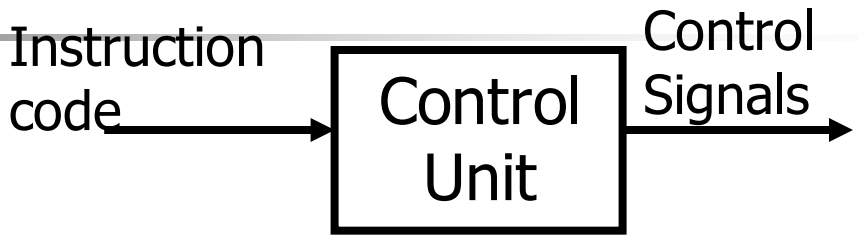
CC 311- Computer Architecture

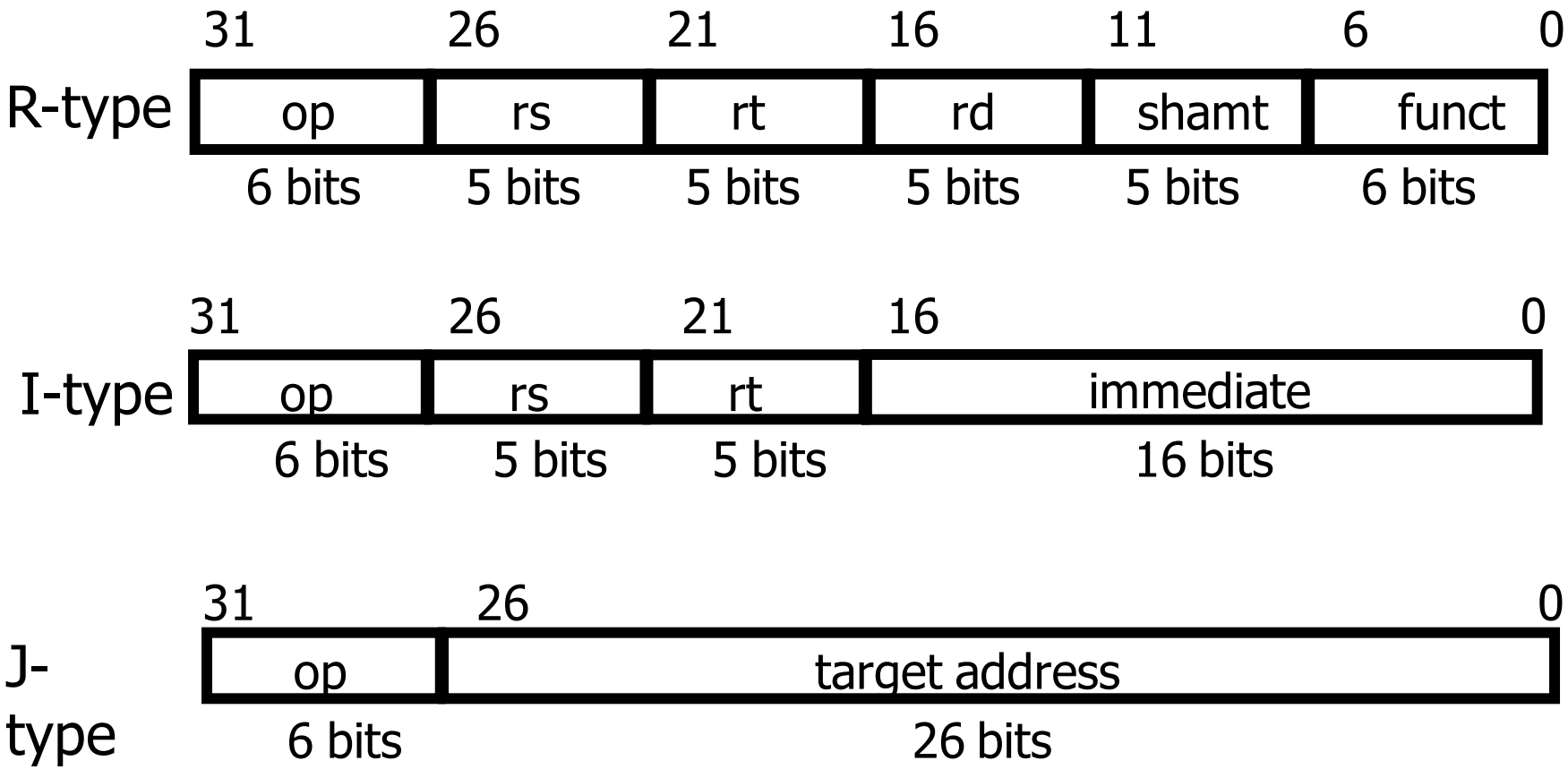
The Processor - Control

Control Unit



- Functions:
  - Select operations to be performed (ALU, read/write, etc.)
  - Control data flow (multiplexor inputs)
- Major components:
  - ALU control
    - ALU's operation is based on instruction type and function code
    - Will be designed first
  - Other controls
- Input:
  - Information comes from the 32 bits of the instruction
- Output:
  - Control signals

Review: MIPS Instruction Format



Observations on Instruction Format

- The *opcode* field is always in bits 31:26. will be referred to as (Op[5:0])
- For R-type, branch equal, and store instructions, the two registers to read are always *rs* and *rt* in bits 25:21 and 20:16 respectively
- For load and store instructions, the base register, *rs*, is always in bits 25:21
- For branch equal, load, and store, the 16-bit offset is always in positions 15:0
- The destination register is in one of two places
  - For load, the destination register (*rt*) is in position 20:16
  - For R-type instructions, destination register (*rd*) is in positions 15:11
  - => A MUX is needed to select which field of the instruction is used to indicate the destination register

ALU Control

- Summary

ALU control	Action
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

- The ALU has 4 control inputs
  - Allows 16 possible combinations
  - Only 6 combinations are used.
  - The rest could be used as don’t-care in minimization

ALU Control

- Remember:
  - ALU is needed for all instruction categories
    - *lw* and *sw* (I-Format):
      - Compute memory address
    - Arithmetic/logic(R-Format):
      - Perform arithmetic / logic operation
    - Branch(*beq*)(I-Format):
      - Subtract registers
  - We need to find ALU control signals from the information in the instruction

ALU Control

- How ALU control bits are set
  - ALUOp = ***00*** or ***01***
    - They are of I-type format
    - Depend on ***"op"*** field & does not depend on ***"funct"*** field

      lw: `100011 sssss ttttt iiiiiiiiiiiiiiii`

      sw: `101011 sssss ttttt iiiiiiiiiiiiiiii`

      beq: `000100 sssss ttttt iiiiiiiiiiiiiiii`

    => Don't care's are used *XXXXXX* for *funct* field
  - ALUOp code = *10*
    - Are of R-type instructions
    - Depend on *"funct"* field

    => *funct* code is used to set the ALU control input

ALU Control

- How the ALU control bits are set depends on ALUOp control bits and the different function codes for R-type instructions
- See fig 5.12, p. 302

Instruction Opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	and	0000
R-type	10	OR	100101	or	0001
R-type	10	set on less than	101010	set on less than	0111
R-type	11	nor	100111	nor	1100

ALU Control

- Truth table for the 4 ALU control bits
  - See Fig 5.13, p. 302
  - Once the truth table is constructed, it can be optimized and turned into gates

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 5.13 The truth table for the three ALU control bits (called Operation). The inpu are the ALUOp and function code field. Only the entries for which the ALU control is asserted are showr Some don't-care entries have been added. For example, the ALUOp does not use the encoding 11, so th truth table can contain entries 1X and X1, rather than 10 and 01. Also, when the function field is used, th first two bits (F5 and F4) of these instructions are always 10, so they are don't-care terms and are replace with XX in the truth table.

Truth Tables for ALU Control Bits

- We need to find the logical functions for O3, O2, O1, and O0

ALUOp		Funct field						Operation				Instruction
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	O3	O2	O1	O0	
0	0	X	X	X	X	X	X	0	0	1	0	lw/sw
X	1	X	X	X	X	X	X	0	1	1	0	beq
1	X	X	X	0	0	0	0	0	0	1	0	add
1	X	X	X	0	0	1	0	0	1	1	0	sub
1	X	X	X	0	1	0	0	0	0	0	0	and
1	X	X	X	0	1	0	1	0	0	0	1	or
1	X	X	X	1	0	1	0	0	1	1	1	sll
1	1	X	X	0	1	1	1	1	1	0	0	nor

Truth Tables for ALU Control Bits

- For Operation O3 = 1

ALUOp		Funct field						Operation	Instruction
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	O3 O2 O1 O0	
0	0	X	X	X	X	X	X	0 0 1 0	lw/sw
X	1	X	X	X	X	X	X	0 1 1 0	beq
1	X	X	X	0	0	0	0	0 0 1 0	add
1	X	X	X	0	0	1	0	0 1 1 0	sub
1	X	X	X	0	1	0	0	0 0 0 0	and
1	X	X	X	0	1	0	1	0 0 0 1	or
1	X	X	X	1	0	1	0	0 1 1 1	sll
1	1	X	X	0	1	1	1	1 1 0 0	nor

ALUOp		Funct field						Operation	Instruction
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	O3 O2 O1 O0	
1	1	X	X	0	1	1	1	1 1 0 0	nor

ALUOp		Funct field					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
1	1	X	X	0	1	1	1

- O3 = ALUOp1 . ALUOp0. F3'.F2.F1.F0

Truth Tables for ALU Control Bits

- For Operation O2 = 1

ALUOp		Funct field						Operation	Instruction
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	O3 O2 O1 O0	
0	0	X	X	X	X	X	X	0 0 1 0	lw/sw
X	1	X	X	X	X	X	X	0 1 1 0	beq
1	X	X	X	0	0	0	0	0 0 1 0	add
1	X	X	X	0	0	1	0	0 1 1 0	sub
1	X	X	X	0	1	0	0	0 0 0 0	and
1	X	X	X	0	1	0	1	0 0 0 1	or
1	X	X	X	1	0	1	0	0 1 1 1	sll
1	1	X	X	0	1	1	1	1 1 0 0	nor

ALUOp		Funct field						Operation	Instruction
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	O3 O2 O1 O0	
X	1	X	X	X	X	X	X	0 1 1 0	beq
1	X	X	X	0	0	1	0	0 1 1 0	sub
1	X	X	X	1	0	1	0	0 1 1 1	sll
1	1	X	X	0	1	1	1	1 1 0 0	nor

ALUOp		Funct field					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
X	1	X	X	X	X	X	X
1	X	X	X	X	0	1	0
1	1	X	X	0	1	1	1

- O2 = ALUOp1.ALUOp0

ALUOp		Funct field					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
1	1	X	X	X	X	X	X

Truth Tables for ALU Control Bits

- For Operation O1 =1

ALUOp		Funct field						Operation	Instruction
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	O3 O2 O1 O0	
0	0	X	X	X	X	X	X	0 0 1 0	lw/sw
X	1	X	X	X	X	X	X	0 1 1 0	beq
1	X	X	X	0	0	0	0	0 0 1 0	add
1	X	X	X	0	0	1	0	0 1 1 0	sub
1	X	X	X	0	1	0	0	0 0 0 0	and
1	X	X	X	0	1	0	1	0 0 0 1	or
1	X	X	X	1	0	1	0	0 1 1 1	sll
1	1	X	X	0	1	1	1	1 1 0 0	nor

ALUOp		Funct field						Operation	Instruction
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	O3 O2 O1 O0	
0	0	X	X	X	X	X	X	0 0 1 0	lw/sw
X	1	X	X	X	X	X	X	0 1 1 0	beq
1	X	X	X	0	0	0	0	0 0 1 0	add
1	X	X	X	0	0	1	0	0 1 1 0	sub
1	X	X	X	1	0	1	0	0 1 1 1	sll

ALUOp		Funct field					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
0	0	X	X	X	X	X	X
X	1	X	X	X	X	X	X
1	X	X	X	X	0	X	0

ALUOp		Funct field					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
0	0	X	X	X	X	X	X
1	1	X	X	X	X	X	X

- O1 = ALUOp1'. ALUOp0' + ALOP1.ALUOp0

Truth Tables for ALU Control Bits

- For Operation O0 =1

ALUOp		Funct field						Operation	Instruction
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	O3 O2 O1 O0	
0	0	X	X	X	X	X	X	0 0 1 0	lw/sw
X	1	X	X	X	X	X	X	0 1 1 0	beq
1	X	X	X	0	0	0	0	0 0 1 0	add
1	X	X	X	0	0	1	0	0 1 1 0	sub
1	X	X	X	0	1	0	0	0 0 0 0	and
1	X	X	X	0	1	0	1	0 0 0 1	or
1	X	X	X	1	0	1	0	0 1 1 1	sll
1	1	X	X	0	1	1	1	1 1 0 0	nor

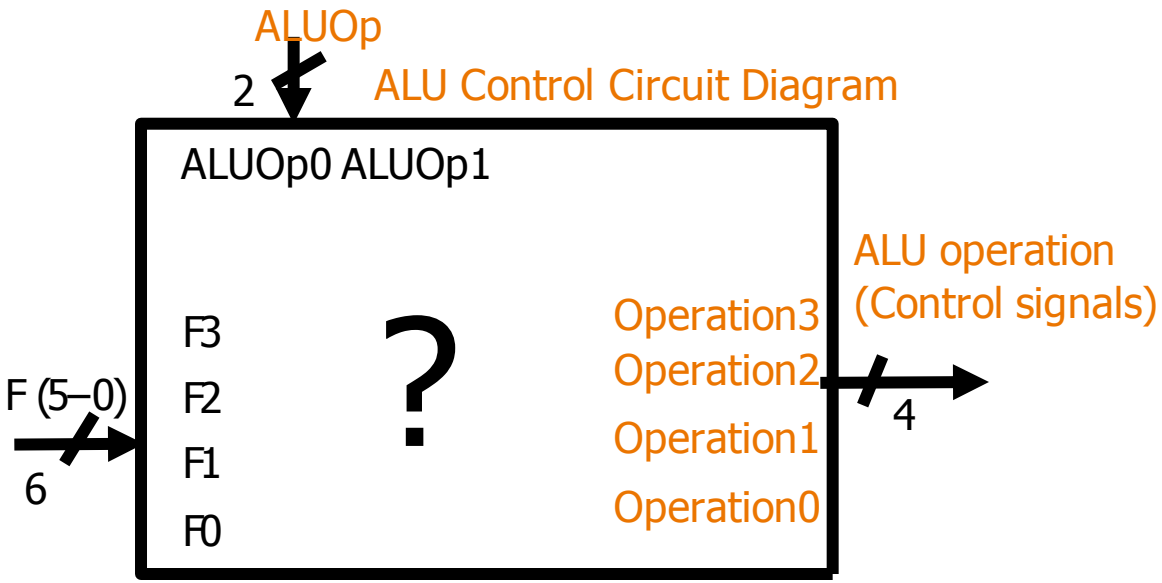
ALUOp		Funct field						Operation	Instruction
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	O3 O2 O1 O0	
1	X	X	X	0	1	0	1	0 0 0 1	or
1	X	X	X	1	0	1	0	0 1 1 1	sll

- O0 = ALUOp1

ALUOp		Funct field					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
1	X	X	X	X	X	X	X

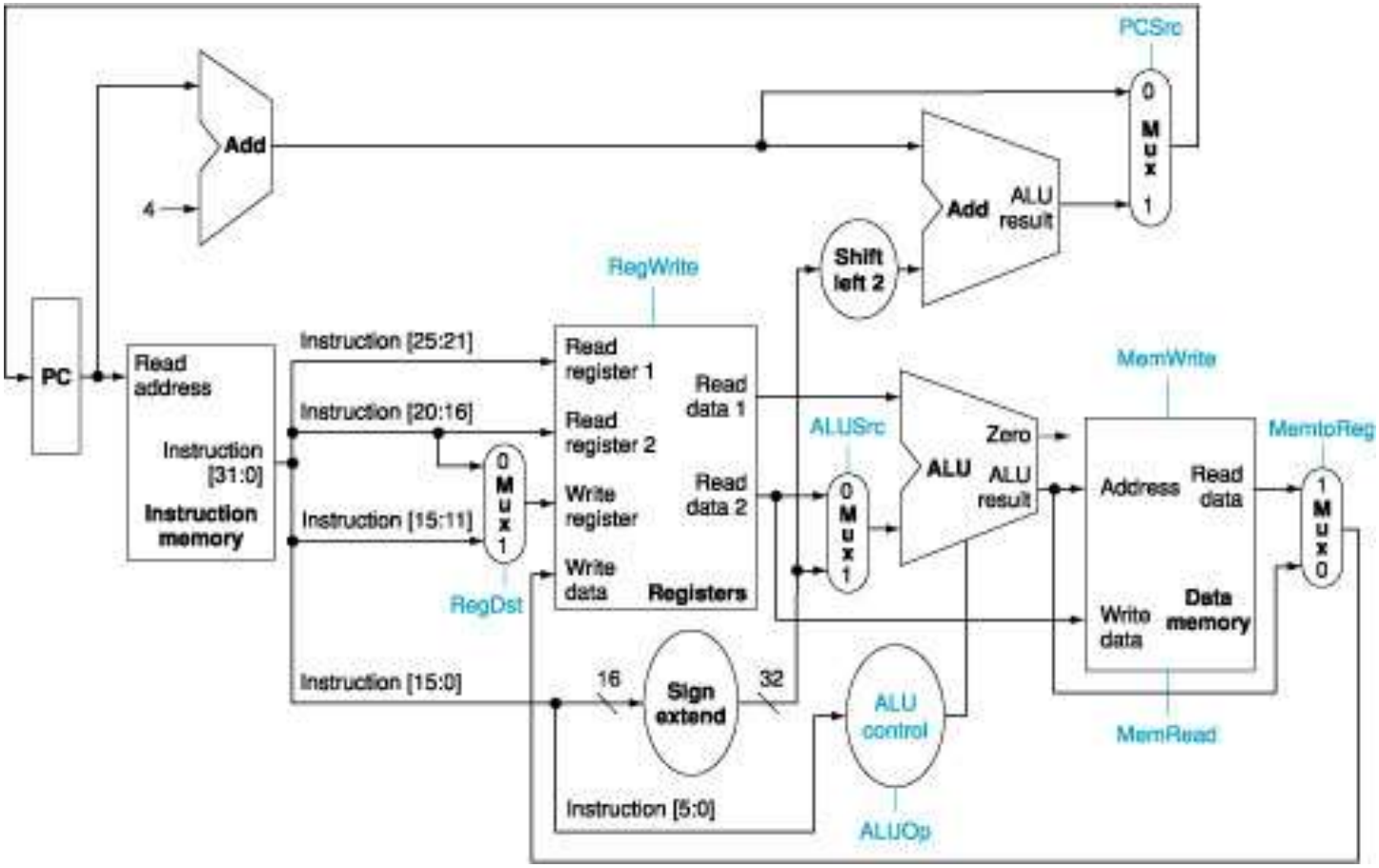
ALU Control: Logic Circuit

- Based on "funct" code & ALUOp
- Details in Appendix C2 (Have some errors/incomplete)
  - O3 = ALUOp1.ALUOp0.F3'.F2.F1.F0
  - O2 = ALUOp1.ALUOp0
  - O1 = O1 = ALUOp1'.ALUOp0' + ALOP1.ALUOp0
  - O0 = ALUOp1
- Exercise:
  - Draw the correct logic diagram



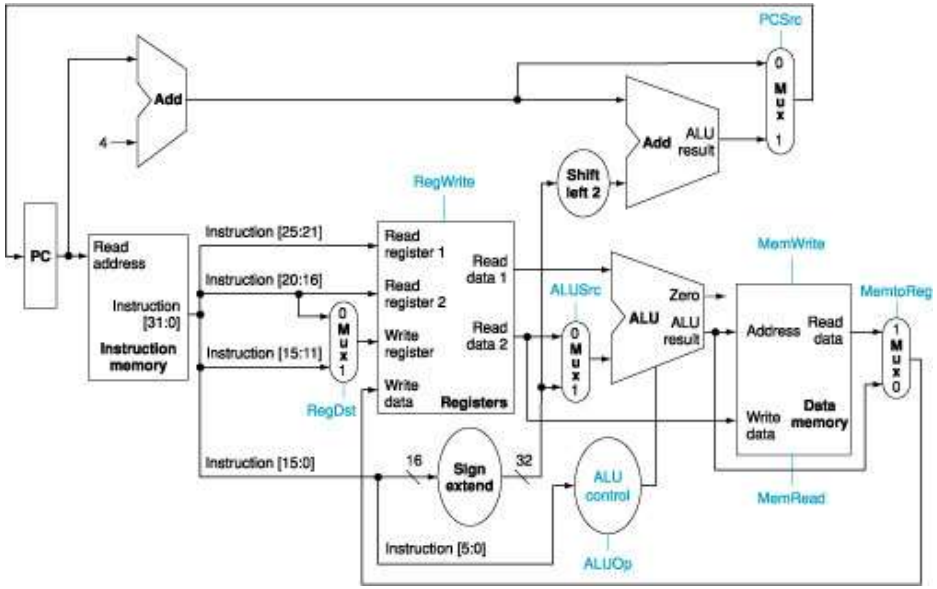
ALU Control

- The datapath with necessary multiplexors and all control lines



ALU Control

- 9 control lines
  - 2 for ALUOp
    - 00 for load/store
    - 10 for R-Format
    - 01 for other operations
  - RegDst
  - RegWrite
  - ALUSrc
  - PCSrc
  - MemRead
  - MemWrite
  - MemtoReg
- All signals except PCSrc are set from the opcode field
- PCSrc is set when the code is for a branch instruction and Zero signal is set
  - To generate PCSrc signal, we use an AND gate with the "zero" signal from ALU
- See fig. 5.16, p. 306 for the effect of the control signals when they are asserted or de-asserted respectively



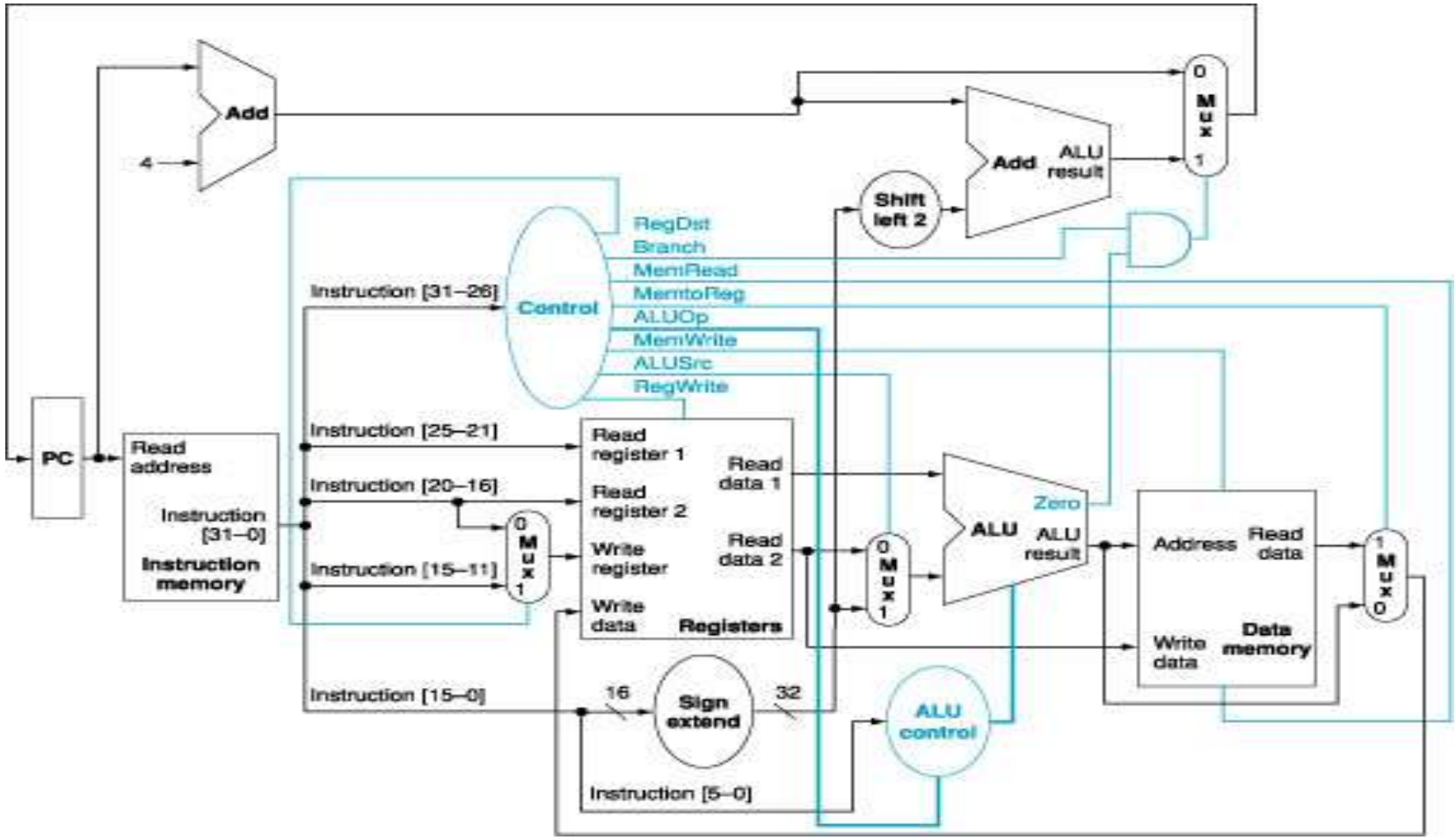
ALU Control

- Effect of the control signals when they are asserted or de-asserted respectively
- See fig. 5.16, p. 306

Signal name	If de-asserted (0)	If asserted (1)
RegDst	Destination register <= rt field (20-16)	Destination register <= rd field (15-11)
RegWrite	None	Write data input => Write register
ALUSrc	Reg Data2 => 2nd ALU operand	Sign-extnd 16-bits of instruction => 2nd ALU operand
PCSrc	PC <= PC + 4 (from adder)	PC <= Branch target (from adder)
MemRead	None	Mem[Address] => to Read data output
MemWrite	None	Mem[Address] <= value on Write data input
MemtoReg	Value to Write data input comes from AL	Value to Write data input comes from data memory

ALU Control

- The simple datapath with the control unit

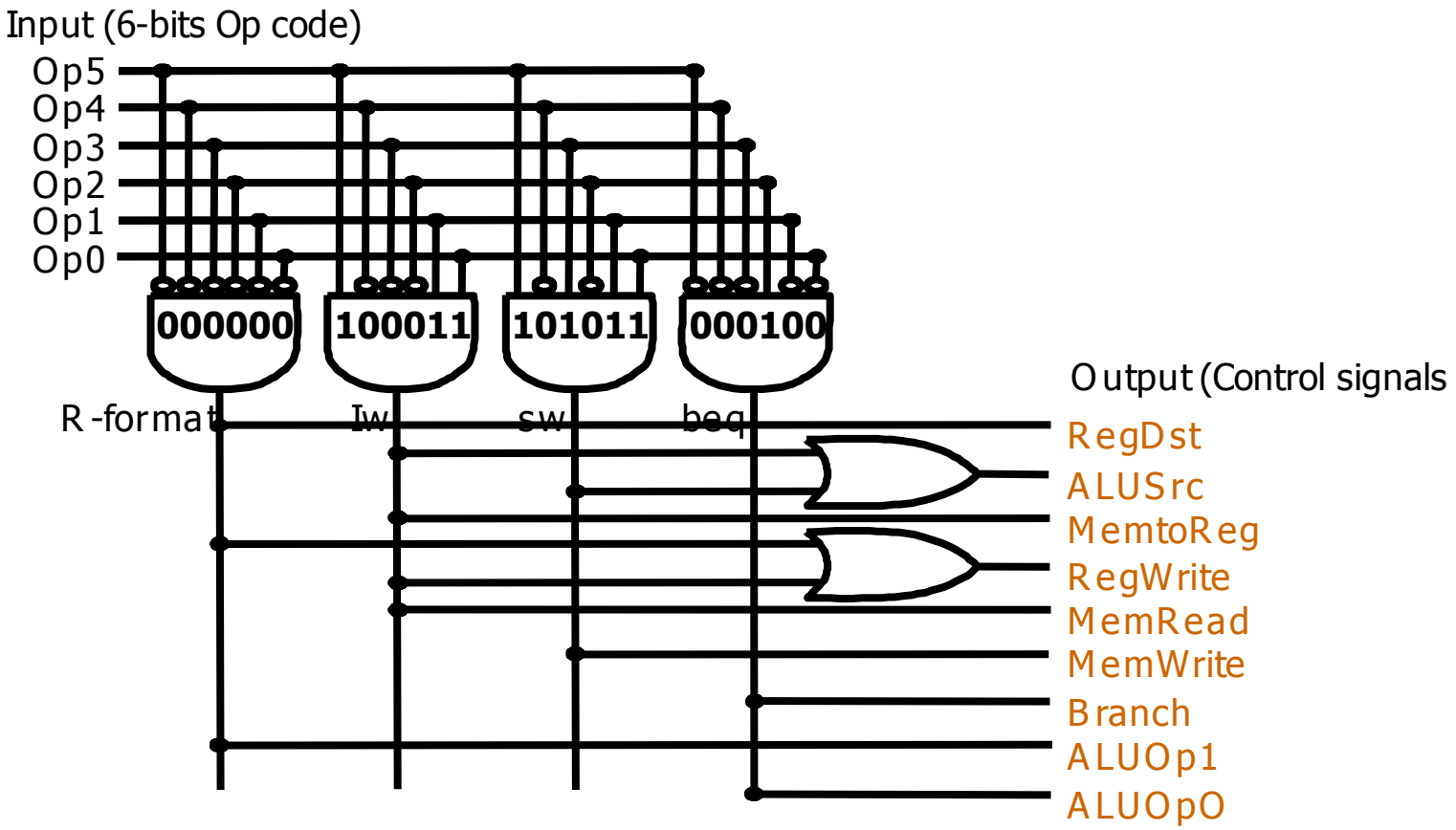


Main Control Unit

- Input:
  - 6-bit opcode field from the instruction
- Output:
  - 9 signals
    - 3 signals for multiplexors
      - RegDest
      - MemtoReg
      - ALUSrc
    - 3 signals to control reads & writes in register file & data memory
      - RegWrite
      - MemRead
      - MemWrite
    - 1 signal to control whether to Branch or not
      - Branch
    - 2 signals for ALU
      - ALUOp

Main Control Unit

- Mapping the Functions to Gates
- See Fig C.2.5, Appendix C, p. c-8



Main Control Unit

- Control lines needed by each instruction
  - R-Format (***add, sub, AND, OR, & slt***)
    - Source register : ***rs & rt***
    - Destination register: ***rd***
    - Writes a register (RegWrite = 1) but neither reads nor writes data memory
    - ALUOp for R-Type format = 10 indicating that ALU control should be generated from function field
  - When Branch:
    - Control signal = 0, PC<= PC +4;
    - Otherwise it is replaced by Branch target
  - For lw
    - MemRead = 1,
  - for sw
    - MemWrite = 1
- See Fig 5.18, p. 308

Instruction	RegDest	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Banch	ALUOp1	ALUOp0
R-Format	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

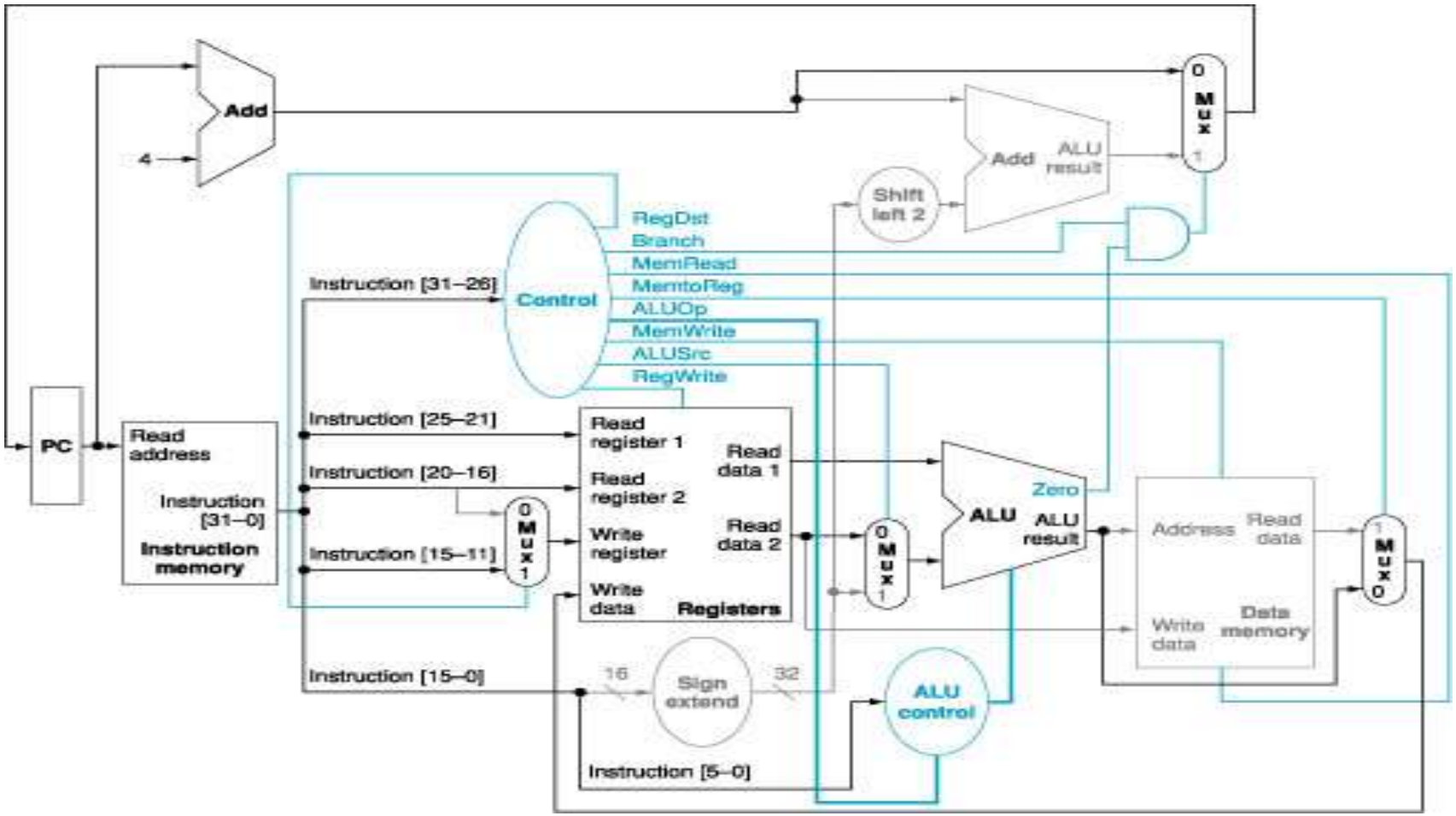
Datapath Operation for R-Format ***(add)***

Add $t1, $t2, $t3

- Step1:
  - Fetch instruction from instruction memory
  - Increment program counter
- Step2:
  - Decode result is ***Add*** operation
  - Read two registers ***$t2 & $t3*** from register file
- Step 3:
  - ALU operates on data, using ***"funct"*** field code to generate the ALU function
- Step4:
  - Idle
- Step5:
  - Write result of step 3 from ALU into register ***$t1***
- Note:
  - Step1 is the same for all instructions

Datapath Operation for R-Format ***(add)***

Add $t1, $t2, $t3 (Fig. 5.19, p. 309)



Multicycle Implementation

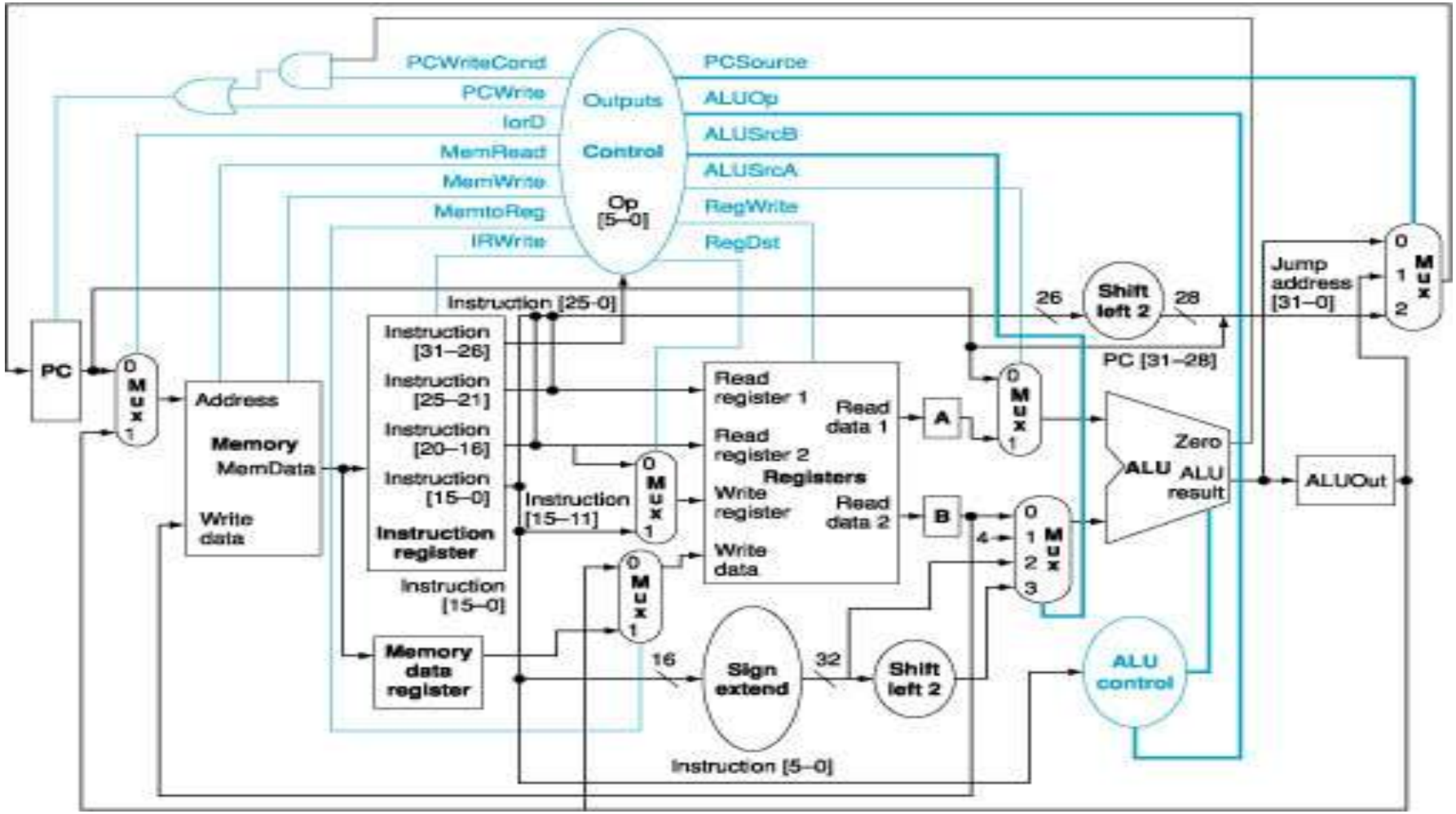
- Each instruction is broken into a series of steps corresponding to the functional unit operations needed
- Each step executes in one clock cycle
- Advantages:
  - Allows functional units to be used more than once per instruction
  - Help reduce HW cost

Multicycle Implementation

- Major difference from single cycle datapath:
  - Single memory unit used for both instruction and data
  - Single ALU instead of three
  - One or more registers are added after every major functional unit to hold the output of that unit until the value is used in subsequent clock cycle
- Added registers
  - **Instruction register (IR)**
    - Saves output from memory for instruction read
  - **Memory data register (MDR)**
    - Saves output from memory for data read
  - **A and B registers**
    - Hold the register operands read from register file
  - **ALUOut register**
    - Holds the output from ALU

Multicycle Implementation

- Multicycle datapath supporting jump and branch instructions



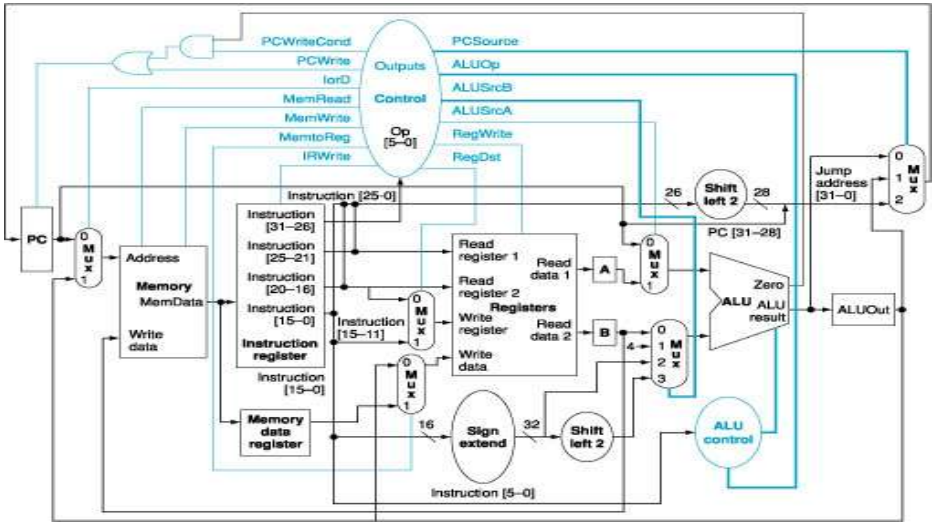
Breaking Instructions

- Each step will contain at most
  - One ALU operation
  - One register file access
  - One memory access
- At the end of each cycle, data values needed for subsequent cycle must be stored into a register
- Since the design is edge triggered, current value of a register can be read during that clock cycle period until the next edge
- All operations listed in one step occur in parallel
- Successive steps occur in a series of different clock cycles
- Each MIPS instruction will need 3 to 5 steps

Breaking Instructions

- Steps:
  1. Instruction fetch
  2. Instruction decode & register fetch
  3. Execution, memory address computation, or branch completion
  4. Memory access or R-type instruction completion
  5. Memory read completion

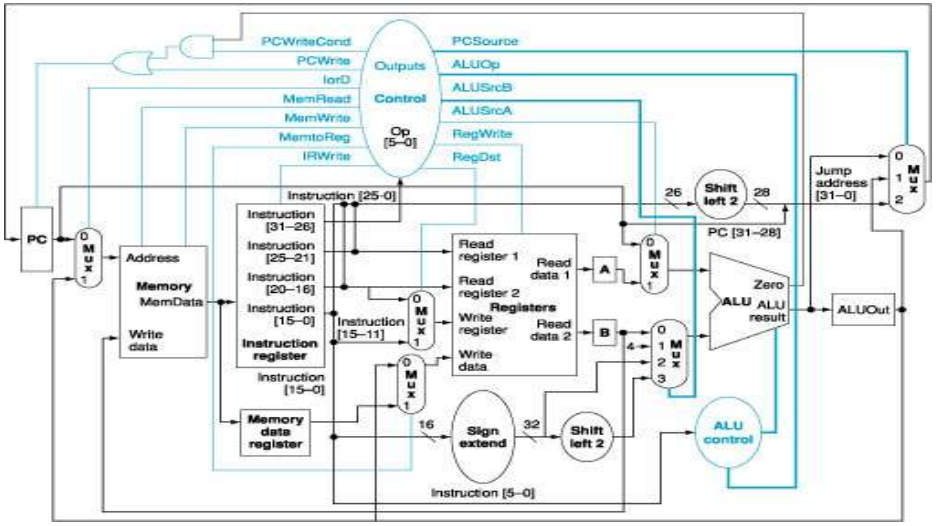
Breaking Instructions



1. Instruction fetch
   - Action taken (in Verilog terminology)
     - IR <= Memory[PC]
     - PC <= PC + 4
   - Control signals required
     - MemRead asserted
     - IRWrite asserted
     - IorD set to 0 (to select PC)
     - ALUSrcA set to 0
     - ALUSrcB set to 01 (to send 4 to ALU)
     - ALUOp set to 00 (choose addition)
     - PCSource set to 00
     - PCWrite asserted

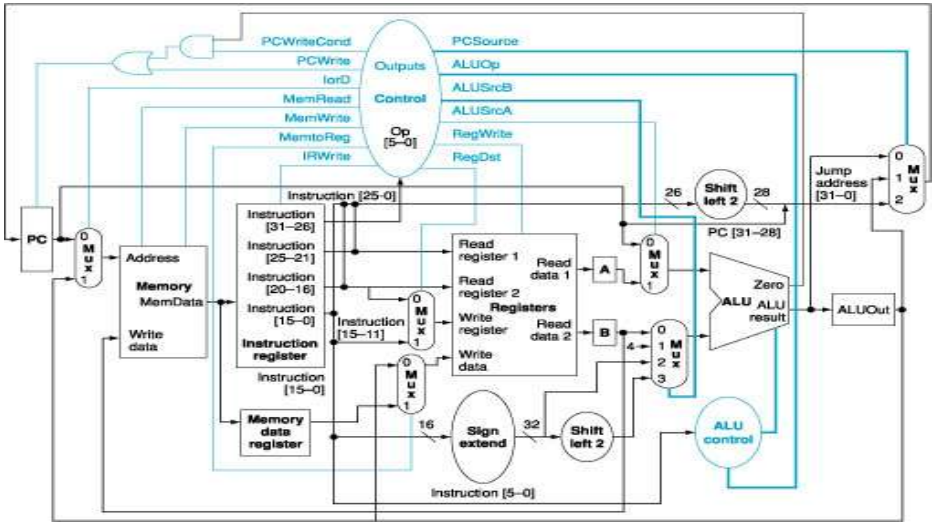
Breaking Instructions

2. Instruction decode & register fetch
   - Action taken
     - A <= Reg[IR[25:21]];
     - (R-Format) B <= Reg[IR[20:16]];
     - (for branch instruction) ALUOut <= PC + (sign-extend (IR[15:0]) << 2);
   - Control signals required
     - ALUSrcA set to 0
     - ALUSrcB set to 11 (sign extend and shift offset sent to ALU)
     - ALUOp set to 00 (choose addition)



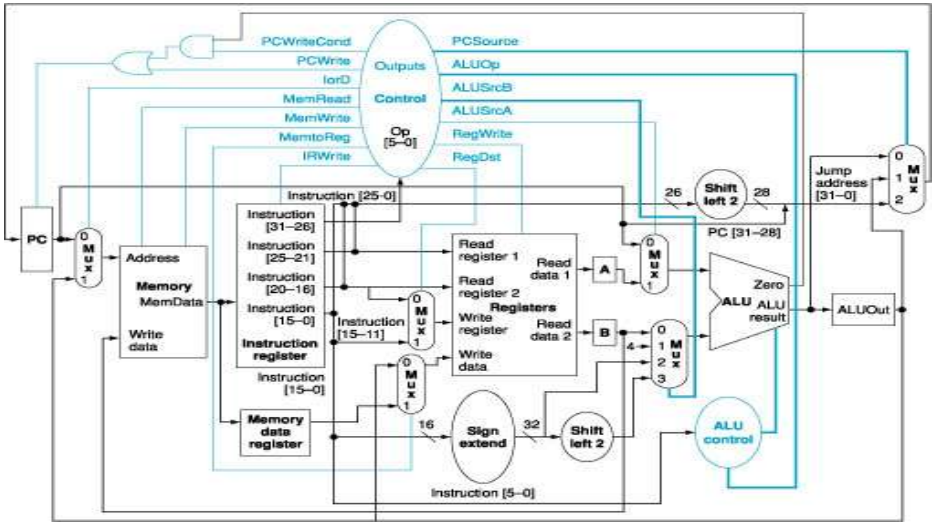
Breaking Instructions

3. Execution, memory address computation, or branch completion
   - Determined by the instruction class
     - Memory reference instruction
       - Action taken
         - ALUOut <= A + sign-extend (IR[15:0]);
       - Control signals required
         - ALUSrcA set to 1
         - ALUSrcB set to 10 (sign extend and shift offset sent to ALU)
         - ALUOp set to 00 (choose addition)
     - Arithmetic-logical
       - Action taken
         - ALUOut <= A op B;
       - Control signals required
         - ALUSrcA set to 1
         - ALUSrcB set to 00
         - ALUOp set to 10(determined by funct field)



Breaking Instructions

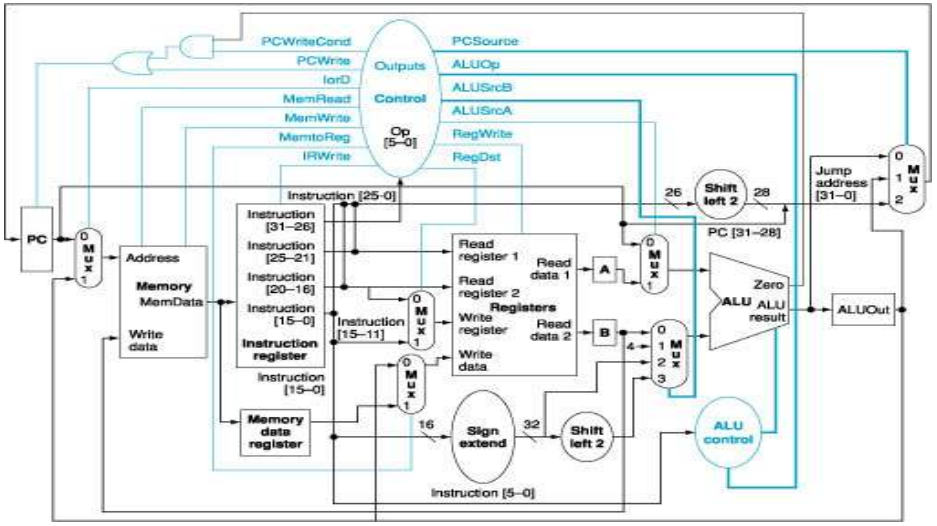
3. Execution, memory address computation, or branch completion
   - Determined by the instruction class
     - Branch
       - Action taken
         - if (A == B) PC <= ALUOut;
       - Control signals required
         - ALUSrcA set to 1
         - ALUSrcB set to 00
         - ALUOp set to 01 (for subtract)
         - PCWrite asserted
         - PCSource seet to 01
     - Jump
       - Action taken
         - PC <= {PC [31:28], IR[25:0]], 2'b00)};
       - Control signals required
         - PCWrite asserted
         - PCSource seet to direct the jump address to the PC



Breaking Instructions

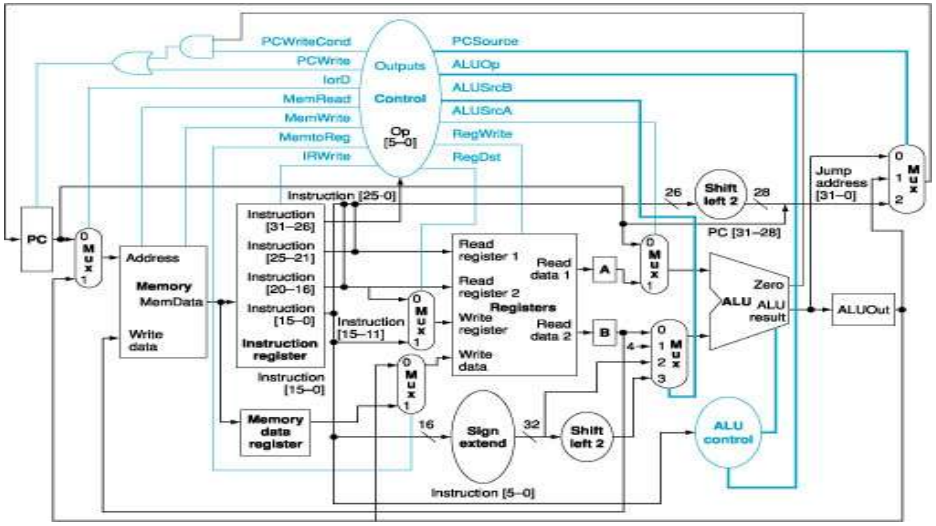
4. Memory access or R-type instruction completion
   - Determined by the instruction class
     - Memory reference (load or store)
       - Action taken
         - MDR <= Memory[ALUOut];

         or
         - Memory [ALUOut] <= B;
       - Control signals required
         - MemRead asserted (for load)
         - MemWrite asserted (for store)
         - IorD set to 1
     - Arithmetic-logical (R-type)
       - Action taken
         - Reg [IR [15:11]] <= ALUOut;
       - Control signals required
         - MemtoReg set to 0
         - RegWrite asserted



Breaking Instructions

5. Memory read completion
   - Load
     - Action taken
       - Reg [IR[20:16]] <= MDR;
     - Control signals required
       - MemtoReg set to 1 (to write result from memory)
       - RegWrite asserted (to cause a write)
       - RegDst set to 0 (to choose rt bits 20:16)

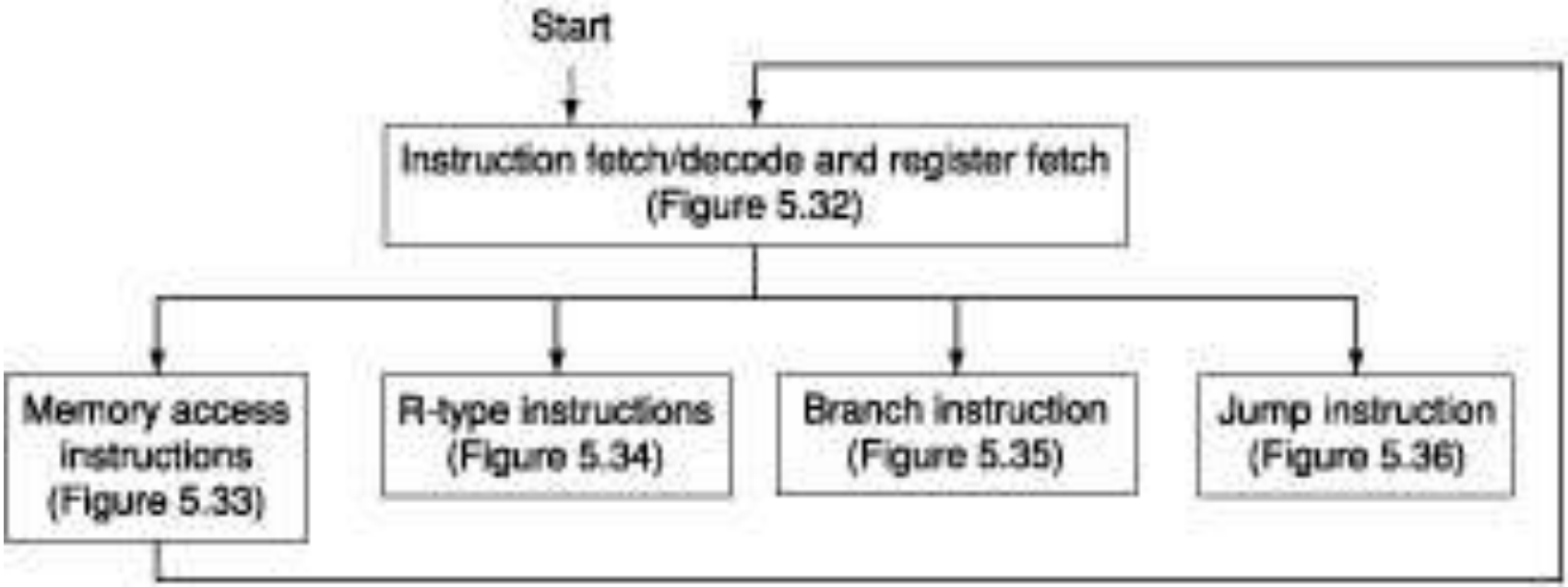


Finite State Machine

- Can be used to specify multi-cycle control
- A sequential logic function consisting of
  - A set of inputs
  - A set of outputs
  - A next state function that maps the current state and the input to a new state
  - An output function that maps the current state and the inputs to a set of asserted outputs

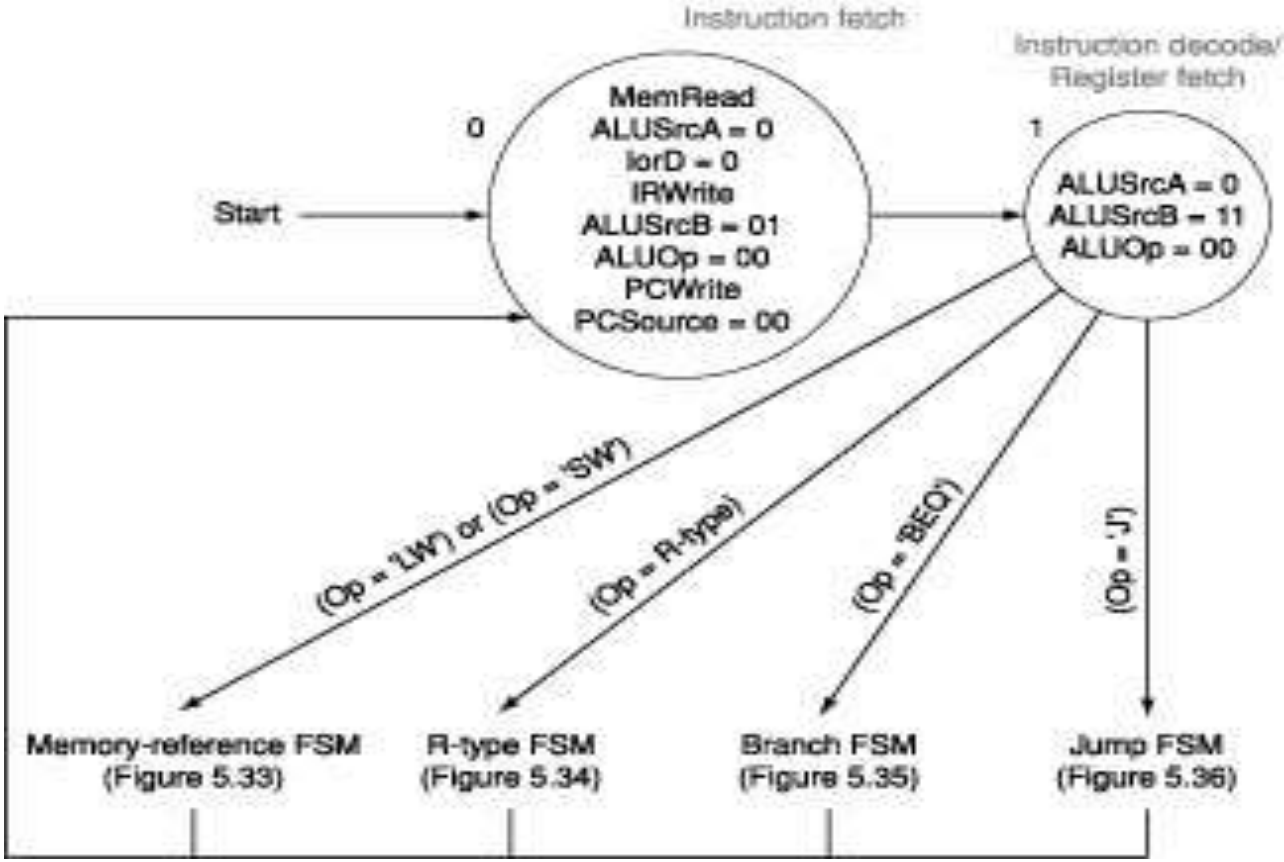
Finite State Machine

- The high-level view of the finite state machine control



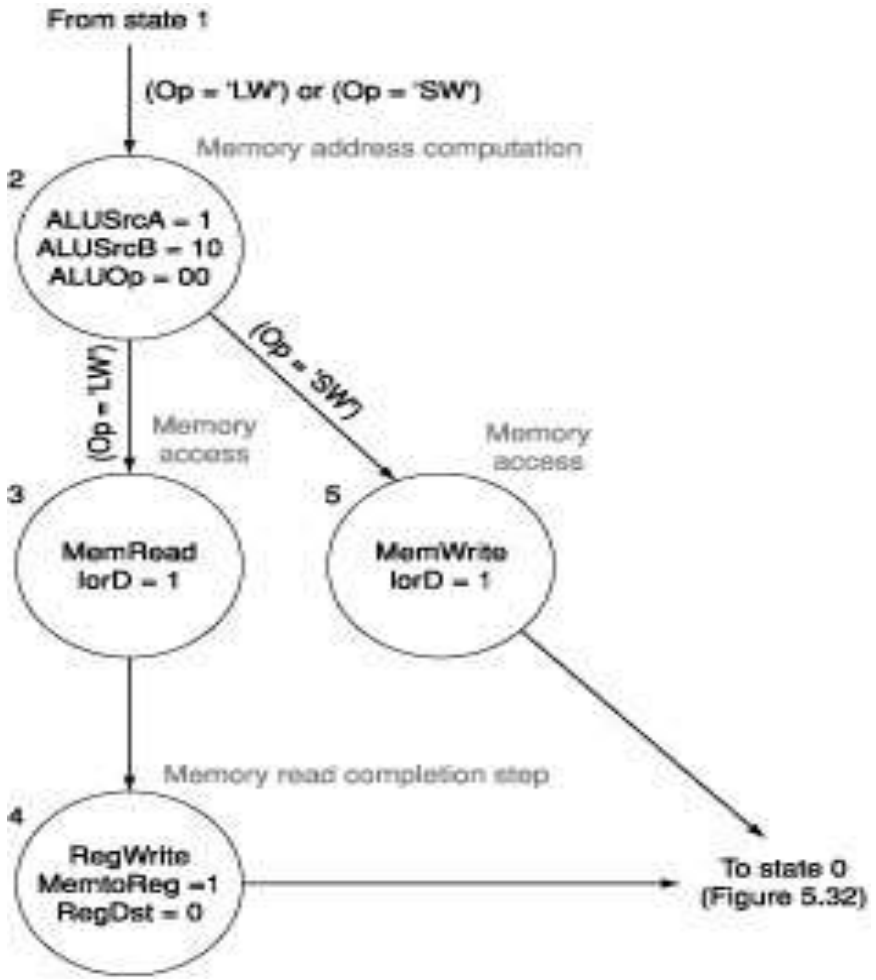
Finite State Machine

- Instruction fetch and decode portion of every instruction is identical



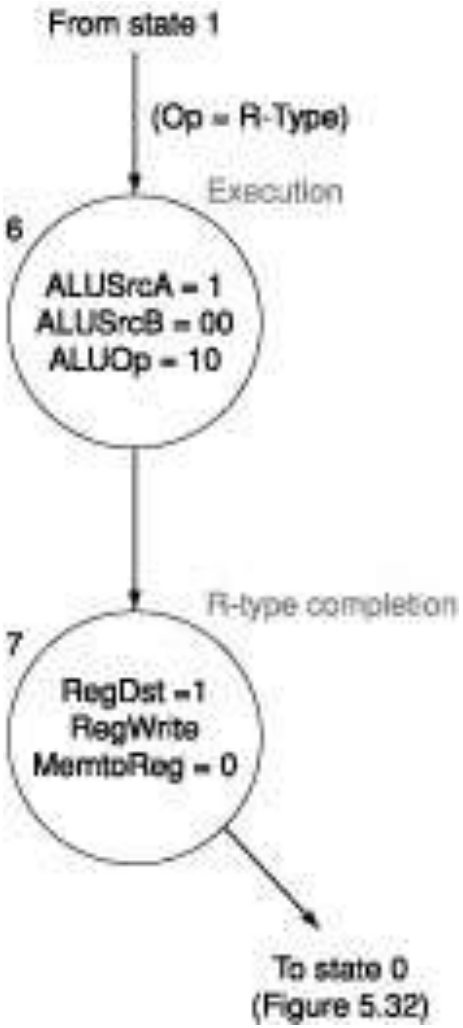
Finite State Machine

- Controlling memory reference instructions



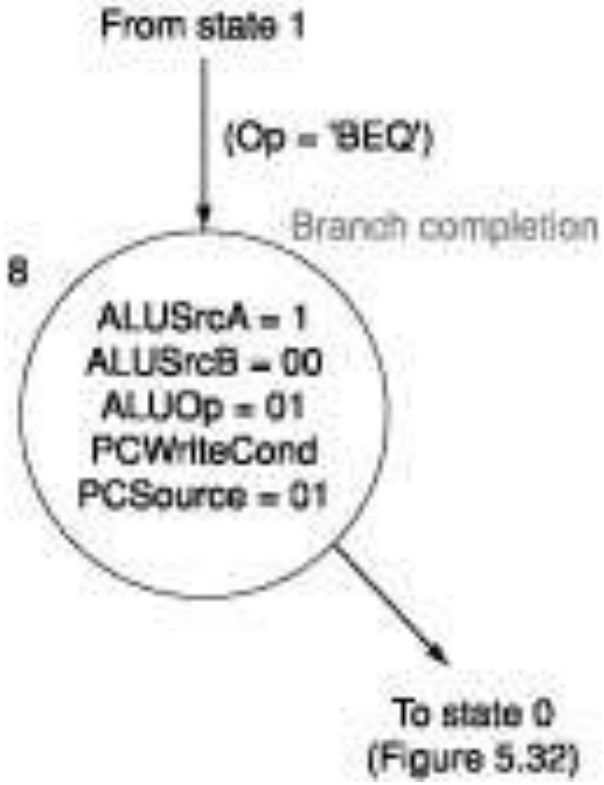
Finite State Machine

- R-type instructions



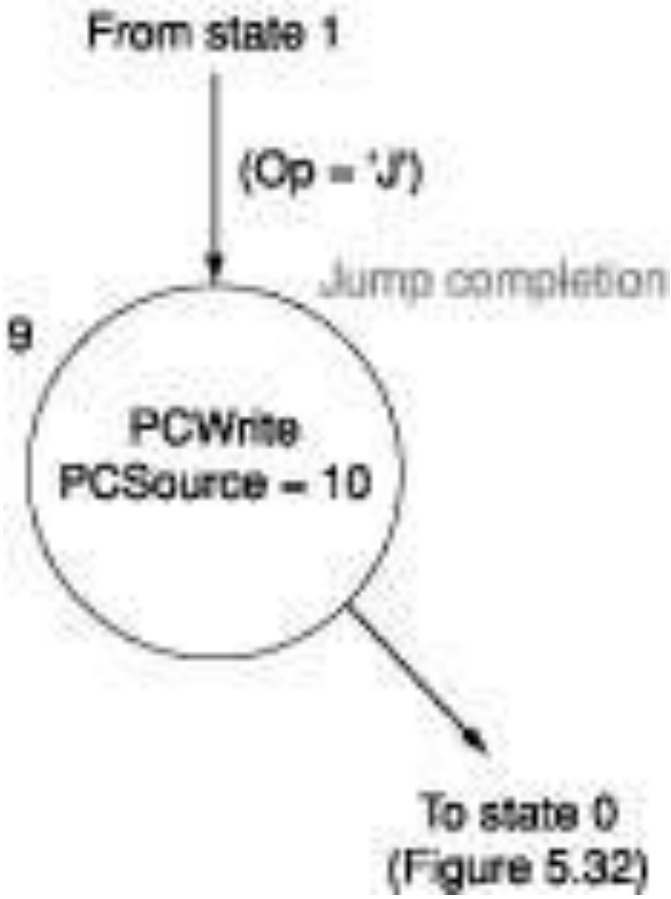
Finite State Machine

- Branch instructions



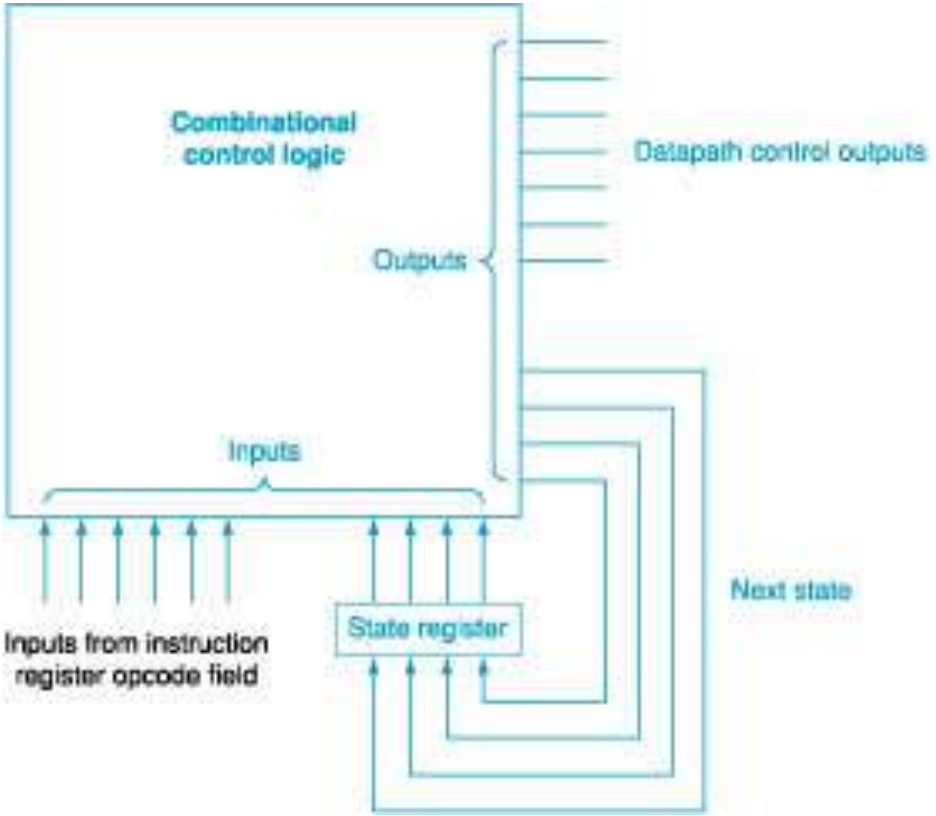
Finite State Machine

- Jump instructions



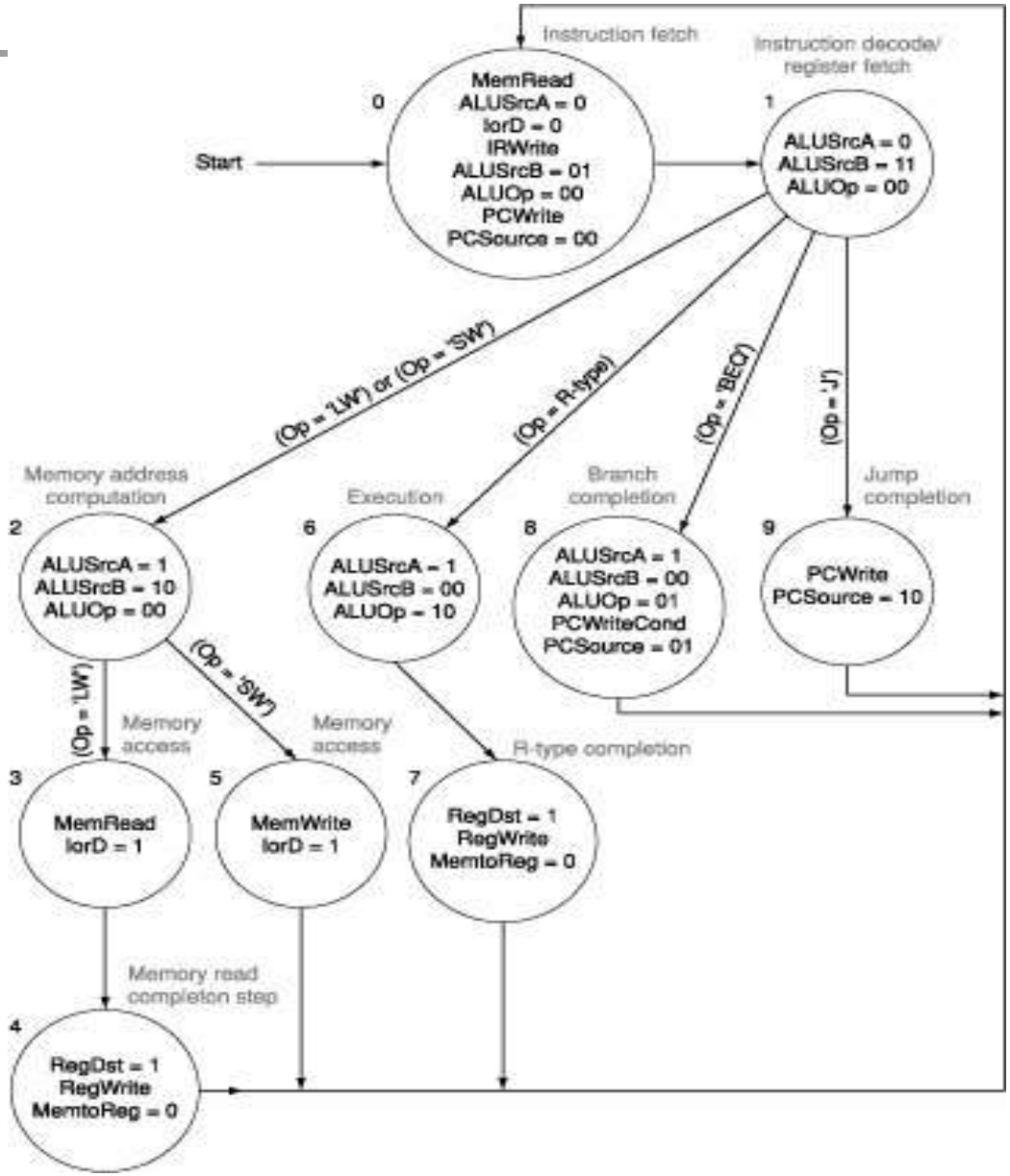
Finite State Machine

- Finite state machine controllers are typically implemented using combinational logic ad a register to hold current state



Finite State Machine

- Complete finite state machine control for the datapath



Review: Actions of 1- Bit Control Signals

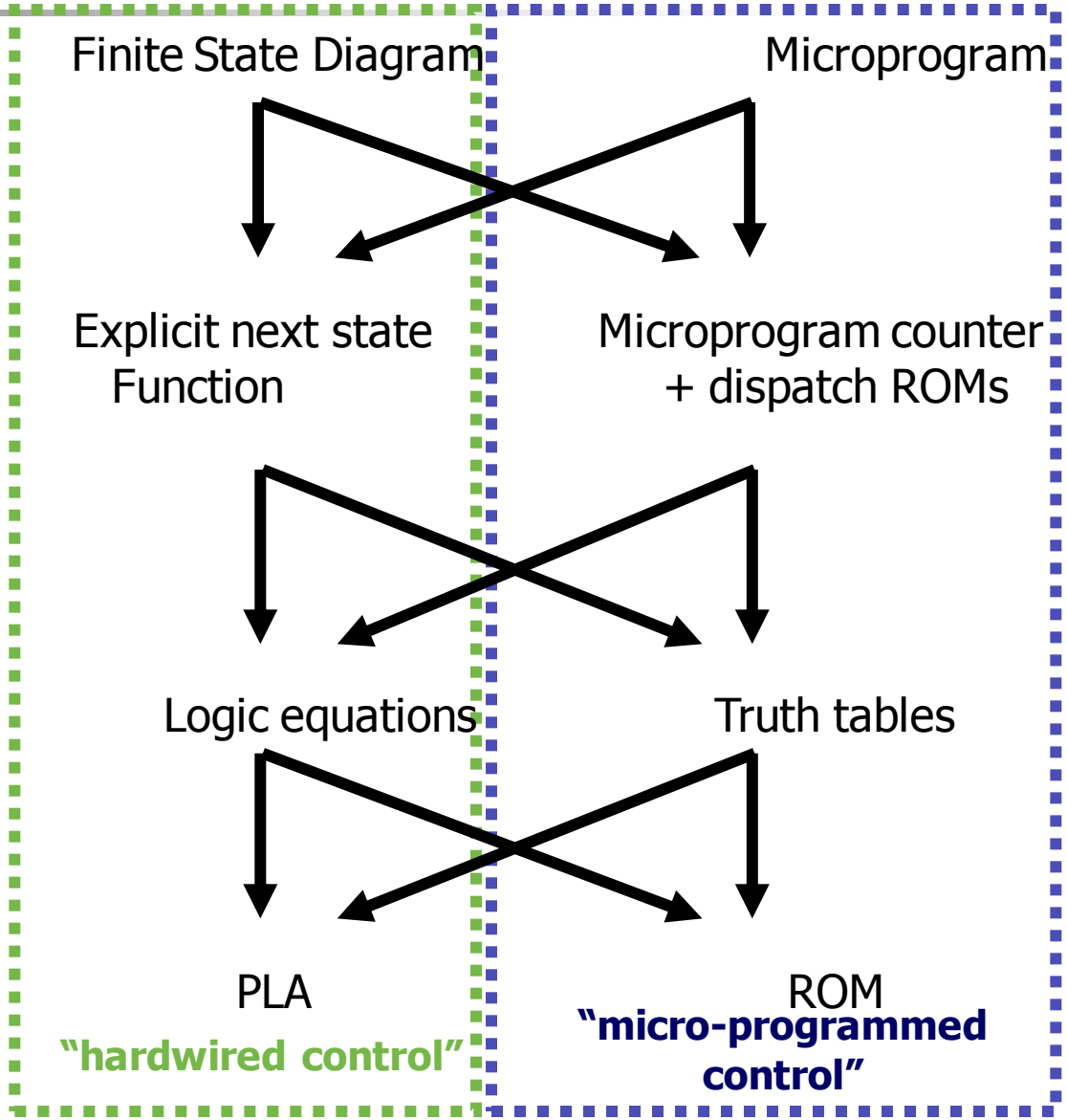
Signal name	Effect when deasserted	Effect when asserted
RegSDt	The register file destination number for the Write register comes from the rt field	The register file destination number for the Write register comes from the rd file
RegWrite	None	The general purpose register selected by the Write register number is written with the value of the Write data input
ALUSrcA	The first ALU operand is the PC	The first ALU operand comes from the A register
MemRead	None	Content of memory at location specified by the Address input is replaced by value on Write data input
MemWrite	None	Memory contents at the location specified by the Address input is replaced by value on Write data input
MemToReg	The value fed to the register file Write data input comes from ALUOut	The value fed to the register file Write data input comes from MDR
IorD	The PC is used to supply the address to the memory unit	ALUOut is used to supply the address to the memory unit
IRWrite	None	The output of the memory is written into IR
PCWrite	None	The PC is written; the source is controlled by PCSource
PCWriteCond	None	The PC is written if the Zero output from the ALU is also active

Review: Actions of 2- Bit Control Signals

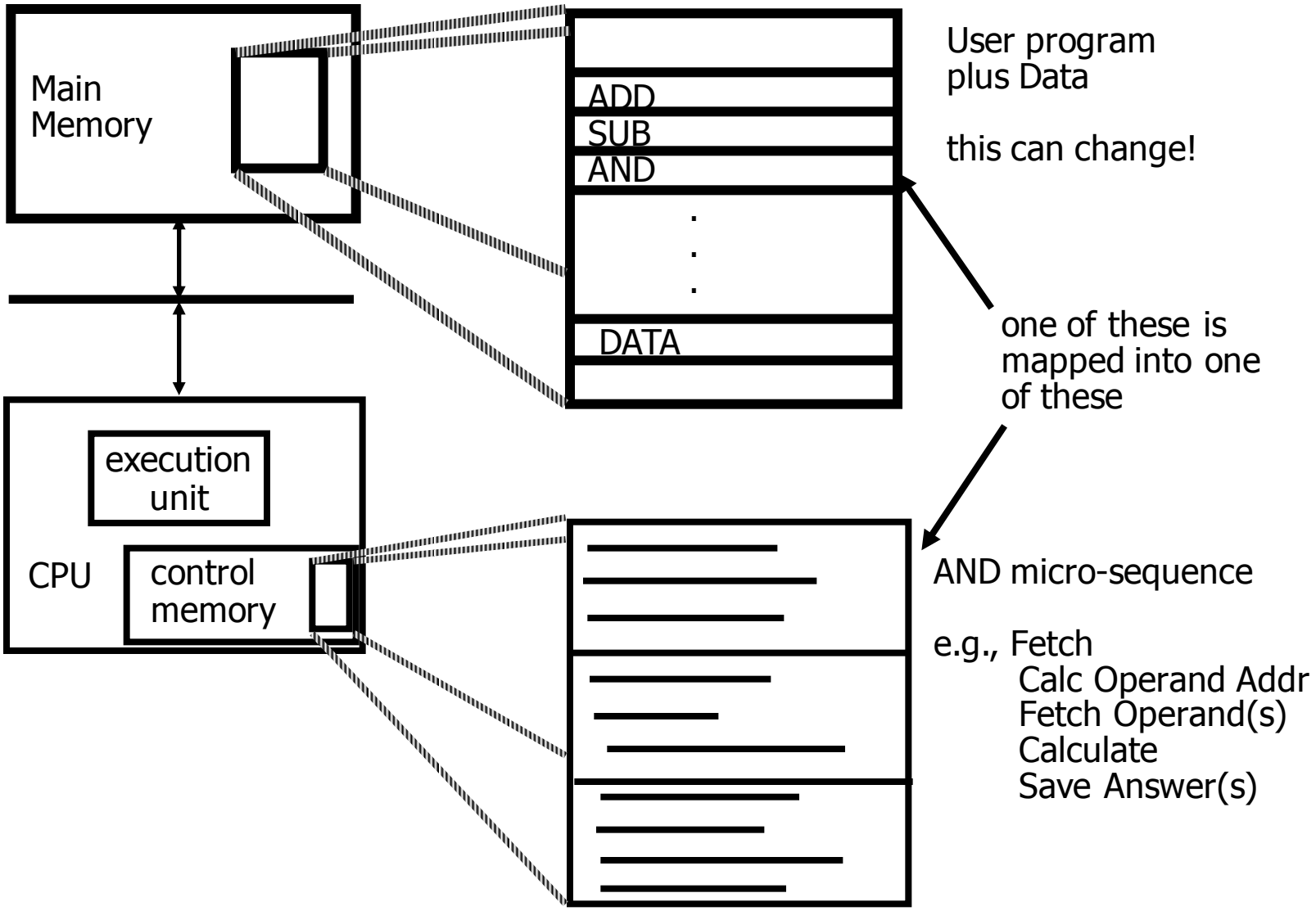
Signal name	Value (Binary)	Effect
ALUOp	00	The ALU performs an add operation
	01	The ALU performs a subtract operation
	10	The funct field of the instruction determines the AKLU operation
ALUSRrcB	00	The second input to the ALU comes from the B register
	01	The second input to the ALU is the constant 4
	10	The second input to the ALU is the sign-extende, lower 16 bits of the IR
	11	The second input to the ALU is the sign-extended, lower 16 bits of the IR shifted left 2 bits
PCSrc	00	Output of the ALU (PC+4) is sent to the PC for writing
	01	The contents of the ALUOut (the branch target address) are sent to the PC for writing
	10	(PC+4[31:28]) is sent to the PC for writing

Alternatives of Designing & Implementing Control

- Initial representation
  - Control may be designed using one of several initial representations
- Sequencing Control
  - The choice of sequence control, and how logic is represented, can then be determined independently
- Logic Representation
- Implementation Technique
  - Control can then be implemented with one of several methods using a structured logic technique



"Macroinstruction" Interpretation



Multiprogramming & Microinstructions

- Can we design the control graphically?
  - Full MIPS instruction set contains > 100 instructions
  - An instruction can take (1-20) clock cycle
  - Control function is complex
  - Control unit might need > 1000's states & 100's different sequences
  - Specifying the control with graphical representation is very cumbersome & difficult to understand
- Solution:
  - Use some ideas of programming to make it easier to understand

Terminology

- Microprogramming:
  - Designing the control as a program that implements machine instructions in terms of simpler microinstructions
- Microinstruction:
  - A set of datapath signals that must be asserted in a given state
  - Usually stored in ROM or PLA (Appendixes B &C)
  - Has memory addresses
- Executing a microinstruction:
  - Asserting the control signals specified by the microinstruction

Terminology

- Inconsistent microinstruction
  - If the instruction requires that a given control signal be set to 2 different values
- Microinstructions are assumed to be executed sequentially, except for branching
- Subroutines in microcode:
  - Idea is also incorporated in microprogramming. Microcode could be reused
  - Microcode control units that are used to implement complex microprograms often provide support for microcode subroutines

Designing a Microinstruction Set

1. Start with list of control signals
2. Group signals together that make sense (vs. random), called "fields"
3. Place fields in some logical order (e.g., ALU operation & ALU operands first and microinstruction sequencing last)
4. Create a symbolic legend for the microinstruction format, showing name of field values and how they set the control signals
   - Use computers to design computers
5. To minimize the width, encode operations that will never be used at the same time

Fields of Microinstruction

- 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 the read or write for the register file, and the source of the value for the write
- Memory:
  - Specify the read or write, and the source for the memory.
  - For a read, specify the destination register
- PCWrite control:
  - Specify the writing for the PC
- Sequencing:
  - Specify how to choose the next microinstruction to be executed

Format of Microinstructions

- Should simplify the representation (easy to write & understand)
- Requirements:
  - A field to control the ALU
  - 3 fields to determine the ALU's source & destination registers
  - Should prevent writing inconsistent microinstructions
    - Could be achieved by specifying non-overlapping set of control signals
    - Signals that that are never asserted simultaneously, can share the same field
- Micro-assemblers check consistency of microcode

Possible Next Microinstruction

1. Sequential:
   - Increment address of current microinstruction
   - Put ***Seq*** in sequencing field
2. Branch:
   - Begin execution of next MIPS instruction (initial microinstruction = state(0))
   - Put ***Fetch*** in sequencing field
3. Dispatch:
   - Next microinstruction based on control unit input
   - Creates a table (Dispatch table) containing the address of target microinstruction
   - Usually there are more than one dispatch table
   - Put ***Dispatch i*** in sequencing field, where ***i*** is the dispatch table number

Format of Microinstructions

Field name	Value	Signals active	Comment
Label	Any string		Used to specify labels to control microcode sequencing. Labels that ends with 1 or 2 are used for dispatching with a jump table that is indexed based on the opcode Other labels are used as direct target in microinstruction sequencing Labels don't generate control signalsdirectly but are used to define contents of dispatch tables and generate control for the Sequencing field
	Add	ALUOp = 00	Cause the ALU to add.
ALU control	Subt	ALUOp = 01	Cause the ALU to subtract; this implements the compare for branches.
	Func code	ALUOp = 10	Use the instruction's function code to determine ALU control.
SRC1	PC	ALUSrcA = 0	Use the PC as the first ALU input.
	A	ALUSrcA = 1	Register A is the first ALU input.
	B	ALUSrcB = 00	Register B is the second ALU input.
SRC2	4	ALUSrcB = 01	Use 4 as the second ALU input.
	Extend	ALUSrcB = 10	Use output of the sign extension unit as the second ALU input.
	Extshft	ALUSrcB = 11	Use the output of the shift-by-two unit as the second ALU input.
	Read		Read two registers using the rs and rt fields of the IR as the register
			numbers and putting the data into registers A and B.
	Write ALU	RegWrite,	Write a register using the rd field of the IR as the register number and
Register		RegDst = 1,	the contents of the ALUOut as the data.
control		MemtoReg = 0	
	Write MDR	RegWrite,	Write a register using the rt field of the IR as the register number and
		RegDst = 0,	the contents of the MDR as the data.
		MemtoReg = 1	
	Read PC	MemRead,	Read memory using the PC as address; write result into IR (and
		lorD = 0	the MDR).
Memory	Read ALU	MemRead,	Read memory using the ALUOut as address; write result into MDR.
		lorD = 1	
	Write ALU	MemWrite,	Write memory using the ALUOut as address, contents of B as the
		lorD = 1	data.
	ALU	PCSource = 00	Write the output of the ALU into the PC.
		PCWrite	
PC write control	ALUOut-cond	PCSource = 01,	If the Zero output of the ALU is active, write the PC with the contents
		PCWriteCond	of the register ALUOut.
	jump address	PCSource = 10,	Write the PC with the jump address from the instruction.
		PCWrite	
	Seq	AddrCtl = 11	Choose the next microinstruction sequentially.
Sequencing	Fetch	AddrCtl = 00	Go to the first microinstruction to begin a new instruction.
	Dispatch 1	AddrCtl = 01	Dispatch using the ROM 1.
	Dispatch 2	AddrCtl = 10	Dispatch using the ROM 2.

Fetch, Decode, & Increment PC

- Microinstructions needed

Microinstruction number	Label	ALU Control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
1	Fetch	Add	PC	4		Read PC	ALU	Seq
2		Add	PC	ExtShft	Read			Dispatch 1

- 1st microinstruction

Fields	Effect
ALU control, SRC1, SRC2	Compute PC + 4. (The value is also written into ALUOut though it will never be read from there.)
Memory	Fetch instruction into IR
PCWrite control	Causes the output of the ALU to be written into the pC
Sequencing	Go to the next microinstruction

- 2nd microinstruction

Fields	Effect
ALU control, SRC1, SRC2	Store PC + sign extension(IR[15:0]) << 2 into ALUOut
Register control	Use the rs & rt fields to read the registers placing the data in A and B
Sequencing	Use dispatch table 1 to choose the next microinstruction address

Memory Reference Instructions

- Microinstructions needed

Microinstruction number	Label	ALU Control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
1	Mem1	Add	A	Extend				Dispatch 2
2-a-1	Lw2					Read ALU		Seq
2-a-2					Write MDR			Fetch
2-b	Sw2					Write ALU		Fetch

Mem1

Fields	Effect
ALU control, SRC1, SRC2	Compute the memory address: Register(rs) + sign-extend (IR[15:0]), writing the result into ALUOut
Sequencing	Use dispatch table 2 to jump to the microinstruction labeled either LW2 or SW2

Lw2-a-1

Fields	Effect
Memory	Read memory using the ALUOut as the address and writing the data into MDR
Sequencing	Go to the next microinstruction

LW2-a-2

Fields	Effect
Register control	Write the contents of the MDR into the register file entry specified by rt
Sequencing	Go to the microinstruction labeled "Fetch"

SW2

Fields	Effect
Memory	Write memory using contents of ALUOut as the address and the contents of B as the value
Sequencing	Go to the microinstruction labeled "Fetch"

R-Type Instructions

- Microinstructions needed

Microinstruction number	Label	ALU Control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
1	RFormat 1	Func code	A	B				Seq
2					Write ALU			Fetch

- 1st Instruction

Fields	Effect
ALU control, SRC1, SRc2	Thte ALU operates on the contents of the A and B registers, using the function field to specify the ALU operation
Sequencing	Go to the next microinstruction

- 2nd Instruction

Fields	Effect
Register control	The value in ALUOut is written into the register file entry specified by the rd field
Sequencing	Go to the microinstruction labeled "Fetch"

Beq Instructions

- Microinstructions needed

Microinstruction number	Label	ALU Control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
1	BEQ1	Subt	A	B			ALUOut-cond	Fetch

Fields	Effect
ALU control,SRC1, SRC2	The ALU subtracts the operands in A and B to generate the Zero outoput
PCWrite control	already in ALUOut, if Zero output of the ALU is true
Sequencing	Go to the microinstruction labeled "Fetch"

Jump Instructions

- Microinstructions needed

Microinstruction number	Label	ALU Control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
1	JUMP1						Jump address	Fetch

Fields	Effect
PCWrite control	Causes the PC to be written using the jump target address
Sequencing	Go to the microinstruction labeled "Fetch"

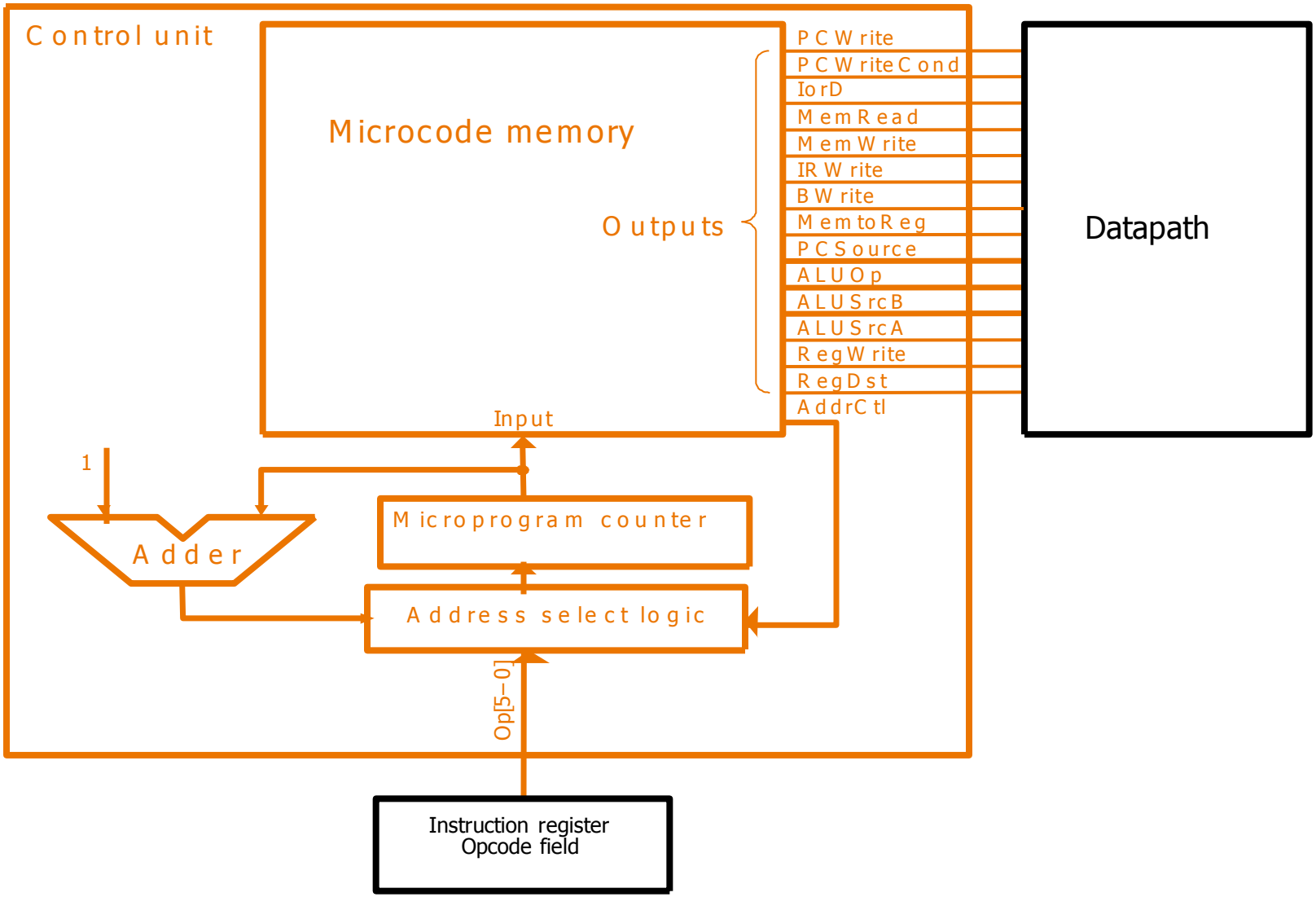
Complete Microprogram

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

Implementing Microprogram

- Micro-programs can be implemented exactly like state machines:
  - PLA
  - ROM
    - Program assembled & stored in microcode storage and is addressed by microprogram counter

Implementing Microprogram



Microprogramming Pros &Cons

- Pros:
  - Ease of design
  - Flexibility
    - Easy to adapt to changes in organization, timing, technology
    - Can make changes late in design cycle, or even in the field
  - Can implement very powerful instruction sets (just more control memory)
  - Generality
    - Can implement multiple instruction sets on same machine.
    - Can tailor instruction set to application.
  - Compatibility
    - Many organizations, same instruction set
- Cons:
  - Costly to implement
  - Slow

Homework Assignment

- What is the meaning of:
  - Control word
  - Micro instruction
  - Micro-program
- Compare between:
  - Hardwired Control
  - Micro-program Control