# LAB5 Report Shreyas Ravishankar (SR 48925)

## Total number of states used- 59 (61,37,26,10,11 are unused)

## VA->PA translation mechanism

- The VA to PA translation must occur whenever memory is being accessed. This happens once to get the code , i.e the value that is pointed by PC. The second translation is needed if it is a load/store/trap instruction which accesses memory as part of the instruction.
- During exception/interrupt handling we also need 3 address translations. 2 of them are needed to push the value of PC and old value of PSR. One of them is needed to translate the virtual address of the interrupt/exception vector table.
- Even in an RTI instruction, we need to do 2 address translations to pop the values of PSR and PC out of the stack.
- Once we are inside the address translation mechanism, we do not need to do another translation to access the page table entry, as we know it will be present in physical memory, i.e we deal with physical addresses within the series of translation steps.
- The unaligned and unknown opcode exception are detected before we enter the address translation states. The protection and page fault exception happen within the address translation states (state 56), so there is a need to handle address translation of the exception handling mechanism when the address translation for the original instruction is going on.

## New control signals added

- **GateALU_EX-** Handles the the various cases of loading the MAR and R6 with different values related to exception and interrupt handling states.
- **DRMUX2- Mux control signal** used to choose R6 as destination for operations involved in exception/interrupt handling. It is also used for selecting R6 as source in the same states.
- **GatePSR-** Tristate control signal that allows the PSR onto the bus.
- **MUXA-** Control signal for mux connected to input 1 to the ALU_EX which is an ALU for exception/interrupt handling
- **MUXB-** Control signal for mux connected to input 2 to the ALU_EX which is an ALU for exception/interrupt handling

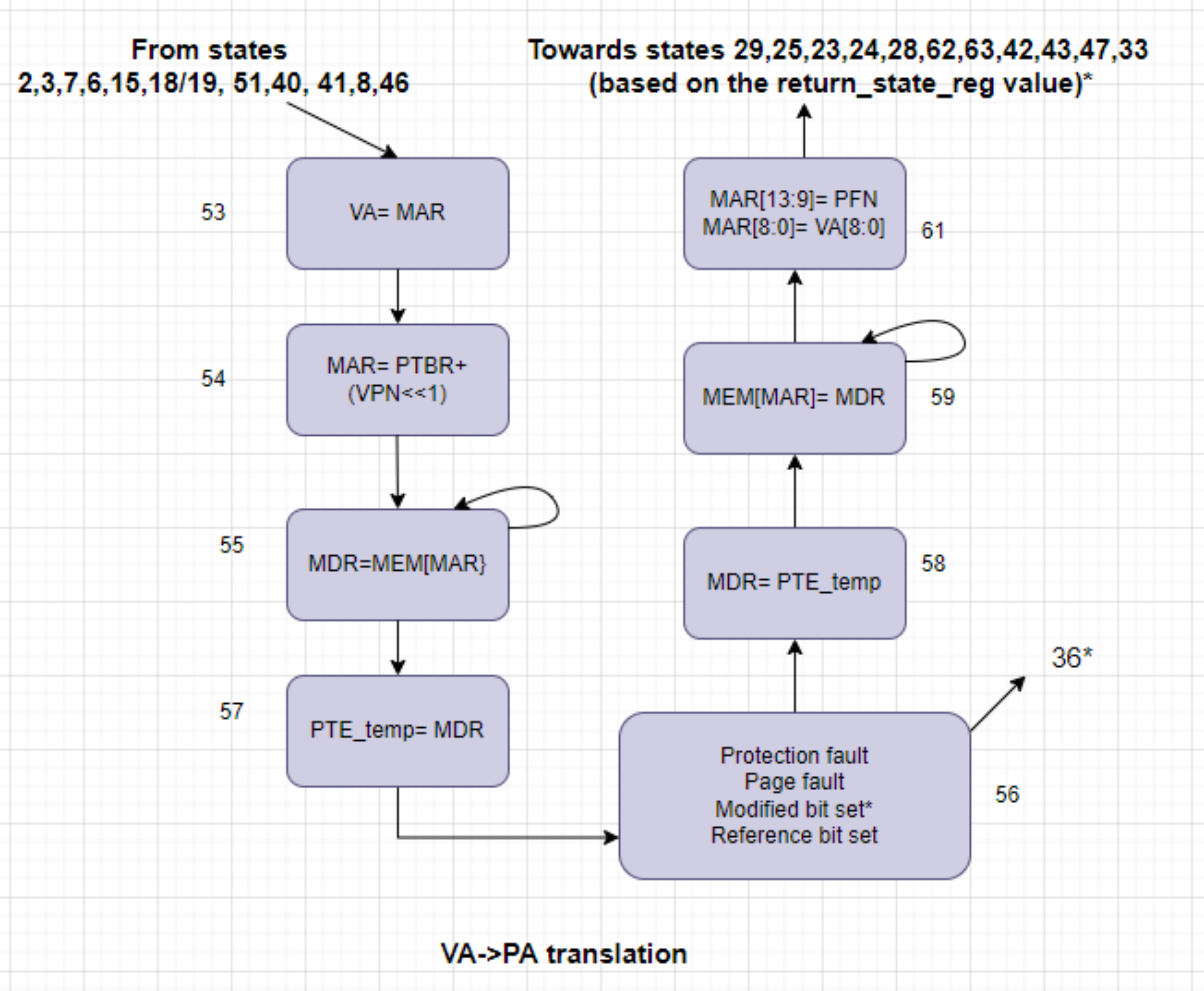
- **MAR_VA-** Used to indicate that virtual address translation is complete, and that we need to go back to using the physical value of MAR to read/write from physical memory.
- **PSRMUX-** Select line for mux that either changes the privilege mode of the PSR or loads its value from the bus.
- **LD_PSR-** Control whether to load the PSR or not in the current cycle.
- **RESET_VEC-** Used to reset the interrupt and exception vector to support nested exceptions/interrupts.
- **GATE_PTBR-** Controls whether the effective address of the PTE entry (PTBR+VPN<<1) goes into the bus or not.
- **GATE_PTE_TEMP-** Signal that tells whether PTE value has to be put on the bus or not.
- **LD_PTE_TEMP-** Control signal indicating whether we have to load PTE value into the PTE_temp register or not.
- **PTE_MUX-** Control signal to the mux that goes into PTE_temp register. Indicates whether we load a value from the bus, or change the protection or modified bit.
- **LD_VA-** Load the value of MAR into VA, this happens at the start of address translation.

## New registers/Latches added

1. **INTV**- Register used to store the interrupt vector
2. **EXCV**- Register used to store the exception vector
3. **SSP**- Constant holding the value of supervisor stack pointer
4. **PSR**- Process status register
5. **EXbit**- Bit to indicate exception has occurred
6. **INTbit**- Bit to indicate interrupt has occured
7. **Vector_Base**- constant containing base of the vector table
8. **USP**- user stack pointer
9. **VA-** Holds the virtual address during address translation.
10. **PTE_temp**- Holds the current value of PTE, for exception checking (page fault and protection), as well as modifying the PTE value before storing it to memory.
11. **VA_mode**- A pulse like signal which is 1 for one cycle when we start address translation.
12. **Return_state_reg**- Holds the return state to jump to after address translation
13. **Translated**- Register which tells us whether translation has been done or not.
14. **PSR_temp**- Holds the old value of PSR to be pushed onto the stack.

## Brief description of states

### VA->PA states




State 53- We enter this state whenever we load values into MAR, as elaborated previously. We don't do this if we are dealing with physical addresses i.e in the address translation states themselves (Ex- state 54).Puts the virtual address (MAR) into a register VA.

State 54- Uses a combination of virtual address and PTBR register to get the address of the PTE entry in the page table.

State 55- Stores the value of PTE entry into MDR. We remain in this state until PTE entry value is loaded from memory.

State 57- Loads the value of MDR (PTE entry) into a temporary register PTE_temp

State 56- Checks for protection and page fault exceptions. If exception go to state 36 (exception handler). If no exception, Sets the reference bit and modified bit appropriately based on whether we are writing to the location or not. This not only

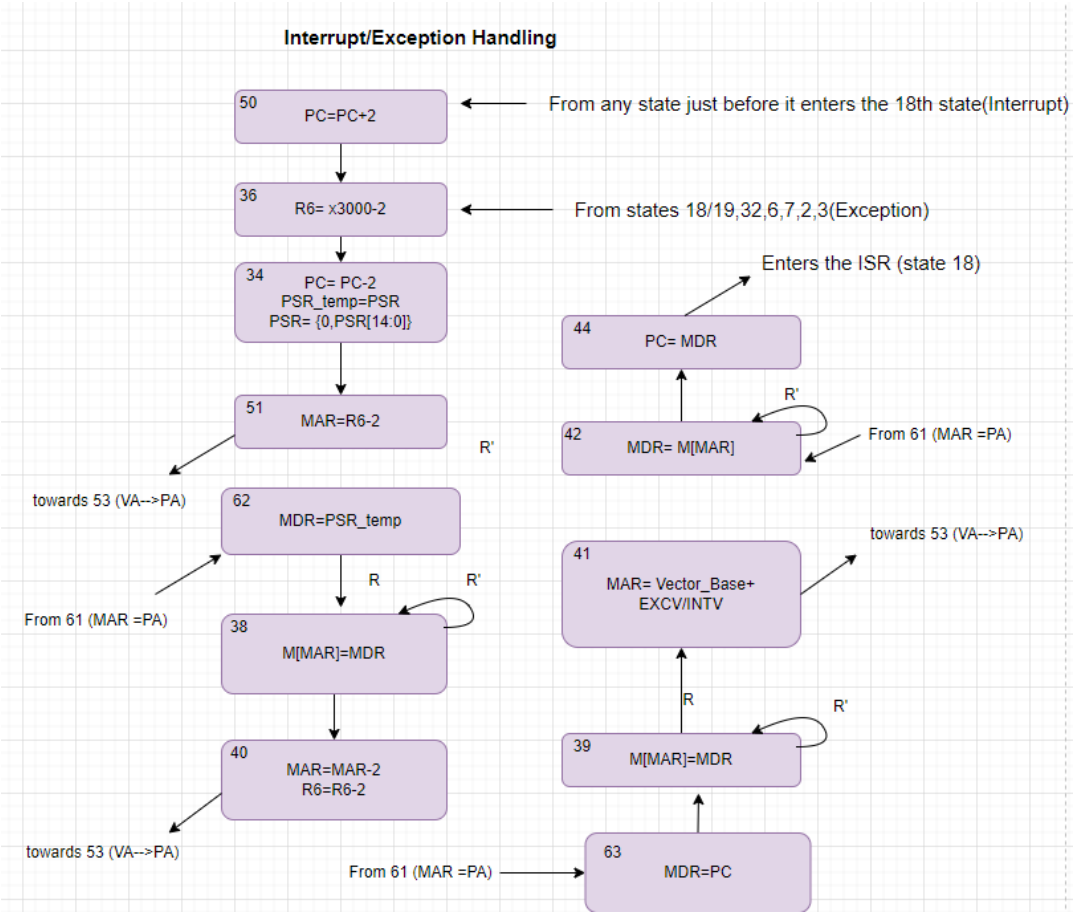
happens on a store instruction but when we push values onto the stack during exception/interrupt handling.

State 58- Put the possibly modified value of PTE_temp into MDR, to load it back into memory.

State 59- Store the MDR into memory. Since MAR has not changed since state 53, it points to the location of the PTE entry, so we are writing to the correct location. We remain in this state until this store takes place.

State 61- Changes the value of MAR, i.e puts the value of physical address onto MAR. This step completes the address translation and we go back to the next state of where address translation took place using the temporary register (return_state_reg).

## Interrupt/exception handling states




State 50- The interrupt handling starts from this state. It increments the PC since we are supposed to return to the next instruction on an interrupt. A later state decrements PC, so this step is required so that the PC that is pushed onto the stack doesn't change for interrupts.

State 36- Starting state for exception handling. R6 is decremented to push the value of PSR onto the stack.

State 34- The PC value is decremented (In-order to re-execute the same instruction when ESR returns). For ISR-> PC+2-2= PC will be eventually stored onto the supervisor stack. We need to change privilege mode to supervisor mode if need be (might not be true incase of nested exceptions and interrupts). This is needed before we access supervisor stack. We store the value of PSR, so that old value can later be stored onto MDR (in state 62).

State 51- The value of R6-2 is put into MAR. We are trying to access the stack which is present in memory, hence we have to proceed via MAR and MDR. Now we go to state 53, to do address translation.

State 62- We return from addr. Translation into this state which loads the old value of PSR (PSR_temp) into MDR.

State 38- Now that value of MAR, and MDR are ready, MDR value is stored in M[MAR]. This operations basically pushes the old value of PSR onto the stack. Until the data is stored , we remain in the same state. Once stored, proceed to state 40.

State 40- Decrements the value of MAR, and stores the value of PC in MDR. We also decrement the value of R6 to update the stack pointer. We are now getting ready to push PC onto the stack. Since MAR has been loaded with virtual address, we need to go to state 53, to do the VA to PA translation.

State 63- We return to this state after address translation, and store the PC value to MDR, since we need to push this onto the supervisor stack.

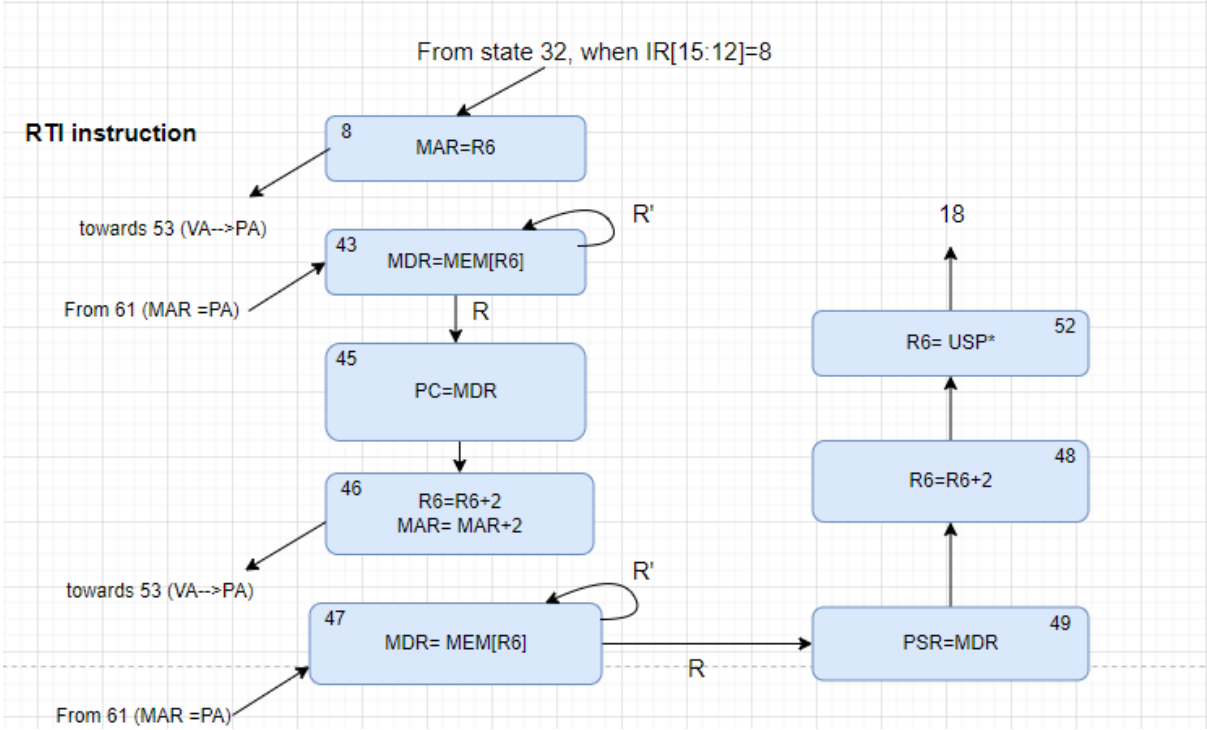
State 39- Value of PC stored onto the supervisor stack. Wait until this store is completed before moving to the next state.

State 41- Now we populate MAR with effective address that is present in the vector table. This involves adding the Vector_base register to the appropriate exception vector or interrupt vector. Go to address translation state to translate this system address.

State 42- Return to this state after translation. Now we retrieve/load the value at that location. This corresponds to the address where we have to jump, i.e the location of the ESR or ISR. We jump to state 44 only on completion of this memory operation.

State 44- The value obtained in the previous step is loaded onto PC, so that we can start executing the ISR. Therefore the next state will be 18.

## RTI states

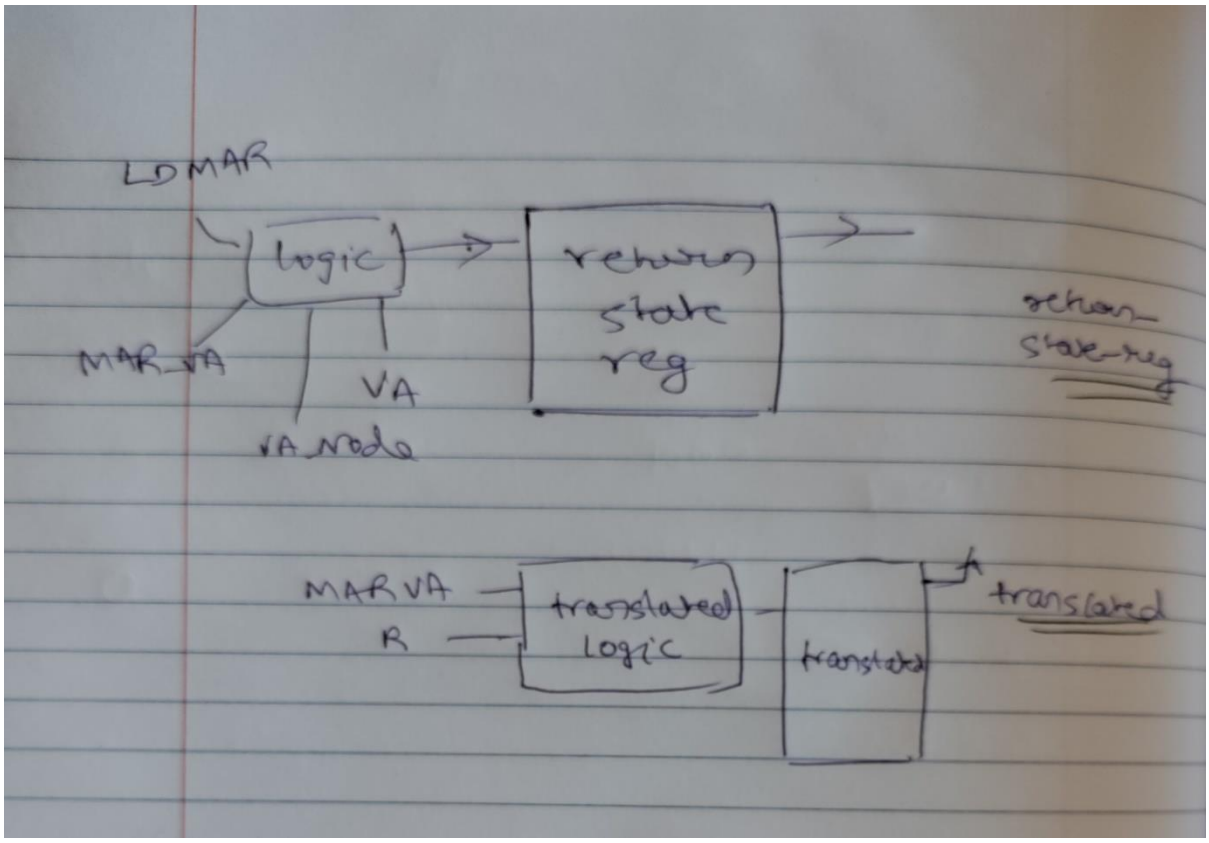


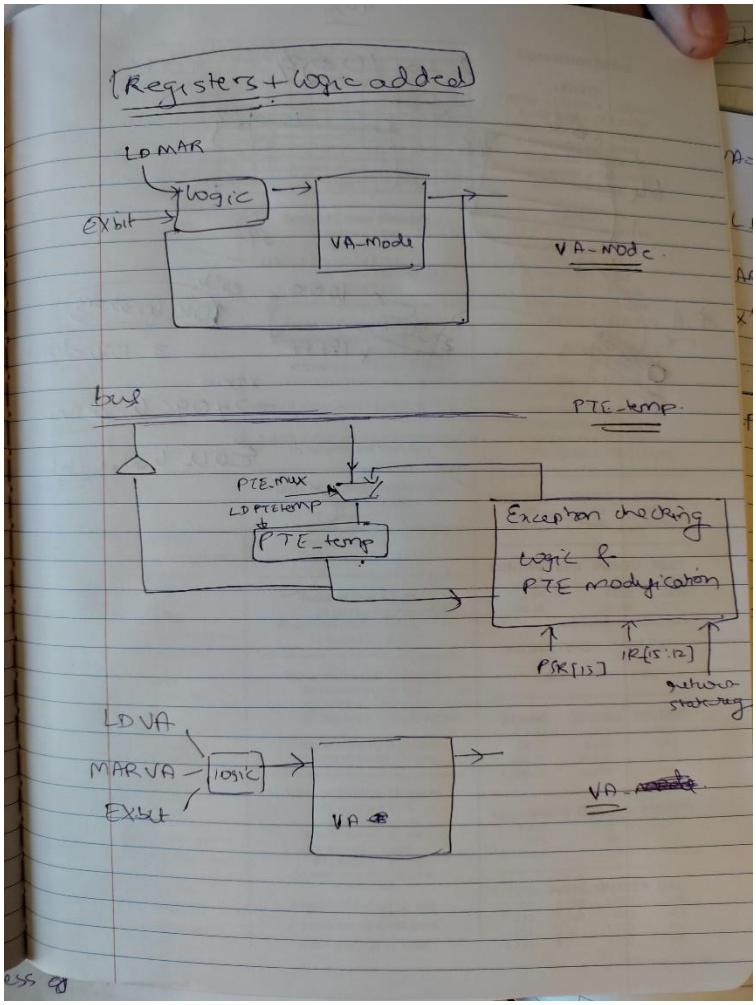

The ISR/ESR usually ends with an RTI instruction, the following describes new states inorder to support this instruction. Basically, we need to undo the changes that were done while switching into the ISR/ESR.

1. State 8- MAR is loaded with R6 which points to the top of the stack. Go to address translation.
2. State 43- Return from addr translation to this state. The value at the top of the stack is retrieved. This memory operation doesn't proceed to state 45 until load is complete.
3. State 45- Since we pushed PC last, we will obtain it first, hence the value obtained is stored in PC.
4. State 46- The supervisor stack pointer is incremented to point to the top of stack since PC is popped out. Go to address translation.
5. State 47- Return from addr translation to this state. The value of PSR is obtained in MDR. Again, this memory operation proceeds to state 49 only after completion.
6. State 49- the value of PSR is restored.
7. State 48- The value of R6 is incremented since PC, and PSR were popped out of the stack.

8. State 52- This state is optional, and only occurs when the ISR/ESR is returning to the user program. Incase of nested interrupts/exceptions, this step is skipped.

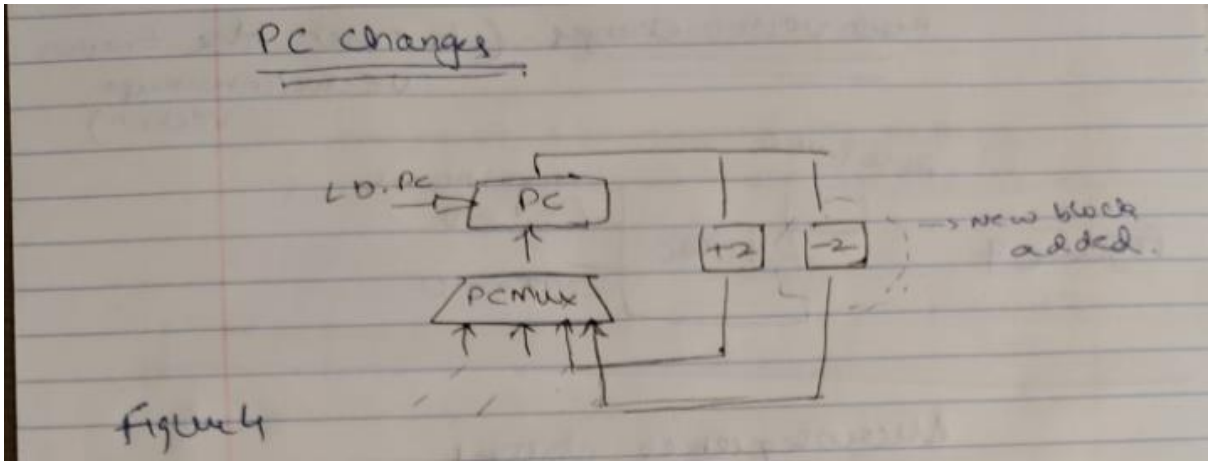
## Extra structures added (VA )






The need for the other structures has already been explained in the system latches/registers added section.

## Exception interrupt handling structures



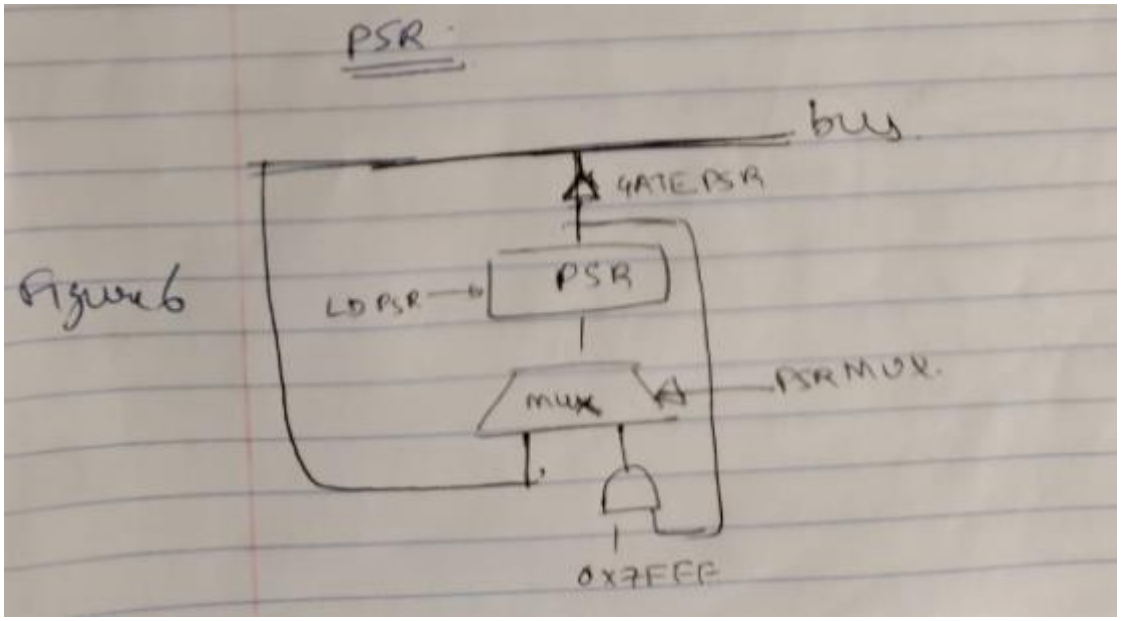


Figure 6

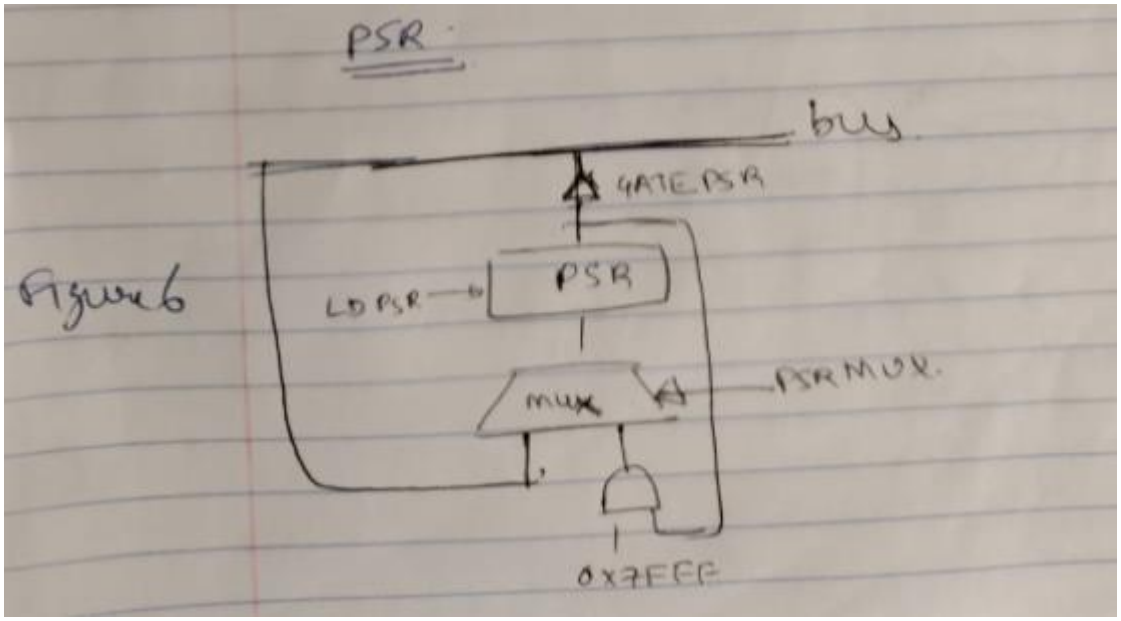
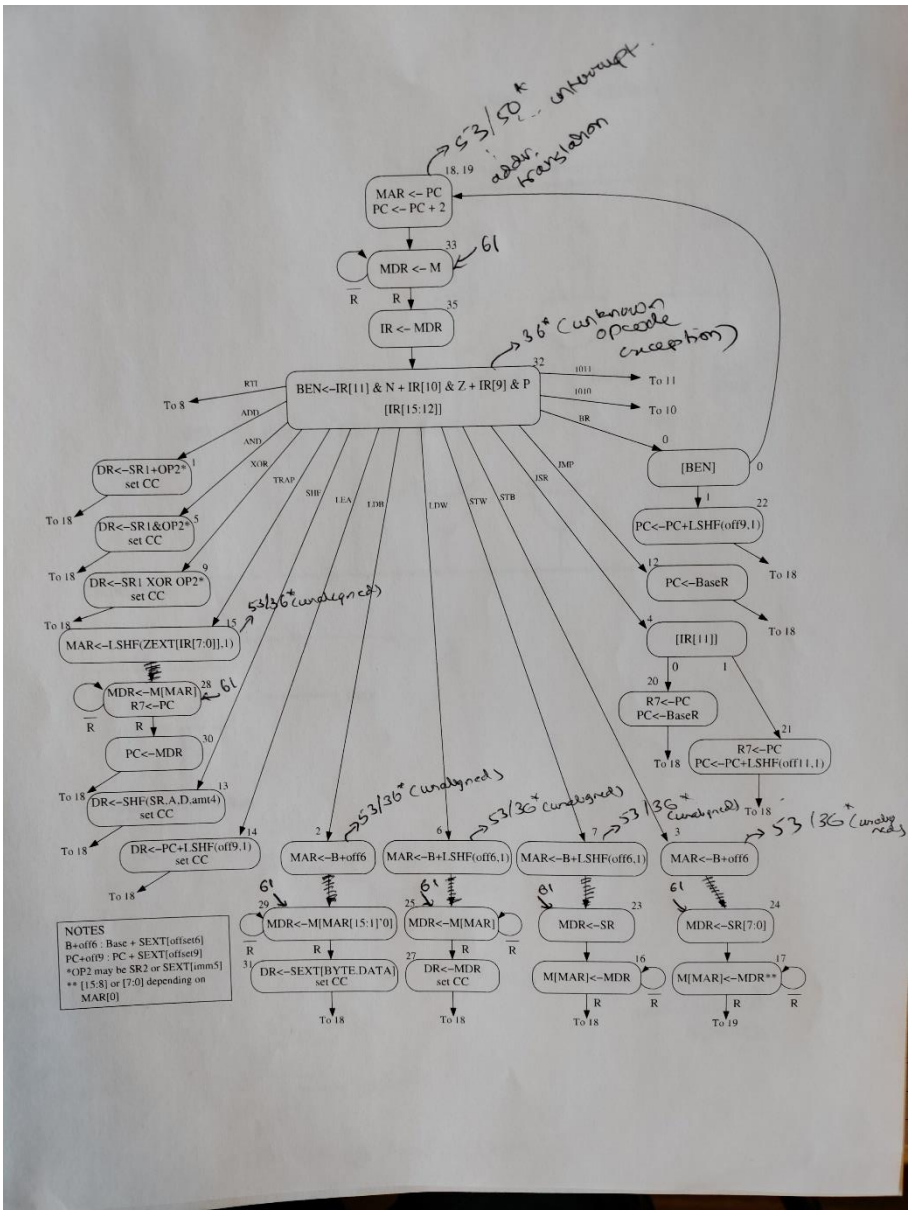


Figure 6

## State diagram



## Microsequencer