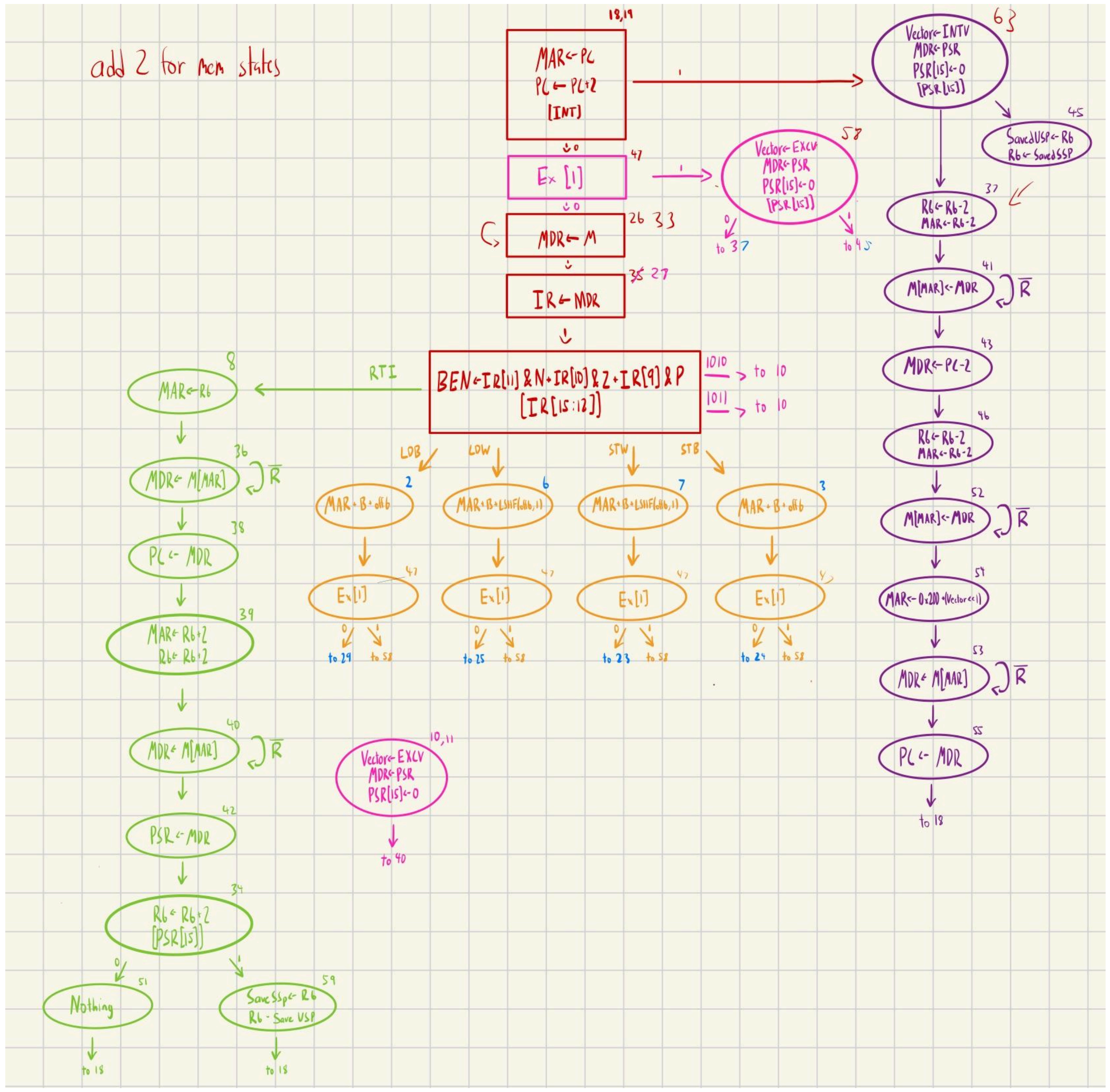


When loading the MAR to fetch the next instruction (states 18, 19), we now check if an interrupt has been asserted. If an interrupt is active, the control logic transitions into states **63–55** to perform the interrupt/exception context switch and begin servicing the interrupt.

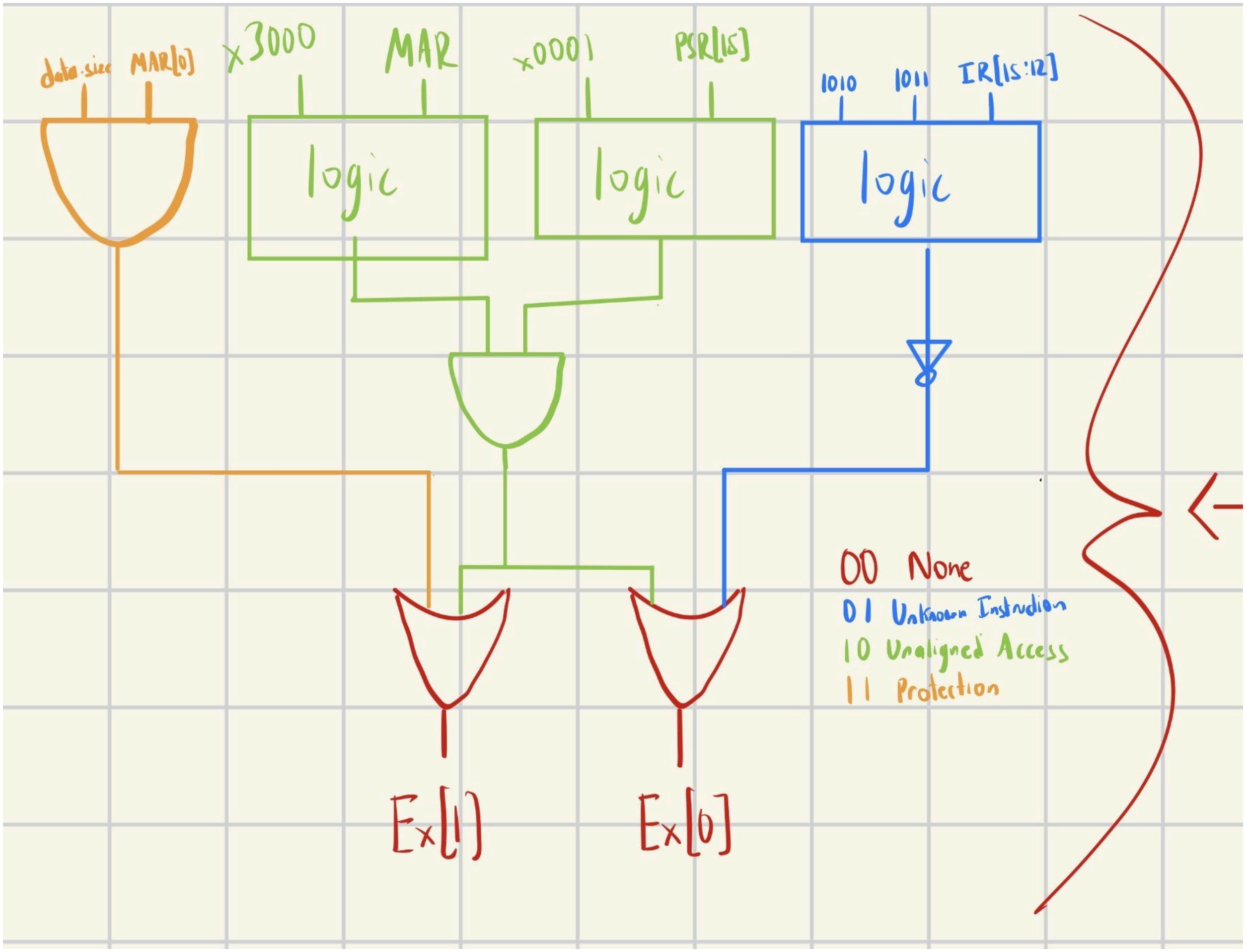
During this sequence, the vector register is loaded with the interrupt vector, the processor switches to supervisor mode if necessary by saving the user stack pointer and restoring the system stack pointer, and the PSR and PC of the user program are pushed onto the system stack. The PC is then updated with the starting address of the Interrupt Service Routine (ISR).

State **47** is responsible for detecting all exceptions other than illegal opcodes. When one of these exceptions is detected, the machine initiates the exception context switch in the same way as interrupts, except the vector register is loaded with **EXCV** instead of **INTV**.

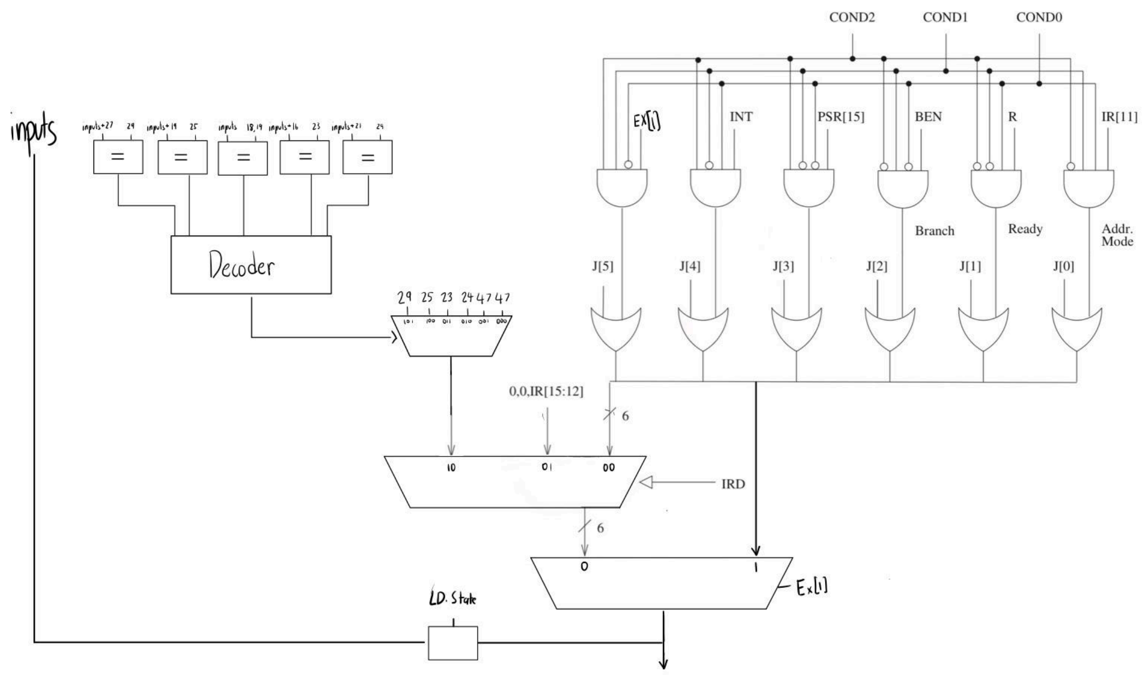
The illegal opcode exception is handled separately in state 32 and directs control to **states 10 and 11** to process it.

Finally, states **8–59** implement the **RTI** instruction. This sequence restores the previous PSR and PC values from the system stack and switches back to user mode if required, saving and reloading the appropriate stack pointers.

You will notice that state State 47 appears multiple times in the microsequencer, each instance corresponding to different input conditions and outcomes depending on both the incoming state and the value of EX[1]. This complexity is managed by an enhanced microsequencer and supporting hardware used to compute the EX signals. Several structural changes were introduced: a new IRD bit, a signal called LD_state added to the control store bits, and an int state register added to the system latches. These enable more flexible next-state control without requiring memory copying each cycle. Microsequencer and exceptions explained below.



To determine exception conditions, EX[1] is set to 1 if the MAR < x3000 while PSR[15] = 1 (indicating a protection fault) or if a byte operation is accessing an unaligned address (i.e., data.size = 1 and MAR[0] = 1). EX[0] is set to 1 if the MAR < x3000 while PSR[15] = 1, or if the IR[15:12] opcode corresponds to 1010 or 1011 (illegal opcodes). Thus, EX = 00 indicates no exception, 01 indicates an unknown opcode, 10 represents an unaligned access, and 11 signals a protection exception.

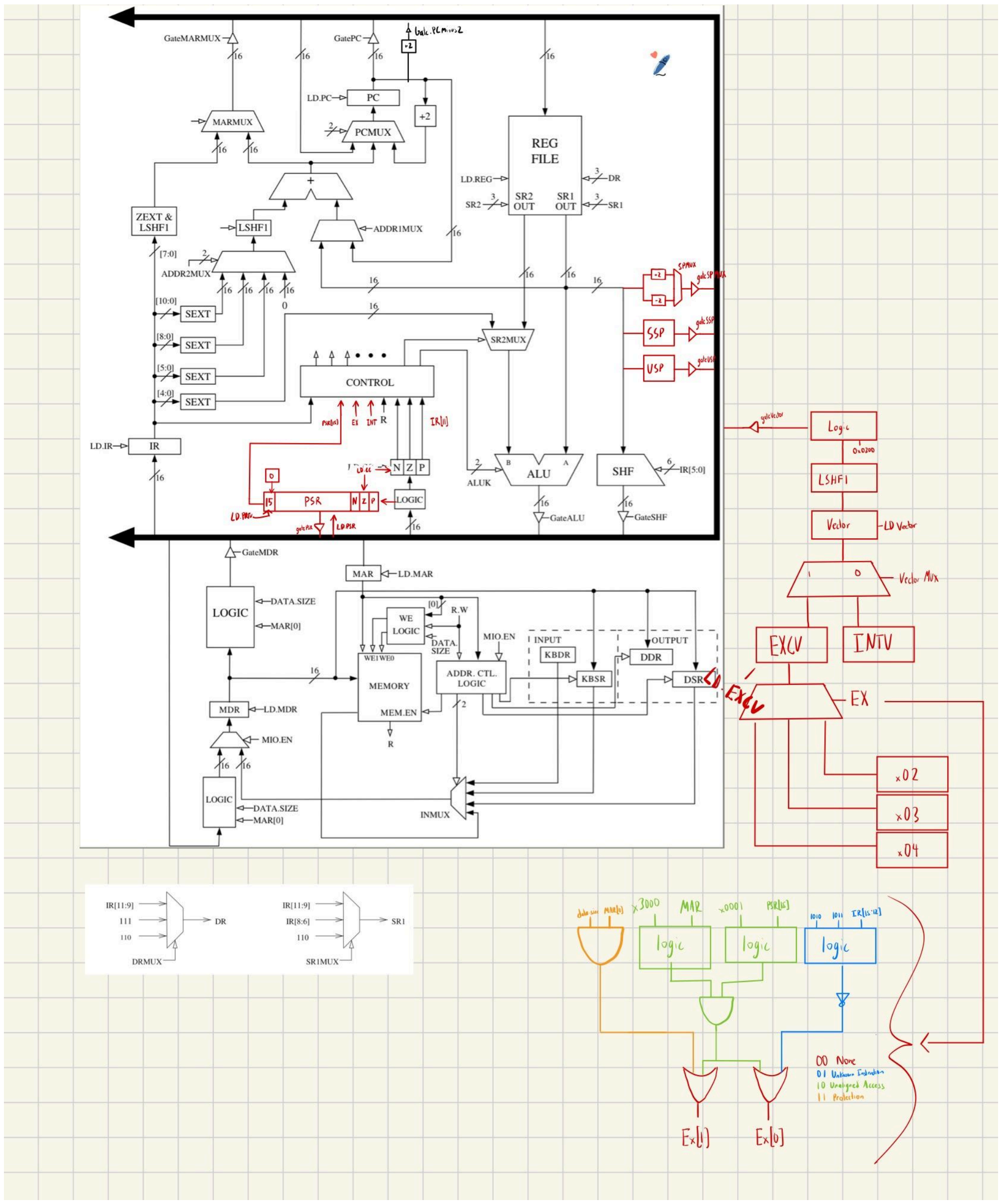


After calculating these values, the next state is determined. If EX[1] = 1, control transitions directly to state 63 to handle the exception. **Otherwise, the sequencer proceeds to evaluate the IRD bits which have been expanded to be 2 bits now.**

- If IRD = 10, the next state depends on the current internal state: states 2, 3, 6, 7, 18, and 19 transition to states 29, 25, 23, 24, 33, and 33, respectively.
- If IRD = 01, the next state is determined by the opcode in the IR (this occurs only during the decode state).
- If IRD = 00, the next state is computed from the normal J bits of the microinstruction.

Before the microsequencer completes, it checks LD_state. If LD_state = 1, the internal state register is updated with the next state value. The purpose of this state register is to remember where to continue execution after an exception check. For example, if the machine is in state 18 and an exception check occurs next, it must know to branch to state 63 when an exception is present. If no exception occurs, the system restores the previously saved state value so execution continues normally. LD_state is asserted during the state immediately preceding an exception check, ensuring that the proper next state is remembered and restored once the exception logic determines that no fault has occurred. I'm super proud of this microsequencer, and although a TA recommended making it simpler, I wanted to keep the design I'd made with 0 outside advice (it did make coding hell though😂).

Other changes to the microsequencer include the INT signal and PSR[15] which are used to microbranch with respect to bit 4 and 3 (respectively) of the state number.



Datapath: Added r6 to Sr1 mux and Drmux. Added hardware for PSR, SSP, +2, -2, and choosing + loading exceptions/interrupts. The Ex[] from before also acts as a mux line input to determine what exception is loaded. Then vectormux chooses if its ex/int, then value is shifted and added and eventually loaded to bus. More explanation below but hardware here is very simple.

ADDED MICROCODE

- **IRD (Added bit)** – Determines the source of the next state. When asserted, the next state comes from the instruction register or a fixed mapping rather than the J bits.
- **COND (Added bit)** – Used to select between interrupt, exception, or normal control flow based on current system conditions.
- **LD_PRIV** – Sets the 15th bit of the PSR to 0, switching the processor into supervisor mode.
- **LD_USP** – Loads the User Stack Pointer (USP) register during a mode switch from supervisor to user mode.
- **LD_SSP** – Loads the System Stack Pointer (SSP) when switching from user to supervisor mode.
- **LD_EXCV** – Loads the exception vector register with the value determined by the **EX** signals.
- **LD_VECTOR** – Loads the final vector register after all interrupt or exception calculations are complete.
- **GATE_PC–2** – Places **PC – 2** on the bus, allowing the processor to store the correct return address on the stack.
- **GATE_PSR** – Drives the PSR value onto the system bus so it can be saved during context switches.
- **GATE_SP** – Places the current Stack Pointer on the bus to enable memory operations that store or retrieve stack data.
- **GATE_VECTOR** – Puts the selected interrupt or exception vector value onto the bus.
- **GATE_SPMUX1 / GATE_SPMUX0** – Selects between incrementing or decrementing the stack pointer by 2, depending on whether data is being pushed or popped.
- **VECTOR_MUX** – Chooses between loading an interrupt or exception vector based on the system state.
- **LD_STATE** – Loads the saved **state** register used by the microsequencer after an exception check, ensuring the system resumes from the correct state if no exception occurs.