appendix C

The Microarchitecture of the LC-3

We have seen in Chapters 4 and 5 the several stages of the instruction cycle that must occur in order for the computer to process each instruction. If a microarchitecture is to implement an ISA, it must be able to carry out this instruction cycle for every instruction in the ISA. This appendix illustrates one example of a microarchitecture that can do that for the LC-3 ISA. Many of the details of the microarchitecture and the reasons for each design decision are well beyond the scope of an introductory course. However, for those who want to understand **how** a microarchitecture can carry out the requirements of each instruction of the LC-3 ISA, this appendix is provided.

C.1 Overview

Figure C.1 shows the two main components of an ISA: the *data path*, which contains all the components that actually process the instructions, and the *control*, which contains all the components that generate the set of control signals that are needed to control the processing at each instant of time.

We say, "at each instant of time," but we really mean **during each clock cycle**. That is, time is divided into *clock cycles*. The cycle time of a microprocessor is the duration of a clock cycle. A common cycle time for a microprocessor today is 0.5 nanoseconds, which corresponds to 2 billion clock cycles each second. We say that such a microprocessor is operating at a frequency of 2 gigahertz.

At each instant of time—or, rather, during each clock cycle—the 49 control signals (as shown in Figure C.1) control both the processing in the data path and the generation of the control signals for the next clock cycle. Processing in the data path is controlled by 39 bits, and the generation of the control signals for the next clock cycle is controlled by 10 bits.

Note that the hardware that determines which control signals are needed each clock cycle does not operate in a vacuum. On the contrary, the control signals needed in the "next" clock cycle depend on all of the following:

1. What is going on in the current clock cycle.
2. The LC-3 instruction that is being executed.
3. The privilege mode of the program that is executing.
4. If that LC-3 instruction is a BR, whether the conditions for the branch have been met (i.e., the state of the relevant condition codes).

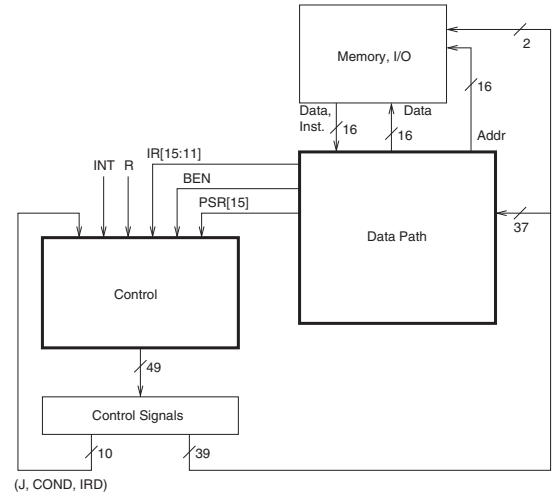

Figure C.1 Microarchitecture of the LC-3, major components

5. Whether or not an external device is requesting that the processor be interrupted.
6. If a memory operation is in progress, whether it is completing during this cycle.

Figure C.1 identifies the specific information in our implementation of the LC-3 that corresponds to these five items. They are, respectively:

1. J[5:0], COND[2:0], and IRD—10 bits of control signals provided by the current clock cycle.
2. inst[15:12], which identifies the opcode, and inst[11:11], which differentiates JSR from JSRR (i.e., the addressing mode for the target of the subroutine call).
3. PSR[15], bit [15] of the Processor Status Register, which indicates whether the current program is executing with supervisor or user privileges.
4. BEN to indicate whether or not a BR should be taken.
5. INT to indicate that some external device of higher priority than the executing process requests service.
6. R to indicate the end of a memory operation.

C.2 The State Machine

The behavior of the LC-3 microarchitecture during a given clock cycle is completely determined by the 49 control signals, combined with nine bits of additional information (inst[15:11], PSR[15], BEN, INT, and R), as shown in Figure C.1. We have said that during each clock cycle, 39 of these control signals determine the processing of information in the data path and the other 10 control signals combine with the nine bits of additional information to determine which set of control signals will be required in the next clock cycle.

We say that these 49 control signals specify the *state* of the control structure of the LC-3 microarchitecture. We can completely describe the behavior of the LC-3 microarchitecture by means of a directed graph that consists of nodes (one corresponding to each state) and arcs (showing the flow from each state to the one[s] it goes to next). We call such a graph a *state machine*.

Figure C.2 is the state machine for our implementation of the LC-3. The state machine describes what happens during each clock cycle in which the computer is running. Each state is active for exactly one clock cycle before control passes to the next state. The state machine shows the step-by-step (clock cycle–by–clock cycle) process that each instruction goes through from the start of its FETCH phase to the end of that instruction, as described in Section 4.2.2. Each node in the state machine corresponds to the activity that the processor carries out during a single clock cycle. The actual processing that is performed in the data path is contained inside the node. The step-by-step flow is conveyed by the arcs that take the processor from one state to the next.

For example, recall from Chapter 4 that the FETCH phase of every instruction cycle starts with a memory access to read the instruction at the address specified by the PC. Note that in the state numbered 18, the MAR is loaded with the address contained in PC, the PC is incremented in preparation for the FETCH of the next LC-3 instruction, and, if there is no interrupt request present (INT = 0), the flow passes to the state numbered 33. We will describe in Section C.6 the flow of control if INT = 1, that is, if an external device is requesting an interrupt.

Before we get into what happens during the clock cycle when the processor is in the state numbered 33, we should explain the numbering system—that is, why 18 and 33. Recall, from our discussion of finite state machines in Chapter 3, that each state must be uniquely specified and that this unique specification is accomplished by means of the state variables. Our state machine that implements the LC-3 ISA requires 52 distinct states to describe the entire behavior of the LC-3. Figure C.2 shows 31 of them plus pointers to three others (states 8, 13, and 49). Figure C.7 shows the other 18 states (plus 8, 13, and 49) that are pointed to in Figure C.2. We will come into contact with all of them as we go through this appendix. Since k logical variables can uniquely identify 2^k items, six state variables are needed to uniquely specify 52 states. The number next to each node in Figure C.2 is the decimal equivalent of the values (0 or 1) of the six state variables for the corresponding state. Thus, the state numbered 18 has state variable values 010010.

Now, then, back to what happens after the clock cycle in which the activity of state 18 has finished. Again, if no external device is requesting an interrupt,

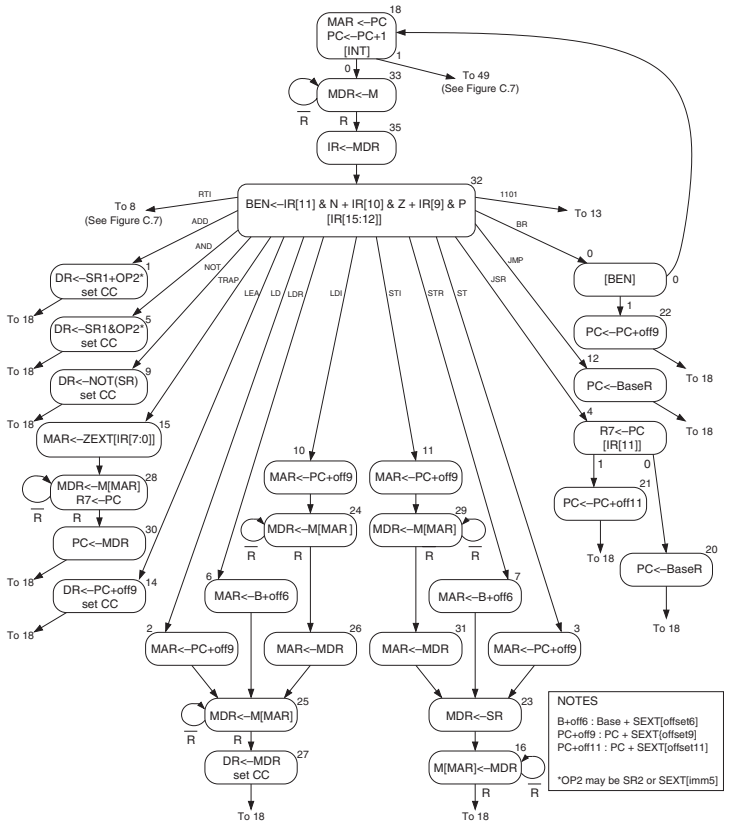

Figure C.2 A state machine for the LC-3

the flow passes to state 33. In state 33, since the MAR contains the address of the instruction to be processed, this instruction is read from memory and loaded into the MDR. Since this memory access can take multiple cycles, this state continues to execute until a ready signal from the memory (R) is asserted, indicating that the memory access has completed. Thus the MDR contains the valid contents of the memory location specified by MAR. The state machine then moves on to state 35, where the instruction is loaded into the instruction register (IR), completing the fetch phase of the instruction cycle.

Note that the arrow from the last state of each instruction cycle (i.e., the state that completes the processing of that LC-3 instruction) takes us to state 18 (to begin the instruction cycle of the next LC-3 instruction).

C.3 The Data Path

The data path consists of all components that actually process the information during a cycle—the functional units that operate on the information, the registers that store information at the end of one cycle so it will be available for further use in subsequent cycles, and the buses and wires that carry information from one point to another in the data path. Figure C.3, an expanded version of what you have already encountered in Figure 5.9, illustrates the data path of our microarchitecture of the LC-3.

Note the control signals that are associated with each component in the data path. For example, ALUK, consisting of two control signals, is associated with the ALU. These control signals determine how the component will be used each cycle. Table C.1 lists the set of control signals that control the elements of the data path and the set of values that each control signal can have. (Actually, for readability, we list a symbolic name for each value, rather than the binary value.) For example, since ALUK consists of two bits, it can have one of four values. Which value it has during any particular clock cycle depends on whether the ALU is required to ADD, AND, NOT, or simply pass one of its inputs to the output during that clock cycle. PCMUX also consists of two control signals and specifies which input to the MUX is required during a given clock cycle. LD.PC is a single-bit control signal, and is a 0 (NO) or a 1 (YES), depending on whether or not the PC is to be loaded during the given clock cycle.

During each clock cycle, corresponding to the "current state" in the state machine, the 39 bits of control direct the processing of all components in the data path that are required during that clock cycle. The processing that takes place in the data path during that clock cycle, as we have said, is specified inside the node representing the state.

C.4 The Control Structure

The control structure of a microarchitecture is specified by its state machine. As described earlier, the state machine (Figure C.2) determines which control signals are needed each clock cycle to process information in the data path and which

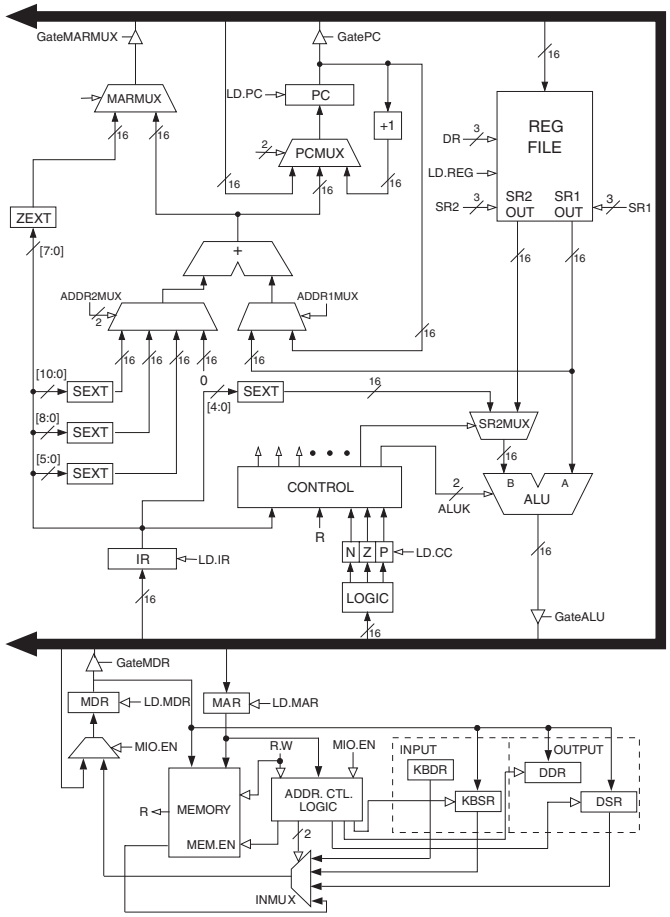

Figure C.3 The LC-3 data path

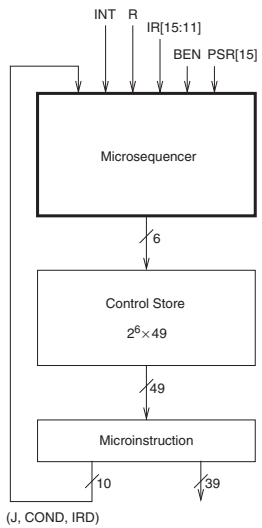

Figure C.4 The control structure of a microprogrammed implementation, overall block diagram

control signals are needed each clock cycle to direct the flow of control from the currently active state to its successor state.

Figure C.4 shows a block diagram of the control structure of our implementation of the LC-3. Many implementations are possible, and the design considerations that must be studied to determine which of many possible implementations should be used is the subject of a full course in computer architecture.

We have chosen here a straightforward microprogrammed implementation. Each state of the control structure requires 39 bits to control the processing in the data path and 10 bits to help determine which state comes next. These 49 bits are collectively known as a *microinstruction*. Each microinstruction (i.e., each state of the state machine) is stored in one 49-bit location of a special memory called the control store. There are 52 distinct states. Since each state corresponds to one microinstruction in the control store, the control store for our microprogrammed implementation requires six bits to specify the address of each microinstruction.

Table C.1 Data Path Control Signals

Signal Name	Signal Values	
LD.MAR/1:	NO, LOAD	
LD.MDR/1:	NO, LOAD	
LD.IR/1:	NO, LOAD	
LD.BEN/1:	NO, LOAD	
LD.REG/1:	NO, LOAD	
LD.CC/1:	NO, LOAD	
LD.PC/1:	NO, LOAD	
LD.Priv/1:	NO, LOAD	
LD.SavedSSP/1:	NO, LOAD	
LD.SavedUSP/1:	NO, LOAD	
LD.Vector/1:	NO, LOAD	
GatePC/1:	NO, YES	
GateMDR/1:	NO, YES	
GateALU/1:	NO, YES	
GateMARMUX/1:	NO, YES	
GateVector/1:	NO, YES	
GatePC-1/1:	NO, YES	
GatePSR/1:	NO, YES	
GateSP/1:	NO, YES	
PCMUX/2:	PC+1	;select pc+1
	BUS	;select value from bus
	ADDER	;select output of address adder
DRMUX/2:	11.9	;destination IR[11:9]
	R7	;destination R7
	SP	;destination R6
SR1MUX/2:	11.9	;source IR[11:9]
	8.6	;source IR[8:6]
	SP	;source R6
ADDR1MUX/1:	PC, BaseR	
ADDR2MUX/2:	ZERO	;select the value zero
	offset6	;select SEXT[IR[5:0]]
	PCoffset9	;select SEXT[IR[8:0]]
	PCoffset11	;select SEXT[IR[10:0]]
SPMUX/2:	SP+1	;select stack pointer+1
	SP−1	;select stack pointer−1
	Saved SSP	;select saved Supervisor Stack Pointer
	Saved USP	;select saved User Stack Pointer
MARMUX/1:	7.0	;select ZEXT[IR[7:0]]
	ADDER	;select output of address adder
VectorMUX/2:	INTV	
	Priv.exception	
	Opc.exception	
PSRMUX/1:	individual settings, BUS	
ALUK/2:	ADD, AND, NOT, PASSA	
MIO.EN/1:	NO, YES	
R.W/1:	RD, WR	
Set.Priv/1:	0	;Supervisor mode
	1	;User mode

Table C.2 Microsequencer Control Signals

Signal Name	Signal Values
J/6:	
COND/3:	COND0 ;Unconditional COND1 ;Memory Ready COND2 ;Branch COND3 ;Addressing Mode COND4 ;Privilege Mode COND5 ;Interrupt test
IRD/1:	NO, YES

Table C.2 lists the function of the 10 bits of control information that help determine which state comes next. Figure C.5 shows the logic of the microsequencer. The purpose of the microsequencer is to determine the address in the control store that corresponds to the next state, that is, the location where the 49 bits of control information for the next state are stored.

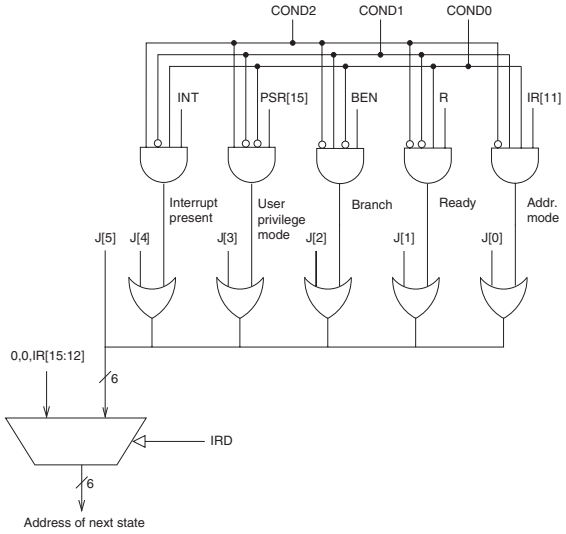

Figure C.5 The microsequencer of the LC-3

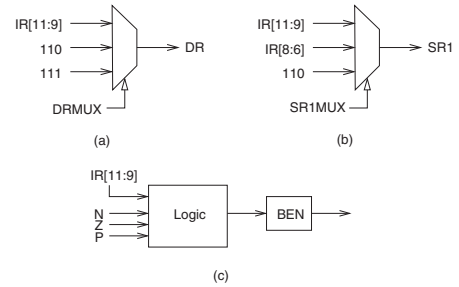

Figure C.6 Additional logic required to provide control signals

Note that state 32 of the state machine (Figure C.2) has 16 "next" states, depending on the LC-3 instruction being executed during the current instruction cycle. This state carries out the DECODE phase of the instruction cycle described in Chapter 4. If the IRD control signal in the microinstruction corresponding to state 32 is 1, the output MUX of the microsequencer (Figure C.5) will take its source from the six bits formed by 00 concatenated with the four opcode bits IR[15:12]. Since IR[15:12] specifies the opcode of the current LC-3 instruction being processed, the next address of the control store will be one of 16 addresses, corresponding to the 15 opcodes plus the one unused opcode, IR[15:12] = 1101. That is, each of the 16 next states is the first state to be carried out after the instruction has been decoded in state 32. For example, if the instruction being processed is ADD, the address of the next state is state 1, whose microinstruction is stored at location 000001. Recall that IR[15:12] for ADD is 0001.

If, somehow, the instruction inadvertently contained IR[15:12] = 1101, the unused opcode, the microarchitecture would execute a sequence of microinstructions, starting at state 13. These microinstructions would respond to the fact that an instruction with an illegal opcode had been fetched. Section C.6.3 describes what happens.

Several signals necessary to control the data path and the microsequencer are not among those listed in Tables C.1 and C.2. They are DR, SR1, BEN, INT, and R. Figure C.6 shows the additional logic needed to generate DR, SR1, and BEN.

The INT signal is supplied by some event external to the normal instruction processing, indicating that normal instruction processing should be interrupted and this external event dealt with. The interrupt mechanism was described in Chapter 8. The corresponding flow of control within the microarchitecture is described in Section C.6.

The remaining signal, R, is a signal generated by the memory in order to allow the LC-3 to operate correctly with a memory that takes multiple clock cycles to read or store a value.

Suppose it takes memory five cycles to read a value. That is, once MAR contains the address to be read and the microinstruction asserts READ, it will take five cycles before the contents of the specified location in memory are available to be loaded into MDR. (Note that the microinstruction asserts READ by means of two control signals: MIO.EN/YES and R.W/RD; see Figure C.3.)

Recall our discussion in Section C.2 of the function of state 33, which accesses an instruction from memory during the FETCH phase of each instruction cycle. For the LC-3 to operate correctly, state 33 must execute five times before moving on to state 35. That is, until MDR contains valid data from the memory location specified by the contents of MAR, we want state 33 to continue to re-execute. After five clock cycles, the memory has completed the "read," resulting in valid data in MDR, so the processor can move on to state 35. What if the microarchitecture did not wait for the memory to complete the read operation before moving on to state 35? Since the contents of MDR would still be garbage, the microarchitecture would put garbage into IR in state 35.

The ready signal (R) enables the memory read to execute correctly. Since the memory knows it needs five clock cycles to complete the read, it asserts a ready signal (R) throughout the fifth clock cycle. Figure C.2 shows that the next state is 33 (i.e., 100001) if the memory read will not complete in the current clock cycle and state 35 (i.e., 100011) if it will. As we have seen, it is the job of the microsequencer (Figure C.5) to produce the next state address.

The 10 microsequencer control bits for state 33 are as follows:

```
IRD/0       ; NO
COND/001    ; Memory Ready
J/100001
```

With these control signals, what next state address is generated by the microsequencer? For each of the first four executions of state 33, since R = 0, the next state address is 100001. This causes state 33 to be executed again in the next clock cycle. In the fifth clock cycle, since R = 1, the next state address is 100011, and the LC-3 moves on to state 35. Note that in order for the ready signal (R) from memory to be part of the next state address, COND had to be set to 001, which allowed R to pass through its four-input AND gate.

C.5 Memory-Mapped I/O

As you know from Chapter 8, the LC-3 ISA performs input and output via memory-mapped I/O, that is, with the same data movement instructions that it uses to read from and write to memory. The LC-3 does this by assigning an address to each device register. Input is accomplished by a load instruction whose effective address is the address of an input device register. Output is accomplished by a store instruction whose effective address is the address of an output device register. For example, in state 25 of Figure C.2, if the address in MAR is xFE02,

Table C.3 Truth Table for Address Control Logic

MAR	MIO.EN	R.W	MEM.EN	IN.MUX	LD.KBSR	LD.DSR	LD.DDR
xFE00	0	R	0	x	0	0	0
xFE00	0	W	0	x	0	0	0
xFE00	1	R	0	KBSR	0	0	0
xFE00	1	W	0	x	1	0	0
xFE02	0	R	0	x	0	0	0
xFE02	0	W	0	x	0	0	0
xFE02	1	R	0	KBDR	0	0	0
xFE02	1	W	0	x	0	0	0
xFE04	0	R	0	x	0	0	0
xFE04	0	W	0	x	0	0	0
xFE04	1	R	0	DSR	0	0	0
xFE04	1	W	0	x	0	1	0
xFE06	0	R	0	x	0	0	0
xFE06	0	W	0	x	0	0	0
xFE06	1	R	0	x	0	0	0
xFE06	1	W	0	x	0	0	1
other	0	R	0	x	0	0	0
other	0	W	0	x	0	0	0
other	1	R	1	mem	0	0	0
other	1	W	1	x	0	0	0

MDR is supplied by the KBDR, and the data input will be the last keyboard character typed. On the other hand, if the address in MAR is a legitimate memory address, MDR is supplied by the memory.

The state machine of Figure C.2 does not have to be altered to accommodate memory-mapped I/O. However, something has to determine when memory should be accessed and when I/O device registers should be accessed. This is the job of the address control logic shown in Figure C.3.

Table C.3 is a truth table for the address control logic, showing what control signals are generated, based on (1) the contents of MAR, (2) whether or not memory or I/O is accessed this cycle (MIO.EN/NO, YES), and (3) whether a load or store is requested (R.W/Read, Write). Note that, for a memory-mapped load, data can be supplied to MDR from one of four sources: memory, KBDR, KBSR, or DSR. The address control logic provides the appropriate select signals to the INMUX. For a memory-mapped store, the data supplied by MDR can be written to memory, KBSR, DDR, or DSR. The address control logic supplies the appropriate enable signal to the corresponding structure.

C.6 Interrupt and Exception Control

The final piece of the state machine needed to complete the LC-3 story are those states that control the initiation of an interrupt, those states that control the return from an interrupt (the RTI instruction), and those states that control the initiation

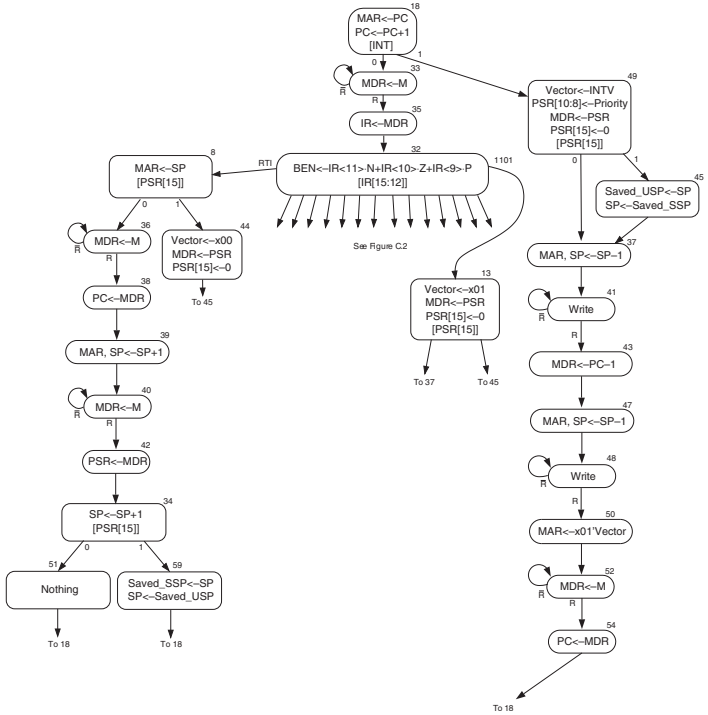

Figure C.7 LC-3 state machine showing interrupt control

of one of the two exceptions specified by the ISA. The two exceptions are a privilege mode violation and an illegal opcode. Figure C.7 shows the state machine that carries these out. Figure C.8 shows the data path, after adding the additional structures to Figure C.3 that are needed to make interrupt and exception processing work.

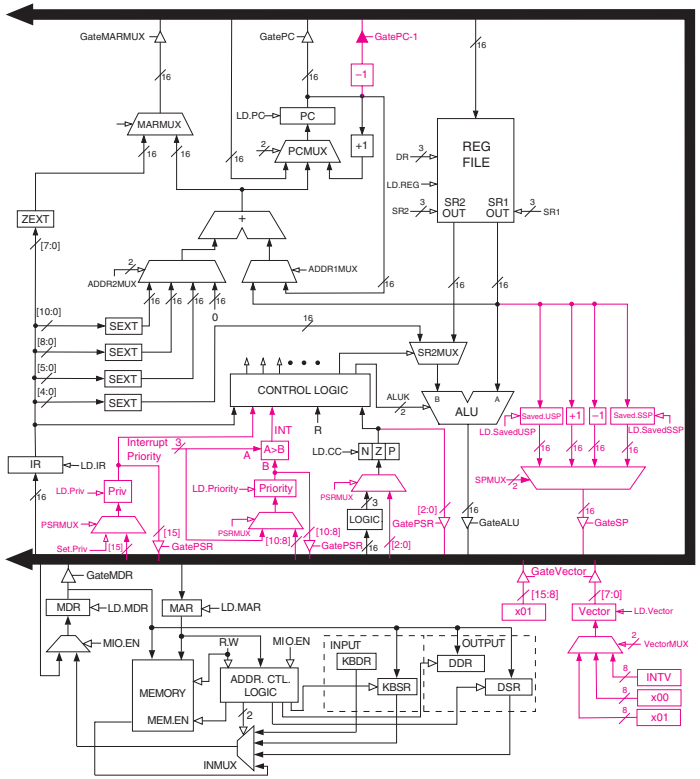

Figure C.8 LC-3 data path, including additional structures for interrupt control

C.6.1 Initiating an Interrupt

While a program is executing, an interrupt can be requested by some external event so that the normal processing of instructions can be preempted and the control can turn its attention to processing the interrupt. The external event requests an interrupt by asserting its interrupt request signal. Recall from Chapter 8 that if the priority level of the device asserting its interrupt request signal is higher than the priority level of the currently executing program, INT is asserted and INTV is loaded with the appropriate interrupt vector. The microprocessor responds to INT by initiating the interrupt. That is, the processor puts itself into supervisor mode, pushes the PSR and PC of the interrupted process onto the supervisor stack, and loads the PC with the starting address of the interrupt service routine. The PSR contains the privilege mode PSR[15], priority level PSR[10:8], and condition codes PSR[2:0] of a program. It is important that when the processor resumes execution of the interrupted program, the privilege mode, priority level, and condition codes are restored to what they were when the interrupt occurred.

The microarchitecture of the LC-3 initiates an interrupt as follows: Recall from Figure C.2 that in state 18, while MAR is loaded with the contents of PC and PC is incremented, INT is tested.

State 18 is the only state in which the processor checks for interrupts. The reason for only testing in state 18 is straightforward: Once an LC-3 instruction starts processing, it is easier to let it finish its complete instruction cycle (FETCH, DECODE, etc.) than to interrupt it in the middle and have to keep track of how far along it was when the external device requested an interrupt (i.e., asserted INT). If INT is only tested in state 18, the current instruction cycle can be aborted early (even before the instruction has been fetched), and control directed to initiating the interrupt.

The test is enabled by the control signals that make up COND5, which are 101 only in state 18, allowing the value of INT to pass through its four-input AND gate to contribute to the address of the next state. Since the COND signals are not 101 in any other state, INT has no effect in any other state.

In state 18, the 10 microsequencer control bits are as follows:

```
IRD/0      ; NO
COND/101   ; Test for interrupts
J/100001
```

If INT = 1, a 1 is produced at the output of the AND gate, which in turn makes the next state address not 100001, corresponding to state 33, but rather 110001, corresponding to state 49. This starts the initiation of the interrupt (see Figure C.7).

Several functions are performed in state 49. The PSR, which contains the privilege mode, priority level, and condition codes of the interrupted program, are loaded into MDR, in preparation for pushing it onto the Supervisor Stack. PSR[15] is cleared, reflecting the change to Supervisor mode, since all interrupt service routines execute in Supervisor mode. The 3-bit priority level and 8-bit interrupt vector (INTV) provided by the interrupting device are recorded. PSR[10:8] is loaded with the priority level. The internal register Vector is loaded with INTV.

PSR[10:80] <== 3'b111 (PL7, not device's priority level)

Finally, the processor must test the old PSR[15] to see which stack R6 points to before pushing PSR and PC.

If the old PSR[15] = 0, the processor is already operating in Supervisor mode. R6 is the Supervisor Stack Pointer (SSP), so the processor proceeds immediately to states 37 and 44 to push the PSR of the interrupted program onto the Supervisor Stack. If PSR[15] = 1, the interrupted process was in User mode. In that case, the USP (the current contents of R6) must be saved in Saved_USP and R6 must be loaded with the contents of Saved_SSP before moving to state 37. This is done in state 45.

The control flow from state 49 to either 37 or 45 is enabled by the 10 microsequencer control bits, as follows:

```
IRD/0      ; NO
COND/100   ; Test PSR[15], privilege mode
J/100101
```

If PSR[15] = 0, control goes to state 37 (100101); if PSR[15] = 1, control goes to state 45 (101101).

In state 37, R6 (SSP) is decremented (preparing for the push), and MAR is loaded with the address of the new top of the stack.

In state 41, the memory is enabled to WRITE (MIO.EN/YES, R.W/WR). When the write completes, signaled by R = 1, PSR has been pushed onto the Supervisor Stack, and the flow moves on to state 43.

In state 43, the PC is loaded into MDR. Note that state 43 says MDR is loaded with PC-1. Recall that in state 18, at the beginning of the instruction cycle for the interrupted instruction, PC was incremented. Loading MDR with PC-1 adjusts PC to the correct address of the interrupted instruction.

In states 47 and 48, the same sequence as in states 37 and 56 occurs, only this time, the PC of the interrupted process is pushed onto the Supervisor Stack.

The final task to complete the initiation of the interrupt is to load the PC with the starting address of the interrupt service routine. This is carried out by states 50, 52, and 54. It is accomplished in a manner similar to the loading of the PC with the starting address of a TRAP service routine. The event causing the INT request supplies the 8-bit interrupt vector INTV associated with the interrupt, similar to the 8-bit trap vector contained in the TRAP instruction. This interrupt vector is stored in the 8-bit register INTV, shown on the data path in Figure C.8.

The interrupt vector table occupies memory locations x0100 to x01FF. In state 50, the interrupt vector that was loaded into Vector in state 49 is added to the base address of the interrupt vector table (x0100) and loaded into MAR. In state 52, memory is enabled to READ. When R = 1, the read has completed and MDR contains the starting address of the interrupt service routine. In state 54, the PC is loaded with that starting address, completing the initiation of the interrupt.

It is important to emphasize that the LC-3 supports two stacks, one for each privilege mode, and two stack pointers (USP and SSP), one for each stack. R6 is the stack pointer and is loaded from the Saved_SSP when privilege changes from User mode to Supervisor mode, and from Saved_USP when privilege changes from Supervisor mode to User mode. Needless to say, when the Privilege mode

changes, the current value in R6 must be stored in the appropriate "Saved" stack pointer in order to be available the next time the privilege mode changes back.

C.6.2 Returning from an Interrupt, RTI

The interrupt service routine ends with the execution of the RTI instruction. The job of the RTI instruction is to restore the computer to the state it was in when the interrupt was initiated. This means restoring the PSR (i.e., the privilege mode, priority level, and the values of the condition codes N, Z, P) and restoring the PC. Recall that these values were pushed onto the stack during the initiation of the interrupt. They must, therefore, be popped off the stack in the reverse order.

The first state after DECODE is state 8. Here we load the MAR with the address of the top of the Supervisor Stack, which contains the last thing pushed (that has not been subsequently popped)—the state of the PC when the interrupt was initiated. At the same time, we test PSR[15] since RTI is a privileged instruction and can only execute in Supervisor mode. If PSR[15] = 0, we can continue to carry out the requirements of RTI.

PSR[15] = 0 ; RTI Completes Execution

States 36 and 38 complete the operation of restoring PC to the value it had when the interrupt was initiated. In state 36, the memory is read. When the read is completed, MDR contains the address of the instruction that was to be processed next when the interrupt occurred. State 38 loads that address into the PC.

States 39, 40, 42, and 34 restore the privilege mode, priority level, and condition codes (N, Z, P) to their original values. In state 39, the Supervisor Stack Pointer is incremented so that it points to the top of the stack after the PC was popped. The MAR is loaded with the address of the new top of the stack. State 40 initiates the memory READ; when the READ is completed, MDR contains the interrupted PSR. State 42 loads the PSR from MDR, and state 34 increments the stack pointer.

The only thing left is to check the privilege mode of the interrupted program to see whether the stack pointers have to be switched. In state 34, the microsequencer control bits are as follows:

```
IRD/0       ; NO
COND/100    ; Test PSR[15], privilege mode
J/110011
```

If PSR[15] = 0, control flows to state 51 (110011) to do nothing for one cycle. If PSR[15] = 1, control flows to state 59 where R6 is saved in Saved_SSP and R6 is loaded from Saved_USP. In both cases control returns to state 18 to begin processing the next instruction.

PSR[15] = 1 ; Privilege Mode Exception

If PSR[15] = 1, the processor has a privilege mode violation. It is attempting to execute RTI while the processor is in User mode, which is not allowed.

The processor responds to this situation by pushing the PSR and the address of the RTI instruction onto the Supervisor Stack and loading the PC with the starting address of the service routine that handles privilege mode violations. The processor does this in a way very similar to the mechanism for initiating interrupts.

First, in state 44, three functions are performed. The Vector register is loaded with the 8-bit vector that points to the entry in the interrupt vector table that contains the starting address of the Privilege mode violation exception service routine. This 8-bit vector is x00. The MDR is loaded with the PSR of the program that caused the violation. Third, PSR[15] is set to 0, since the service routine will execute with Supervisor privileges. Then the processor moves to state 45, where it follows the same flow as the initiation of interrupts.

The main difference between this flow and that for the initiation of interrupts flow comes in state 50, where MAR is loaded with x01'Vector. In the case of interrupts, Vector had previously been loaded in state 49 with INTV, which is supplied by the interrupting device. In the case of the privilege mode violation, Vector was loaded in state 44 with x00.

Two other minor differences reflect the additional functions performed in state 49 if an interrupt is initiated. First, the priority level is changed, based on the priority of the interrupting device. We do not change the priority in handling the privilege mode violation. The service routine executes at the same priority as the program that caused the violation. Second, a test to determine the privilege mode is performed for an interrupt. This is unnecessary for a privilege mode violation since the processor already knows it is executing in User mode.

C.6.3 The Illegal Opcode Exception

At the outset of Section C.6, we said the LC-3 ISA specifies two exceptions, a privilege mode violation and an illegal opcode. The privilege mode violation, as you have just seen, occurs when the processor tries to execute the RTI instruction while in User mode. The illegal opcode exception occurs if the instruction being processed specifies the undefined opcode (i.e., 1101) in bits [15:12] of the instruction. The action the processor takes is very similar to what happens when a privilege mode exception is detected. That is, the PSR and PC of the program are pushed onto the Supervisor Stack and the PC is loaded with the starting address of the Illegal Opcode Exception service routine. That initiates the service routine. From there, the service routine does whatever has been specified as the corrective action when an illegal opcode is detected.

The fact that the processor is in state 13 is enough to know that an illegal opcode is being processed. The reason: the only way it could get there is via the IR decode state 32. State 13 starts the initiation of the exception. State 13 is very similar to state 49, which starts the initiation of an interrupt, and state 44, which starts the initiation of a privilege mode violation. As with states 49 and 44, the Vector register is loaded in preparation for vectoring to the Interrupt Vector Table to find the starting address of the service routine. The exception vector in this case is x01. As with states 49 and 44, state 13 sets the Privilege mode to Supervisor (PSR[15] ← 0), since the service routine executes in Supervisor mode. Also like

those states, it loads the PSR into the MDR to start the process of pushing the PSR onto the Supervisor Stack.

Like state 44, it does not change the priority of the running program, since the urgency of handling the exception is the same as the urgency of executing the program that contains it. Like state 49, it tests the Privilege mode of the program that contains the illegal opcode, since if the currently executing program is in User mode, the stack pointers need to be switched as was described in Section C.6.1. Like state 49, the processor then microbranches either to state 37 if the stack pointer is already pointing to the Supervisor Stack, or to state 45 if the stack pointers have to be switched. From there, the initiating sequence continues in states 37, 41, 43, etc., identical to what happens when an interrupt is initiated (Section C.6.1) or a privilege mode exception is initiated (Section C.6.2). The PSR and PC are pushed onto the Supervisor Stack and the starting address of the service routine is loaded into the PC, completing the initiation of the exception.

C.7 Control Store

Figure C.9 completes our microprogrammed implementation of the LC-3. It shows the contents of each location of the control store, corresponding to the 49 control signals required by each state of the state machine. We have left the exact entries blank to allow you, the reader, the joy of filling in the required signals yourself. The solution is available from your instructor.

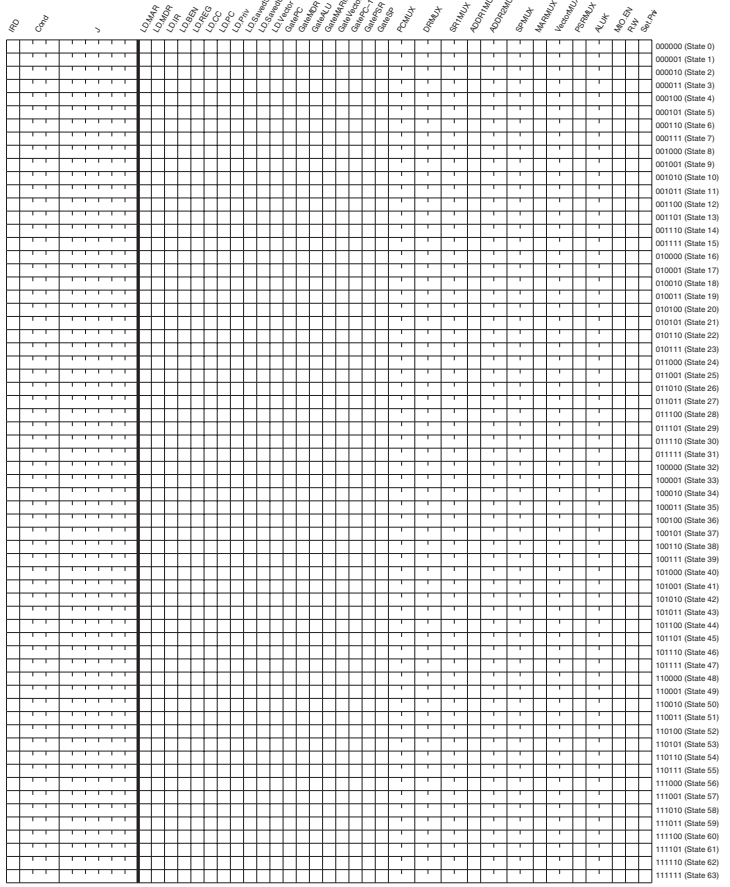

Figure C.9 Specification of the control store

NB–Table is missing one column.