes based on whether EXCV is non-zero or INTV is non-zero.
To ISR/ESR. State 18 in case of ISR

The above diagram represents the states added before going to the ISR routine.

It is crucial that the below texts in RED executes only when PSR[15] = 1 (i.e the interrupt or exception has occurred from the user mode) to support the nesting of interrupts and exceptions.

In the case of nesting of interrupts or exceptions, the value of R6 will not be reloaded with SSP for every nested interrupt/exception. This helps in R6 being maintained as a stack pointer and pushes/pops registers in the sequence of operation.

The interrupts are being serviced only after the current instruction is completely executed. For example, if an interrupt occurs in between a STB instruction (say state 3), then the interrupt is not serviced immediately in the next cycle – rather it waits for the complete instruction to be executed. When the next state is scheduled to be state 18, we check if there are any pending interrupts by looking at the INTV register and IE register. Any interrupts or exceptions that occur when IE bit is 0 will not be serviced.

Exceptions are handled a bit differently. Whenever any type of exception occurs, we move into servicing the exception starting from the next cycle itself instead of waiting for entire instruction to be completed as in the case of interrupts. The exception handlers in this lab by default contain the HALT instruction. Hence, the program will get terminated once an exception occurs. If the exception handler contains some generic instructions followed by an RTI instruction towards the end, then the instruction that caused the exception would be re-executed again and again in an infinite loop.

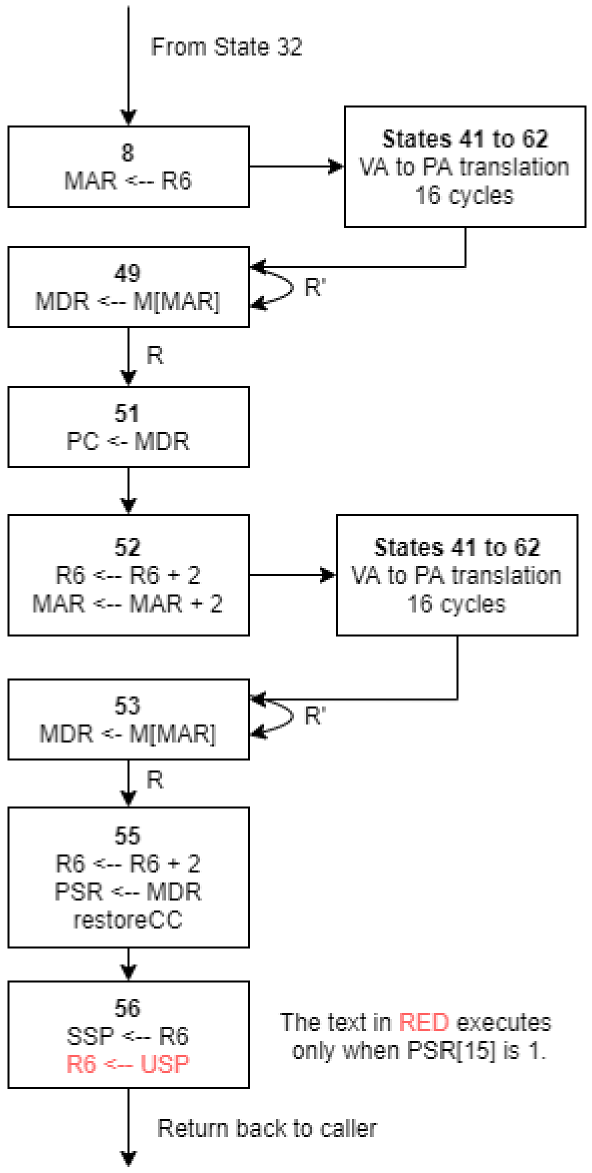
Unknown opcode exceptions are checked at State 32 when every instruction is decoded via its opcode bits to decide if it’s a valid opcode or not.

Unaligned access exceptions are checked at places wherever MAR is being loaded and accessed further. Unaligned access exceptions can occur at any point. Specifically, they are checked at States 18 and 19 (Fetch stages), State 15 (TRAP) and States 6 and 7 (Load and Store – memory access instructions)

Protected and Page fault exceptions are checked only during the Virtual to physical address translation flow.

The pushing of the PSR and the PC values happens in state 40 and 43.

d. RTI flow



The below diagram represents the states added for the RTI instruction. RTI instruction starts from State 18 and goes through the regular flow until the Decode state (State 32). At the decode state, the next state would be set to 8 and it follows the flow as described below.

The popping of the PC and PSR registers from the stack is described in states 49 and 53. The condition codes are also restored when PSR is being restored.

New Control Signals Added/Modified Existing Control Signals

a. **For Lab 4**

LD.PSR

This signal is used to control the load of PSR register in a particular cycle.

LD.SSP

This signal is used to control the load of SSP register in a particular cycle.

LD.USP

This signal is used to control the load of USP register in a particular cycle. Even though this control signal was not used in this Lab, this signal was added to support USP operations in the future.

PCMUX

This already set of 2 control signals was modified to include PC-2 with a select value of 3.

(4:1 MUX)

00 – PC+2

01 – Input from BUS

10 – Output from Address Adder

11 – PC-2

ADDR1MUX

This already set of 1 control signal was modified to include another control signal (making it two control signals in total). Selecting IVT_BASE register value was added with a select value of 2.

(4:1 MUX)

00 – PC

01 – BaseR

10 – IVT_BASE

ADDR2MUX

This already set of 2 control signals was modified to include another control signal (making it three control signals in total). Selecting INTV/EXCV register value was added with a select value of 4.

(8:1 MUX)

000 – 0

001 – offset6

010 – PCoffset9

011 – PCoffset11

100 – INTV/EXCV

REGMUX

This set of 2 control signals was added to select from the below options.

(4:1 MUX)

00 – Output from SPMUX

01 – Input from BUS

10 – REG+2

11 – REG-2

ALUMUX

This control signal was added to select from the below options

(2:1 MUX)

0 - Output from REGFILE

1 – PSR

MAR2MUX

This control signal was added to select from the below options

(4:1 MUX)

00 – Input from BUS

01 – MAR+2

10 – MAR-2

b. **For Lab 5**

LD.VA

This signal is used to control the load of Virtual address into the VA register in a particular cycle.

GateMAR

This signal is used to control the load of MAR value on to the BUS in a particular cycle.

GateVA

This signal is used to control the load of VA value on to the BUS in a particular cycle.

GatePTBR

This signal is used to control the load of PTBR value on to the BUS in a particular cycle.

ADDR1MUX

Selecting PTBR register value was added with a select value of 3.

(4:1 MUX)

00 – PC

01 – BaseR

10 – IVT_BASE

11 – PTBR

ADDR2MUX

Selecting VA[15:9] register value was added with a select value of 5.

(8:1 MUX)

000 – 0

001 – offset6

010 – PCoffset9

011 – PCoffset11

100 – INTV/EXCV

101 – VA[15:9]

MAR2MUX

Selecting translated MAR register value was added with a select value of 3.

(4:1 MUX)

00 – Input from BUS

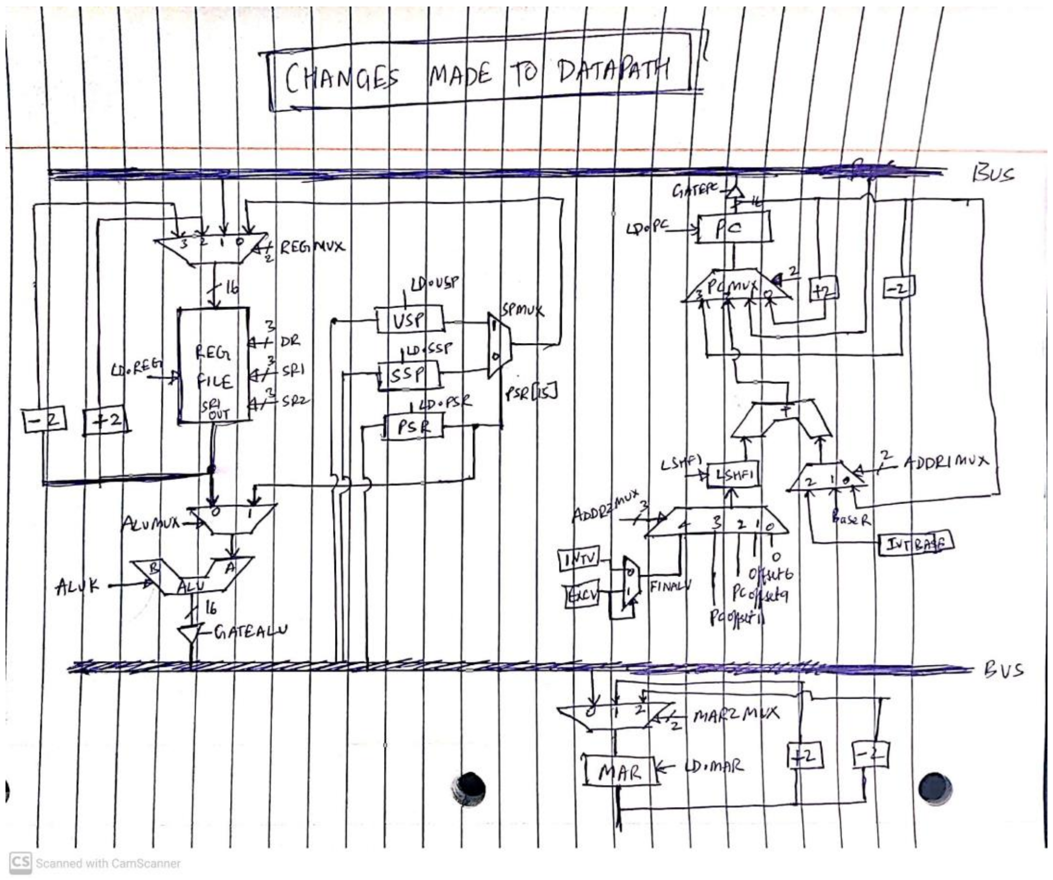
01 – MAR+2

10 – MAR-2

11 – Translated (VA-PA) MAR value

Changes made to the Datapath

a. **For Lab 4**



A 4:1 MUX named REGMUX (having two control signals) is added on top of the REGFILE structure. This MUX helps in routing the R6+2, R6-2 and Loading SSP/USP to R6 operations in addition to loading the registers present in the REGFILE from the BUS.

The REGFILE component in the data path remains undisturbed. The SR2OUT remains unchanged and SR1OUT output from the REGFILE is given as one of the inputs to a 2:1 MUX named ALUMUX (having one control signal). The other input to the ALUMUX is from the PSR register. This path is needed for the loading of PSR onto MDR via the BUS. It makes use of the PASSA operation via the ALUK control signal. The generic output from the REGFILE is routed onto the BUS using the appropriate ALUMUX and ALUK control signals.

Three new registers named USP (User stack pointer), SSP (Supervisor Stack pointer) and PSR (Program Status register) are added to the data path. Each of the registers have a LD control signal which controls when the registers would be loaded. All these registers take the inputs from the BUS. This facilitates the operations such as SSP being loaded with R6 value, PSR being loaded with MDR value etc.

Both SSP and USP are connected as inputs to a 2:1 MUX having the PSR[15] as the select line. This is used for the stack switching operation. When PSR[15]=0, it is in supervisor mode and R6 is loaded with SSP. When PSR[15]=1, it is in user mode and R6 is loaded with USP.

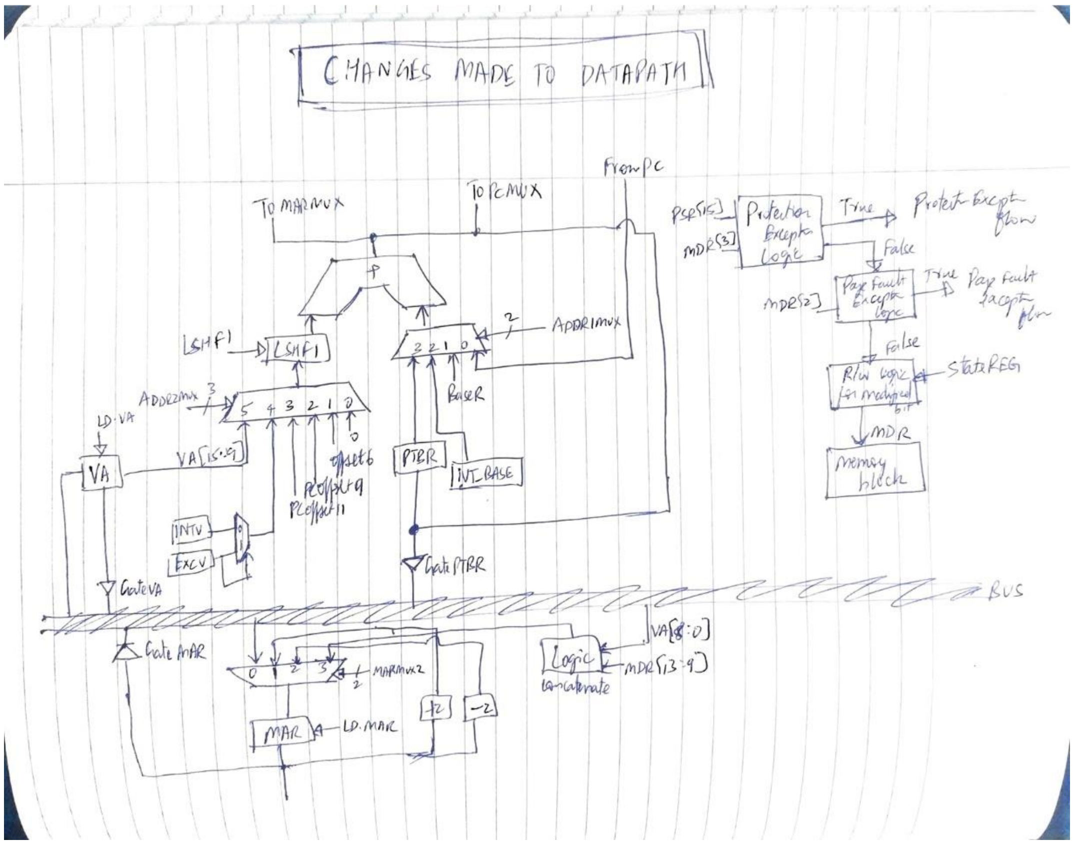
A 4:1 MUX having MAR2MUX (2 control signals) as the select lines are being added above the MAR register. This facilitates the MAR+2, MAR-2 and loading of BUS values onto the MAR register.

On the existing PCMUX, a new input PC-2 is added as the 4th input to a 4:1 MUX. This helps in performing PC-2 operation which is needed in the case of exceptions (State 37).

Existing ADDR1MUX is modified from a 2:1 MUX to a 4:1 MUX by adding IVT_BASE register as one of the extra inputs. Even though this lab assumes the IVT_BASE to be a constant, I have added the register so that it can be loaded with different values and makes the data path generic.

A 2:1 MUX having INTV and EXCV as inputs are added to the data path having the EXCV as the select line. The corresponding values of INTV and EXCV are cleared just before entering the ISR/ entering the ESR (State 48) respectively. Hence the lifetime of the exact values of INTV and EXCV lives only from the detection of interrupt till entering the ISR/ESR. If the values of INTV/ESCV are not cleared, then in the above design, once an exception occurs, any further interrupts will not be detected and serviced. Hence, it is crucial to clear the values of INTV and EXCV appropriately. The resultant output at any time instant from the above MUX is named FINALV which will be used for the start address calculation of the ISR/ESR routine. The existing ADDR2MUX was changed from a 4:1 MUX to a 8:1 MUX for adding the FINALV value as the 5th input value to the MUX. The start address of this new ISR/ESR routine is calculated and is routed to the MAR over the BUS.

b. <u>For Lab 5</u>



The above diagram shows the consolidated list of changes made to the datapath on top of Lab 4 updates.

Two new registers namely VA and PTBR are created to store the values of the virtual address and the Page Table Base Register respectively. There are also other TriState drivers created namely GateVA, GatePTBR and GateMAR which are used to load the virtual address, load the calculated physical address and the current MAR value onto the BUS respectively.

VA[15:9] value is chosen using the ADDR2MUX control signal and the PTBR register value is chosen using the ADDR1MUX control signal.

VA[8:0] from the BUS and MDR[13:9] value are used as inputs to a logic block which will concatenate both the inputs to generate a corresponding physical address for the virtual address present in VA register. This calculated physical address is loaded onto the MAR register using the MAR2MUX control signal.

In order to detect exceptions, PSR[15], MDR[3] and IR[15:12] (to detect TRAP) are used as inputs to a Protection exception Logic block to check whether it is a valid protection exception or not. If there is no protection exception, then MDR[2] is used as an input to the Page Fault exception Logic block to detect a page fault.

A temporary register named STATE_REG is created to hold the next state number when returning from the Virtual to Physical address translation flow. If there is no protection/page fault exception, then this State REG is used an input to determine whether the PTE needs to have the modified bit set or not. The state numbers 23, 24, 39, 54 indicate that it is a STORE operation and hence in those cases, the modified bit will be set to the corresponding PTE.