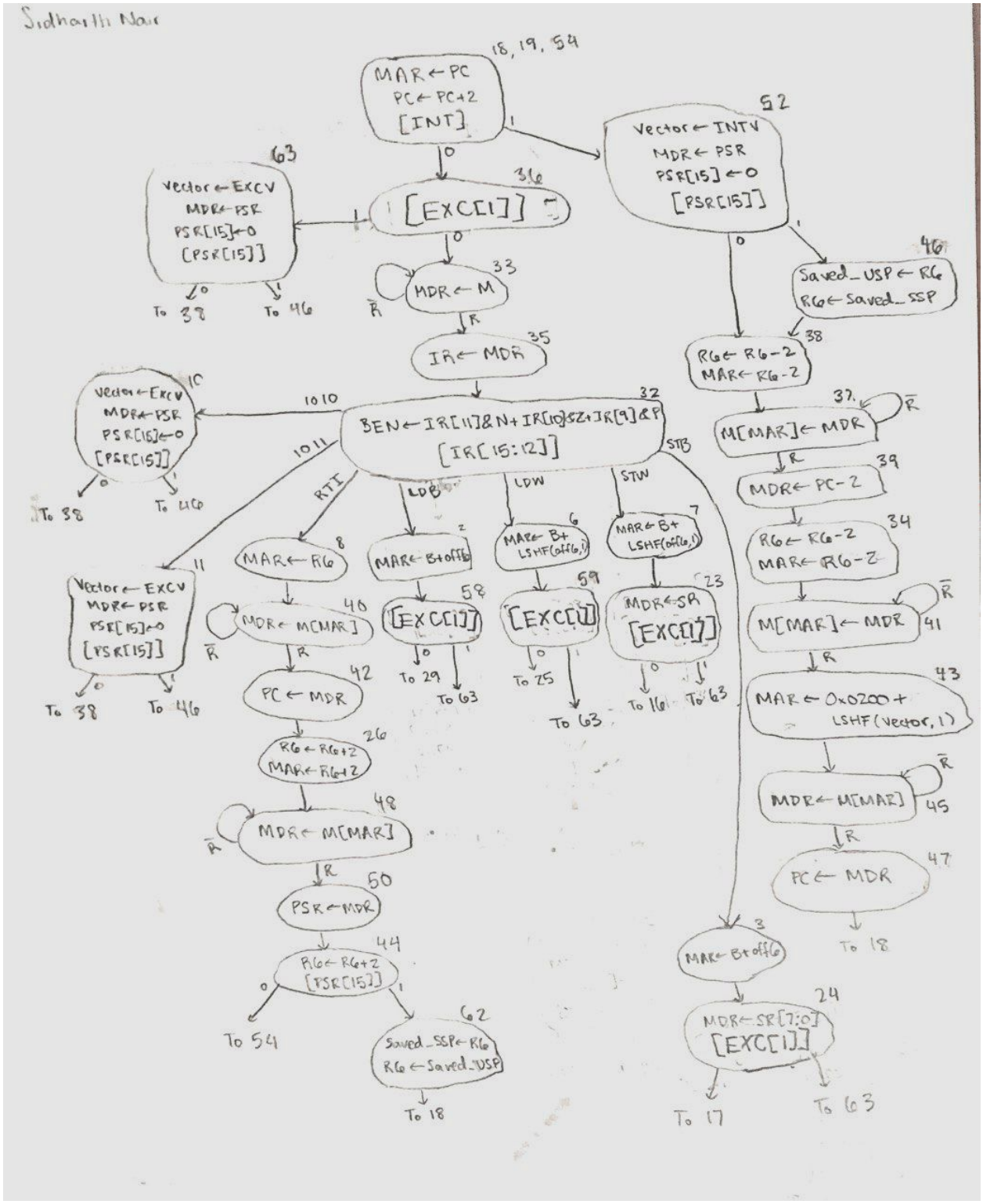
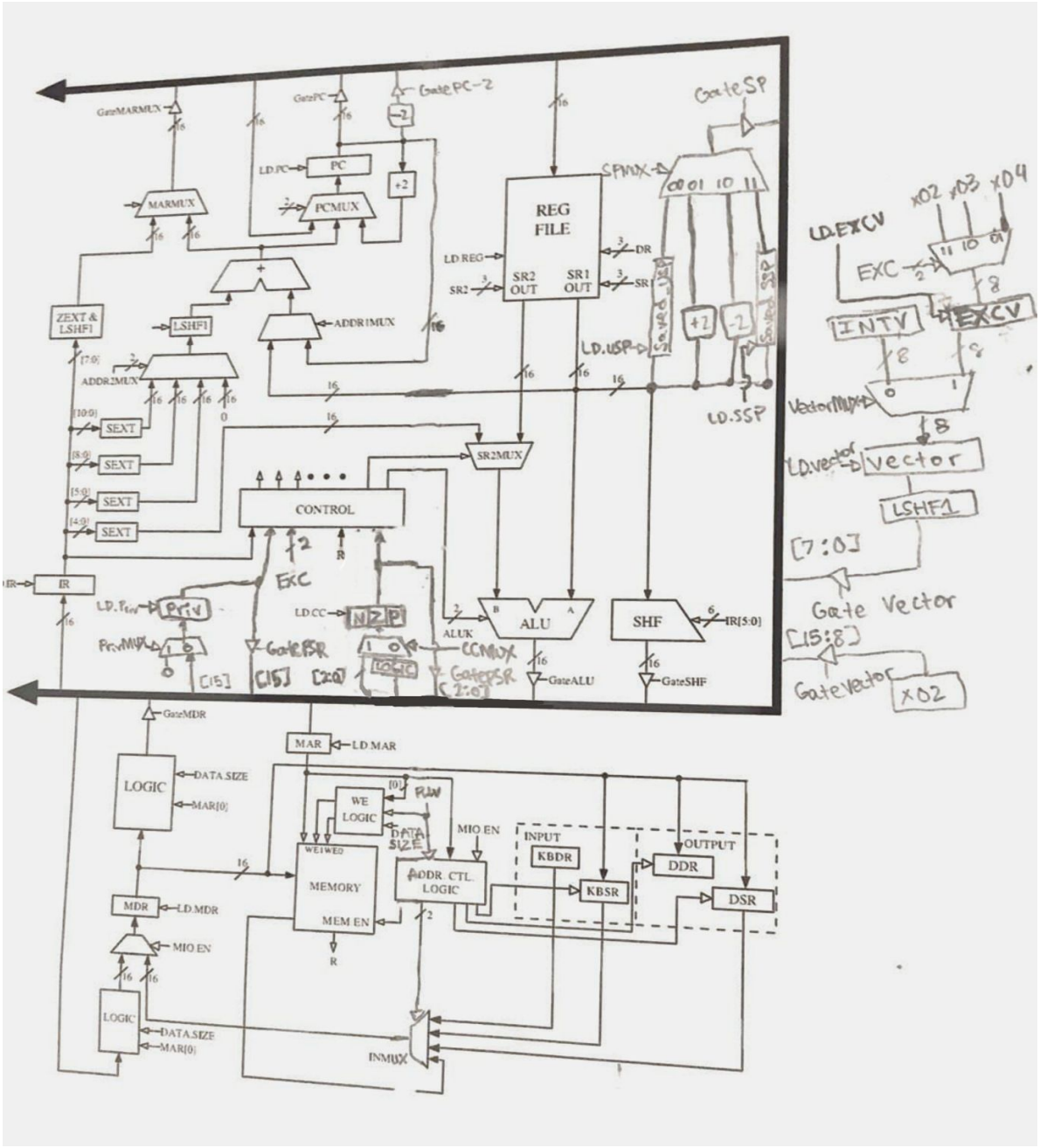


Modifications to the state machine: When we load the MAR to get the next instruction (state 18, 19, 54) we check to see if an interrupt signal has been asserted. If so, we enter a series of states (states 52, 46/38, 37, 39, 34, 41, 43, 45, 47) to perform the interrupt context switch so that we can execute the interrupt subroutine. This context switch consists of loading the interrupt vector into the vector register (state 52),

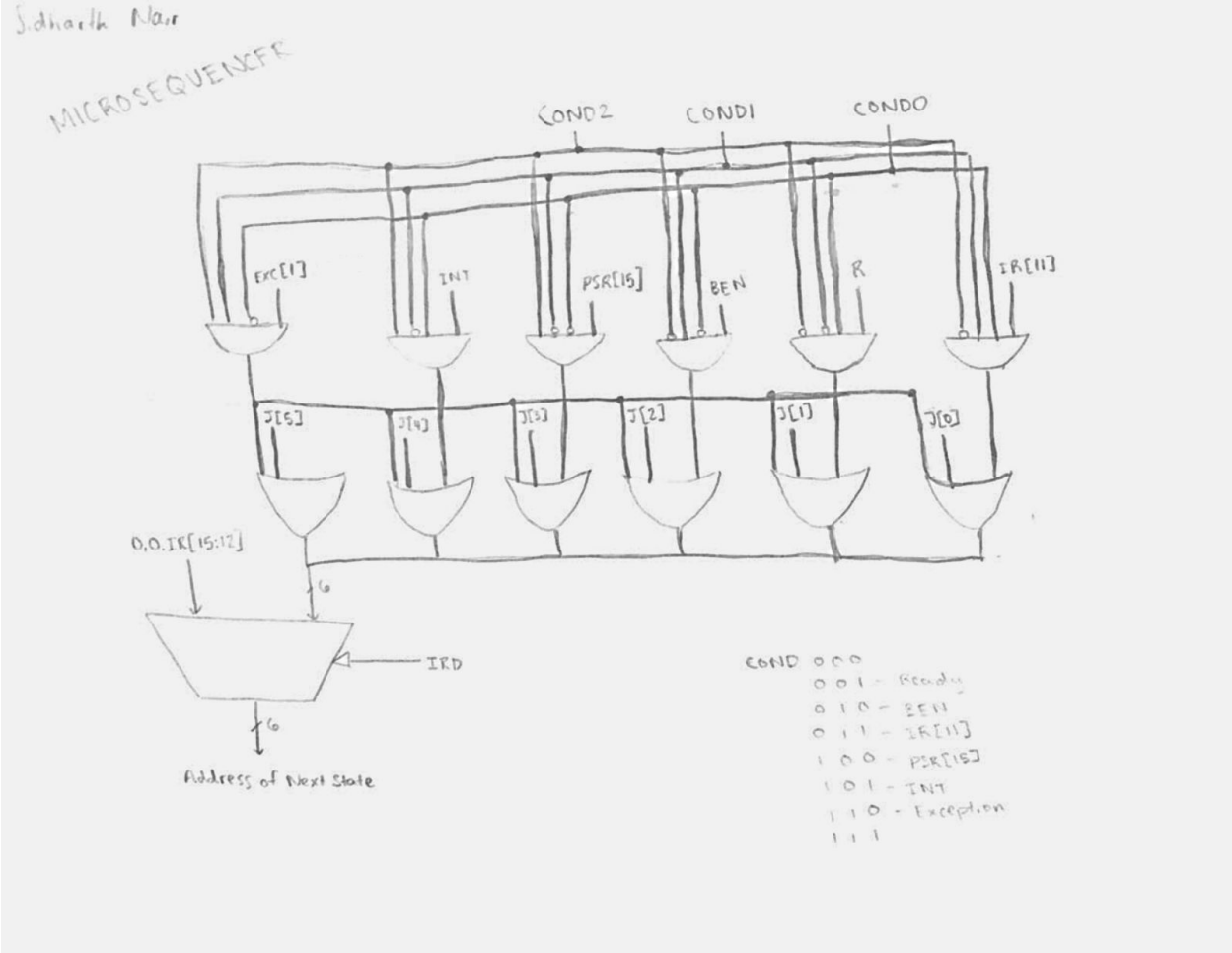
checking to see if we need to switch into supervisor mode and performing this switch by saving the user stack pointer and loading the saved system stack pointer if necessary (states 52, 46), pushing the PSR and PC of the user program on the system stack so that we can return to it after the ISR (states 38, 37, 39, 34, 41), and finally loading the PC with the starting address of the interrupt service routine (states 43, 45, 47). State 36 serves to check whether the PC is set to a protected region of memory while in user mode or that the PC is unaligned (caused by a JSRR or JMP instruction). If one of these exceptions is detected (indicated by the EXC[1] signal) we will begin the exception handler context switch (state 63). Since the context switch is very similar to the interrupt context switch, the same states were used after the initial state which only differed from the initial state of the interrupt context switch by loading vector with EXCV instead of INTV. The check for unaligned accesses or unprivileged accesses also occurs in states 58, 59, 23, and 24 (23 and 24 were modified from the original state machine to include this check) which are right before memory accesses for the cycles corresponding to LDB, LDW, STW, and STB respectively. States 10 and 11 are used to handle the unknown opcode exception. In these states EXC[0] will be 1 and EXC[1] will be 0. Finally, states 8, 40, 42, 26, 48, 50, 44, and 62) are used to handle the RTI instruction. It consists of popping the old PC and PSR from the stack and swapping from privileged to user mode (saving system stack pointer and loading saved user stack pointer) if necessary.



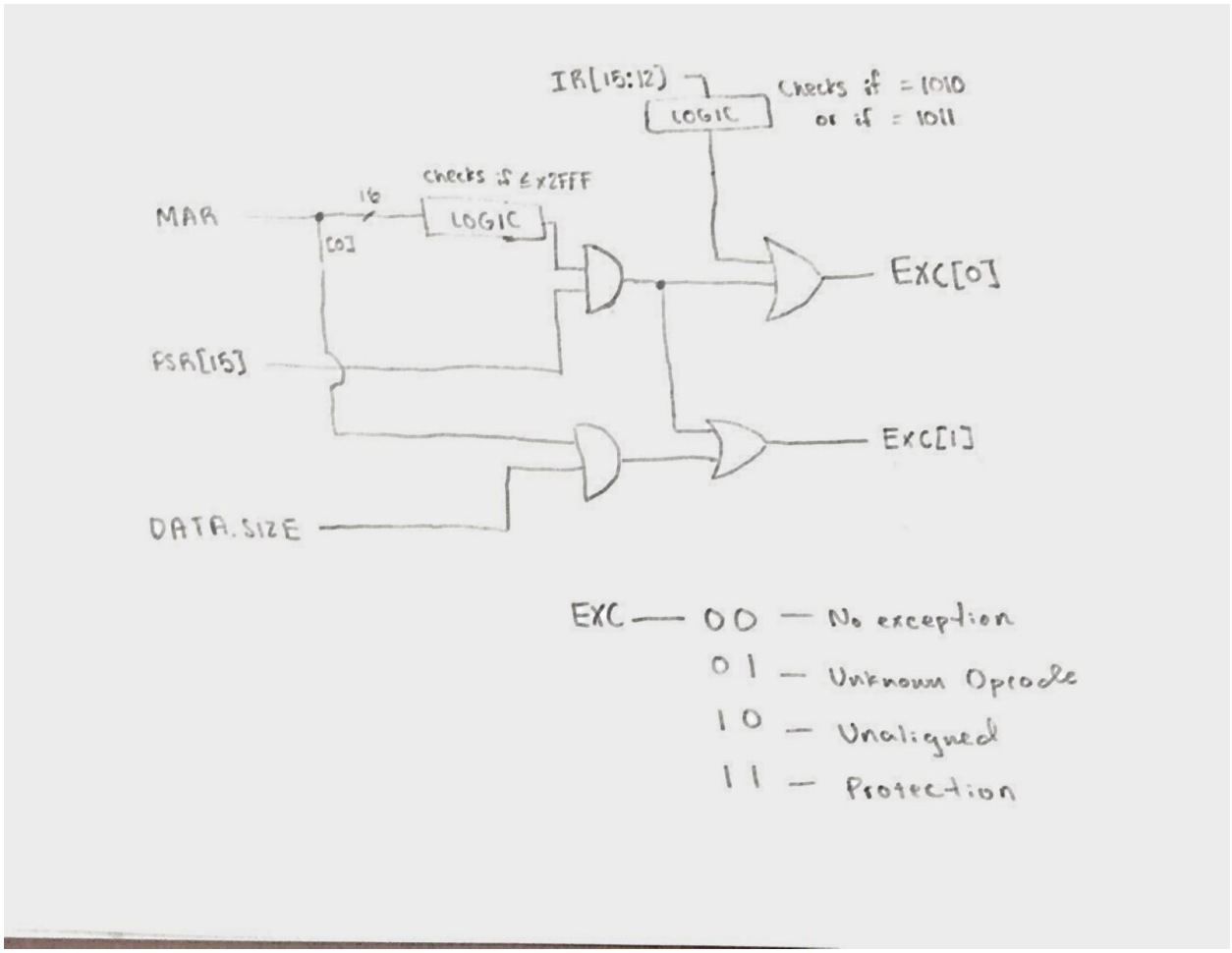
Changes to the data path: The privilege bit of the PSR was added and changes (PrivMUX, CCMUX) were made so that the privilege bit and CC could be saved and modified during a context switch. Registers were added to save the user and system stack pointers and allow the stack pointer to be sent to the BUS with a +/-2 offset for push and pop operations (Saved_USP, Saved_SSP, SPMUX). We also allow the PC to be gated on the bus with a -2 offset so that it will be easier to return to the user program once done with an ISR. Finally, the interrupt and exception vector structures can be seen on the right side where registers are used to hold the respective vector before being left shifted by 1 and concatenated with x02 and loaded on the bus (INTV, EXCV, EXCVMUX, Vector, VectorMUX).

The new control signals added are: LD.Priv, LD.USP, LD.SSP, LD.EXCV, LD.Vector, GatePSR, GateSP, GatePC-2, GateVector, PrivMUX, CCMUX, VectorMUX, and SPMUX (2 bits). The LD signals

are used to load the registers with a new value, the gate signals are used to gate the value from a register or the output of a mux to the bus, and the MUX signals are used to select which value should be loaded into a register or to select which value should be sent to the bus (in the case of the SPMUX signal). The EXC signal is a new control signal but is not part of the control store, rather it is defined by the MAR, PSR[15], and DATA.SIZE.
A few changes were made to existing control signals which are: adding a COND3 bit to allow for exception, interrupt, and privilege check microbranches and adding an extra bit to both DRMUX and SRMUX so that if they are 10 they will load or source R6 (respectively).



New Microsequencer: The INT signal and PSR[15] are used to microbranch with respect to bit 4 and 3 (respectively) of the state number. However, what is more interesting is the EXC[1] signal which indicates that an unaligned or protected access exception has occurred. In order to save states that would be wasted from only wiring this signal to a single bit for microbranches, I wired all of the 6 J bits to this so that if EXC[1] is asserted and the COND bits are 110, then we will always branch to state 63.



This shows how the EXC[0] and EXC[1] bits are defined based on the MAR, PSR[15], and DATA.SIZE.