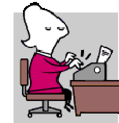
Administrative Matters

- Homework #4
  - Due November 5th
  - Watch out for "regular printing" vs. "revised printing" issue
- Extra Credit Quiz #1
  - Wednesday, November 7th
  - Take about 10 minutes
  - Cover chapter 5 lecture homework 4, and some chapter 6 lecture
  - Worth extra 5 quiz points!
- Midterm #2
  - Monday, November 19th
  - Cover chapter 5, 6, and a little bit 7
  - Cover homework 4 and 5
  - 15% of your grade



©2004 Morgan Kaufmann Publishers

Chapter Five

The Processor: Datapath and Control

(continue)

©2004 Morgan Kaufmann Publishers

Compute CPU Time

$$\text{CPU Time (or, Execution Time)} = \frac{\text{\# of instructions}}{\text{program}} \times \frac{\text{\# of cycles}}{\text{instruction}} \times \frac{\text{\# of seconds}}{\text{cycle}}$$

$$= \text{instruction count} \times \text{CPI} \times \text{cycle time}$$

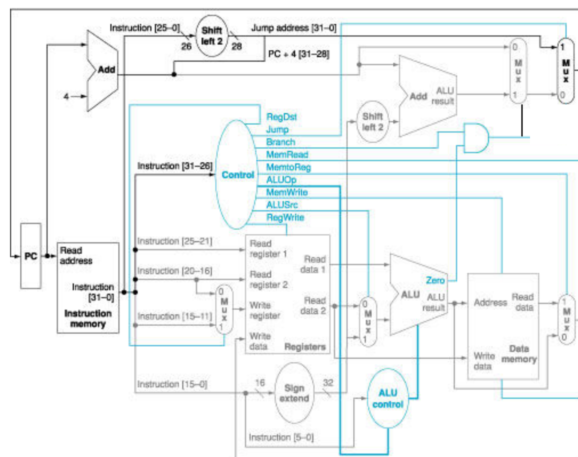
$$= \text{instruction count} \times \text{CPI} \times \frac{1}{\text{clock rate}}$$

- Instruction count is determined
  - By selection of ISA (chapter 2) and compiler (elsewhere)
- Can still play with CPI and cycle time
  - By ways of implementing control and datapath (now)
  - By pipelining (chapter 6)
  - By memory hierarcy (chapter 7)

©2004 Morgan Kaufmann Publishers

Single Cycle Implementation

- **Calculate cycle time assuming negligible delays except:**
  - **memory (2ns), ALU and adders (2ns), register file access (1ns)**



©2004 Morgan Kaufmann Publishers

Single Cycle – How long is the cycle?

Inst. Type	Inst. Mem.	Reg. File (read)	ALU (s)	Data Mem.	Reg. File (write)	Total	Inst. %
R-type	2	1	2	0	1	6 ns	44
Load	2	1	2	2	1	8 ns	24
Store	2	1	2	2	0	7 ns	12
Branch	2	1	2	0	0	5 ns	18
Jump	2	0	0	0	0	2 ns	2

The cycle time must accommodate the longest operation: *lw*.
Cycle time $\geq$ 8 ns but the CPI = 1.

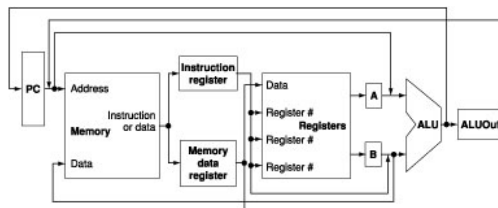
If we can accommodate variable number of cycles for each instruction and a cycle time of 1ns.
CPI = 6*44% + 8*24% + 7*12% + 5*18% + 2*2% = 6.3

How much faster would this machine be?

©2004 Morgan Kaufmann Publishers

Where we are headed

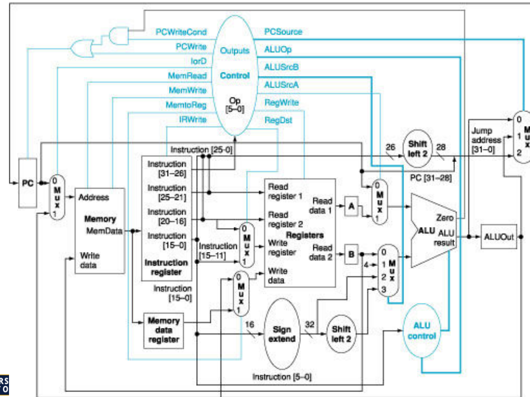
- **Single Cycle Problems:**
  - **what if we had a more complicated instruction like floating point?**
- **One Solution:**
  - **use a "smaller" cycle time**
  - **have different instructions take different numbers of cycles**
  - **a "multicycle" datapath:**

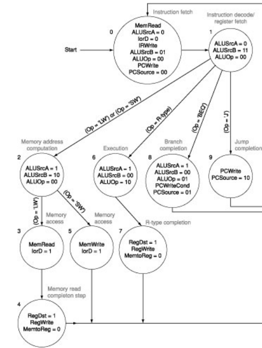


©2004 Morgan Kaufmann Publishers

Multicycle Approach

ALU used to compute address and to increment PC
Memory used for instruction and data
Control signal need to "stay" with instruction as it progresses
- **We'll use a finite state machine for control**

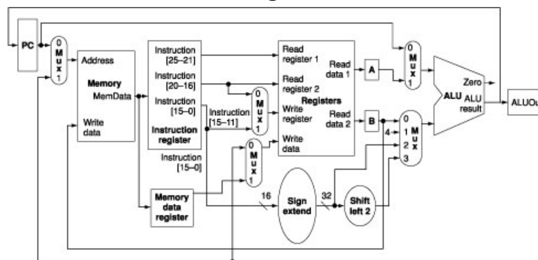




©2004 Morgan Kaufmann Publishers

Multicycle Approach

- **Break up the instructions into steps, each step takes a cycle**
  - **balance the amount of work to be done**
  - **restrict each cycle to use only one major functional unit**
    - Why?
- **At the end of a cycle**
  - **store values for use in later cycles**
  - **introduce additional "internal" registers**



UNIVERSITY of CALIFORNIA Riverside COMPUTER SCIENCE & ENGINEERING

©2004 Morgan Kaufmann Publishers

Instructions from ISA perspective

- Consider each instruction from perspective of ISA.
- Example:
  - The add instruction changes a register.
  - Register specified by bits 15:11 of instruction.
  - Instruction specified by the PC.
  - New value is the sum ("op") of two registers.
  - Registers specified by bits 25:21 and 20:16 of the instruction

```
Reg[Memory[PC][15:11]] <=  Reg[Memory[PC][25:21]] op
                            Reg[Memory[PC][20:16]]
```

  - In order to accomplish this we must break up the instruction.

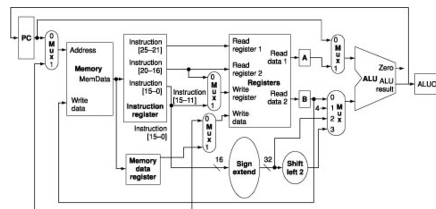
©2004 Morgan Kaufmann Publishers

Breaking down an instruction

- ISA definition of arithmetic:

```
Reg[Memory[PC][15:11]] <= Reg[Memory[PC][25:21]]  op
                          Reg[Memory[PC][20:16]]
```

- Could break down to:
  - `IR <= Memory[PC]`
  - `A <= Reg[IR[25:21]]`
  - `B <= Reg[IR[20:16]]`
  - `ALUOut <= A op B`
  - `Reg[IR[20:16]] <= ALUOut`



- Then, get to the next instruction
  - `PC <= PC + 4`

Why not
Reg[IR[20:16]] <= A op B ?

©2004 Morgan Kaufmann Publishers

Idea behind multicycle approach

- We define each instruction from the ISA perspective
- Break it down into steps
  - following rule that data flows through at most one major functional unit
  - (e.g., balance work across steps)
- Introduce new registers as needed
  - (e.g, A, B, ALUOut, MDR, etc.)
- Finally try and pack as much work into each step
  (avoid unnecessary cycles)
  while also trying to share steps where possible
  (minimizes control, helps to simplify solution)
- Result: a multicycle Implementation!

©2004 Morgan Kaufmann Publishers

Five Execution Steps

- Instruction Fetch
- Instruction Decode and Register Fetch
- Execution, Memory Address Computation, or Branch Completion
- Memory Access or R-type instruction completion
- Write-back step

INSTRUCTIONS TAKE FROM 3 - 5 CYCLES!

©2004 Morgan Kaufmann Publishers

Step 1: Instruction Fetch

- Use PC to get instruction and put it in the Instruction Register.
- Increment the PC by 4 and put the result back in the PC.
- Can be described succinctly using RTL "Register-Transfer Language"

```
IR <= Memory[PC];
PC <= PC + 4;
```

Can we figure out the values of the control signals?

What is the advantage of updating the PC now?

©2004 Morgan Kaufmann Publishers

Step 2: Instruction Decode and Register Fetch

- Read registers rs and rt in case we need them
- Compute the branch address in case the instruction is a branch
- RTL:

```
A <= Reg[IR[25:21]];
B <= Reg[IR[20:16]];
ALUOut <= PC + (sign-extend(IR[15:0]) << 2);
```

- Question: Why not PC+4?
- We still aren't setting any control lines based on the instruction type
  (we are busy "decoding" it in our control logic)
  - Would A, B, and ALUOut value always be:
    - Useful?
    - Meaningful?

©2004 Morgan Kaufmann Publishers

Step 3 (instruction dependent)

- **ALU is performing one of three functions, based on instruction type**
- **Memory Reference:**

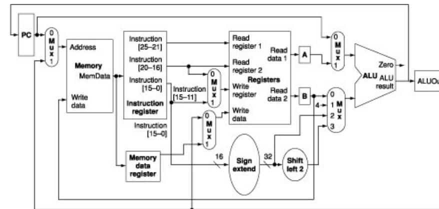
```
ALUOut <= A + sign-extend(IR[15:0]);
```

- **R-type:**

```
ALUOut <= A op B;
```

- **Branch:**

```
if (A==B) PC <= ALUOut;
```



©2004 Morgan Kaufmann Publishers

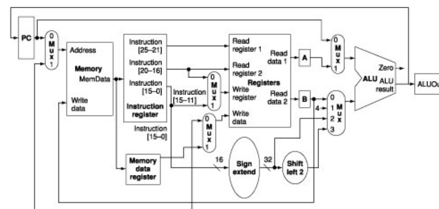
Step 4 (R-type or memory-access)

- **Loads and stores access memory**

```
MDR <= Memory[ALUOut];
       or
Memory[ALUOut] <= B;
```

- **R-type instructions finish**

```
Reg[IR[15:11]] <= ALUOut;
```



©2004 Morgan Kaufmann Publishers

Write-back step

- `Reg[IR[20:16]] <= MDR;`

Which instruction needs this?

©2004 Morgan Kaufmann Publishers

Summary:

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR <= Memory[PC] PC <= PC + 4			
Instruction decode/register fetch	A <= Reg [IR[25:21]] B <= Reg [IR[20:16]] ALUOut <= PC + (sign-extend (IR[15:0]) << 2)			
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + sign-extend (IR[15:0])	If (A == B) PC <= ALUOut	PC <= {PC [31:28], (IR[25:0]],2'b00)}
Memory access or R-type completion	Reg [IR[15:11]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B		
Memory read completion		Load: Reg[IR[20:16]] <= MDR		

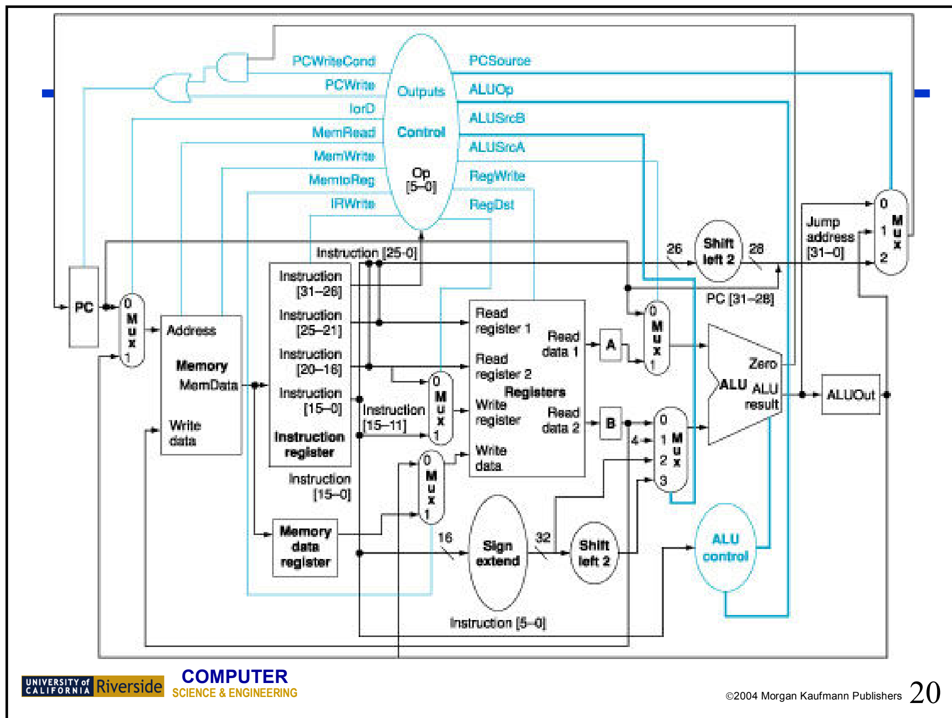
©2004 Morgan Kaufmann Publishers

Simple 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
          add $t5, $t2, $t3
          sw $t5, 8($t3)
Label:    ...
```

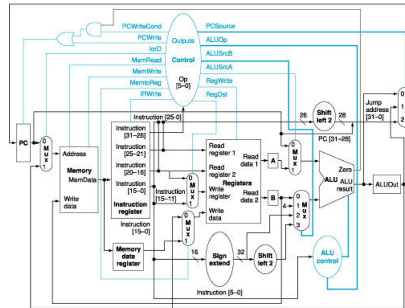
©2004 Morgan Kaufmann Publishers



©2004 Morgan Kaufmann Publishers

Implementing the Control

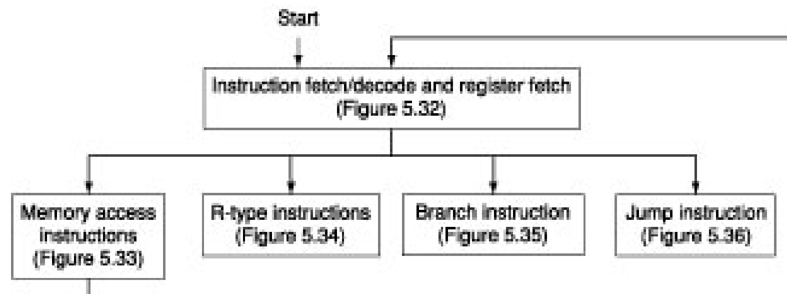
- Value of control signals is dependent upon:
  - what instruction is being executed
  - which step is being performed
- Use the information we've accumulated to specify a finite state machine
  - specify the finite state machine graphically, or
  - use microprogramming



2004 Morgan Kaufmann Publishers

Building the control

- Finite State Machine
  - First two steps are identical for all instructions
  - Last 3-5 steps are different
- Not yet pipelined

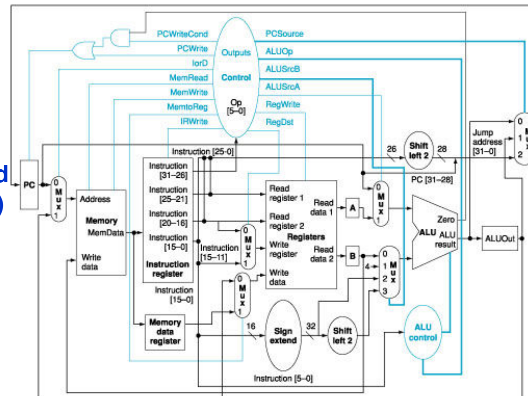


©2004 Morgan Kaufmann Publishers

IF (again)

- Assert
  - MemRead
  - IRWrite
- IorD to instruction
- ALUSrcA to PC
- ALUSrcB to 4
  ALUop to add

- PC+4 happen now to avoid conflict later (execution…)

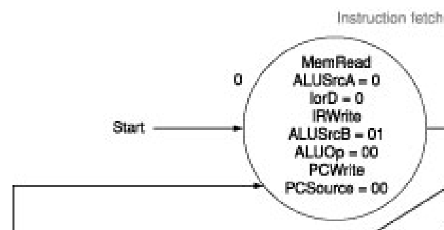


©2004 Morgan Kaufmann Publishers

IF

- Assert
  - MemRead
  - IRWrite
- IorD to instruction
- ALUSrcA to PC
- ALUSrcB to 4
  ALUop to add

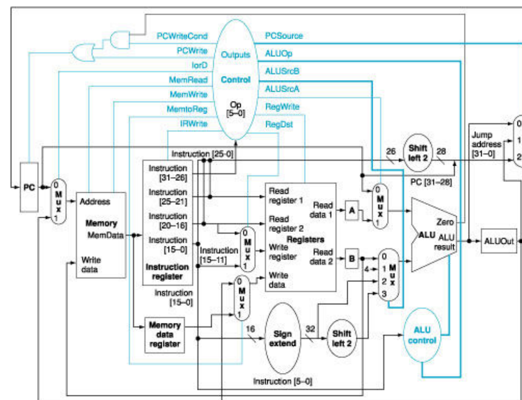
- PC+4 happen now to avoid conflict later (execution…)



©2004 Morgan Kaufmann Publishers

ID (again)

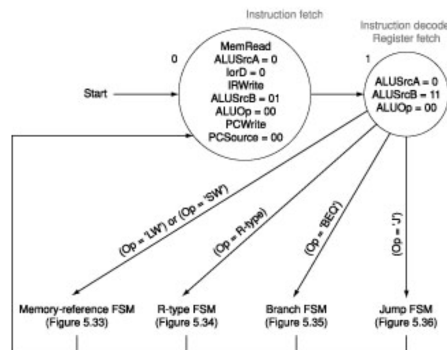
- ALUSrcA to PC
- ALUSrcB to branch offset
- ALUop to add
  - Compute branch target
- Fetch Operand A and B
- Still don't know (care) what instruction it is



©2004 Morgan Kaufmann Publishers

IF and ID FSM

- ALUSrcA to PC
- ALUSrcB to branch offset
- ALUop to add
  - Compute branch target
- Fetch Operand A and B
- Still don't know (care) what instruction it is



©2004 Morgan Kaufmann Publishers

Memory Reference

From state 1

(Op = 'LW') or (Op = 'SW')

Memory address computation

2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00

(Op = 'LW') Memory access

(Op = 'SW') Memory access

3
MemRead
IorD = 1

5
MemWrite
IorD = 1

Memory read completion step

4
RegWrite
MemtoReg = 1
RegDst = 0

To state 0 (Figure 5.32)

©2004 Morgan Kaufmann Publishers

R-Type Instruction

From state 1

(Op = R-Type)

Execution

6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10

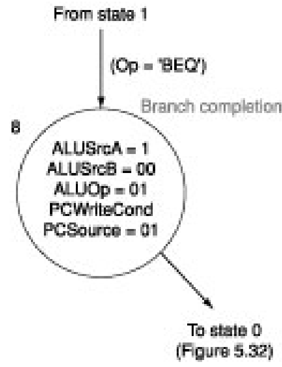
R-type completion

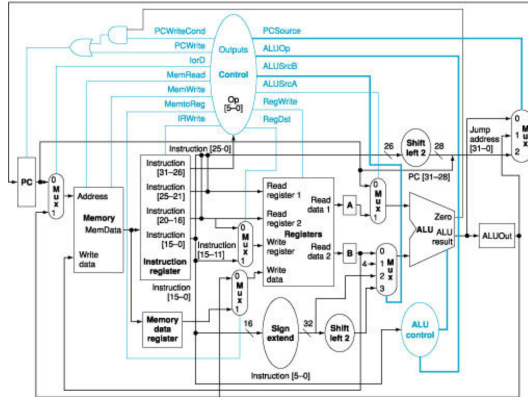
7
RegDst = 1
RegWrite
MemtoReg = 0

To state 0 (Figure 5.32)

©2004 Morgan Kaufmann Publishers

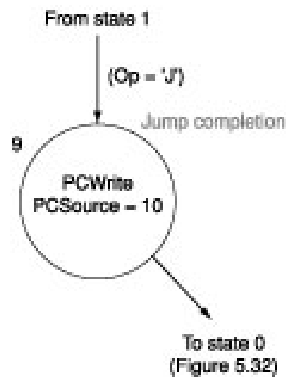
Branch

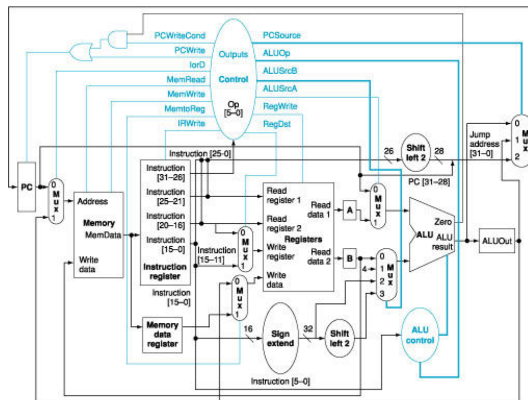




©2004 Morgan Kaufmann Publishers

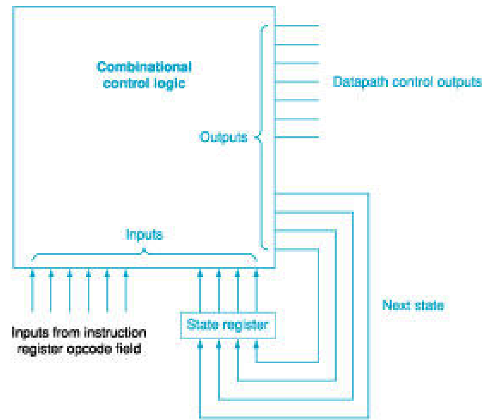
Jump

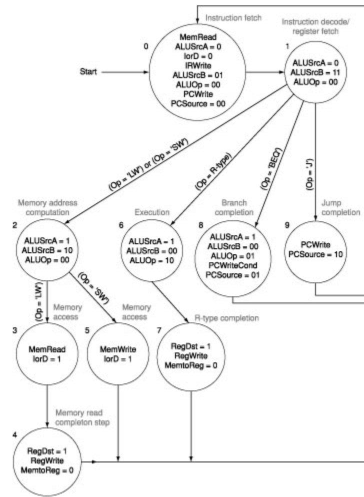




©2004 Morgan Kaufmann Publishers

Control path





©2004 Morgan Kaufmann Publishers