

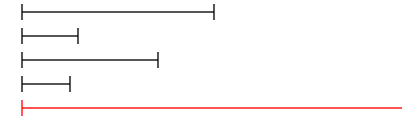
Multi-Cycle CPU: Datapath and Control

CSE 141, S2'06

Jeff Brown

Why a Multiple Clock Cycle CPU?

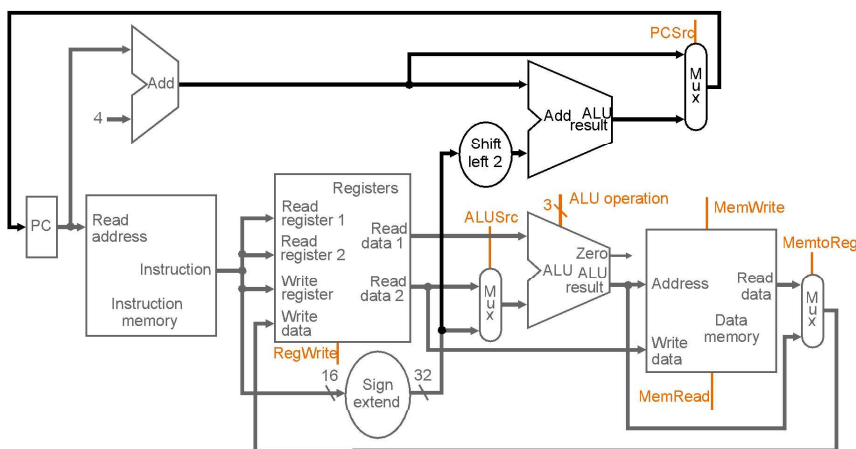
- the problem => single-cycle cpu has a cycle time long enough to complete the **longest** instruction in the machine
- the solution => break up execution into smaller tasks, each task taking a cycle, different instructions requiring different numbers of cycles or tasks
- other advantages => reuse of functional units (e.g., alu, memory)
- $ET = IC * CPI * CT$



CSE 141, S2'06

Jeff Brown

High-level View



CSE 141, S2'06

Jeff Brown

Breaking Execution Into Clock Cycles

- We will have five execution steps (not all instructions use all five)
 - fetch
 - decode & register fetch
 - execute
 - memory access
 - write-back
- We will use Register-Transfer-Language (RTL) to describe these steps

CSE 141, S2'06

Jeff Brown

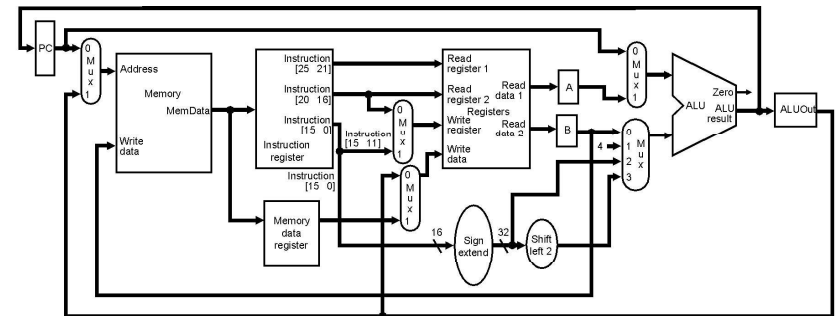
Breaking Execution Into Clock Cycles

- Introduces extra registers when:
 - signal is **computed** in one clock cycle and **used** in another, AND
 - the inputs to the functional block that outputs this signal can **change** before the signal is written into a state element.
- Significantly complicates control. **Why?**
- The goal is to **balance** the amount of work done each cycle.

CSE 141, S2'06

Jeff Brown

Multicycle datapath



1. Fetch

$IR = Mem[PC]$

$PC = PC + 4$

(may not be final value of PC)

2. Instruction Decode and Register Fetch

$A = Reg[IR[25-21]]$

$B = Reg[IR[20-16]]$

$ALUOut = PC + (sign-extend(IR[15-0]) \ll 2)$

- compute target before we know if it will be used (may not be branch, branch may not be taken)
- target is a new state element (temp register)
- everything up to this point must be Instruction-independent, because we still haven't decoded the instruction.
- everything instruction (opcode)-dependent from here on.

CSE 141, S2'06

Jeff Brown

CSE 141, S2'06

Jeff Brown

3. Execution, memory address computation, or branch completion

- Memory reference (load or store)
 $ALUOut = A + \text{sign-extend}(IR[15-0])$
- R-type
 $ALUOut = A \text{ op } B$
- Branch
 $\text{if } (A == B) \quad PC = ALUOut$

At this point, Branch is complete, and we start over; others require more cycles.

4. Memory access or R-type completion

- Memory reference
 - load
 $MDR = Mem[ALUOut]$
 - store
 $Mem[ALUOut] = B$
- R-type
 $Reg[IR[15-11]] = ALUOut$

R-type is complete

5. Memory Write-Back

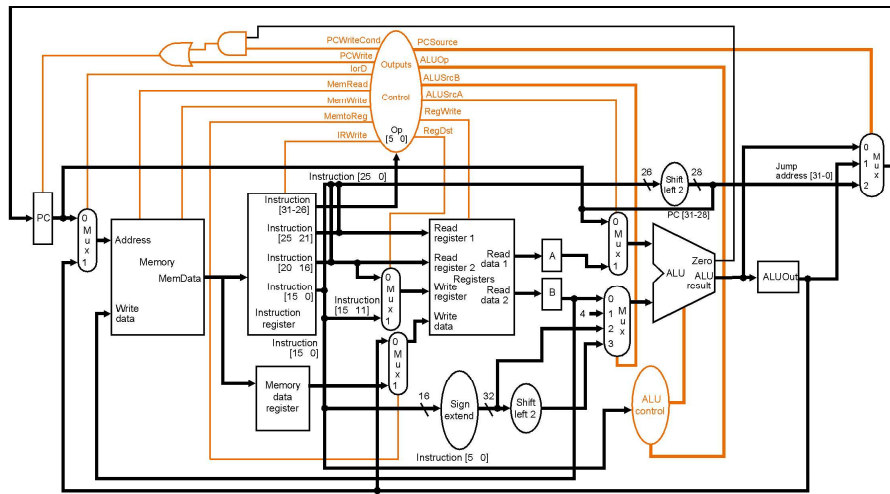
$Reg[IR[20-16]] = MDR$

memory instruction is complete

Summary of execution steps

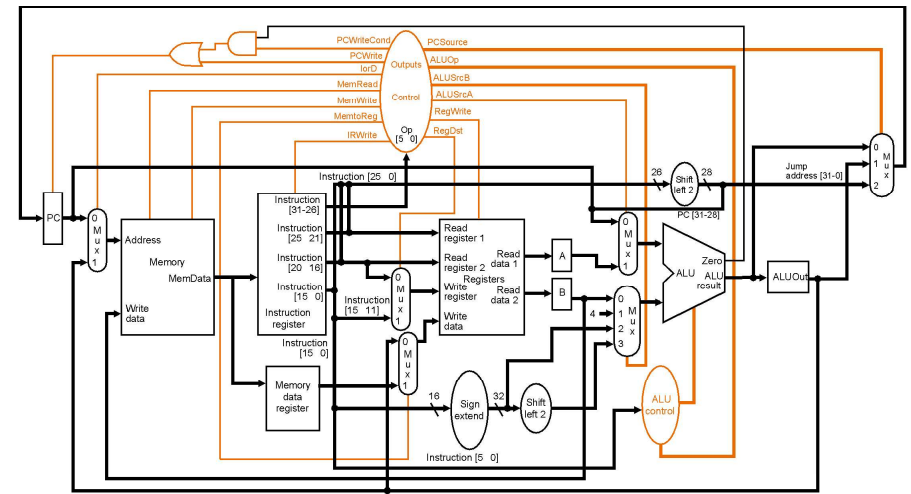
Step	R-type	Memory	Branch
Instruction Fetch	$IR = Mem[PC]$ $PC = PC + 4$		
Instruction Decode/ register fetch	$A = Reg[IR[25-21]]$ $B = Reg[IR[20-16]]$ $ALUOut = PC + (\text{sign-extend}(IR[15-0]) \ll 2)$		
Execution, address computation, branch completion	$ALUOut = A \text{ op } B$	$ALUOut = A + \text{sign-extend}(IR[15-0])$	$\text{if } (A == B) \text{ then } PC = ALUOut$
Memory access or R-type completion	$Reg[IR[15-11]] = ALUOut$	$\text{memory-data} = Mem[ALUOut]$ or $Mem[ALUOut] = B$	
Write-back		$Reg[IR[20-16]] = \text{memory-data}$	

Complete Multicycle Datapath



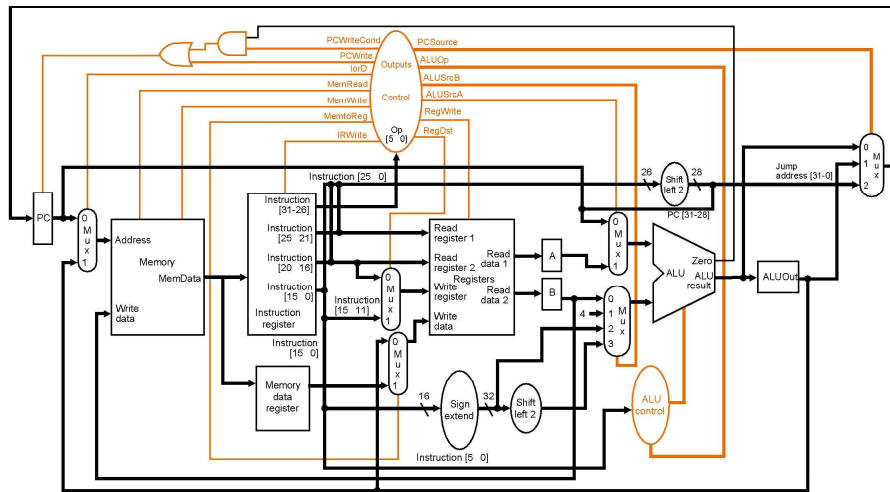
(support for what instruction just got added?)

1. Instruction Fetch



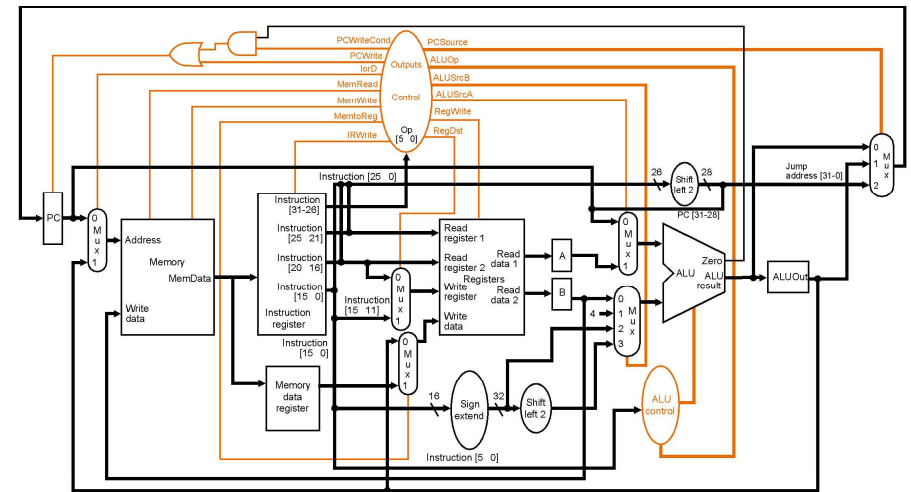
$IR = \text{Memory}[PC]$
 $PC = PC + 4$

2. Instruction Decode and Reg Fetch



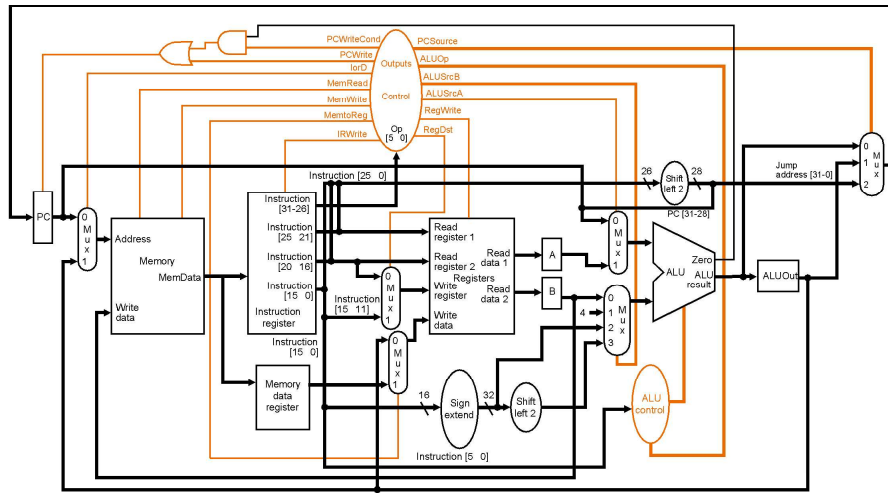
$A = \text{Register}[IR[25:21]]$
 $B = \text{Register}[IR[20:16]]$
 $ALUOut = PC + (\text{sign-extend}(IR[15:0]) \ll 2)$

3. Execution (R-type)



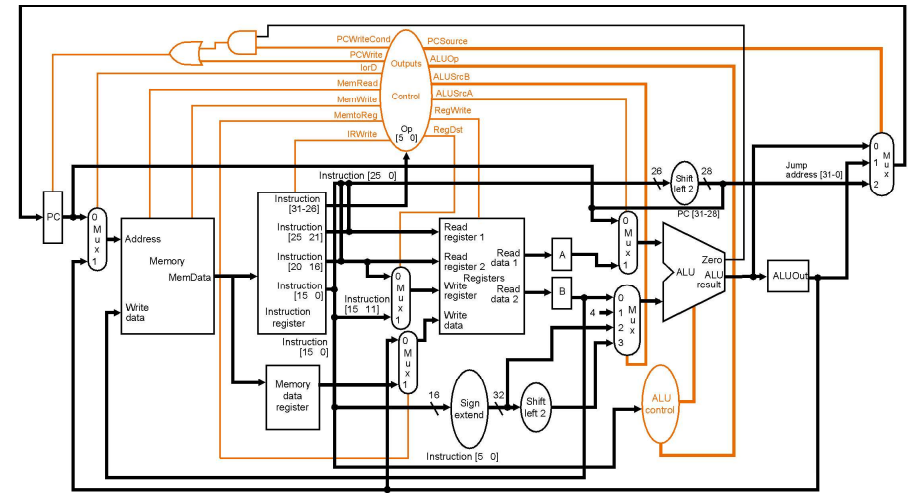
$ALUOut = A \text{ op } B$

4. R-type Completion



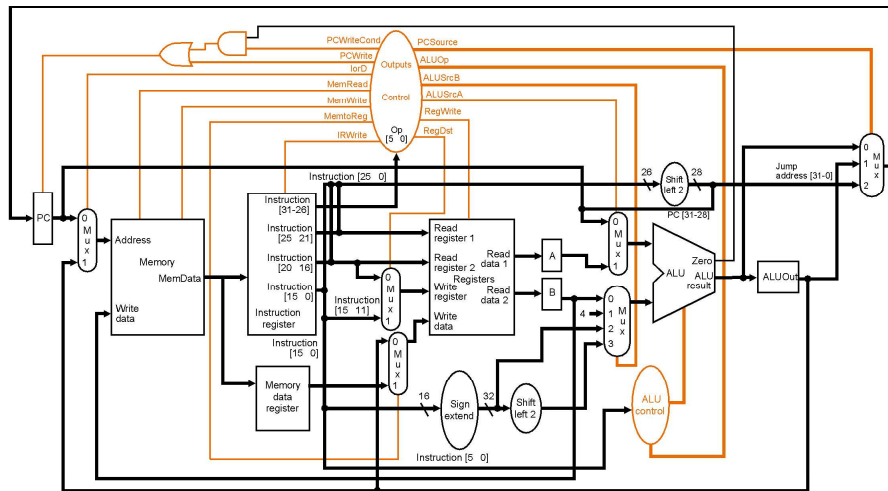
Reg[IR[15-11]] = ALUOut

3. Branch Completion



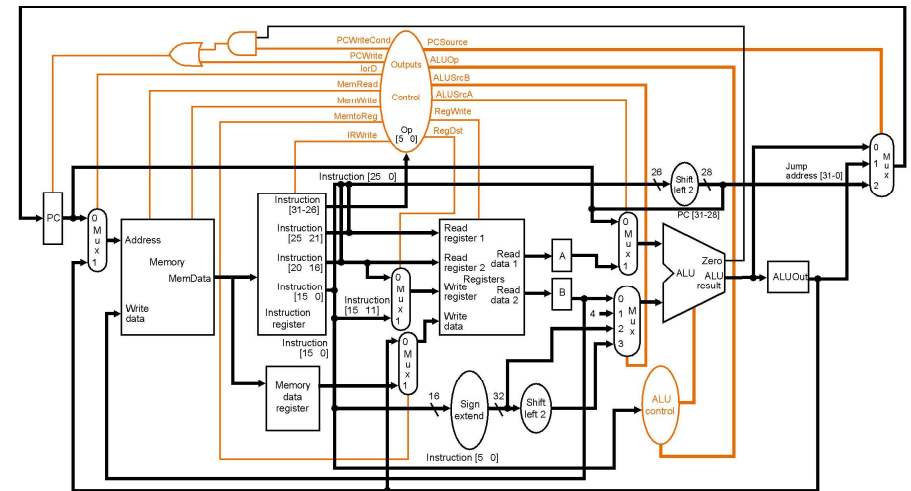
if (A == B) PC = ALUOut

3. Memory Address Computation



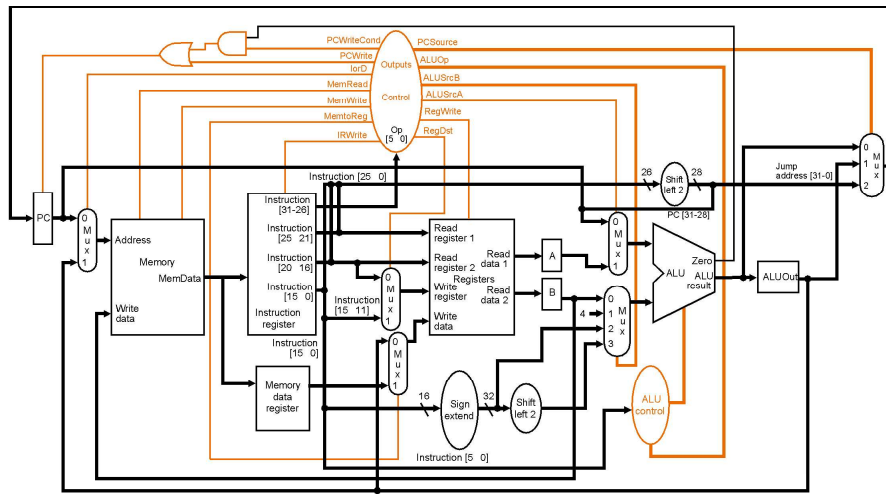
ALUOut = A + sign-extend(IR[15-0])

4. Memory Access



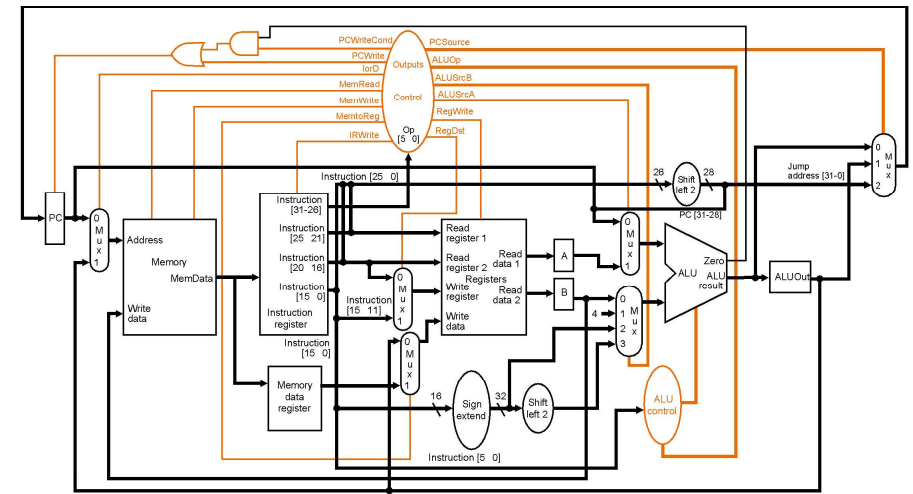
memory-data = Memory[ALUOut], or
Memory[ALUOut] = B

5. Write-back



$\text{Reg}[\text{IR}[20-16]] = \text{memory-data}$

3. JMP Completion

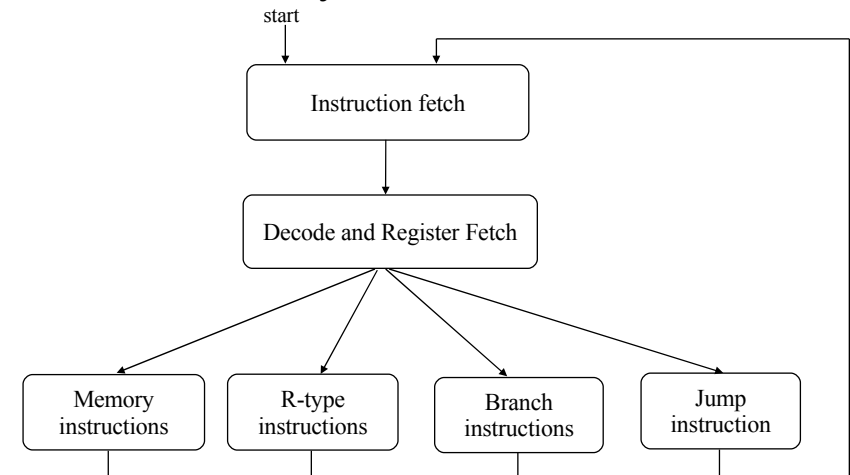


$\text{PC} = \text{PC}[31-28] \mid (\text{IR}[25-0] \ll 2)$

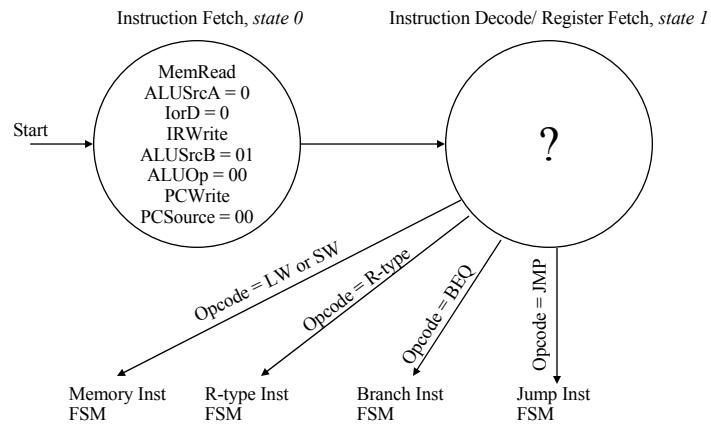
Multicycle Control

- Single-cycle control used **combinational** logic
- Multi-cycle control uses **??**
- FSM defines a succession of states, transitions between states (based on inputs), and outputs (based on state)
- First two states same for every instruction, next state depends on opcode

Multicycle Control FSM



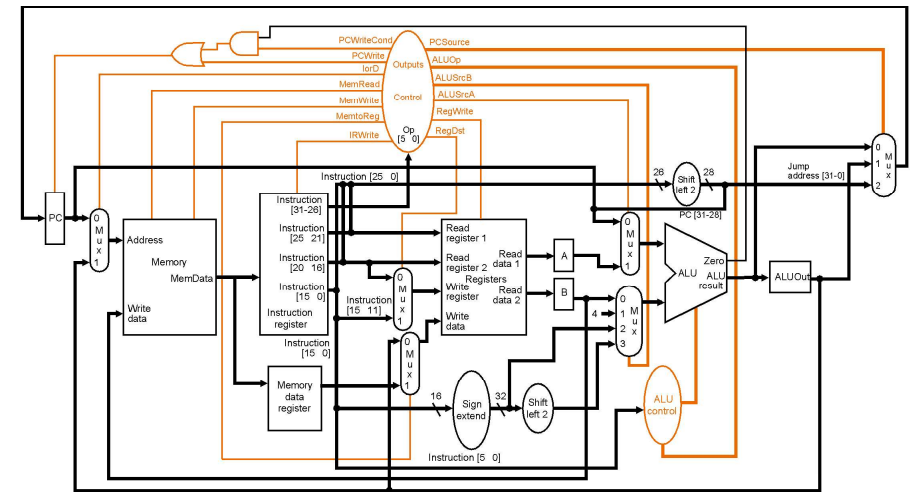
First two states of the FSM



CSE 141, S2'06

Jeff Brown

Instruction Decode and Reg Fetch

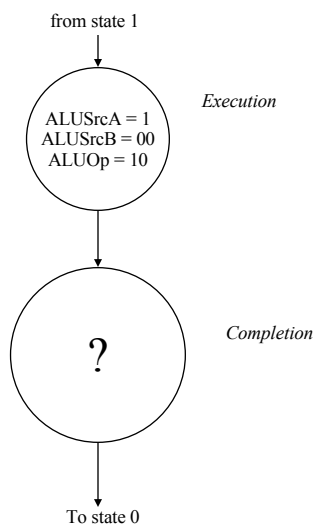


```
A = Register[IR[25-21]]
```

```
B = Register[IR[20-16]]
```

Target = PC + (sign-extend (IR[15-0]) << 2)

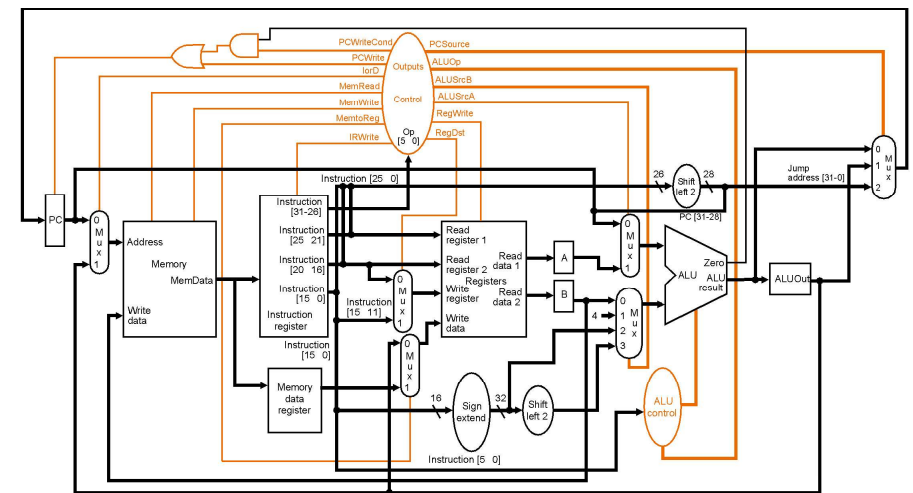
R-type Instructions



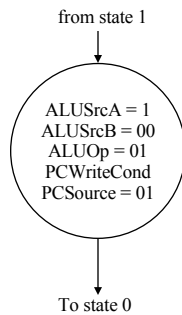
CSE 141, S2'06

Jeff Brown

4. R-type Completion


$$\text{Reg}[\text{IR}[15-11]] = \text{ALUout}$$

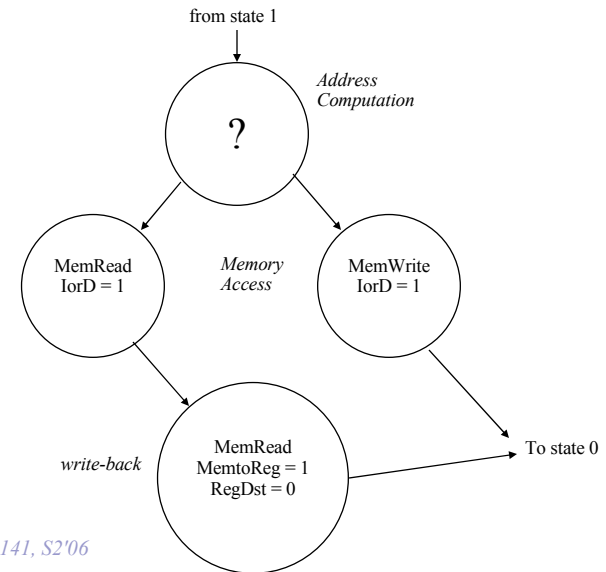
BEQ Instruction



CSE 141, S2'06

Jeff Brown

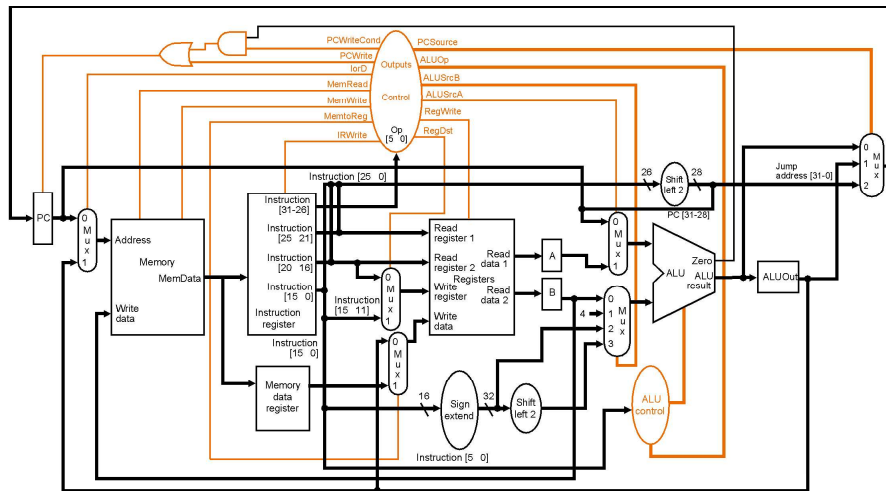
Memory Instructions



CSE 141, S2'06

Jeff Brown

3. Memory Address Computation

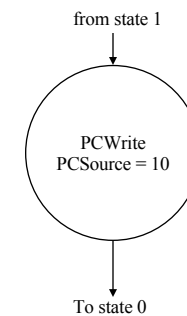


$$\text{ALUOut} = A + \text{sign-extend}(\text{IR}[15:0])$$

CSE 141, S2'06

Jeff Brown

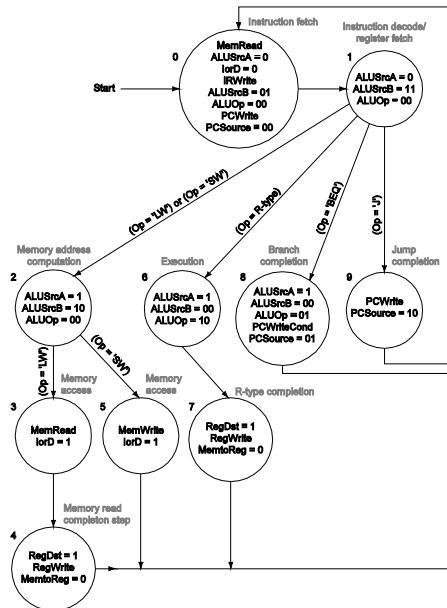
JMP Instruction



CSE 141, S2'06

Jeff Brown

The Whole FSM



CSE 141, S2'06

Jeff Brown

Some Questions

- How many cycles will it take to execute this code?

```
lw $t2, 0($t3)
lw $t3, 4($t3)
beq $t2, $t3, Label #assume not taken
add $t5, $t2, $t3
sw $t5, 8($t3)
Label: ...
```

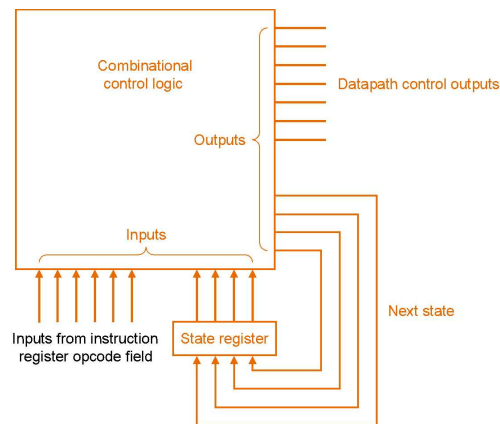
- What is going on during the 8th cycle of execution?
- In what cycle does the actual addition of \$t2 and \$t3 take place?
- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?

CSE 141, S2'06

Jeff Brown

Finite State Machine for Control

- Implementation:

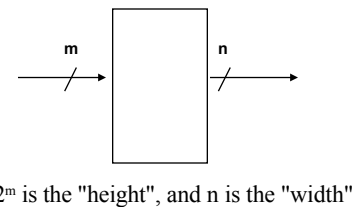


CSE 141, S2'06

Jeff Brown

ROM Implementation

- ROM = "Read Only Memory"
 - values of memory locations are fixed ahead of time
- A ROM can be used to implement a truth table
 - if the address is m -bits, we can address 2^m entries in the ROM.
 - our outputs are the bits of data that the address points to.



0	0	0	0	0	0	1	1
0	0	1	1	1	0	0	0
0	1	0	1	1	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	1
1	1	0	0	1	1	0	0
1	1	1	0	1	1	1	1

CSE 141, S2'06

Jeff Brown

ROM Implementation

- How many inputs are there?
6 bits for opcode, 4 bits for state = 10 address lines
(i.e., $2^{10} = 1024$ different addresses)
- How many outputs are there?
16 datapath-control outputs, 4 state bits = 20 outputs
- ROM is $2^{10} \times 20 = 20\text{K}$ bits (and a rather unusual size)
- Rather wasteful, since for lots of the entries, the outputs are the same
— i.e., opcode is often ignored

CSE 141, S2'06

Jeff Brown

Multicycle CPU Key Points

- Performance gain achieved from **variable-length** instructions
- $ET = IC * CPI * \text{cycle time}$
- Required very few new state elements
- More, and **more complex**, control signals
- Control requires **FSM**

CSE 141, S2'06

Jeff Brown