EE 460N Lab 4
Raiyan Chowdhury
rac4444

This README describes my implementation of the Lab 4 assignment.

The State Diagram

The enhanced state diagram for this lab is given in Figure A.1 (all relevant figures are attached at the end). In order to carry out an interrupt, state 18 has been modified to check the new INT bit, which is set when an external device requests an interrupt (INTV is also set when this happens). In the case of this lab, INT and INTV are set during cycle 300. Figure A.3 shows changes to the microsequencer including the INT bit. When this bit is set, the state machine micro-branches to state 47 which begins the execution of an interrupt.

In state 47, the Vector register is loaded with the appropriate interrupt vector (zero-extended and left-shifted by 1), and the PSR is loaded into the MDR before going into supervisor mode (where bit 0 is changed to 0). After this, R6's current value is saved in the USP register (whether R6 was actually being used as the user stack pointer up until now is up to the user) and it is then set to the SSP value. From here, R6 is used to push the PSR and decremented PC onto the stack. The MAR is then loaded with the address of the appropriate interrupt service routine by adding 0x0200 to the value previously loaded into the Vector register so that the vector table may be read, and the ISR's address is read and loaded into the PC. The processor then returns to state 18 to begin execution of the ISR.

Additional states were also added for the RTI instruction. It begins with state 8 where the MAR is loaded with the value of R6 (the supervisor stack pointer) so that the PC and PSR can be popped off the stack and restored to their respective registers. This is done by reading memory at the addresses specified by R6 while incrementing it after each pop. The SSP register is then loaded with the value inside R6, and R6 is loaded with the value inside the USP register (its original value prior to the interrupt being handled).

States to handle exceptions have also been added. In the case of an unknown opcode exception, the processor need only see that the machine has entered either state 10 or 11 to understand that an invalid opcode has been provided. In these states, the Vector register is loaded with the appropriate EXCV value (0x04 in this case, zero-extended and left-shifted by 1). Other than this difference, states 10 and 11 function exactly the same as 47. The following state is state 26, as the following steps for handling exceptions are the same as for interrupts--set R6 equal to the SSP, push the PSR and PC, and get the address for the exception handler from the vector table.

In the case of protection and unaligned access exceptions, some different steps are taken. An additional bit called “EXC” (similar to INT) is fed into the CONTROL LOGIC block (see Figure A.2), and it is set to 1 whenever a protection or unaligned access is detected (EXCV is also set to either 0x02 or 0x03 as appropriate when this happens). When EXC is set to 1, the state machine automatically branches to state 49, which functions the same way as state 10 and 11, except for the fact that it handles protection and unaligned access exceptions instead. In some areas, additional states that do nothing but check for the EXC bit had to be added in order to avoid accessing memory before the system has determined that an exception has occurred.

The Data Path

Figure A.2 shows the additions made to the LC-3b’s data path. Some changes are also shown in Figure A.3.

Firstly, some additions were made to account for the PSR. Because we are only using bits [15] and [2:0] from the PSR, it seemed unnecessary to add an entire register for the PSR, so I only added a single bit register (PSR_PRIV) to hold the privilege level (bit 15) of the PSR, and the condition codes register was sufficient already to use for PSR[2:0]. A load control signal accompanies the privilege register, and it is set whenever the PSR’s value is being changed. If we are simply changing the PSR’s privilege mode, then a mux (PrivMUX) selects the value 0 to be loaded into this register, otherwise it selects bit 15 from the bus (where the bus would contain the “full” value of the PSR). A tristate driver GatePSR was added connecting the bus to both the privilege and condition code registers for when the PSR is being loaded into some other device (the MDR, in this case) via the bus. The CC register also now receives its input from a mux (CCMux), which chooses between changing the condition codes normally or directly reading bits [2:0] off of the bus when the bus contains the “full” PSR value.

Structures were added to handle the interrupt and exception vectors as well. INTV is set by an external device when that device requests an interrupt. A series of logic is used to set EXCV. Figure A.2 shows a logic block taking the MAR, IR[15:12], the PSR’s privilege value and DATA.SIZE as inputs. These inputs are used to determine whether or not a protection exception or unaligned exception has occurred. If there is no exception, the logic block does not select any vector. If the MAR has been loaded with an address in system space and the privilege is set to 1 (user), then this is a protection exception and the logic block's output is used to select the exception vector 0x02 from a mux, which loads EXCV with 0x02. If the MAR's address is in user space, but MAR[0] and DATA.SIZE are both equal to 1, then the 0x03 vector is chosen to signify an unaligned access exception. Lastly, if IR[15:12] is 10 or 11, then this signifies an unknown opcode exception and the 0x04 vector is chosen. Moreover, the aforementioned logic block's output is fed into another logic block--this second logic block determines whether or not a protection or unaligned access exception has been detected based on the previous logic, and sets EXC appropriately.

Once INTV or EXCV have been set, a mux (VecMUX) determines which of the two vectors is to be loaded into the Vector register (which has its own LD.Vector control signal), which depends on the machine's current state (see Figure A.1). In state 47, INTV is chosen, and in states 10, 11, and 49, EXCV is chosen. Before being loaded into the Vector register, these vectors are first zero-extended to 16 bits and left-shifted by 1. Then, when it is time to load the MAR with the appropriate vector table address, the value inside the Vector register is added to the base address of 0x0200 and loaded onto the bus. A new tristate driver called GateVector is set whenever it is time to load this value onto the bus.

The CONTROL LOGIC block is fed with two new inputs, as mentioned before, which are the EXC and INT bits. This will be expanded upon when discussing changes to the microsequencer later in this document.

Structures have also been added to swap R6 between user and stack pointer values correctly, and also to handle pushing and popping on the supervisor stack. The output of SR1MUX (or SR1 OUT in Figure A.2) is latched to 4 structures--USP and SSP registers (both of which have load signals), as well as +2 and -2 blocks which are used when incrementing and decrementing R6 before loading the modified R6 into R6 again or the MAR (in the context of this lab). A mux called SPMUX determines whether or not R6 is being incremented or decremented before its value is loaded onto the bus. When changing R6 to the SSP, the USP register is loaded with R6's current value, and SPMUX selects SSP's value to be loaded onto the bus. When changing R6 to the USP, the SSP register is instead loaded by R6 and SPMUX selects USP's value to be loaded onto the bus. A tristate driver called GateSP has been added so that these values can be loaded onto the bus appropriately.

Because all of these stack operations use R6 specifically, some changes had to made to DRMUX and SR1MUX, which can be seen in Figure A.3. Both can now select a third input (which is simply the number 6), and their select signals have been expanded to 2 bits in order to accommodate. So, whenever the state machine wants to modify the value of R6, DRMUX selects it. Also, whenever we want R6's value from SR1 OUT to be used for the aforementioned stack pointer logic, SR1MUX selects it.

Lastly, one additional structure was added so that the decremented PC can be pushed onto the stack. A new -2 block takes the output of the PC's register and latches onto the bus, with a new tristate driver called GateDecPC. For the purposes of this lab, whenever the MDR is being loaded with the decremented PC, GateDecPC is set and the current PC value minus 2 is loaded onto the bus.

New Control Signals

The additional control signals were described in the previous section, but this section clarifies each one briefly.

GateDecPC is set whenever we want to load the bus with the value of the decremented PC (PC - 2). GatePSR is set when we want to load the bus with the value of the PSR. GateVector is set when we want to load the bus with the the value inside the Vector register + 0x0200. GateSP is set when we want to load the bus with the output of SPMUX, which may be the USP, SSP, or an incremented/decremented register value (in this case, R6).

LD.Priv is set when we want to change the privilege value of the PSR. LD.USP and LD.SSP are set when changing the values of the USP and SSP registers, which are used when swapping the value of R6 between one or the other. LD.Vector is set when we want to load the Vector table with either INTV or EXCV (both zero-extended to 16 bits and left-shifted by 1).

PrivMUX is used when determining whether we want to forcefully change PSR[15] (or, in the data path, the privilege register PSR_PRIV) to 0, or change PSR[15] based on the value on the bus. VecMUX is used to choose between INTV and EXCV, depending on whether or not the system is handling an interrupt or exception, respectively. CCMUX is used to determine whether the condition codes are loaded based on the value on the bus (e.g., set N after the ADD instruction yields a negative value) or based on the last 3 bits on the bus directly (i.e. when the bus holds the value of the PSR outright, for example in state 54 where we load PSR from the MDR). SPMUX is used to choose between sending the USP, SSP, the incremented stack pointer or decremented stack pointer to the bus.

In addition to existing control signals, DRMUX and SR1MUX are now 2-bit signals in order to accommodate for the new option of choosing R6 directly for both. There is also an additional COND bit, which will be elaborated on in the following section.

The New Microsequencer

Figure A.3 details the changes to the microsequencer logic.

As mentioned before, an INT bit has been added and it is set whenever an interrupt is requested by an external device (for this lab, it is set during cycle 300). The value of INT is checked during state 18 (or 19). A third COND bit has been added to check for interrupts, and during state 18/19 the value of COND is set to 4 (100 in binary). As shown in Figure A.3, this allows the INT bit to pass through its AND gate when it is set to 1, which sets the value of J[3] to 1. So, when the INT bit is set, state 18 micro-branches to state 47 instead of state 39 (note: state 39 was added to check for protection and unaligned access exceptions before memory access occurs in state 33). From here, interrupt handling occurs as described previously in this document.

As mentioned before, an EXC bit has also been added to the LC-3b system for the purpose of determining whether or not a protection or unaligned access exception has occurred. When EXC bit is set to 1, we want the system to micro-branch to the same state no matter what the current state is, and because there are many states in which this can happen, it is easier to latch EXC to the existing mux that initially used the IRD bit as a select signal, rather than implement something like we did for the INT bit. Doing this preserves the number of states that need to be added. So, the mux now uses IRD and EXC as select signals. This mux functions as it always did in regard to the IRD bit, but now when the EXC bit is set (and the IRD bit is not--so, in other words, when the mux sees a select signal equal to 2, or 10 in binary) the mux selects the number 49, which corresponds to state 49 in the state machine which marks the start of exception handling for protection and unaligned access violations. The system micro-branches to state 49 no matter what state it is in currently, and continues from here as described previously. In the case of this lab, this will only happen in states 23, 24, 39, 55, and 56, as shown in the state diagram in Figure A.1; these are the states in which the preceding state loads a value into the MAR or PC, which is when protection and unaligned exceptions must be detected. Note that the TRAP instruction is an exception to this rule, as it is allowed to access system space in state 28, and it always left-shifts its trap vector so it is impossible for an unaligned access to occur.

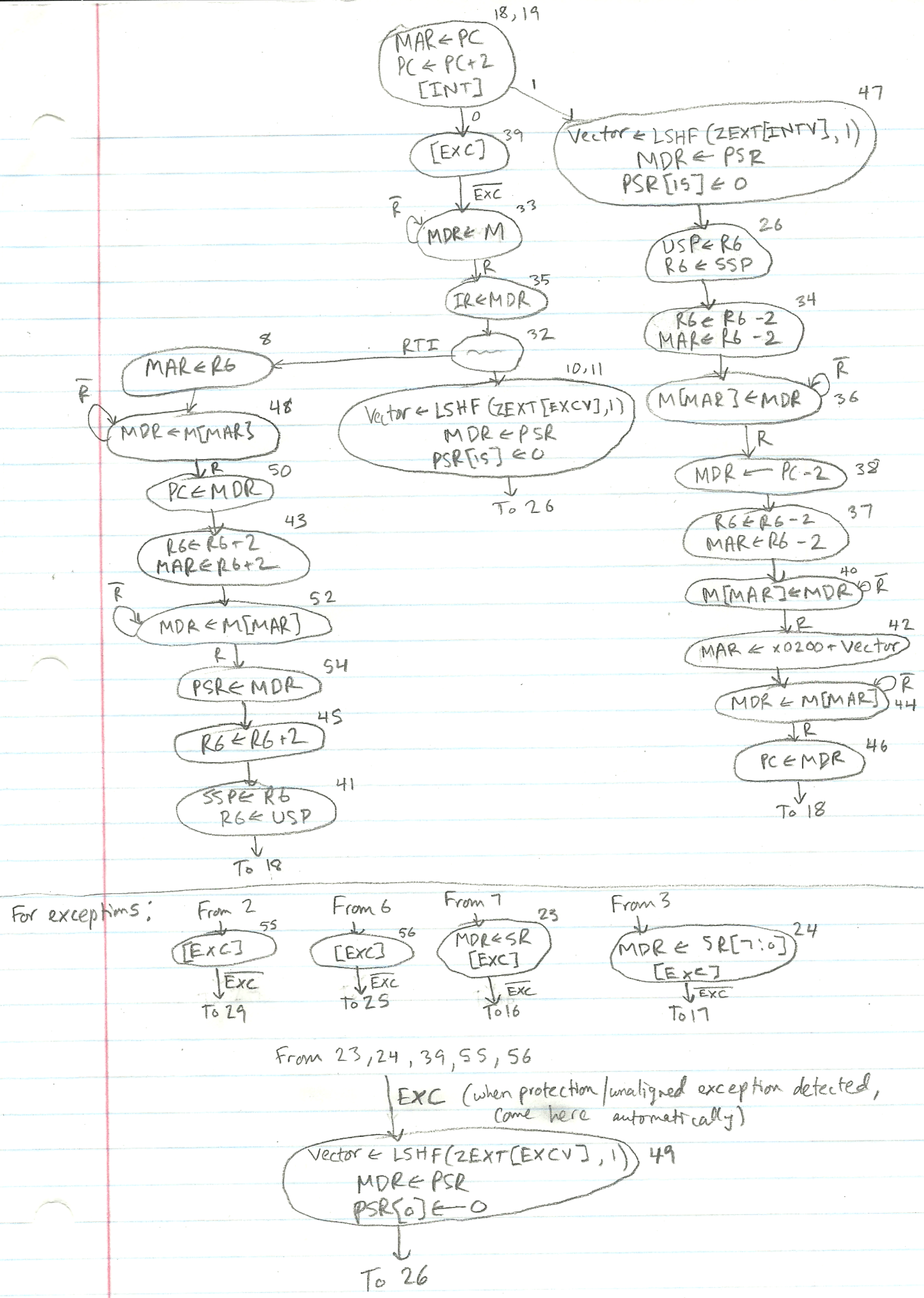

18, 19
MAR ← PC
PC ← PC+2
[INT]
1
47
Vector ← LSHF (ZEXT[INTV], 1)
MDR ← PSR
PSR[15] ← 0
0
39
[EXC]
EXC
R
33
MDR ← M
R
35
IR ← MDR
32
RTI
8
MAR ← R6
10,11
Vector ← LSHF (ZEXT[EXCV], 1)
MDR ← PSR
PSR[15] ← 0
To 26
R
48
MDR ← M[MAR]
R
50
PC ← MDR
43
R6 ← R6+2
MAR ← R6+2
R
52
MDR ← M[MAR]
R
54
PSR ← MDR
45
R6 ← R6+2
41
SSP ← R6
R6 ← USP
To 18
26
USP ← R6
R6 ← SSP
34
R6 ← R6 -2
MAR ← R6 -2
R
36
M[MAR] ← MDR
R
38
MDR ← PC -2
37
R6 ← R6 - 2
MAR ← R6 - 2
40
M[MAR] ← MDR
R
R
42
MAR ← x0200 + Vector
R
44
MDR ← M[MAR]
R
46
PC ← MDR
To 18
For exceptions;
From 2
55
[EXC]
EXC
To 29
From 6
56
[EXC]
EXC
To 25
From 7
23
MDR ← SR
[EXC]
EXC
To 16
From 3
24
MDR ← SR[7:0]
[EXC]
EXC
To 17
From 23, 24, 39, 55, 56
EXC (when protection/unaligned exception detected, come here automatically)
49
Vector ← LSHF(ZEXT[EXCV], 1)
MDR ← PSR
PSR[0] ← 0
To 26

FIGURE A.1

PrivMUX: 0, PSR[15]

VecMUX: EXCV, INTV

CCMUX: BUS, PSR[2:0]

SPMUX: USP, +2, -2, SSP

*This logic helps determine the exception vector, using info from the IR and MAR. If x0000 ≤ MAR ≤ x2FFF, x02 is selected. If DATA.SIZE and MAR[0] = 1, x03 is selected. If IR[15:12] is 10 or 11, x04 is selected.

* This logic sets EXC to 1 when a protection or unaligned exception is detected.

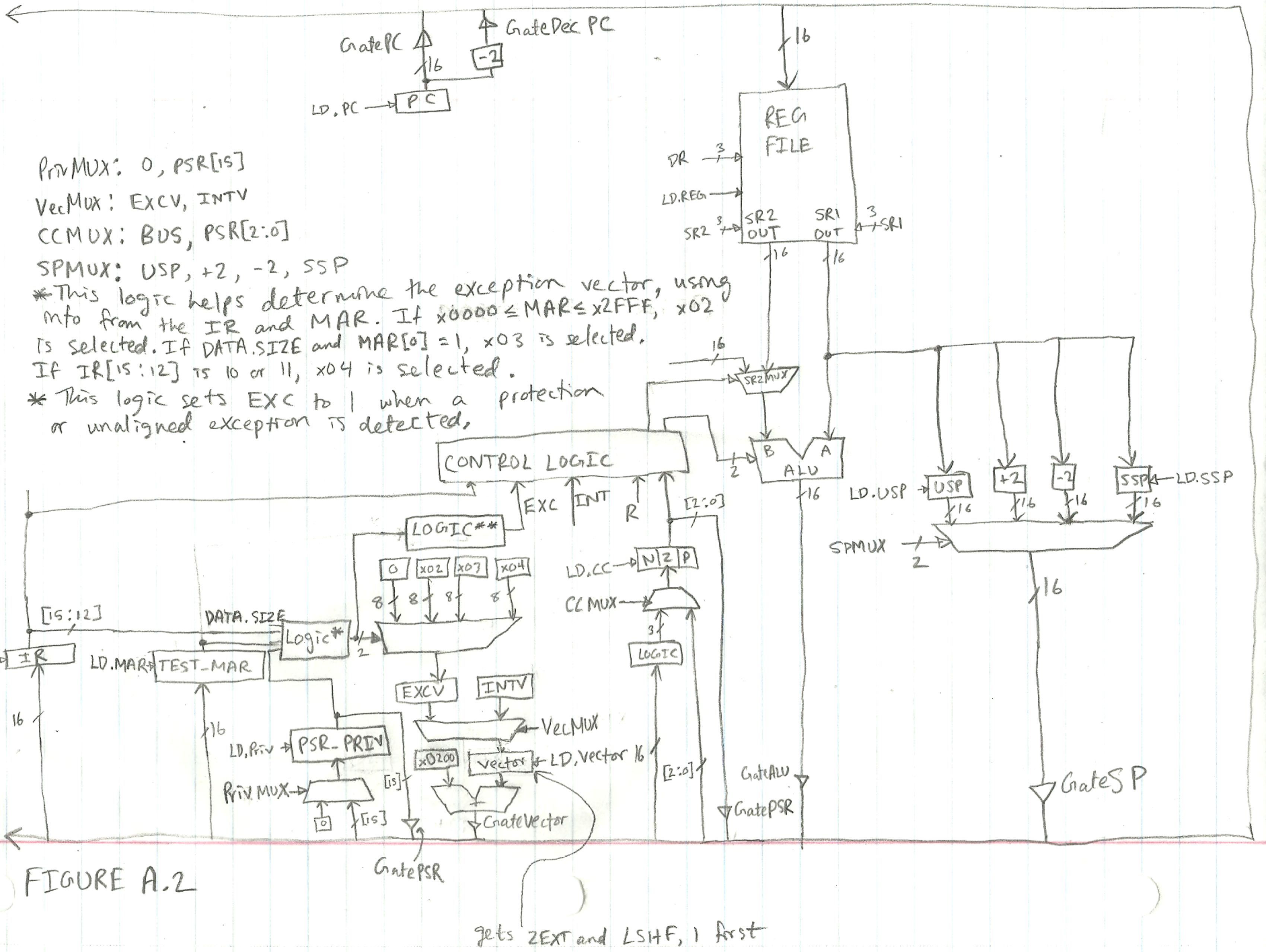
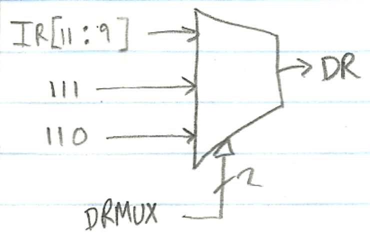
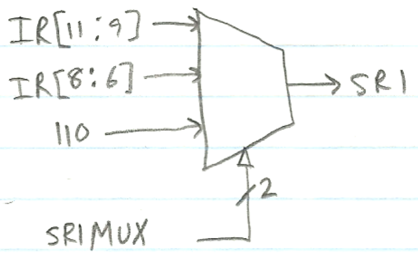


FIGURE A.2

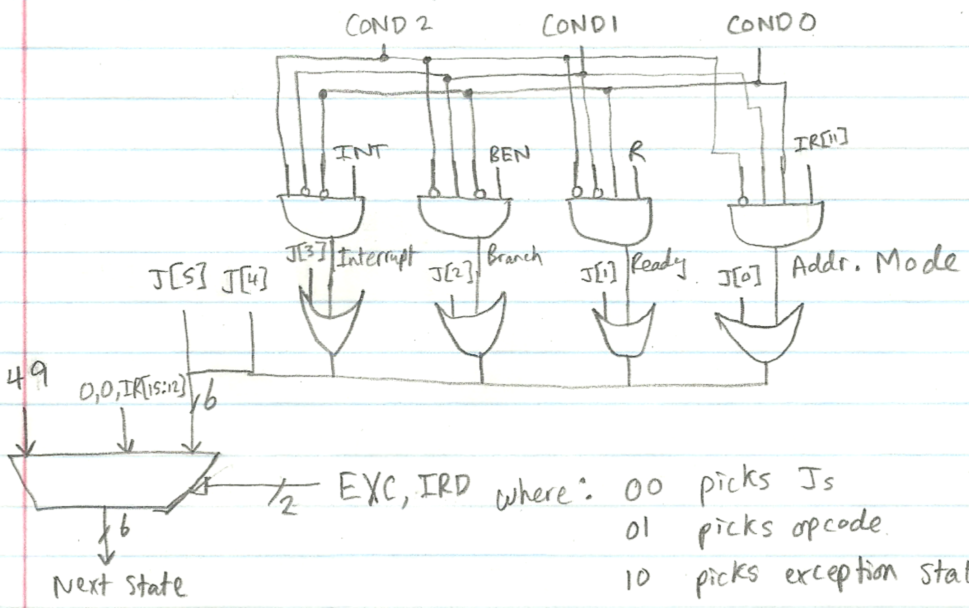




DRMUX/2 :	11.9
	R7
	R6

SR1MUX/2 :	11.9
	8.6
	R6

COND/3 :	COND₀ ; Unconditional	→ 000
	COND₁ ; Memory ready	→ 001
	COND₂ ; Branch	→ 010
	COND₃ ; Addressing Mode	→ 011
	COND₄ ; Interrupt	→ 100



where : 00 picks Js
01 picks opcode
10 picks exception state 49
(i.e. EXC is set, IRD is not)

FIGURE A.3