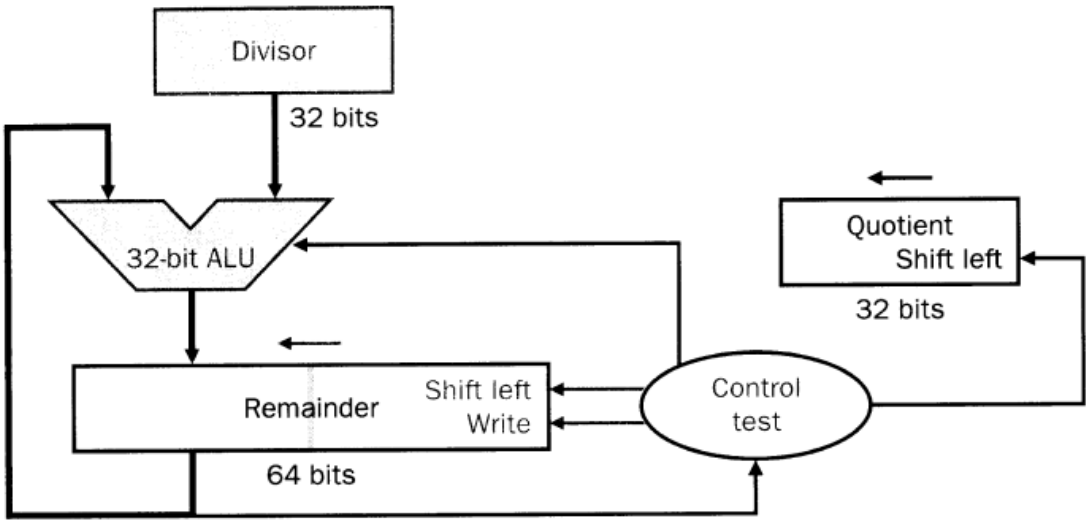
Department of Electrical Engineering
Introduction to Computer Engineering 1
Assignment 6: Computer Architecture

This assignment is not to be handed in, but is intended as a tutorial to help you understand the key elements of Chapter 5. All figure and page numbers are from Patterson and Hennessy, 3rd edition (with numbers from the 2nd edition given in parentheses afterwards).

Question 1:

Consider the following datapath:



Like the more complex control unit of the MIPS cpu, e.g. Figure 5.40, p 345 (Fig 5.50, p. 416, 2nd ed.), the control can be implemented as a finite state machine. To make the problem more interesting, assume that a non-restoring algorithm is used for division. What changes are required in the datapath? Next, draw the finite state machine diagram for the modified datapath (assume numbers are positive).

Question 2

(5.8) Suppose there were a MIPS instruction, called `bcp`, that copied a block of words from one address to another. Assume that this instruction requires that the starting address of the source block is in register $1 and the destination address is in $2, and the number of words to copy is in $3 (which is > 0). Furthermore, assume that the values of these registers as well as register $4 can be destroyed in executing this instruction (so that the registers can be used as temporaries to execute the instruction).

Write the MIPS assembly language program to implement block copy. How many instructions will be executed to perform a 100-word block copy? Using the CPI of the instructions in the multicycle implementation, how many cycles are needed for the 100-word block copy?

Question 3

(5.9) Microcode has been used to add more powerful instructions to an instruction set; let's explore the potential benefits of this approach. Give a microprogram to implement the `bcp` instruction. To implement this instruction, we will need to extend the microinstruction format. In the extended format we allow the SRCI and SRC2 fields to contain either an explicit register designator, and the SRC2 field to contain a small constant (five bits in length). We also allow the ALU destination field to contain an explicit register specifier. Finally, we will need to have microinstructions that can conditionally branch, since implementing `bcp` will require a loop. Assume the sequencing field is extended to allow a branch based on the 0 bit out of the ALU. The label specifies another microinstruction.

How many microinstructions will be executed to copy a block of 100 words? How does this compare to the number of MIPS instructions required? Assuming each microinstruction takes one cycle, how does the cycle count of the microcode implementation compare to the implementation using MIPS instructions in the previous question? How do you explain the difference?

Question 4

(5.10) To implement the `bcp` instruction in Question 3, we needed to expand the microinstruction. Assume that each field of the microinstruction is encoded separately and that there will be at most 1024 microinstructions. Find the width of each field in the original and extended microinstruction and the total widths. Remember to include bits that describe fields that can have different types of values (e.g., SRC1 in the extended microinstruction).

Question 5:

(5.11) We wish to add the instruction addiu (Add Immediate Unsigned) to the singlecycle datapath described in Chapter 5. This instruction is described in Chapter 3. Add any necessary datapaths and control signals to the single clock datapath of Figure 5.17 on page 307 (Fig. 5.19, p 360, 2nd ed.). You can photocopy the existing datapath to make it faster to show the additions.

Question 6:

(5.12) Show the additions to the table in Figure 5.18 on page 308 (Fig 5.20, p. 361, 2nd ed.) needed to set the control lines that were added in Question 2 for the instruction addiu.

Question 7:

(5.13) We wish to add the datapath parts and control needed to implement the addiu instruction in the multiclock datapath and control. Show the additions to the datapath and control lines of Figure 5.28 on page 323 (Fig. 5.33, p. 383, 2nd ed.) needed to implement this instruction in the multicycle datapath.

Question 8

(5.14) Show the steps in executing the addiu instruction in the multiclock datapath, using the same breakdown of steps as used in pages 325 through 329 (385 through 388, 2nd. ed.).

Question 9

(5.15) Show the additions to the finite state machine of Figure 5.38 on page 339 (5.42, p. 396, 2nd ed.) needed to implement the addiu instruction.

Question #1

- The modified datapath is shown on the next page (3) for an 8-bit ALU. To work in MIPS all we have to do is enlarge widths by 4x.

- Modifications

  The ALU can add or subtract.

  A 1-bit register is added to record the operation performed on the previous iteration.

- State Diagram

  Is shown on page 4. The bits in curly brackets indicate the datapath settings for that state,

  e.g., {0 1 0 1 0} ⇒ { SL=0, S0=1, S-M1=0, S-M0=1, R-M1=0, R-M1=1 }

Pay special attention to states 8, 9, and 10!

State 8 loads the register without shifting.

If the result from 8 is negative, one additional addition will have to be performed. During this step, the quotient is held constant (shift register does not shift). We don't know the result until we get to State 9.

State 9 is where the decision on the final restore is made. Both registers are held constant, and the multiplexer controlling the D flip-flop is set

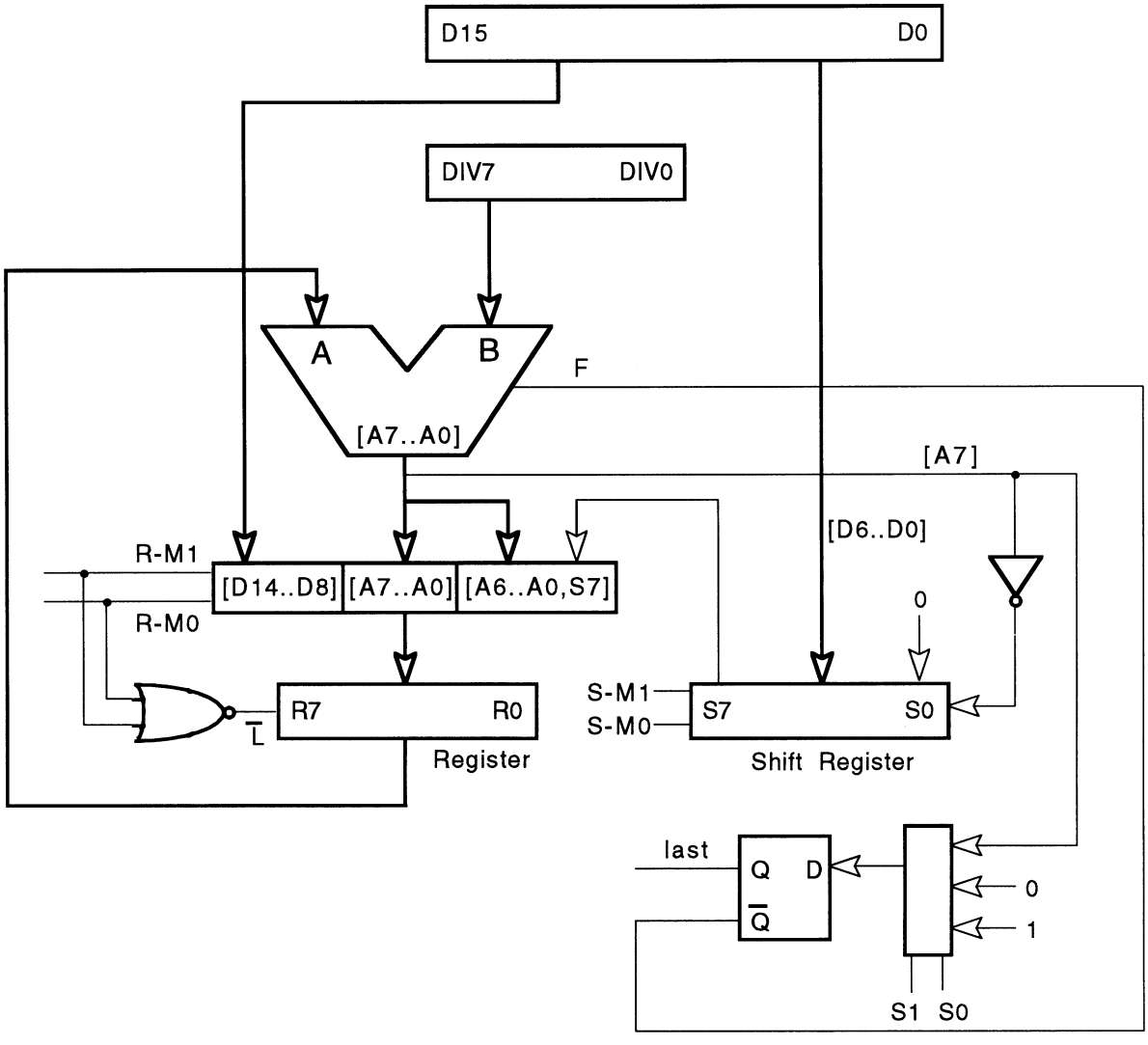
Question # 1: cont.

so that D=1 on the next state transition. The decision on whether to do the add (State 10) or skip the add is made on the basis of the current value of the flip-flop. If last = 1, a negative result was obtained, so we go to State 10. Otherwise we go to 11.

State 11 is just an idle state that freezes the registers so we can observe the result. It stays in 11 until the external input "Go" changes from 0 to 1.

In all, 11 states are needed.
I added an extra state (and an additional input) to give the circuit a "display" feature.

Non-Restoring Divider: 8-bit Datapath



Datapath Operations

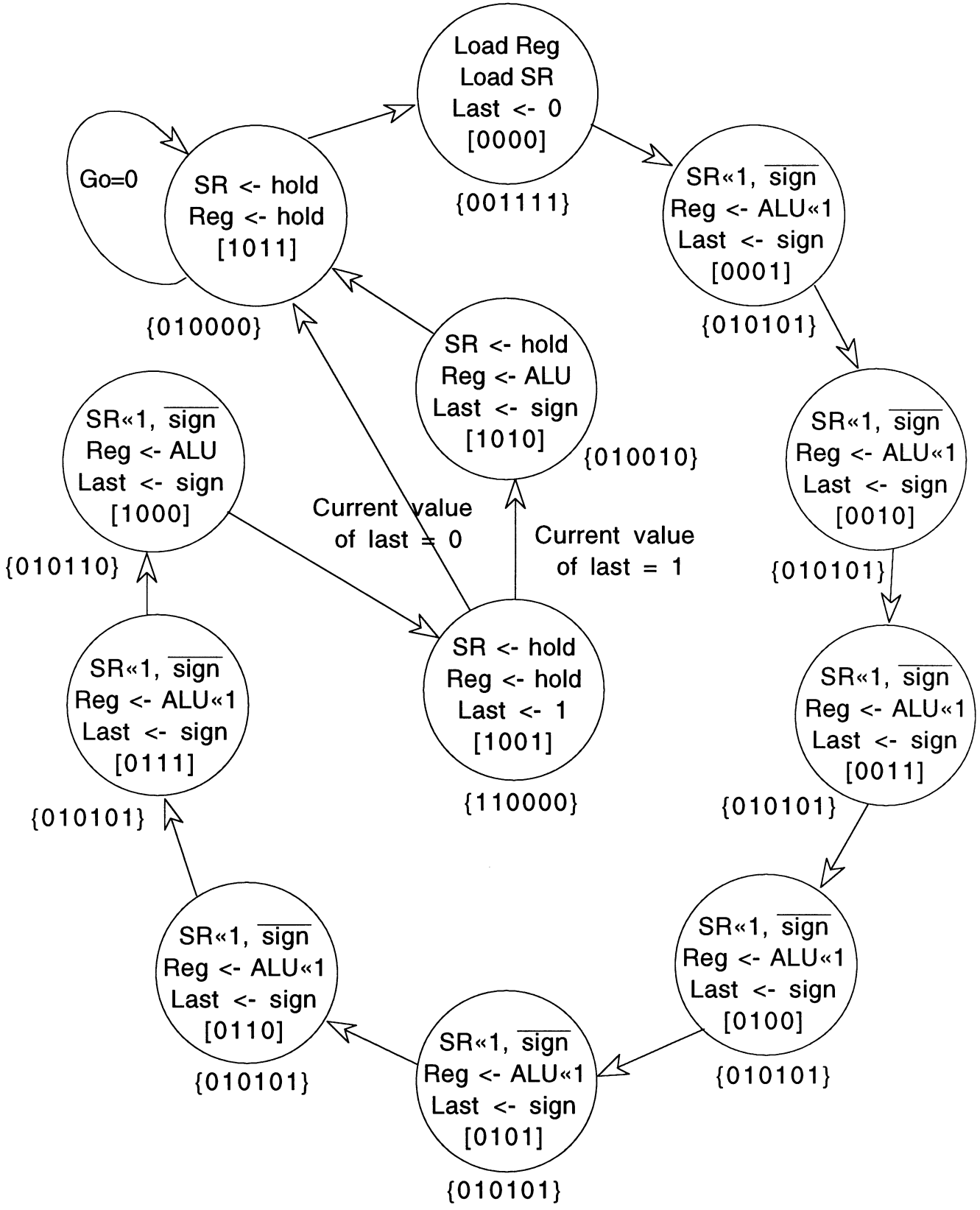
R-M1	R-M0	Function
0	0	[R7..R0] <-- [R7..R0]
0	1	[R7..R0] <-- [A6..A0],[S7]
1	0	[R7..R0] <-- [A7..A0]
1	1	[R7..R0] <-- [D14..D7]

S-M1	S-M0	Function
0	0	[S7..S0] <-- [S7..S0]
0	1	[S7..S0] <-- [S6..S0],[$\overline{A7}$]
1	0	[S7..S0] <-- [0],[S7..S1]
1	1	[S7..S0] <-- [DIV6..DIV0],[0]

S1	S0	Function
0	0	D <-- 0
0	1	D <-- A7
1	0	N/A
1	1	D <-- 1

F	Function
0	[A7..A0] <-- A + B
1	[A7..A0] <-- A - B

State Diagram for Non-Restoring Divider



Legend: {S1 S0 S-M1 S-M0 R-M1 R-M0}

P&H p 358, 5.8

```
void bcp (int *src, int *dst, int length)
{
   int i

   for (i=0; i<length; i++)
      *dst = *src++;
}
```

```
bcp:  lw   $4, 0($1)        # $1, $2, $3, $4 don't have to
      sw   $4, 0($2)        # be saved. Function assumes
      addi $3, $3, -1       # NO 0 length transfers - i.e.
      bne  $3, $0, -bcp     # no error checking.
      jr   $31
```

Cycles for 100 word copy:

lw: 5 cycles
sw: 4 cycles
addi: 4 cycles
bne: 3 cycles
16 cycles x 100 words = 1600 cycles

The jr adds an additional 3 for a total of 1603.

P&H p 358, 5.9

New microinstruction format:

- SRC1 and SRC2 can explicitly specify registers.
- SRC2 may contain a 5-bit constant.
- ALU dest can explicitly specify a register.
- Sequencing field may specify a microinstruction label.
- Branch based on D4 from ALU.

304-221B
April 97

Assignment 6

P&H p 358, 5.9

LABEL	ALU control	SRC1	SRC2	ALU destination	Memory	Memory register	PCWrite control	Sequencing
bcp	Add	$1	Extend					seq
	Add	$1	Extend		Read ALU			seq
	Add	$1	Extend		Read ALU	Write rt		seq
	Add	$2	Extend					
	Add	$2	Extend		Write ALU	Read rt		seq
	Add	$3	-1					seq
	Add	$3	-1	$3				seq
	Subt	$1	$0					fetch
	Subt	$0	$0					bcp
bcp10								

- Microcode is entered via fetch and dispatch table.
- Microprogram is executed as an INSTRUCTION, hence there is no "jr", i.e., return is via "fetch" sequence.

Total cycles: 8 cycles per iteration × 100 = 800 cycles
+ 2 cycles for the fetch = 802 cycles.

N.B. The instruction specifies what register rt is.

Comparing: 802 / 1603

⇒ Microcode saves 2 cycles / iteration as no instructions have to be fetched and decoded. The difference corresponds to 4 instructions × 2 cycles / iteration.

P&H p 359, 5.10

Extended microcode:

- SRC1: Currently selects between PC or rs (IR[21:25]). To support explicit register specification in microcode, an extra multiplexor select is required + 5 bits for register specification.

  ∴ SRC1 is extended by 6 bits. <7-bit field>

- SRC2: Currently selects 1 of 4 sources. Need 1 addl select line + 5 bits.

  ∴ SRC2 is extended by 6 bits <8-bit field>

- ALU destination: Currently selects whether ALU writes to rd or target on write back. To support explicit register specification, we need to override the selected write register by adding an extra port to the write register mux.

  Need an additional select line + 5 bits.

  ∴ ALU dest is extended by 6 bits <7-bit field>

- Sequencing field. Currently supports 4 ways of determining how to select next microcode address <2-bits>. We now add a 5th <1 extra bit> + a 10-bit microcode address.

  ∴ seq field extended by 11-bits <13-bit field>

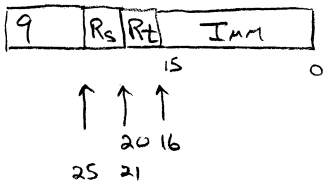
Assignment 6

P&H p 359, 5.10 cont.

	Original	Extended
ALU control:	2	2
SRC1:	1	7
SRC2:	2	8
ALU dest.:	1	7
Memory:	2	2
Memory register:	2	2
PCWrite control:	2	2
Sequencing:	2	13
	14	43

P&H p 359, 5.11

Add an addiu instruction to the single-cycle datapath (fig 5-22).



$R_t = R_s + I_{mm}$

- addi & addiu have the identical datapath to the lw Rt, constant (Rs) with the exception that the data memory mux is set to select ALU result instead of read data.
  ∴ No changes in datapath required to support addi.

- according to the definition of addiu, the only difference with addi is that exceptions are suppressed. Therefore no changes in datapath for addiu either.

P&H p 359, 5.12

Only change required is to "recognize" (i.e., decode addiu and modify the MemtoReg and MemRead control lines.

Instr	RegDst	ALUsrc	MtoReg	RegWr	MemRd	MemWr	Branch	ALUOp1	ALUOp2
R-fmt	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1
addiu	0	1	0	1	0	0	0	0	0

P&H p 359, 5.13

No changes required - same datapath as lw with memory read by-passed. Behaviour of MemtoReg and MemRead altered accordingly.

P&H p 359, 5.14

Execution of addiu in multi-clock datapath.

Step1: Fetch

IR = Memory[PC]
PC = PC+4

MemRead, IRWrite } asserted

IorD ← 0 • select PC
ALUselB ← 01 • ALU + 4
ALUselA ← 0 • select PC
ALUOp ← 00 • add

Step2: Instruction decode & register fetch step

A = Register [IR[25-21]];
B = Register [IR[20-16]];
Target = PC + (sign extend (IR[15-0] << 2);

ALU sel B ← 11 · select (IR[15-0] sign extended) << 2
ALU sel A ← 0 · select PC
ALU Op ← 00 · add

Assert Target Write.

Step 3: Arithmetic-logical. (modified-immediate)

ALU output = A op B

ALU sel A ← 1 · select R_s
ALU sel B ← 10 · select sign-extended constant
ALU Op ← 00 · add

Step 4: R-type completion (modified)

Reg [IR[16:20]] = ALU output.

RegDst ← 0 · select R_t
MemtoReg ← 0 · select ALU output
ALUsel A ← 1
ALUsel B ← 10
ALUOp ← 00
} same as step 3

Assert RegWrite.

P&H p 359, 5.15

FSM Modifications:

(see attached)

P&H p 359, 5.16

LABEL	ALU Ctl.	SRC 1	SRC 2	ALU dest	Memory	Mem Reg	PC Write	Sequencing
ADDI4	Add Add	rs rs	extend extend	rt				seq fetch

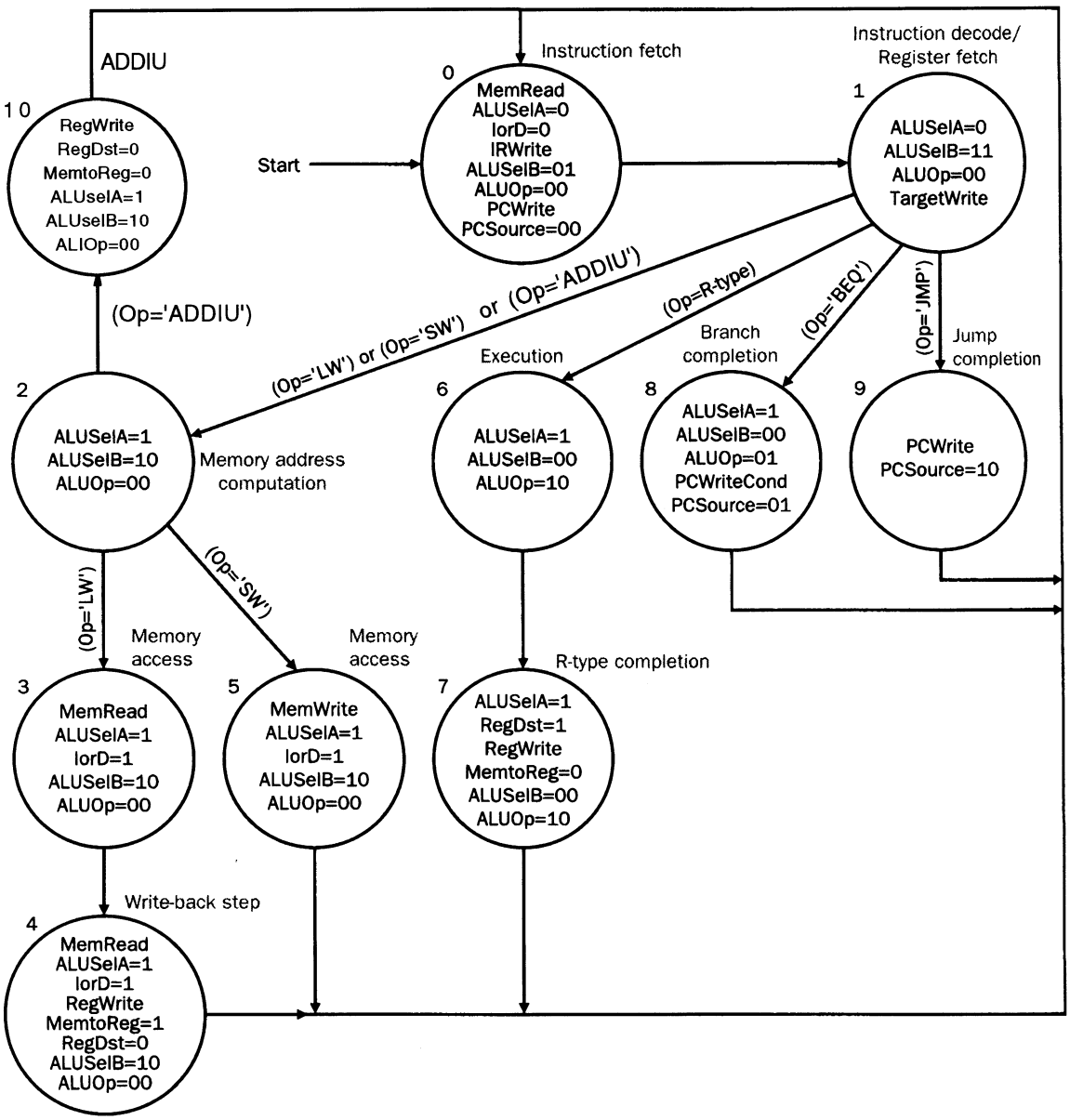


FIGURE 5.47 The complete finite state machine control for the datapath shown in Figure 5.39. The labels on the arcs are conditions that are tested to determine which state is the next state; when the next state is unconditional, no label is given. The labels inside the nodes indicate the output signals asserted during that state; we always specify the setting of a multiplexor control signal if the correct operation requires it. Hence, in some states a multiplexor control will be set to 0. In Appendix C, we will examine how to turn this finite state machine into logic equations and look at how to implement those logic equations.