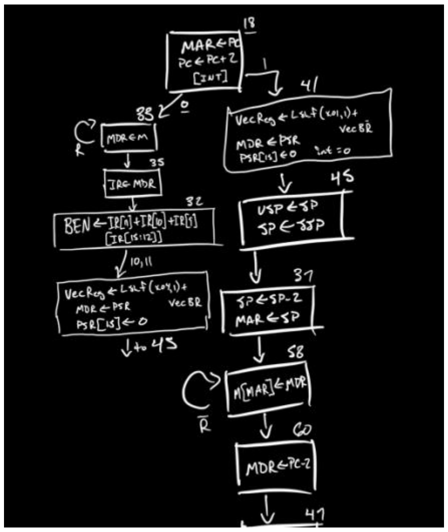
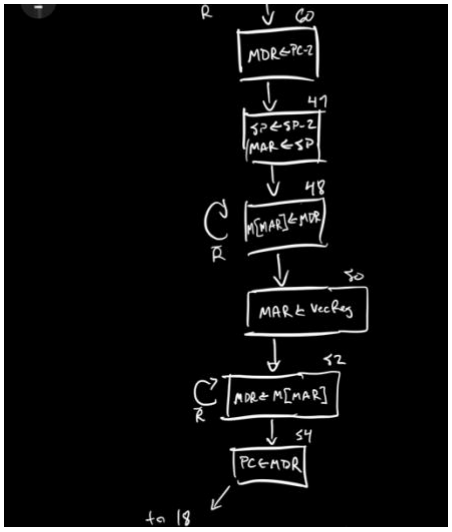
Guy Farmer gcf375

Below are two changes I made to the state diagram. 1.a shows that State 41 refers to the beginning of the interrupt states, and 33 refers to the opcode exception routine. In state 41, there is a data structure that will compute left shift the interrupt vector and add it to the base register and store it in a vector register. The current PSR is loaded to the MDR to prepare for a push, the PSR[15] is flipped to 0, and the interrupt flag is turned off. States 45 – 48 is a part of switching to the supervisor stack pointer and pushing the PSR and PC onto the supervisor stack. States 50 – 54 is when the address specified in the vector register is loaded into the program counter. States 10 & 11 is when there is an illegal opcode, and performs the same functions as 41, but this would be the illegal opcode vector instead. Then it would perform the same states as the ISR.

1.a

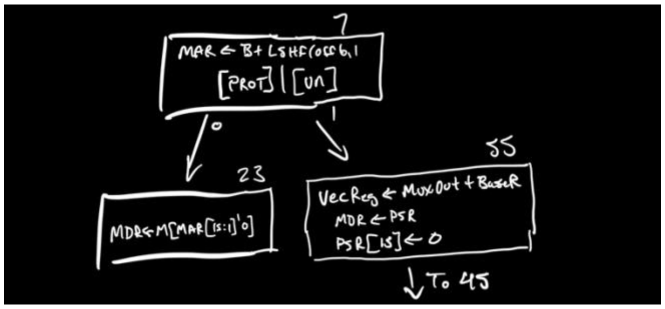


1.b

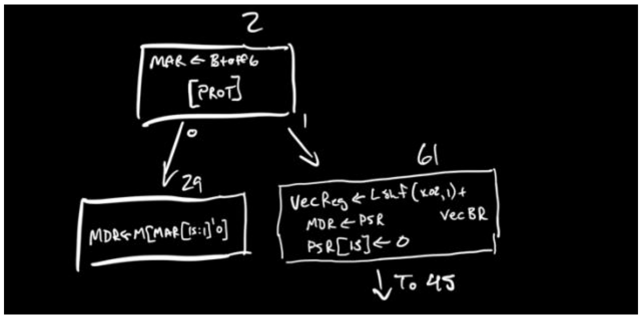


1.c, 1.d, & 1.e refers to the various other exception routines occur. State 6 and 7 check for unaligned and protection exceptions because it deals with memory access. States 2 and 3 check for solely protection exceptions, then the following states load the vector register with the correct routine address, then point to state 45.

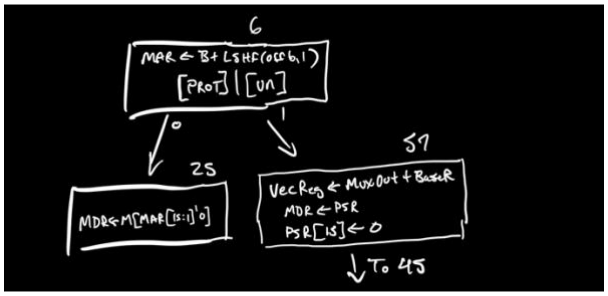
1.c



1.d



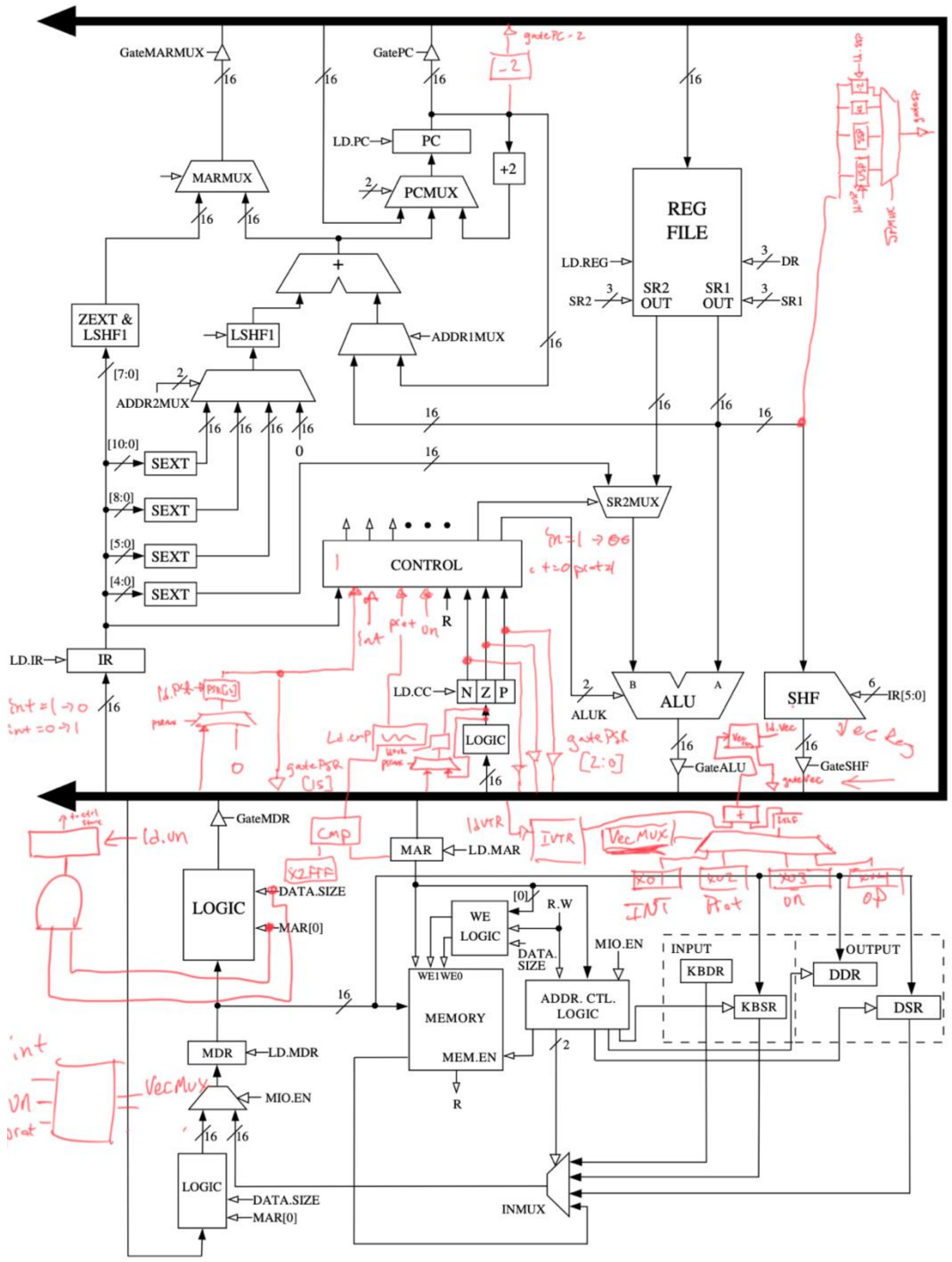
1.e



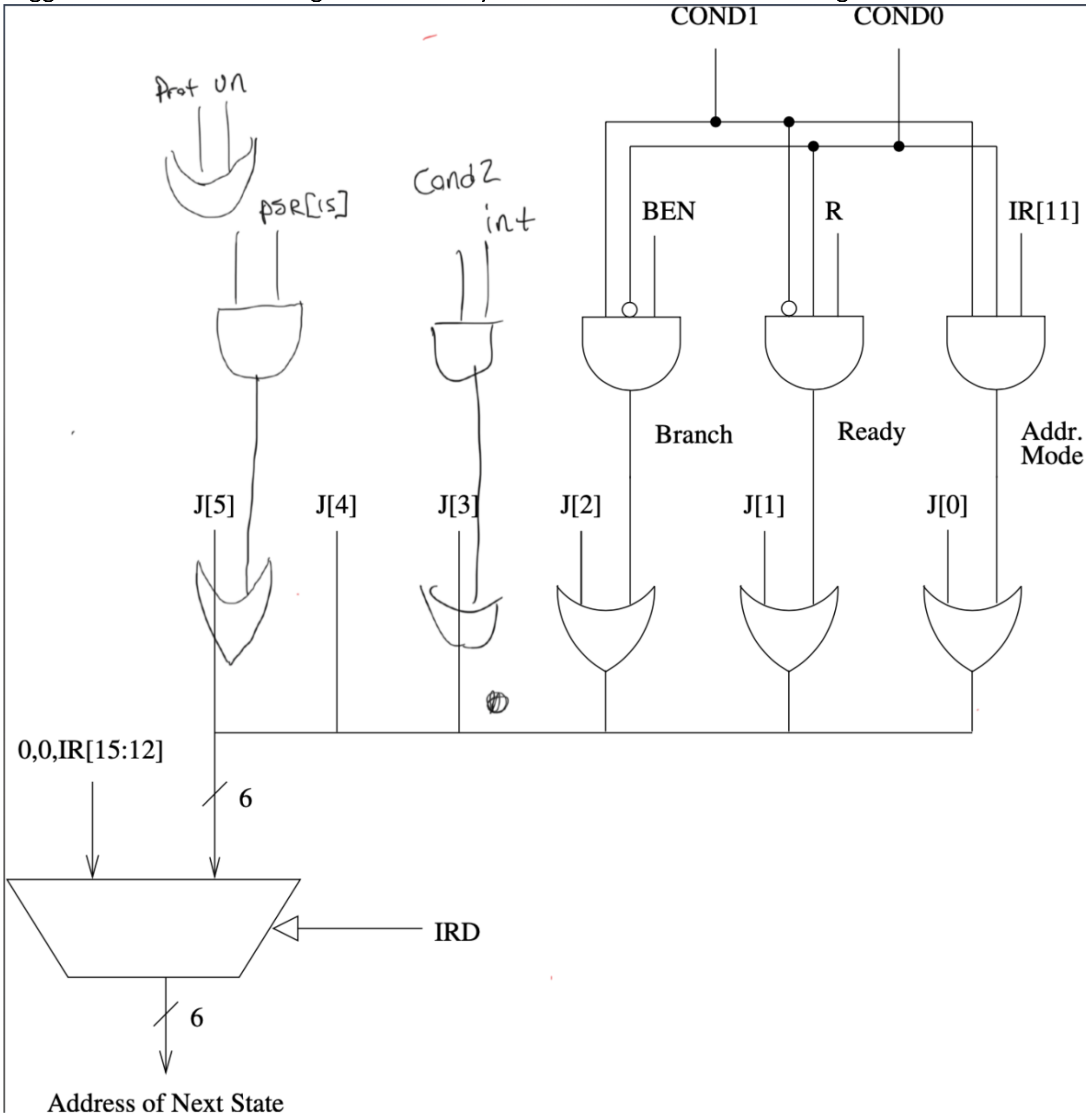
Guy Farmer gcf375

Below describes the additions I made to the datapath. There is a state that requires the program counter to be decremented so I included a structure to decrement included with a gate to pass on to the bus. There are states that need to handle the supervisor stack pointer and user stack pointer with increments and decrements, so I added an additional structure which selects each process and a gate to specify when to load it onto the bus. I've also included a structure which decides which type of vector to left shift and add to the vector base register, which the register is connected to a gate which will load the contents on to the bus. The multiplexor is controlled by the outputs of the interrupt, opcode, protection, and unaligned flags that might occur during the instruction cycle. The unaligned exception is determined through a single and gate that checks whether or not there is a word access at an unaligned address. The protection is determined through a comparator which checks for a supervisor address during user mode. The opcode exception is determined through the control store. Lastly there is a multiplexor which decides what value to load the PSR and whether or not to load it from the bus, flip a 1 to a zero, or continue in a user mode, and there are a series of gates connected to the PSR which determines whether or not to load it onto the bus.

2



Below is the modified microsequencer. The only additions are the two and gates and a single or gate. The logic is that the microsequencer will take in the PSR[15], a protection flag, an interrupt flag, and an unaligned flag. Combined with an additional condition bit, the microsequencer will give the next address if an interrupt is triggered or if an exception is triggered. Otherwise it will go to its already established state without a flag.


COND1
COND0
Prot un
PSR[15]
Cond2
int
BEN
R
IR[11]
Branch
Ready
Addr. Mode
J[5]
J[4]
J[3]
J[2]
J[1]
J[0]
0,0,IR[15:12]
6
IRD
6
Address of Next State