Multi-cycle control: FSM

As in single-cycle case, need to generate control signals for MUXes, etc.
Can no longer be combinational, however. Why?
Use FSM to represent states, inputs, outputs

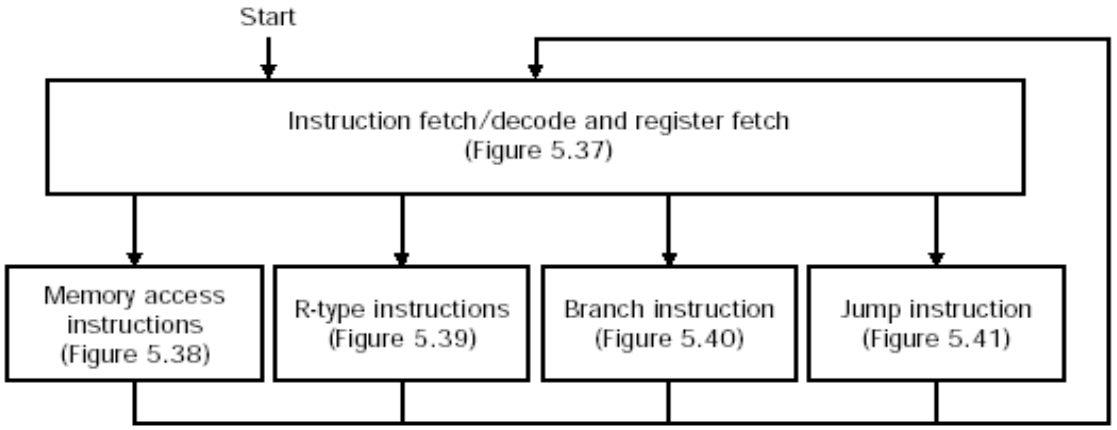


Fig. 5.36

Multi-cycle control: FSM

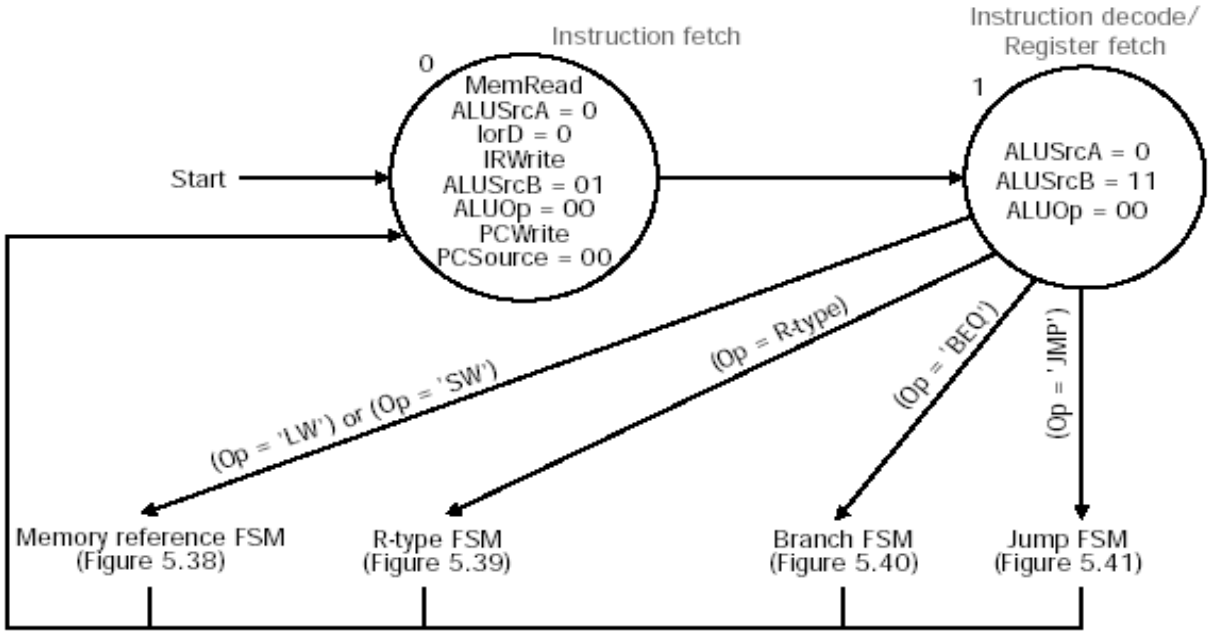


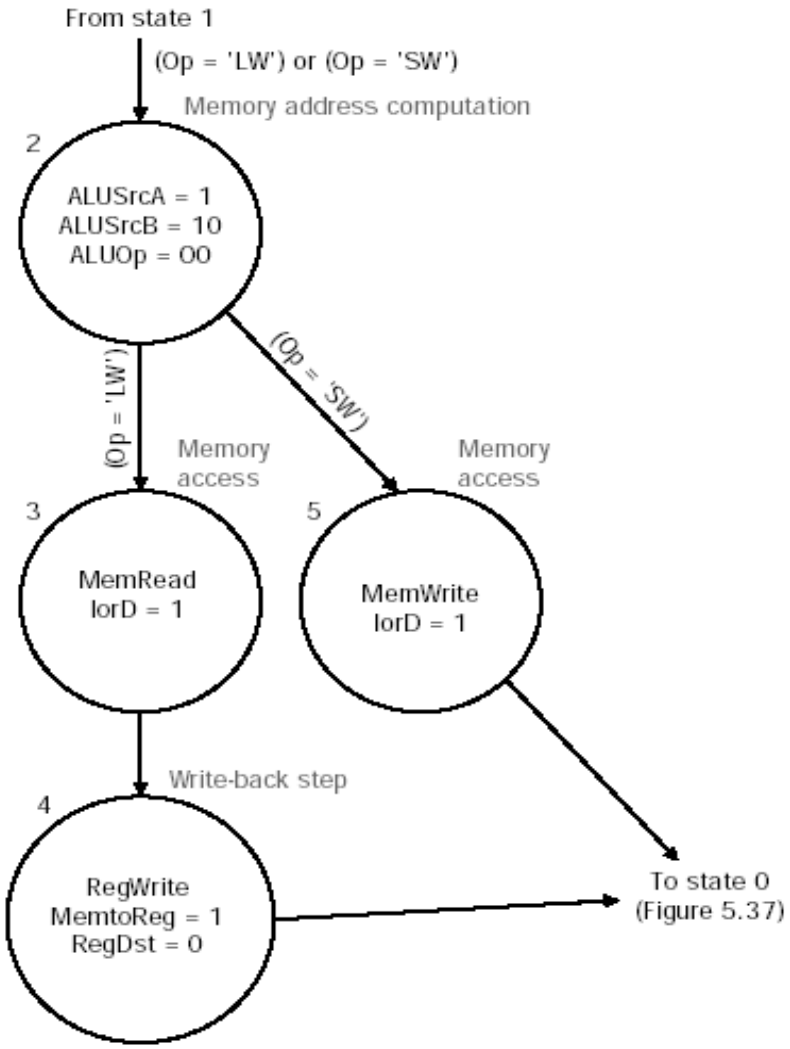
Fig. 5.37

State 0: Instruction fetch

State 1: Decode/Register fetch

OP	Next state
lw/sw	Memory reference FSM
R-type	R-type FSM
beq	Branch FSM
j	Jump FSM

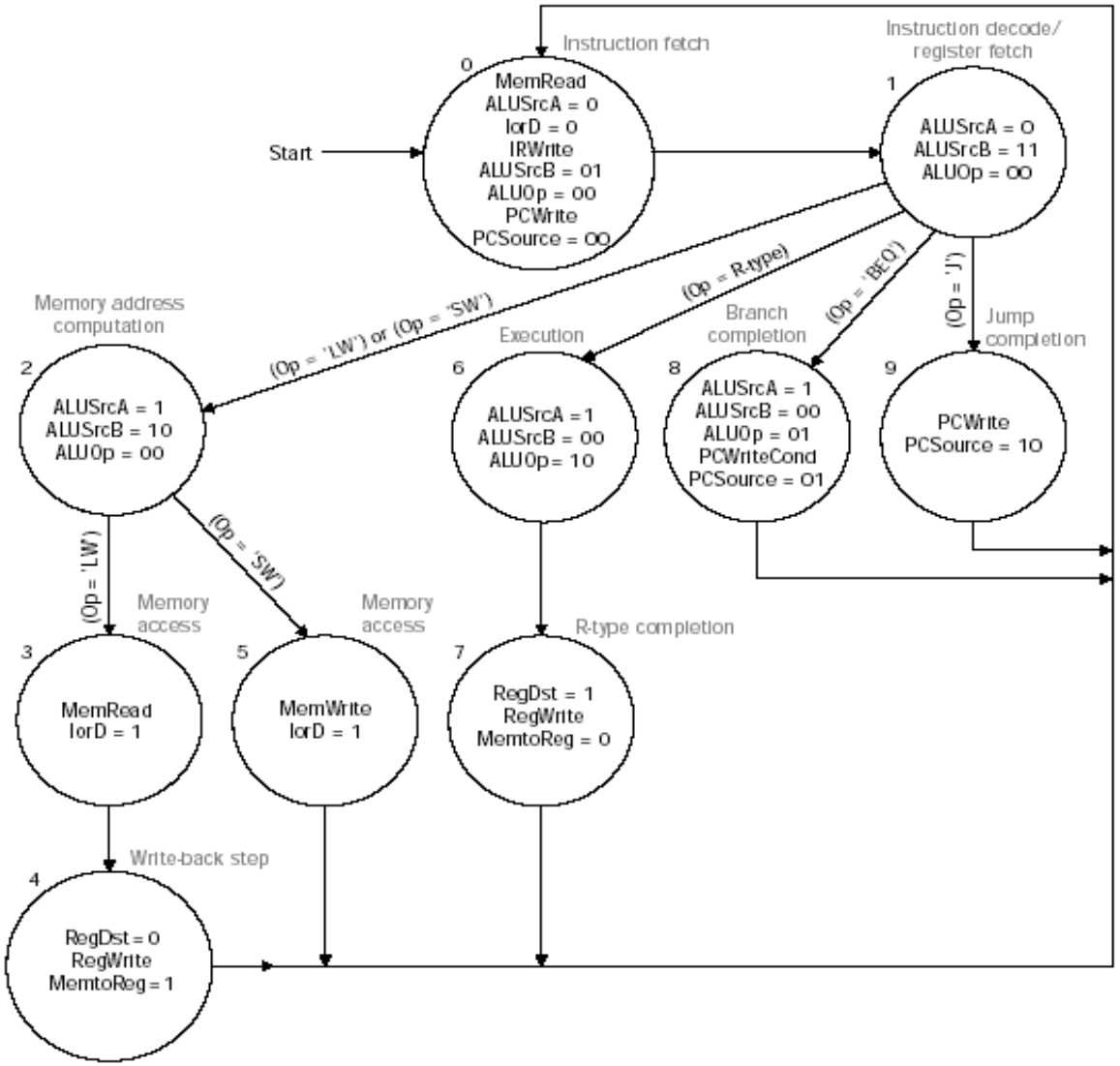
Multi-cycle control: memory-reference FSM



From state 1 (decode/register fetch)
State 2: Memory address calculation
LW: to state 3
SW: to state 5
State 3: Memory read
To state 4
State 5: Memory write
To state 0
State 4: Write register
To state 0 (fetch)

Fig. 5.38

Multi-cycle control: complete FSM



Common part

Instruction-specific

Fig. 5.42

Multi-cycle datapath: performance

Instruction type	Cycles	Distribution
R-type	4	49%
load	5	22%
store	4	11%
branch	3	16%
jump	3	2%

Average cycle time:

$$0.49 * 4 + 0.22 * 5 + 0.11 * 4 + 0.16 * 3 + 0.02 * 3 = 4.03$$

4.03/5 = 81% of critical path time (load)

Control: implementation

Moore machine
Input:
- **6 opcode bits**
  - **Op0-Op5**
- **4 state bits (10 states)**
  - **S0-S3**

Output:
- **13 control signals**
  - **3 are multiple-bit**
- **4 next-state bits**
  - **NS0-NS3**

Current state stored in state register
Control logic: combinational circuit
- **controls depend only on state**
- **next-state depends on**
  - **current state**
  - **opcode**

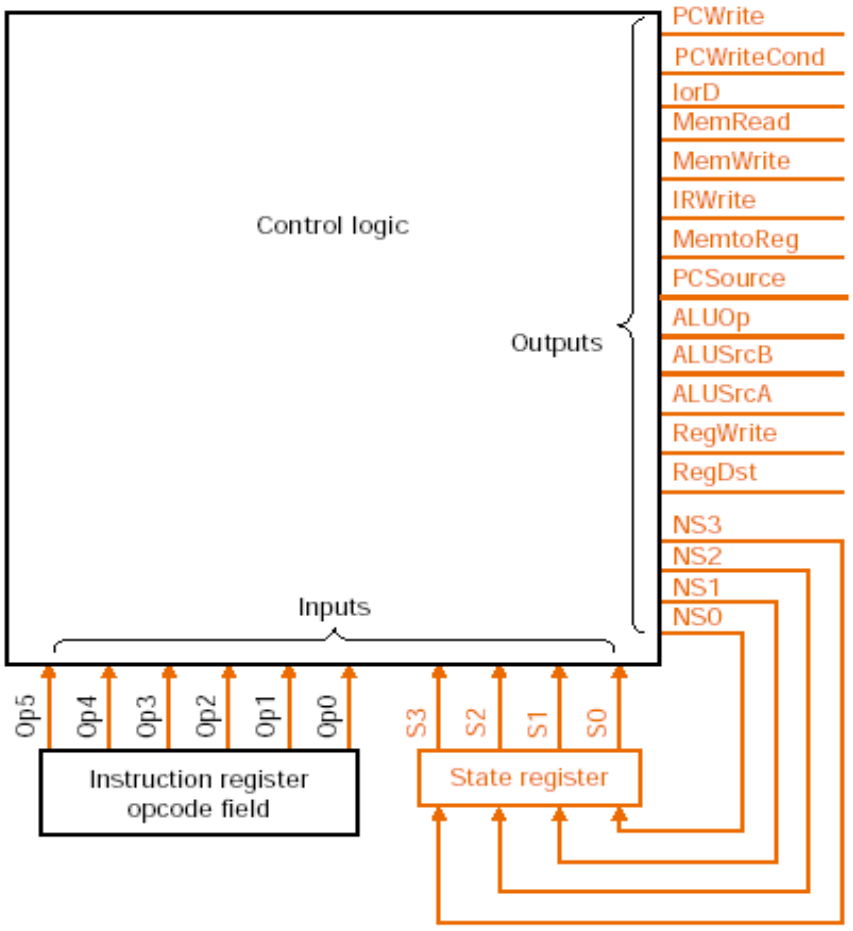


Fig. C.7

Control: PLA

Control logic implemented in PLA

Input:

- **6 opcode bits**
- **4 state bits (10 states)**

Output:

- **16 control bits**
- **4 next-state bits**

Current state stored in state register

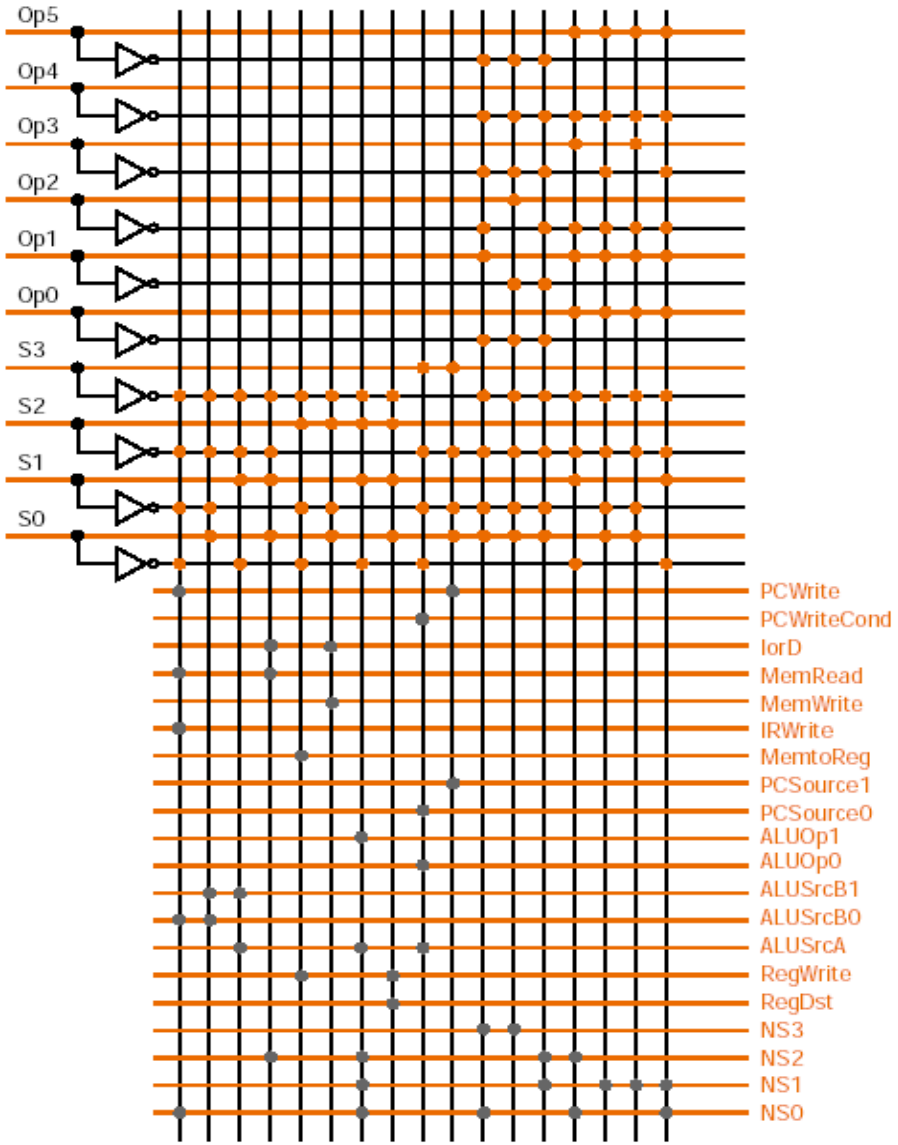


Fig. C.14

Multi-cycle control: microprogram

Disadvantages of FSM

- **Very complex for 100 instructions, even with MIPS architecture**
- **Instructions take 1-20 clock cycles**
- **Large number of states: 100s or more**

Microprogram: another level of abstraction, simplifies control design

- **Each microinstruction specifies the set of control signals in a given state**
- **Executing a microinstruction: assert the specified control signals**
- **Sequencing:**
  - **Unconditional: go to single next state**
  - **Conditional: next state depends on input**
- **Microinstruction format**
  - **Series of fields: each field specifies set of control signals**
  - **Signals never asserted simultaneously may share same field**

Field name	Function of field
ALU control	Specify the operation being done by the ALU during this clock; the result is always written in ALUOut.
SRC1	Specify the source for the first ALU operand.
SRC2	Specify the source for the second ALU operand.
Register control	Specify read or write for the register file, and the source of the value for a write.
Memory	Specify read or write, and the source for the memory. For a read, specify the destination register.
PCWrite control	Specify the writing of the PC.
Sequencing	Specify how to choose the next microinstruction to be executed.

Fig. 5.44

Multi-cycle control: microprogram

Microprogram implementation: ROM or PLA

Each microinstruction has address, represents 1 clock cycle

Selection of next instruction

- **Increment address: seq**
- **Begin executing next MIPS instruction: fetch**
- **Jump to microinstruction based on control input: dispatch**
  - **Table contains addresses of jump targets**
  - **Indexed by control inputs**
  - **Multiple tables indicated by value i in sequencing field**

Blank fields

- **Functional unit or write control not asserted**
- **MUX: don't care**

Example: instruction fetch

Label	ALU Con	SRC 1	SRC 2	RegCntl	Mem	PC Write	Sequence
Fetch	Add	PC	4		ReadPC	ALU	Seq
	Add	PC	Extshft	read			Dispatch 1

First microinstruction:

Fields	Effect
ALU control, SRC 1, SRC 2	Compute PC + 4

ALU control, SRC 1, SRC 2	Compute PC + 4
Memory	Fetch the instruction to IR
PCWrite control	Ouput of ALU is loaded into PC
Sequencing	Go to the next microinstruction

Second microinstruction:

Fields	Effect
ALU control, SRC 1, SRC 2	Store PC + sign extension (IR [15-0])
	<< 2 into ALUOut
Register control	Moves the data from the register file to A & B
Sequencing	Use dispatch table 1 for next address

Similar sets of microinstructions for:

- **memory access (load/store)**
- **R-type instructions**
- **branch**
- **jump**

Multi-cycle control: microprogram

Complete microprogram

	Label	ALU Con	SRC 1	SRC 2	RegCntl	Mem	PC Write	Sequence
fetch	Fetch	Add	PC	4		ReadPC	ALU	Seq
		Add	PC	Extshft	read			Dispatch 1
memory	Mem 1	Add	A	Exend				Dispatch 2
	LW2					Read ALU		seq
					Write Mem			Fetch
	SW2					Write ALU		Fetch
R-type	Rfor1	FuncCode	A	B				seq
					Write ALU			Fetch
branch	BEQ1	Subtr	A	B			AluOutC	Fetch
jump	Jump1						JpAdr	Fetch

Fig. 5.46

Note 10 microinstructions (1 for each state)

Sequence:

Dispatch1: go to label ending in 1 (Mem1, Rfor1, BEQ1, Jump1)

Dispatch2: go to label ending in 2 (LW2, SW2)

More complex machines: 100s or 1000s of microinstructions

May also have more temporary registers for holding intermediate results

Memory access:

Fields	Effect
ALU control, SRC 1, SRC 2	Memory address = rs + sign extend (IR[15-0])
	result available at the ALU output
Sequencing	Use 2nd dispatch tabel for continuation
	at SW 2 or LW 2

Load:

Fields	Effect
Memory	Read memory and write to MDR
Sequencing	Go to the next microinstruction

Fields	Effect
Register control	Write the contents of the MDR to register rt
Sequencing	Go to the label Fetch

Store:

Fields	Effect
Memory	Write memory with ALUOut as address
Sequencing	Go to the label Fetch