

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



Chapter Five

The Processor: Datapath and Control (continue)

Compute CPU Time

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

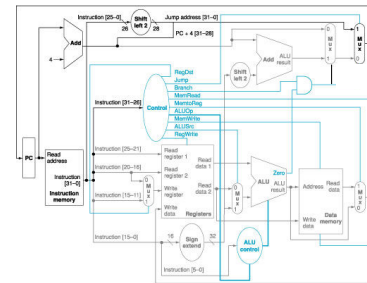
$$= \text{instruction count} \times \text{CPI} \times \text{cycle time}$$

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

- 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 hierarchy (chapter 7)

Single Cycle Implementation

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



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

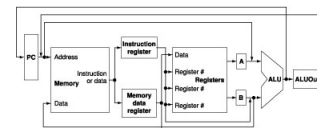
The cycle time must accommodate the longest operation: lw.
Cycle time ≥ 8 ns but the CPI = 1.

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?

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:



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 7

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

©2004 Morgan Kaufmann Publishers 8

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

$$\text{Reg}[\text{Memory}[\text{PC}][15:11]] \leftarrow \text{Reg}[\text{Memory}[\text{PC}][25:21]] \text{ op } \text{Reg}[\text{Memory}[\text{PC}][20:16]]$$

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

©2004 Morgan Kaufmann Publishers 9

Breaking down an instruction

- ISA definition of arithmetic:

$$\text{Reg}[\text{Memory}[\text{PC}][15:11]] \leftarrow \text{Reg}[\text{Memory}[\text{PC}][25:21]] \text{ op } \text{Reg}[\text{Memory}[\text{PC}][20:16]]$$
- Could break down to:
 - $\text{IR} \leftarrow \text{Memory}[\text{PC}]$
 - $\text{A} \leftarrow \text{Reg}[\text{IR}[25:21]]$
 - $\text{B} \leftarrow \text{Reg}[\text{IR}[20:16]]$
 - $\text{ALUOut} \leftarrow \text{A op B}$
 - $\text{Reg}[\text{IR}[20:16]] \leftarrow \text{ALUOut}$
- Then, get to the next instruction
 - $\text{PC} \leftarrow \text{PC} + 4$

Why not
 $\text{Reg}[\text{IR}[20:16]] \leftarrow \text{A op B} ?$

©2004 Morgan Kaufmann Publishers 10

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 11

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!

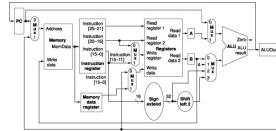
©2004 Morgan Kaufmann Publishers 12

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?



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?

Step 3 (instruction dependent)

- ALU is performing one of three functions, based on instruction type

- Memory Reference:

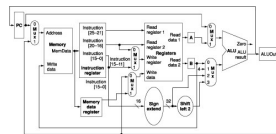
```
ALUOut <= A + sign-extend(IR[15:0]);
```

- R-type:

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

- Branch:

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



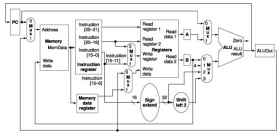
Step 4 (R-type or memory-access)

- Loads and stores access memory

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

- R-type instructions finish

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



Write-back step

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

Which instruction needs this?

Summary:


Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch				
Instruction decode/register fetch				
Execution, address computation, branch/jump completion				
Memory access or R-type completion				
Memory read completion				

Simple Questions

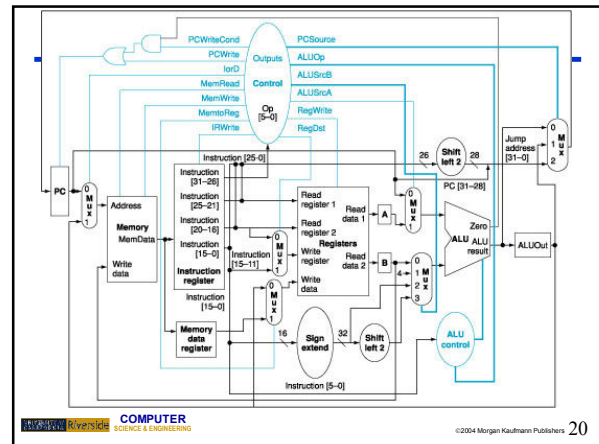
- How many cycles will it take to execute this code?

```

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

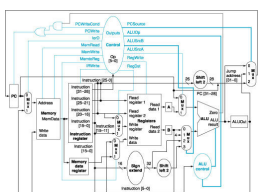


COMPUTER SCIENCE & ENGINEERING ©2004 Morgan Kaufmann Publishers 19

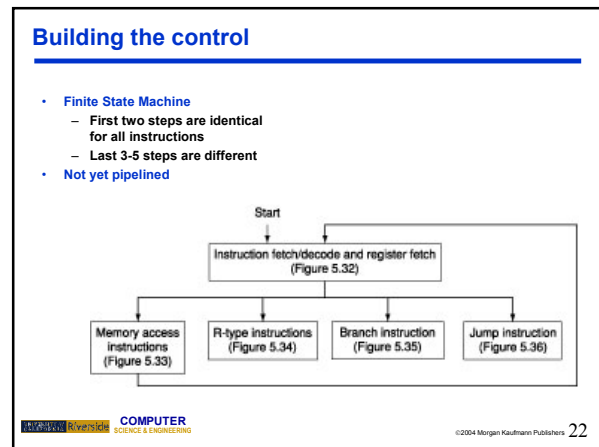


Implementing the Control

- 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



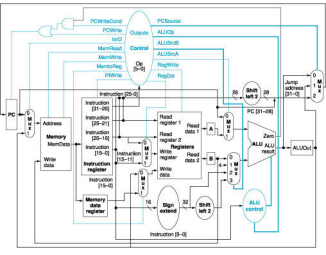
COMPUTER SCIENCE & ENGINEERING ©2004 Morgan Kaufmann Publishers 21



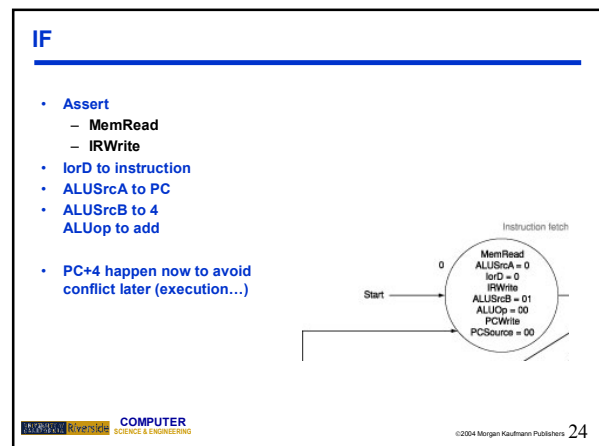
IF (again)

- Assert
 - MemRead
 - IRWrite
- lorD to instruction
- ALUSrcA to PC
- ALUSrcB to 4
- ALUOp to add

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



COMPUTER SCIENCE & ENGINEERING ©2004 Morgan Kaufmann Publishers 23



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 25

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 26

Memory Reference

©2004 Morgan Kaufmann Publishers 27

R-Type Instruction

©2004 Morgan Kaufmann Publishers 28

Branch

©2004 Morgan Kaufmann Publishers 29

Jump

©2004 Morgan Kaufmann Publishers 30

