Sai Koukuntla
EE460N Lab 4

Changes to State Diagram

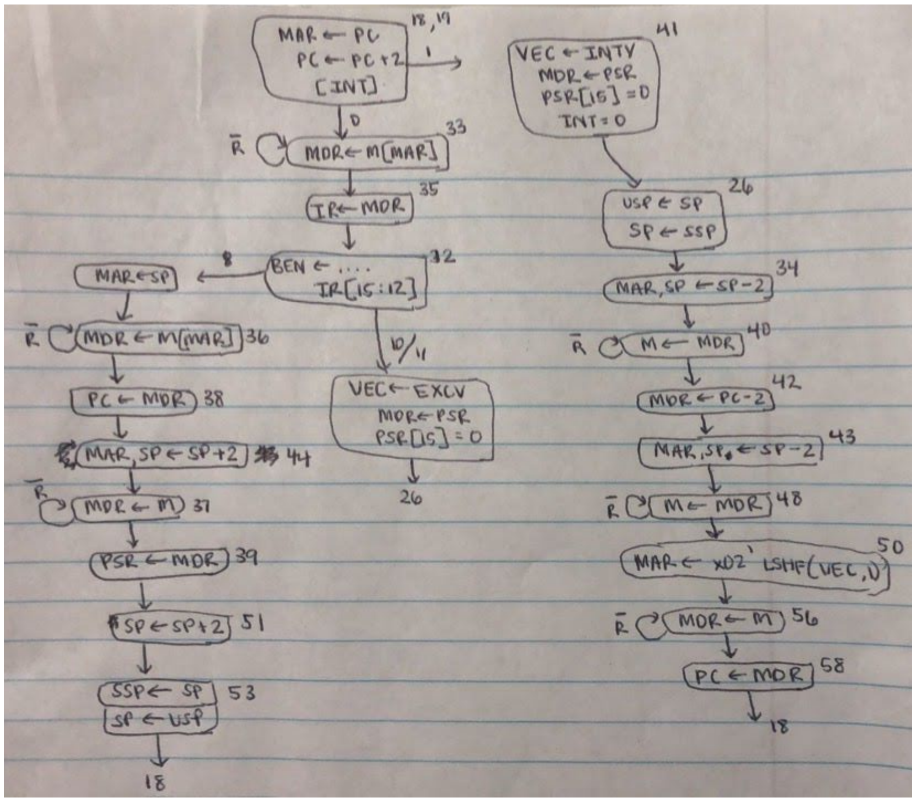

Figure 1: Modified FSM with new paths for interrupts, exceptions, and RTI

<u>Interrupts:</u> The interrupt signal is checked in state 18/19, and if it's high, the FSM branches to state 41 instead. In state 41, the interrupt vector is loaded into the VEC register, PSR is loaded into the MDR, PSR[15] is set to 0 to indicate supervisor mode, and the interrupt signal is de-asserted. In state 26, the user stack pointer is saved and the supervisor stack pointer is loaded into R6. In states 34, 40, and 42, the PSR and PC are pushed on to the supervisor stack in that order. In state 50, the ISR vector table entry address is calculated by concatenating 0x02 with the contents of VEC left-shifted by 1, and then loaded into the MAR. States 56 and 58 obtain the ISR address from the vector table and load it into PC.

<u>Exceptions:</u> All exceptions begin in state 10/11. Unaligned and protected exceptions assert the EXC signal (logic below), which causes the microsequencer to change the next state to 10. Unknown opcodes will enter state 10/11 automatically since state 32 will route opcodes of 10/11 to state 10/11. State 10/11 is very similar to state 41: the exception vector (logic below) is loaded into the VEC register, PSR is loaded into the MDR, and PSR[15] is cleared to indicate supervisor mode. The FSM then enters state 26, where it follows the same path detailed for interrupts above.

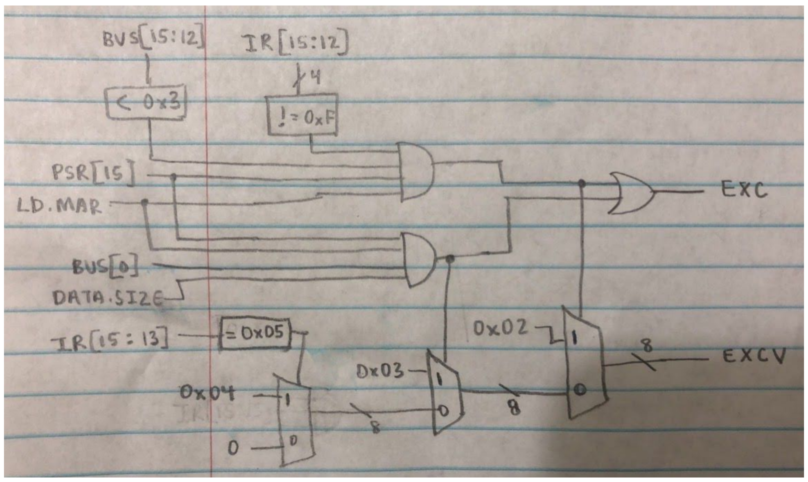

Figure 2: Logic to generate EXC and EXCV. Protected exceptions occur when an address less than 0x3000 is loaded into the MAR in user mode during a non-TRAP instruction. Unaligned exceptions occur when an odd address is loaded into the MAR in user mode during a word access. EXCV is 0x02 when a protection exception occurs, 0x03 when an unaligned exception occurs, and 0x04 when an unknown opcode exception occurs, with the priority of exceptions being protection > unaligned > unknown opcode.

<u>RTI:</u> RTI instructions are routed to state 8 after the decode phase. In states 8, 36, 38, 44, 37, and 39, the PC and PSR are popped off the supervisor stack in that order. In state 51, the SSP is incremented to point to the next stack entry. In state 53, the supervisor stack pointer is saved and the user stack pointer is loaded into R6.

Changes to Datapath

<u>Stack Pointer:</u> I added the 16-bit USP and SSP registers (with SR1 as an input) to store saved values of the user and supervisor stack pointers respectively. Each of the registers has its own LD control signal (LD.USP and LD.SSP). I also added +2 and -2 units with SR1 as an input in order to produce the SP - 2 and SP + 2 values needed in the RTI and interrupt/exception states. The two SP registers, + 2 unit, and -2 unit all output to a 4x1 mux, where the SPMUX/2 control signal determines which one is passed to GateSP. Finally, the GateSP control signal determines whether the output of SPMUX is used to drive the bus (*Figure 3, top right)*. I also expanded the DRMUX and SR1MUX control signals to 2 bits instead of 1 (*Figure 4)*, so that SP (R6) could be accessed.

<u>PSR:</u> I added a 1-bit PRIV register to represent PSR[15], and it has its own LD signal (LD.PRIV). The input to the PRIV register is a 2x1 mux that uses the PSRMUX control signal to pass through either a 0 (so the PRIV bit can be cleared) or bit 15 of the bus (so a stored PSR can be loaded in). Also, the output of the PRIV register goes to a tri-state buffer using the GatePSR control signal, which determines if it is used to drive the bus (*Figure 3).* I added similar logic to the condition codes (PSR[2:0]). A 2x1 mux uses the PSRMUX to pass either calculated condition codes or BUS[2:0] (to load a previous PSR) to the input of the NZP registers (*Figure 5).* The output of the NZP registers goes to another GatePSR tri-state buffer so BUS[2:0] can be driven by the condition codes.

VEC: I added an 8-bit VEC register to hold the current interrupt/exception vector, and it has its own LD signal (LD.VEC). The input to the VEC register is a 2x1 mux that uses the VECMUX control signal to pass through either INTV (interrupt vector) or EXCV (exception vector). The output of the VEC register goes into a GateVEC tri-state buffer. When the GateVEC control signal is high, bits [15:8] of the bus are driven by 0x02 and bits [7:0] are driven by the VEC register (*Figure 3)*.

PC-2: I added a -2 unit coming off of the PC output. The output of the -2 unit goes into a GatePC-2 tri-state buffer so it can be used to drive the bus in state 42 (*Figure 3)*.

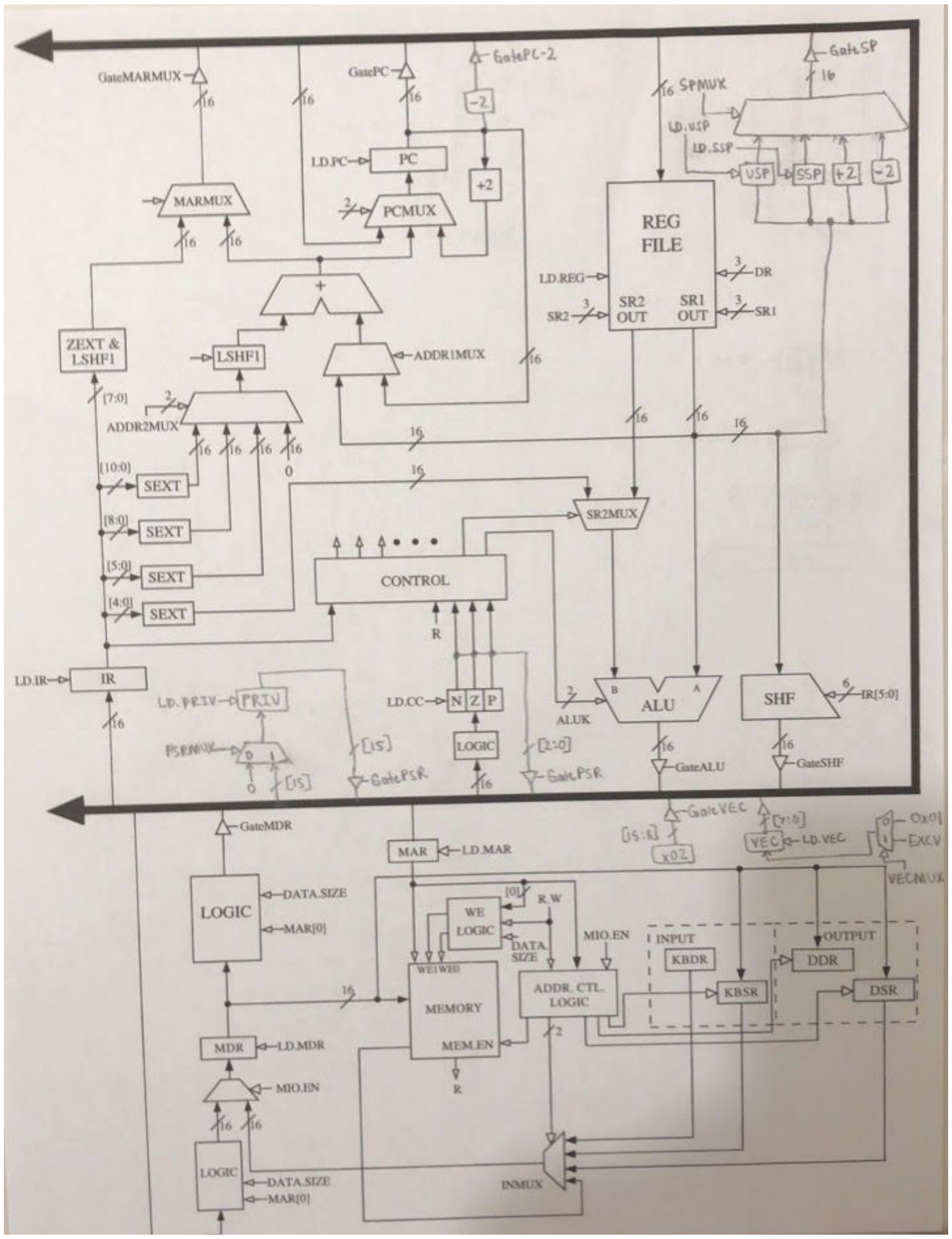

Figure 3: Modified datapath showing structures to support SP, VEC, PSR, and PC-2 usage.

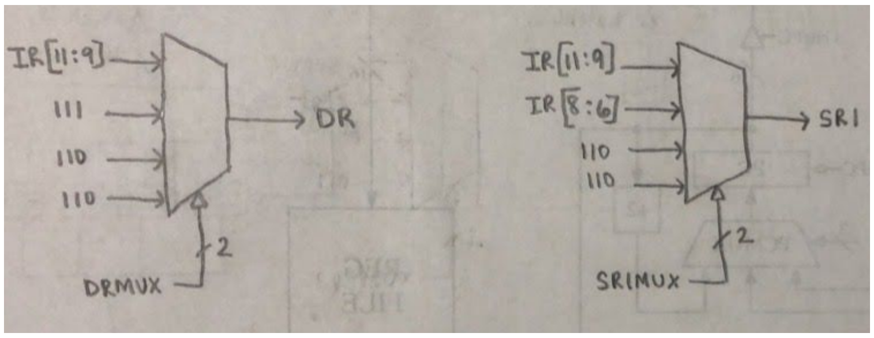

Figure 4: Modified DRMUX and SR1MUX showing exapnsion to support SP (R6) selection.

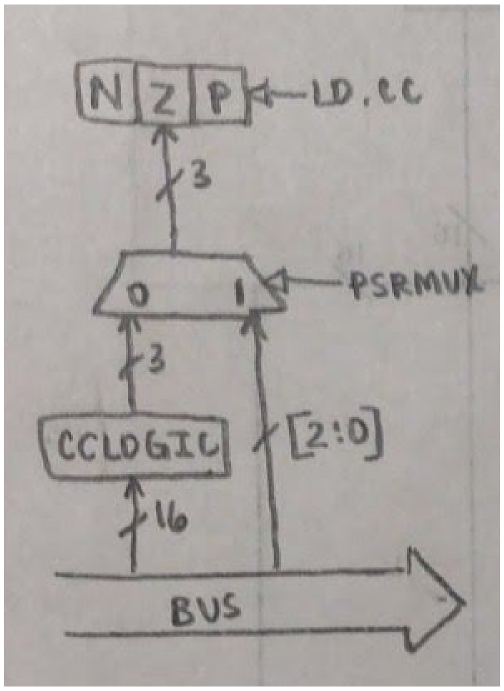

Figure 5: Modified CC input logic, showing the option to calculate CC or load them directly from bus.

New Control Signals

- LD.SSP: Allows a write to the supervisor stack pointer register.
- LD.USP: Allows a write to the user stack pointer register.
- LD.VEC: Allows a write to the VEC register.
- LD.PRIV: Allows a write to the PSR[15].
- GateSP: Allows the output of SPMUX to drive the bus.
- GateVEC: Allows 0x02'(VEC << 1) to drive the bus.
- GatePSR: Allows the PSR (only bits [15] and [2:0] are implemented in hardware) to drive the bus.
- GatePC-2: Allows decremented PC to drive the bus.
- COND: Added a third conditional bit to allow the detection of interrupts in state 18/19.
- DRMUX: Added a second bit to set the destination register to SP (R6)
- SR1MUX: Added a second bit to set the source register to SP (R6)

- SPMUX: Determines whether USP, SSP, SR1+2, or SR1-2 are the input to GateSP.
- VECMUX: Determines whether INTV or EXCV is passed to the VEC register.
- PSRMUX: Determines whether PSR bits can be set using logic (clear [15], calculate [2:0] from CC logic) or should be loaded from the bus.

Control signal values are detailed below:

LD.SSP/1:	NO, LOAD	
LD.USP/1:	NO, LOAD	
LD.VEC/1:	NO, LOAD	
LD.PRIV/1:	NO, LOAD	
GateSP/1:	NO, YES	
GateVEC/1:	NO, YES	
GatePSR/1:	NO, YES	
GatePC-2/1:	NO, YES	
COND/3:	$COND_0$	; Unconditional
	$COND_1$	; Memory Ready
	$COND_2$	; Branch
	$COND_3$	; Addressing Mode
	$COND_4$	; Interrupt
DRMUX/2	11.9	; destination IR[11:9]
	R7	; destination R7
	R6	; destination R6
	R6	; destination R6
SR1MUX/2	11.9	; source IR[11:9]
	8.6	; source IR[8.6]
	R6	; destination R6
	R6	; destination R6
SPMUX/2	USP	; user stack pointer
	SSP	; system stack pointer
	SR1 + 2	; R6 + 2
	SR1 - 2	; R6 - 2
VECMUX/1	INT	; interrupt vector
	EXCV	; exception vector
PSRMUX/1	LOGIC	; clear PSR[15], set PSR[2:0] (CC) from logic
	BUS	; load from bus

Changes to Microsequencer

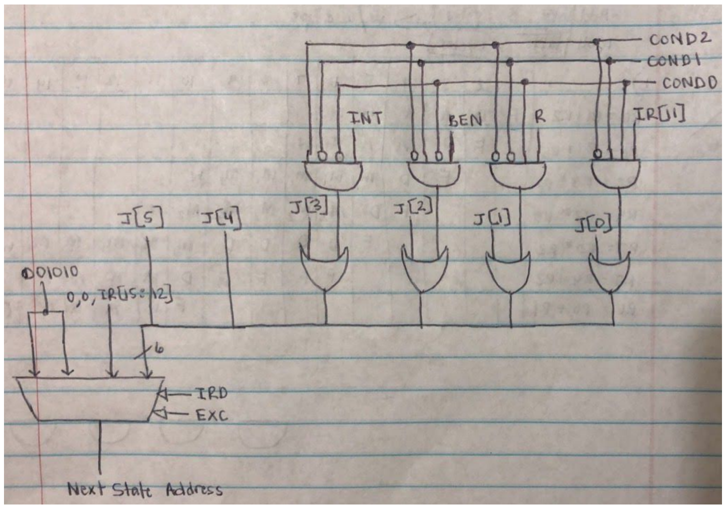

Figure 6: Modified microsequencer with support for detecting interrupts and exceptions.

If COND[2:0] = 100, and the interrupt signal is high, bit 3 of the next state is set to a 1. This allows the FSM to enter state 41 instead of 33 if the interrupt signal is high during state 18/19. The mux immediately before the microsequencer output has been expanded from a 2x1 mux to a 4x1 mux to allow for exception detection. If the exception signal is high, the next state is hardwired to state 10 so that the FSM enters the exception pathway. The functionality of the microsequencer is unchanged otherwise.