

Lab 5 Readme Lab Report
Dhruv Sandesara (djs3967)

Note: Only the changes I made on top of lab 4 are mentioned in this document. The immediate next pages contain the final state diagram, micro-sequencer and the data-path for lab5 with the changes made highlighted. For reference the original Lab 4 readme has also been attached afterwards.

1 Changes made to State diagram:

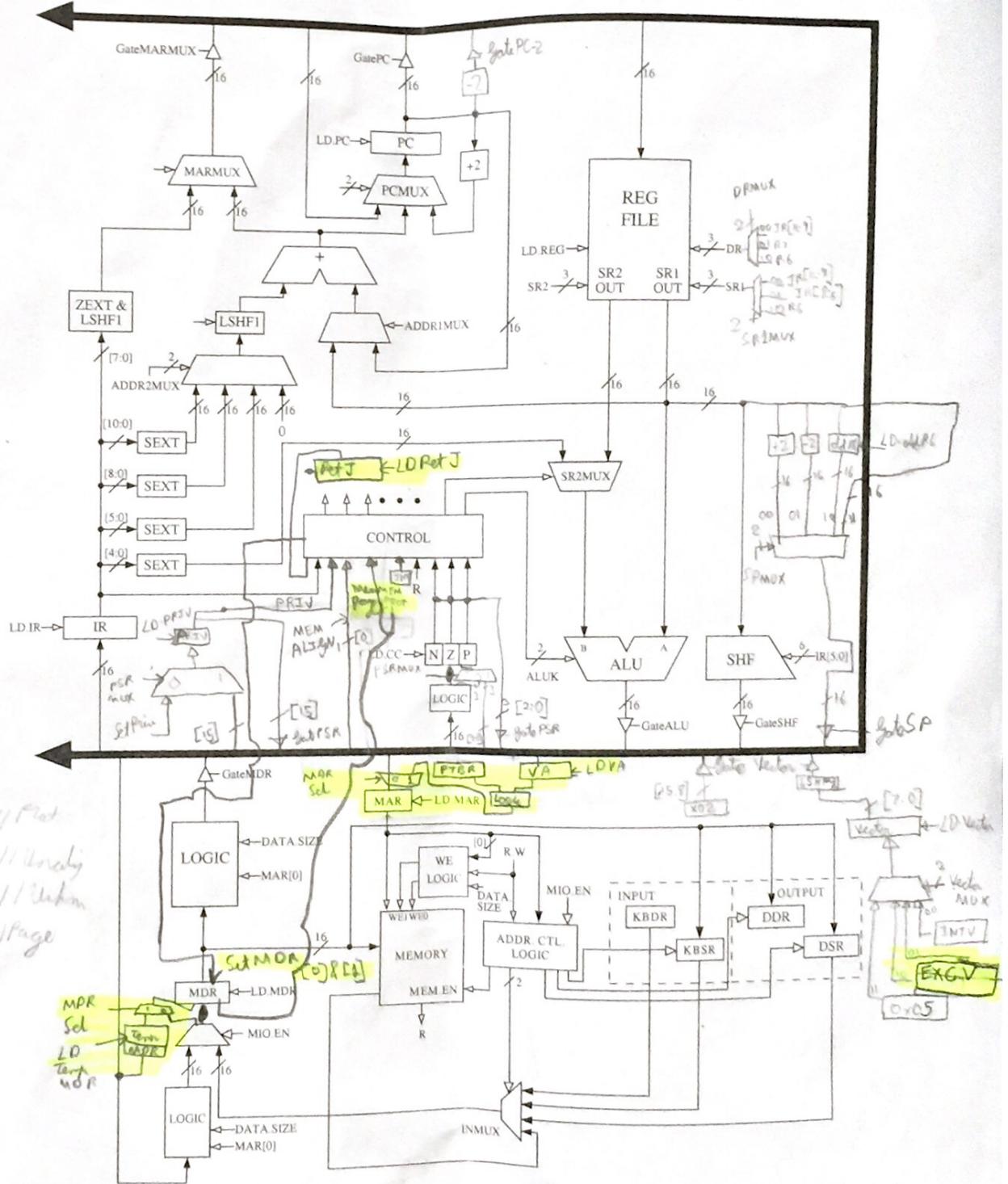
- The biggest change that we made for memory access is that whenever we try to put a value in the MAR we save a return state number for where to go after the VA to PA translation is completed.
- We also load the MAR values to the VA and then we decide where to direct the state.
- The states with the aforementioned changes are states 8,26(new number), 2, 6, 7, 3, 50,46,37, 18,19.
- Then the redirection falls in 3 categories. In case of Word sized access we go to state 45 and 56 for the LDB and the STB instructions. There we check if the BUs value latched to the MAR had a 0 in bit 1 and then throws an exception. If not then the redirection happens to one of the 2 remaining redirections.
- One of them is state 51 which is the Common entry point for any load mem instruction. Or to state 57 which is the common entry point for the store mem instruction. The entry state and the 2 states following that do the same thing.
- They Load the MAR with the PTBR added with the Virtual page number left shifted by one as the PTE is 16 bits.
- Then we wait for the mdr to be loaded with the PTE. When ready we put pte entries in the MDR but at the same time check for a Mem_PROT or Page Fault.
- This logic is by looking at bit 3 and the Priv and if Priv is 1 and Prot is 0 then we send the next control signal for mem prot as 1.
- We also check the bit 2 to get if the page entry is valid or not. And if not we set the next control signal for Page Fault to 1.
- Then we have the third state for Load and Store diverge as we change us the MRD ref bit and set it to 1 for both. But we only set the dirty/ modified bit to 1 in Store.
- In this state we also calculate the Exec vector incase the Prot or Page Fault control signals were detected.
- Then if an exception was detected we go to the common Exception state of 39.
- If not we store back our Modified PTE to the page table.
- This was done in state 41 which is again common for both the load and store which when ready goes to State 43 which loads up the MAR with the PFN with the VA offset bits in 8:0.
- Then in this special state we go back to the ret state number that was stored in the reg before.
- Another major change that I made to make sure the prog fit in 64 state was to consolidate the exception handler vector loading case to 1 common one instead of the 4 different one for lab 4.
- Thus my state 39 loads up the Vector with the EXCV and also loads the PSR to the Temp MDR as the MDR will be overwritten in the address translation.
- This happens in the Interrupt handler case state 59 as well. This temp mdr is popped off into MDR in the common int/exc handler case of state 43.
- One last change was for state 10 and 11 I kept the states vector go to a hard coded 0x05 value.
- Now including all the LDVA Setting Ret addresses we have covered all the state diagram changes

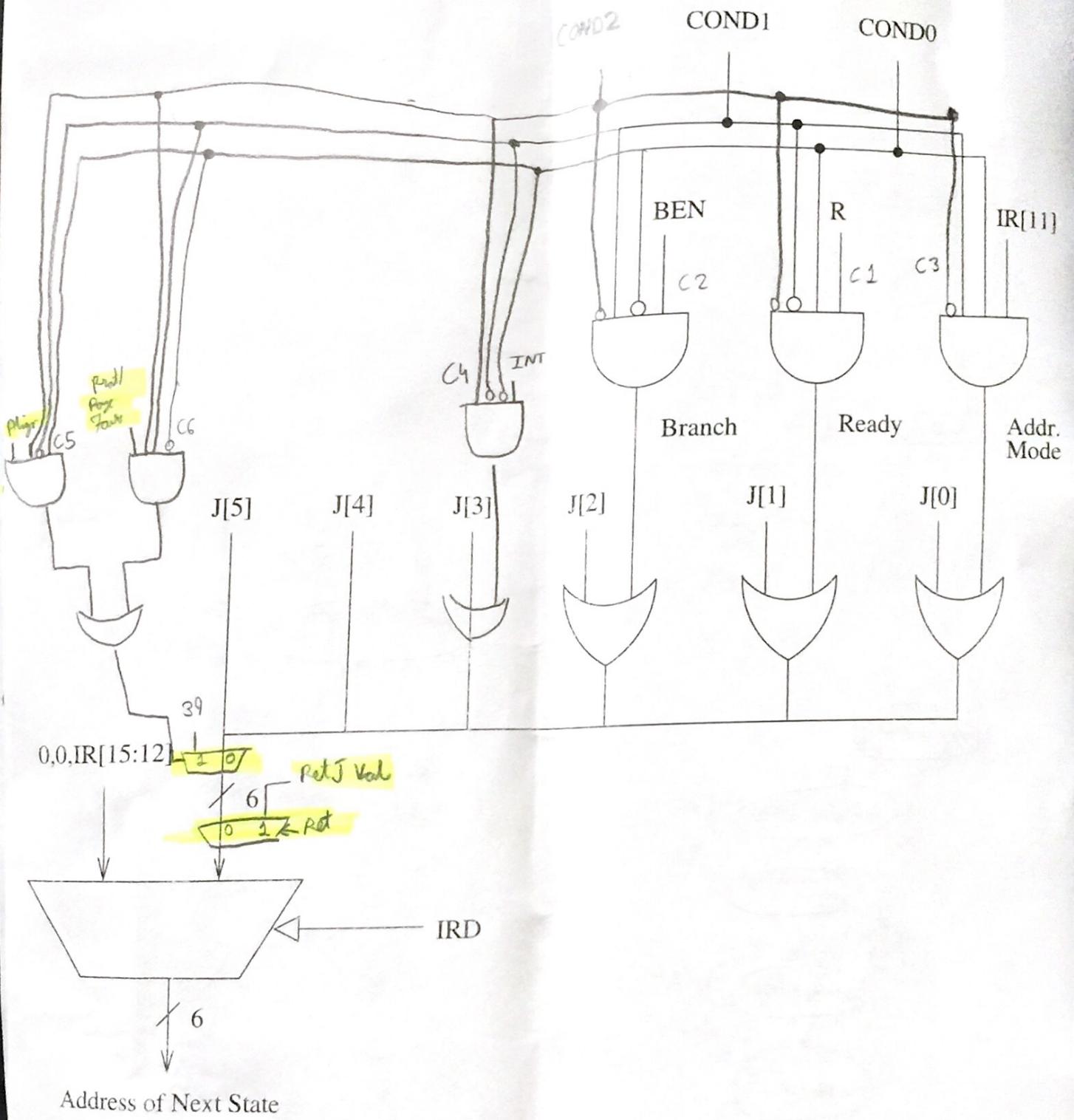
2&3 Changes to the data-path with each control signals role explained:

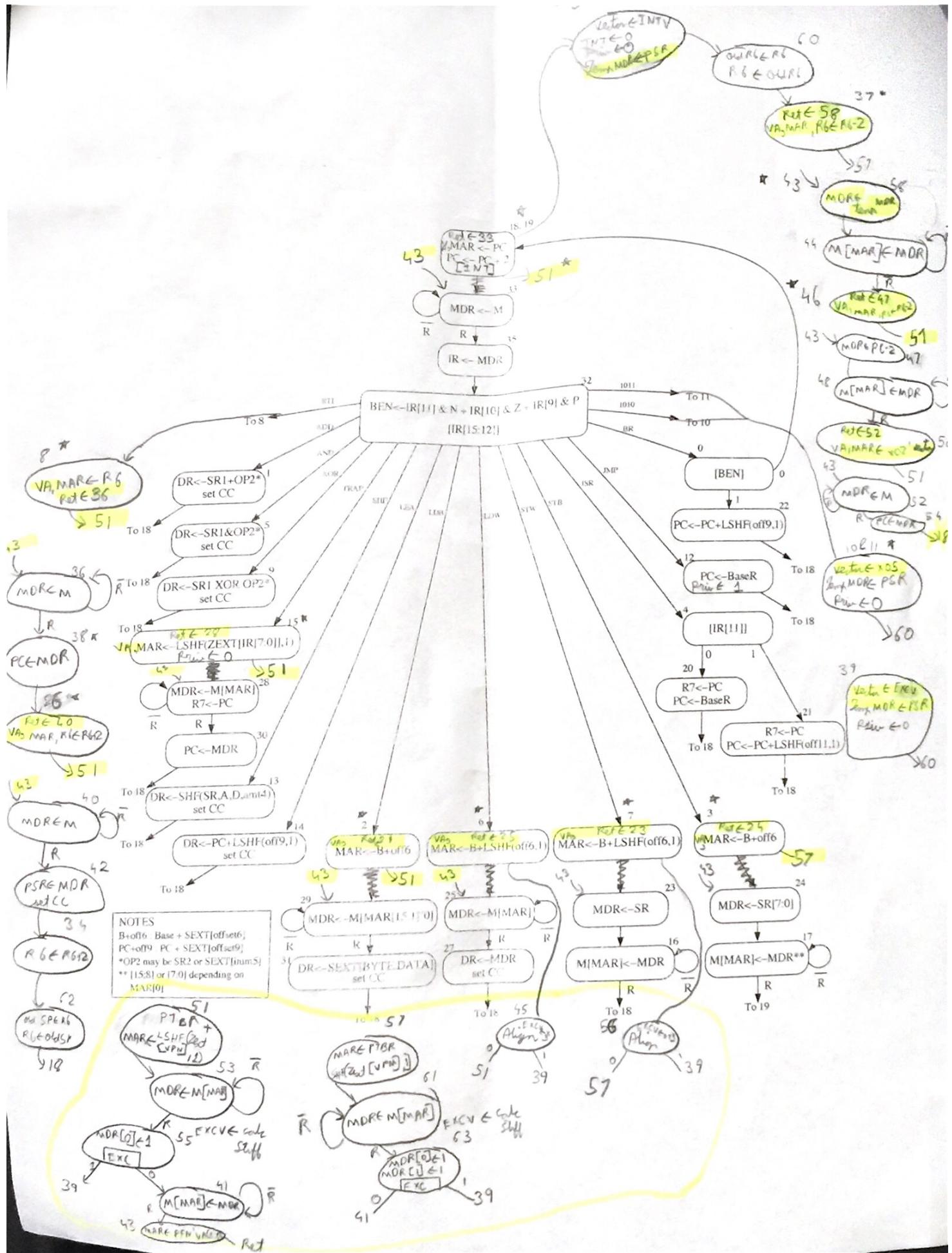
- The first thing I added was the RetJ register. This was to know which state number to go back to once address translation is done. This has with it associated the R bits which have the state. The LDRetJ which loads the value and the GotoRet which basically selects this value as the next state.
- The next 2 values which were modified and added were the Mem_Prot and Mem_Page_Fault. The way the page fault works is to basically look at bit 2 of MDR to see if it is resident in mem.
- The other is the Mem_Prot which checks bit 3 to check if it is a protected page and if it is checks if the Priv is in user mode. If it is a Prot error is thrown. This is to detect a mem PROT access
- The next logic is the PTBR which holds the PTBR values and has nothing associated with it.
- Next change was the adding of VA with the LDVA signal to load the VA so we can use it without writing over it with the physical address.
- Then we have a MAR SEL to select the value of the bus or to use the calculated values from the PTBR + VPN LSHF by1 or the PFN with the VA offset. The diagram shows 1 bit MUX but it actually is a 2 bit so select one of these 3 values.
- Next we also use a TempMDR so as that our original value is not overwritten by PTE entry. So we have a LD signal with it. We can select the MDR input from either the previous logic or the TempMDR.
- Next we also have SetMDR bit 0 and 1 signals to write back the modified and reference values in the page table
- Finally we have the EXCV replacing the hardcoded values of vectors 0x02 and 0x03. This also has a Ld signal with it which is unfortunately forgot to include on the paper but the code does it. It checks if an alignment error had occurred then we would write the 0x03 vector else if it was a prot error, then write vector 4 else if page error then vector 2. The INTV and unknown vector are hardcoded in their own states.
- That does it for the changes to the data-path and control signals.

4 Changes made to the MicroSeq:

- I removed whatever logic we had for the Mem Prot and Mem align. We still have the Interrupt signal changing bit 3 to 1 when condition code is 4 to either go to state 51 or 59.
- We then check if it is condition code 5 and if alignment problems occurred. And we check if the Condition is 6 and a Prot or Page problem occurred. If they did we go to common state 39 and if not we just go to the J bits
- Then finally we check if we want to go back to the Ret value. If so we load J with the RetJ Reg value and if not just go to the original J values.
- That does it for the changes to the MicroSeq.
-







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) Then 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

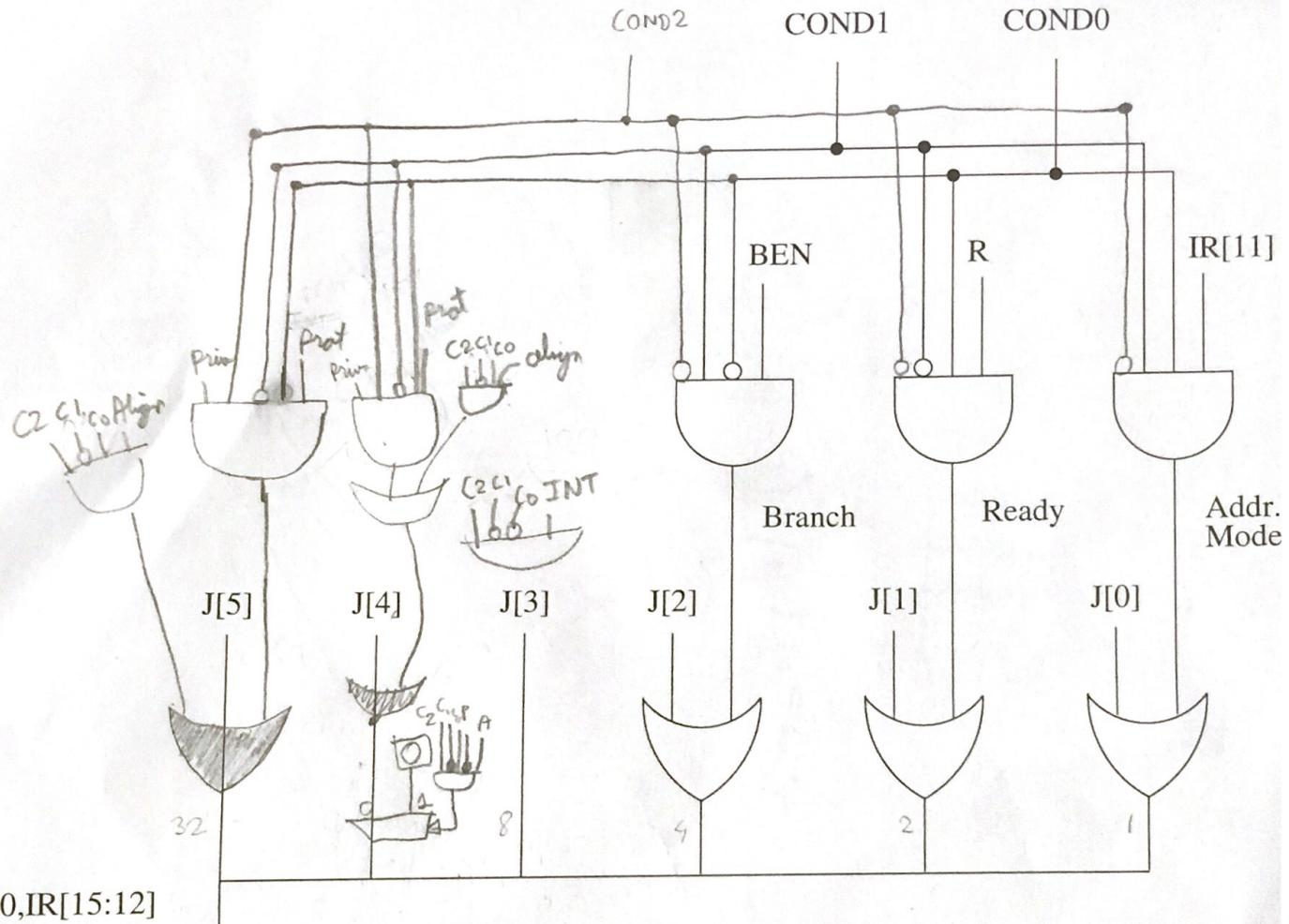
we set the Priv to 0 to make our next executions occur in supervisor mode. And we also clear the interrupt bit to signify that we have handled the interrupt.

- 19) In 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 to 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.

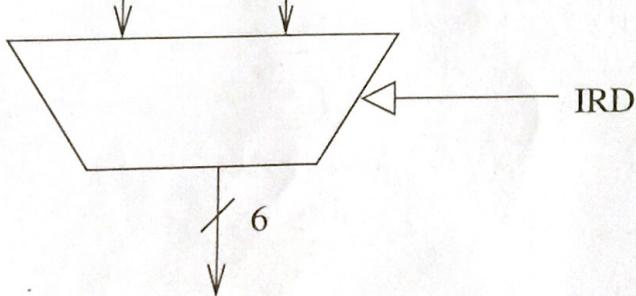
Cord 161

Prot 0
Align 1



0,0,IR[15:12]

↓ 6



Address of Next State

Cord
000 stay
001 Relado,
010 Branch
011 Add mode
100 Interrupt

