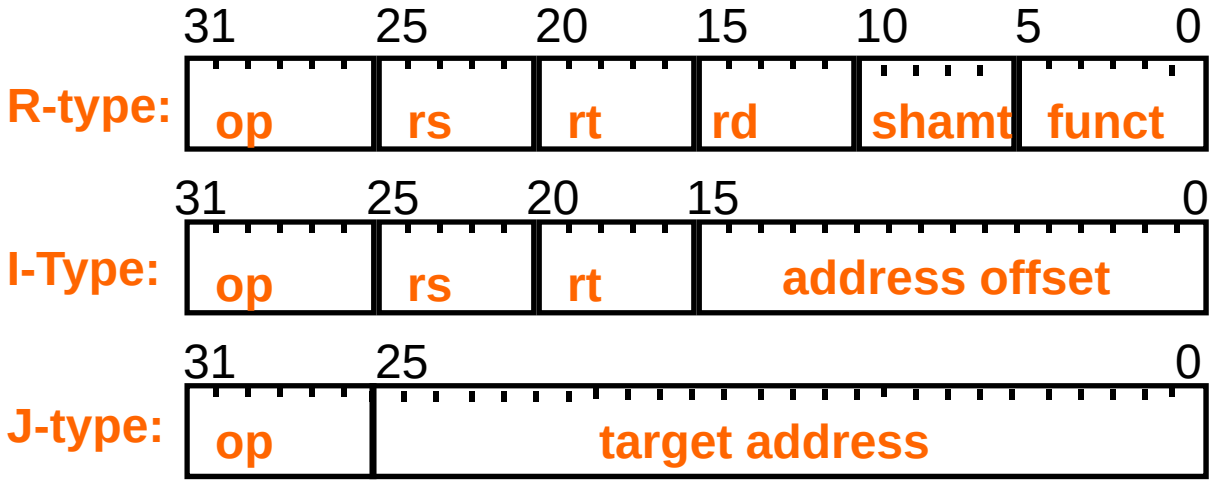
# Implementing the Simple MIPS Machine

## MIPS has 3 Instruction Types:



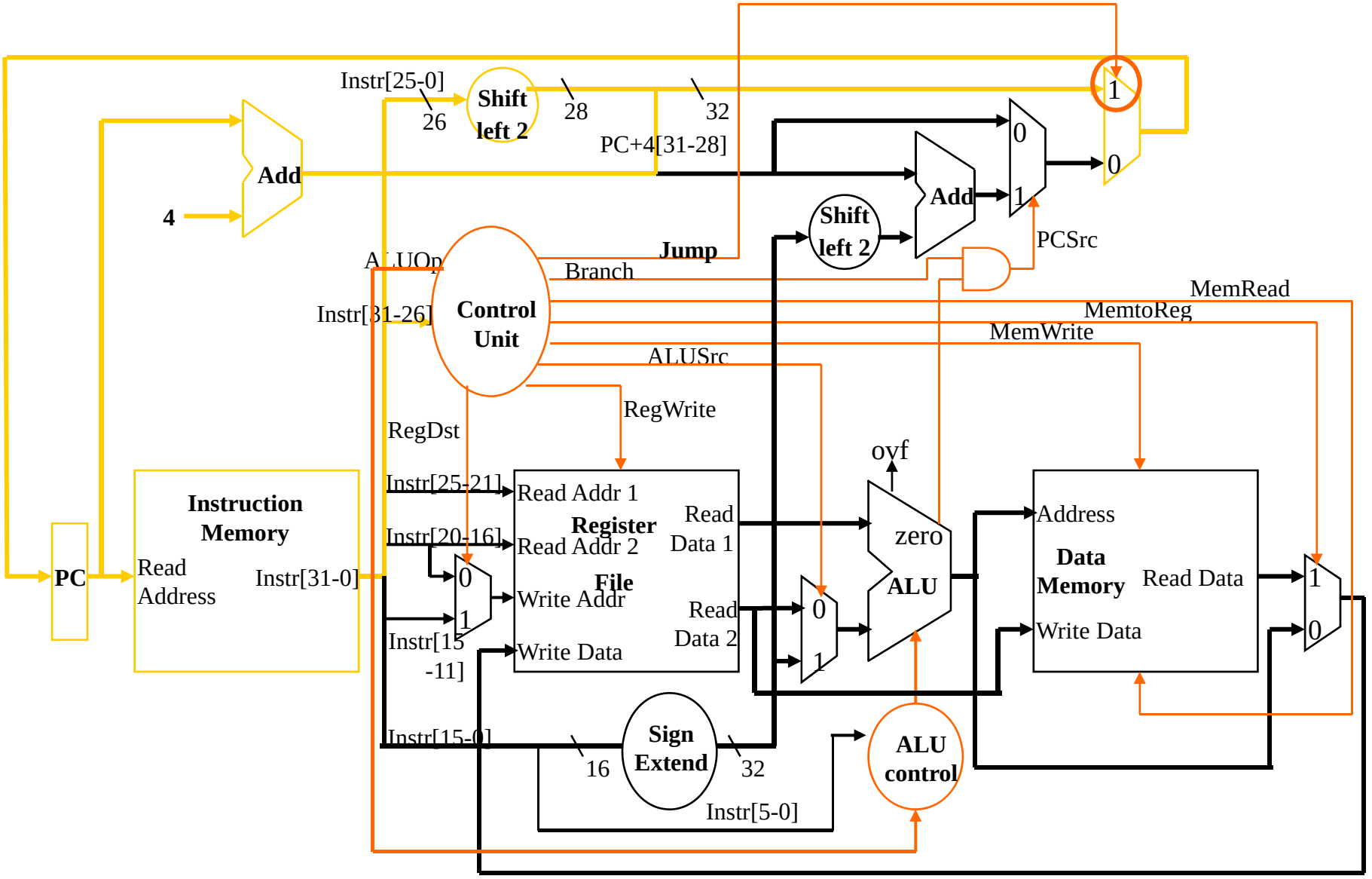

## The Simple Machine Implements:

R-Types: ADD, SUB, AND, OR, NOR, SLT

I-Types: LW, SW, & BEQ

J-Types: J

# Single Cycle Datapath with Control Unit



# Generating the ALUOp1 & ALUOp0 Signals

ALUOp1 & ALUOp0 are generated from the 6 bit OPCODE:

Instruction Types: Instructions (Opcodes) → ALUOp1 &ALUp0

R-Type: ADD, SUB, AND, OR, NOR, SLT (000000) → 10

I-Type: LW (100010) & SW (101011) → 00

I-Type: BEQ (000100) → 01

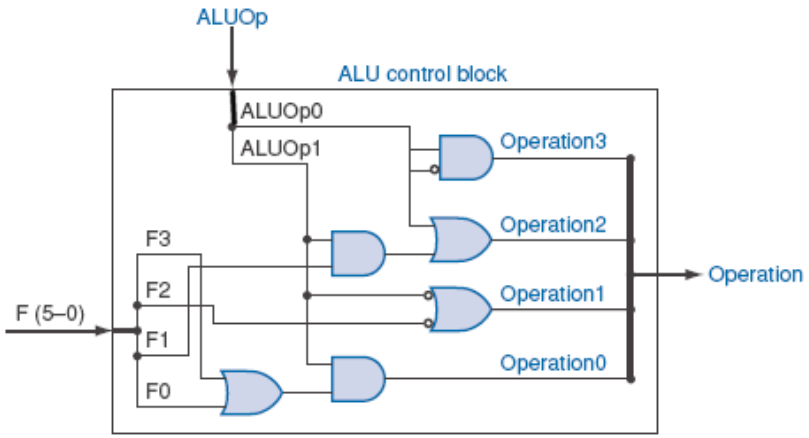
J-Type: J (000010) → XX (Doesn't use the ALU)

| OpCodes | ALUOp1 | ALUOp0 |
|---|---|---|
| 000000 | 1 | 0 |
| 100010 | 0 | 0 |
| 101011 | 0 | 0 |
| 000100 | 0 | 1 |
| 000010 | X | X |

# Generating the ALU Control Signals

| ALUOp | | Funct field | | | | | | Operation |
|---|---|---|---|---|---|---|---|---|
| ALUOp1 | ALUOp0 | F5 | F4 | F3 | F2 | F1 | F0 | |
| 0 | 0 | X | X | X | X | X | X | 0010 |
| X | 1 | X | X | X | X | X | X | 0110 |
| 1 | X | X | X | 0 | 0 | 0 | 0 | 0010 |
| 1 | X | X | X | 0 | 0 | 1 | 0 | 0110 |
| 1 | X | X | X | 0 | 1 | 0 | 0 | 0000 |
| 1 | X | X | X | 0 | 1 | 0 | 1 | 0001 |
| 1 | X | X | X | 1 | 0 | 1 | 0 | 0111 |

**FIGURE C.2.1 The truth table for the 4 ALU control bits (called Operation) as a function of the ALUOp and function code field.** This table is the same as that shown in Figure 5.13.



# Generating the One Clock Control Signals

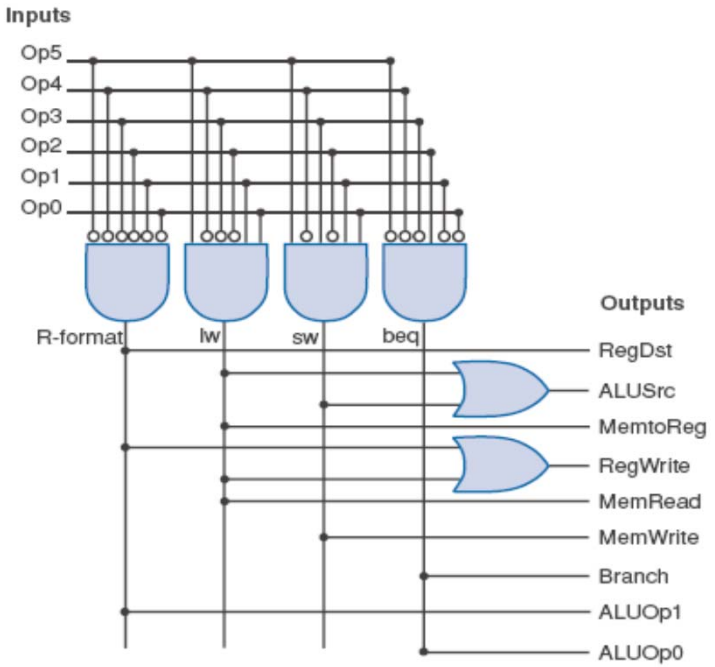
| Control | Signal name | R-format | lw | sw | beq |
|---|---|---|---|---|---|
| Inputs | Op5 | 0 | 1 | 1 | 0 |
| | Op4 | 0 | 0 | 0 | 0 |
| | Op3 | 0 | 0 | 1 | 0 |
| | Op2 | 0 | 0 | 0 | 1 |
| | Op1 | 0 | 1 | 1 | 0 |
| | Op0 | 0 | 1 | 1 | 0 |
| Outputs | RegDst | 1 | 0 | X | X |
| | ALUSrc | 0 | 1 | 1 | 0 |
| | MemtoReg | 0 | 1 | X | X |
| | RegWrite | 1 | 1 | 0 | 0 |
| | MemRead | 0 | 1 | 0 | 0 |
| | MemWrite | 0 | 0 | 1 | 0 |
| | Branch | 0 | 0 | 0 | 1 |
| | ALUOp1 | 1 | 0 | 0 | 0 |
| | ALUOp0 | 0 | 0 | 0 | 1 |

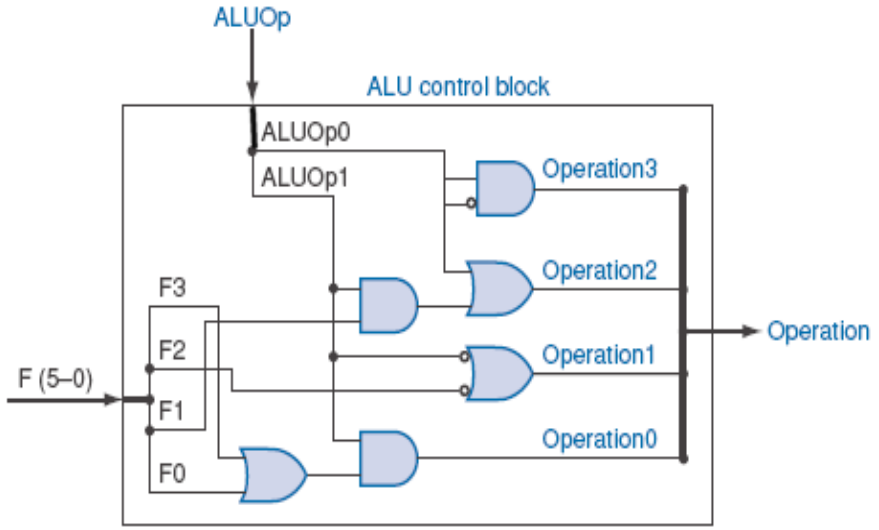
**FIGURE C.2.4 The control function for the simple one-clock implementation is completely specified by this truth table.** This table is the same as that shown in Figure 5.22.

# Generating the One clock Control Signals

| Control | Signal name | R-format | lw | sw | beq |
|---|---|---|---|---|---|
| Inputs | Op5 | 0 | 1 | 1 | 0 |
| | Op4 | 0 | 0 | 0 | 0 |
| | Op3 | 0 | 0 | 1 | 0 |
| | Op2 | 0 | 0 | 0 | 1 |
| | Op1 | 0 | 1 | 1 | 0 |
| | Op0 | 0 | 1 | 1 | 0 |
| Outputs | RegDst | 1 | 0 | X | X |
| | ALUSrc | 0 | 1 | 1 | 0 |
| | MemtoReg | 0 | 1 | X | X |
| | RegWrite | 1 | 1 | 0 | 0 |
| | MemRead | 0 | 1 | 0 | 0 |
| | MemWrite | 0 | 0 | 1 | 0 |
| | Branch | 0 | 0 | 0 | 1 |
| | ALUOp1 | 1 | 0 | 0 | 0 |
| | ALUOp0 | 0 | 0 | 0 | 1 |

**FIGURE C.2.4 The control function for the simple one-clock implementation is c pletely specified by this truth table.** This table is the same as that shown in Figure 5.22.



# Implementing the Multiple Clock Cycle Machine

- The Multiple Cycle Machine cannot be implemented simply as combinational logic.
  Why?

  Because each instruction requires 3 -5 states (clock cycles) to complete.

- It will require a Finite State machine (FSM).

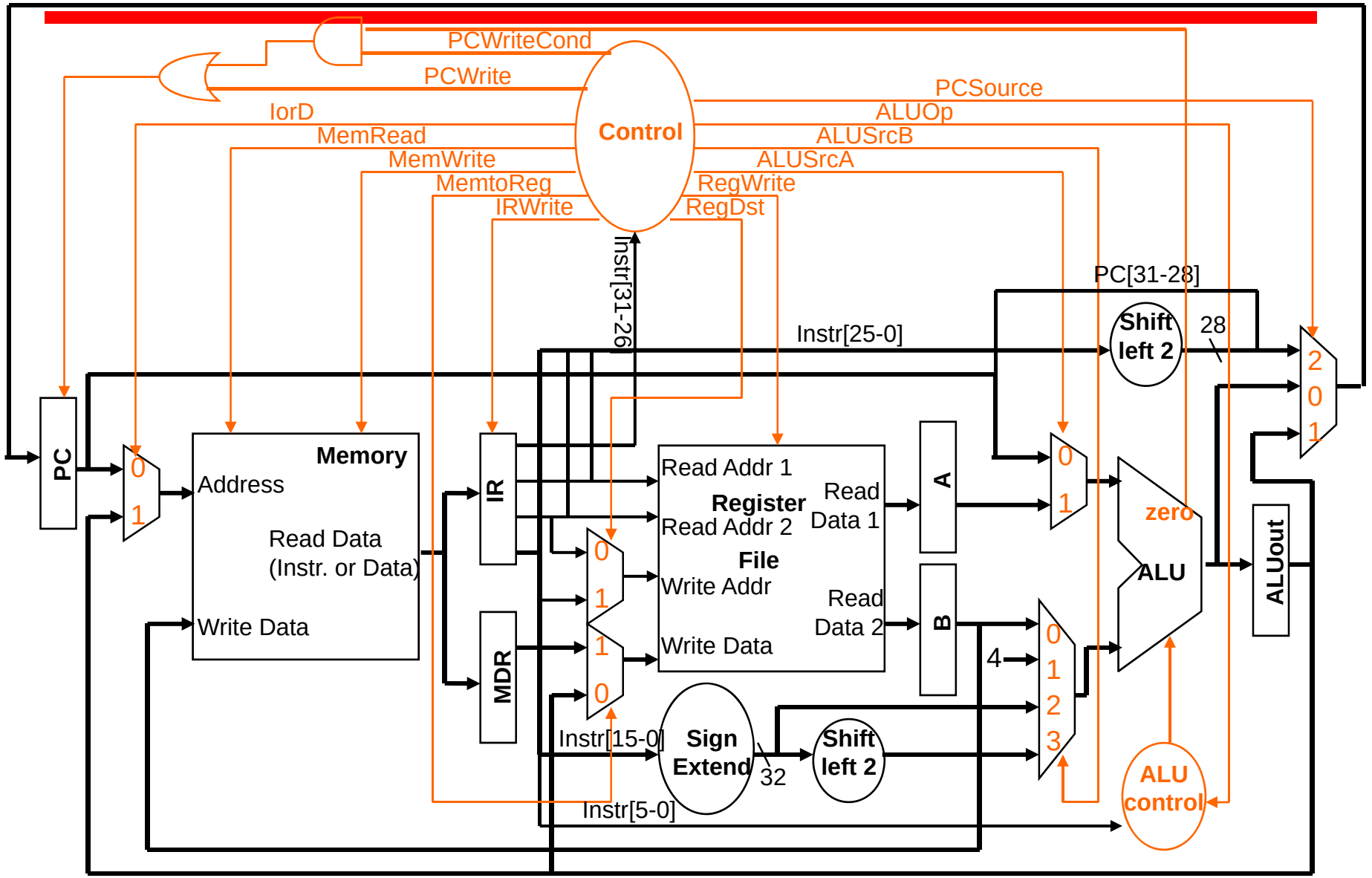
  A set of states. How many?

  Combinational logic for control signals.

  Combinational logic for next state inputs.

- It can be implemented with a PLA(s).

# The Multicycle Datapath with Control Signals



# Multiple Clock Cycle Finite State Machine

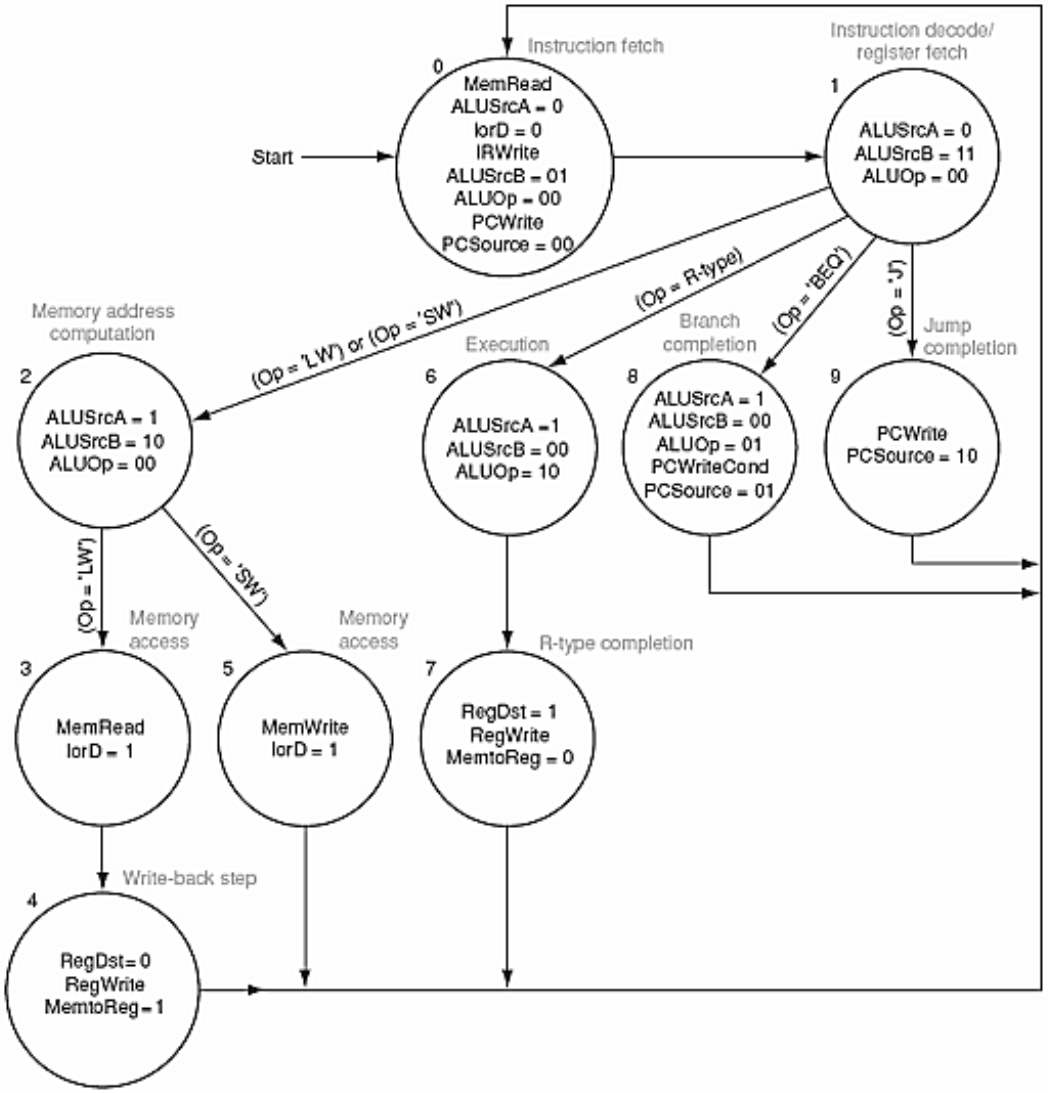
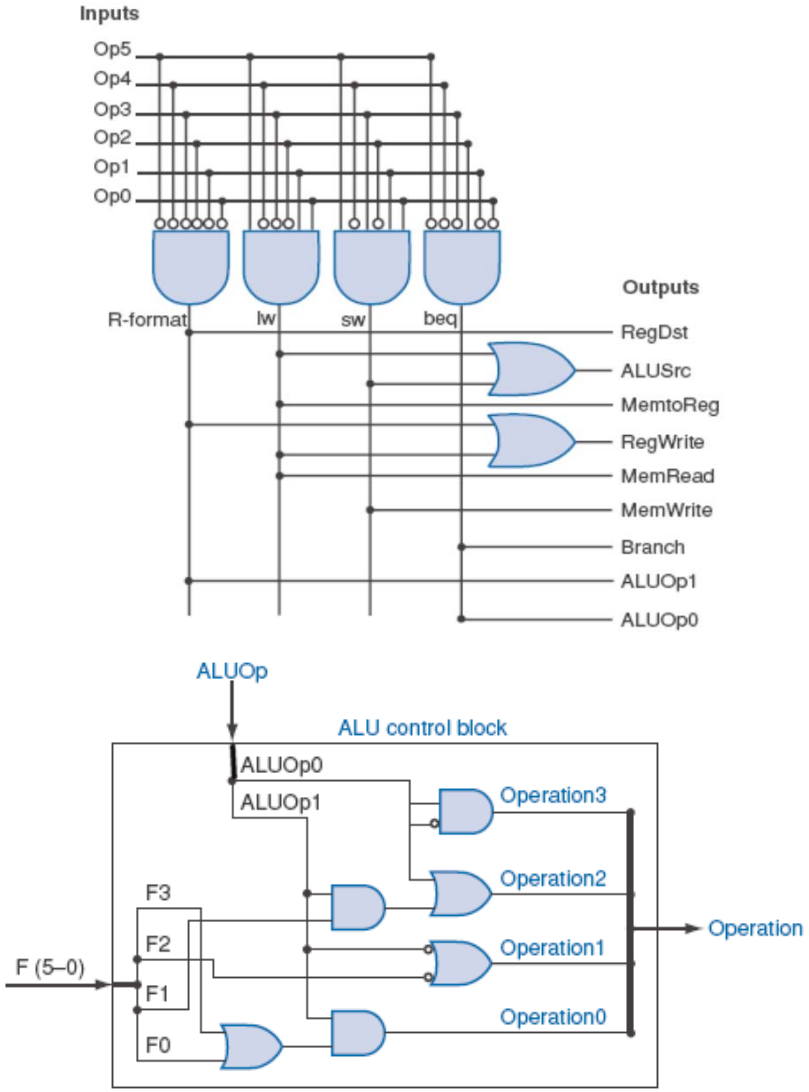
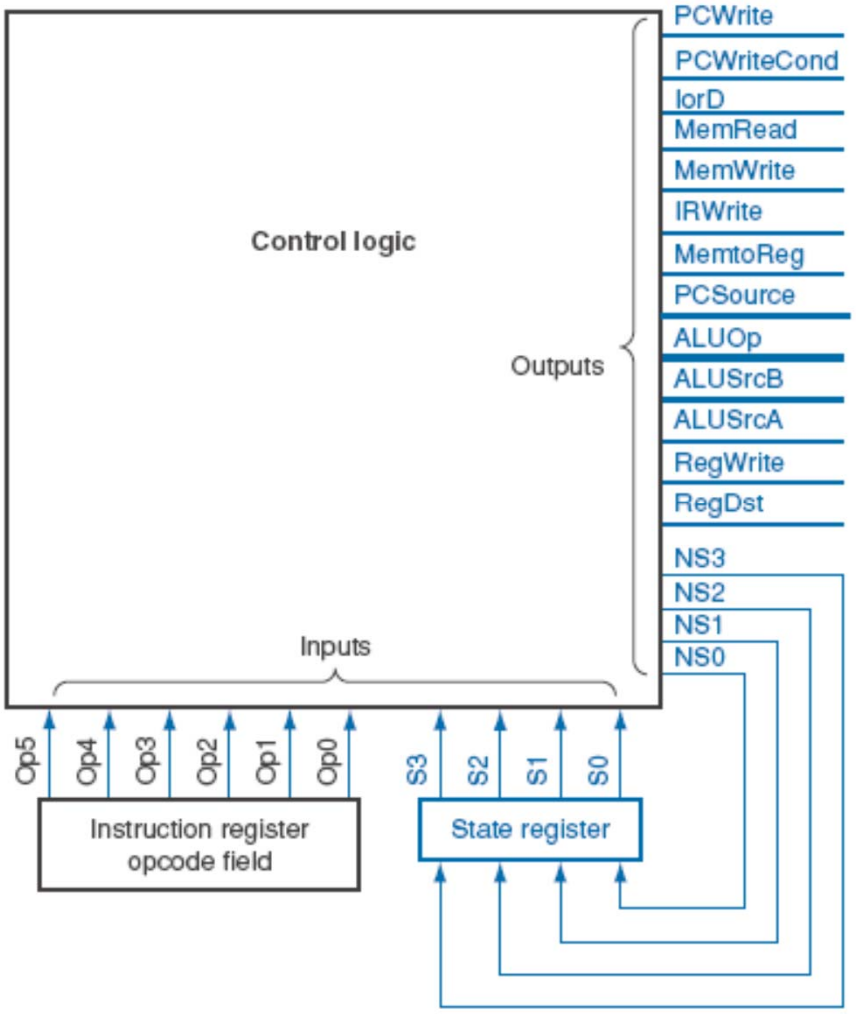



FIGURE C.3.1 The finite-state diagram that was developed in Chapter 5; it is identical to Figure 5.37.

# Comparing Single and Multiple Clock Machines



# Generating the Next State Equations

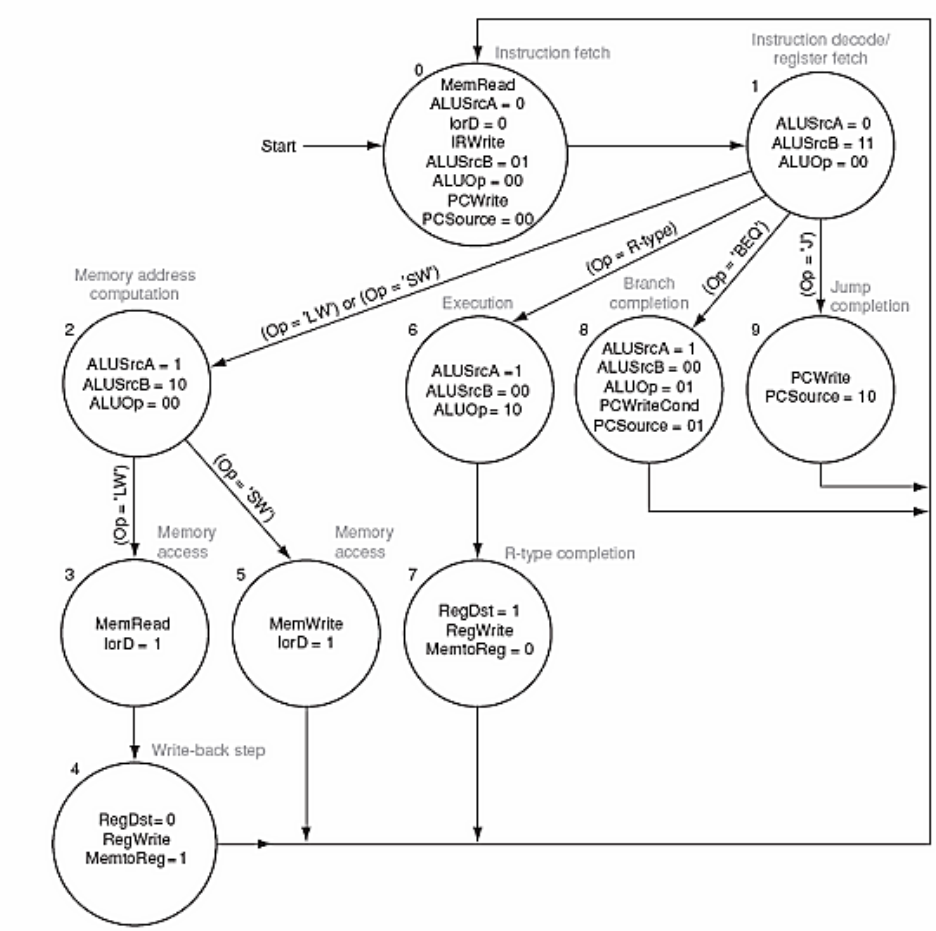
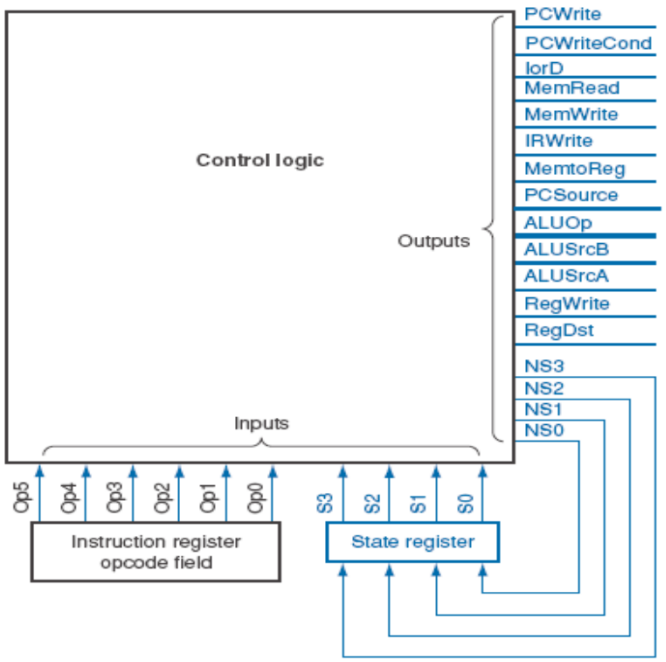


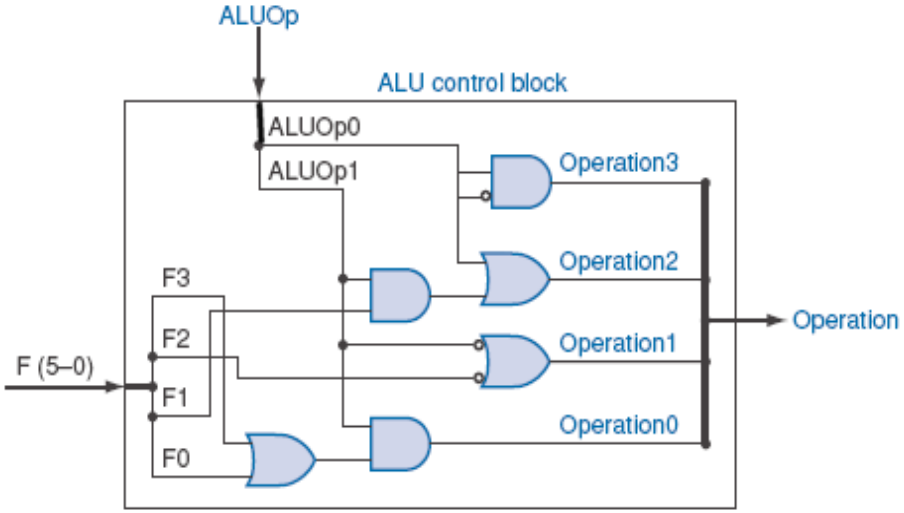

FIGURE C.3.1 The finite-state diagram that was developed in Chapter 5; it is identical to Figure 5.37.

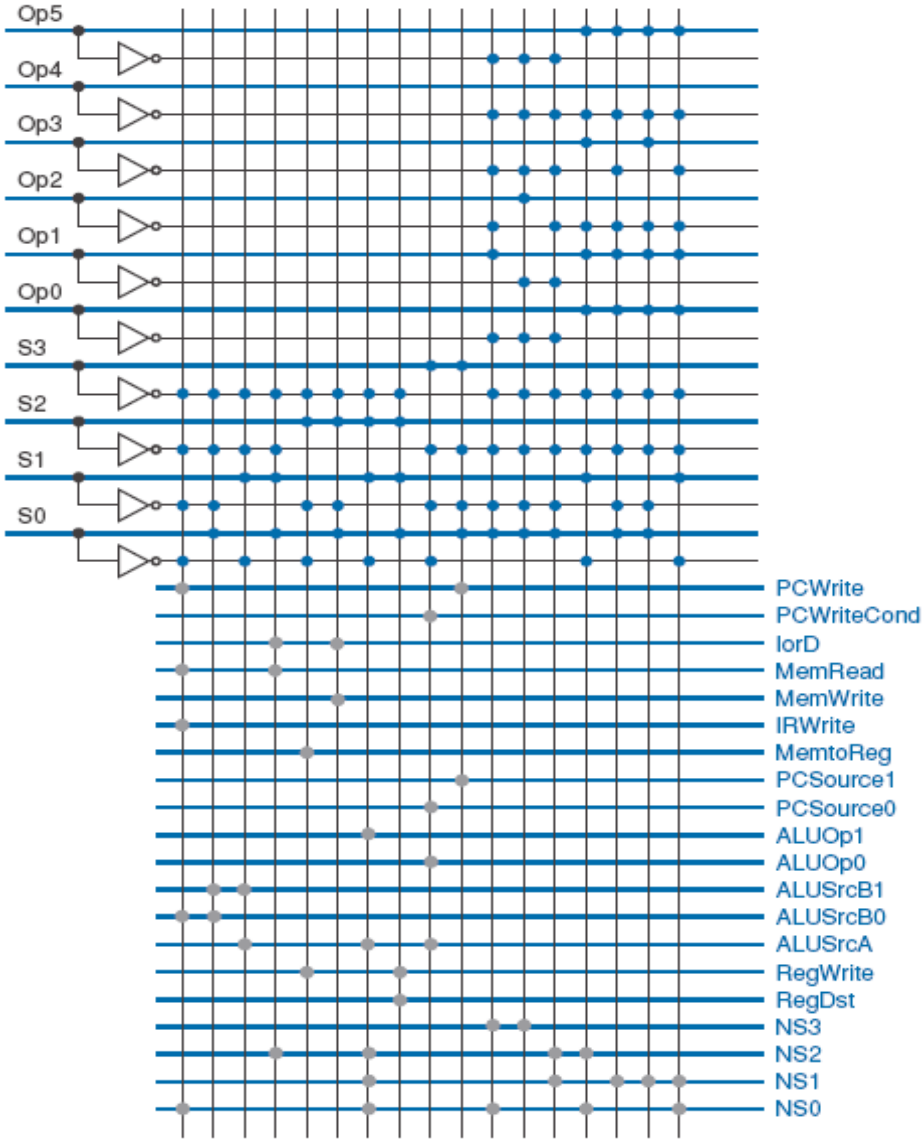
| Output | Current states | Op |
|---|---|---|
| PCWrite | state0 + state9 | |
| PCWriteCond | state8 | |
| IorD | state3 + state5 | |
| MemRead | state0 + state3 | |
| MemWrite | state5 | |
| IRWrite | state0 | |
| MemtoReg | state4 | |
| PCSource1 | state9 | |
| PCSource0 | state8 | |
| ALUOp1 | state6 | |
| ALUOp0 | state8 | |
| ALUSrcB1 | state1 +state2 | |
| ALUSrcB0 | state0 + state1 | |
| ALUSrcA | state2 + state6 + state8 | |
| RegWrite | state4 + state7 | |
| RegDst | state7 | |
| NextState0 | state4 + state5 + state7 + state8 + state9 | |
| NextState1 | state0 | |
| NextState2 | state1 | (Op = 'lw') + (Op = 'sw') |
| NextState3 | state2 | (Op = 'lw') |
| NextState4 | state3 | |
| NextState5 | state2 | (Op = 'sw') |
| NextState6 | state1 | (Op = 'R-type') |
| NextState7 | state6 | |
| NextState8 | state1 | (Op = 'beq') |
| NextState9 | state1 | (Op = 'jmp') |

FIGURE C.3.3 The logic equations for the control unit shown in a shorthand form.

# PLA Implementation of Multi Clock FSM



# Micro-Programmed Control

- An alternative to a wired implementation.
- Most Computers today are at least partially implemented with Micro-program control
- The concept is to build a simpler internal computer (micro-computer) that implements the control sequences (micro-instructions) that are stored in a micro-computer memory (ROM)
- The advantage is ease of design, flexibility, and adaptability to "families of computers".

# Control Unit Organization

The Control Memory contains sequences of microinstructions that provide the control signals to execute instruction cycles, e.g. Fetch, Indirect, Execute, and Interrupt.

Tasks of Control Unit:

- Microinstruction sequencing
- Microinstruction execution