

Register File

□ Built using D flip-flops

Read register number 1
Read register number 2

Register 0
Register 1
...
Register n-1
Register n

MUX
MUX

Read data 1
Read data 2

Read register number 1
Read data 1
Read register number 2
Read data 2
Write register
Write data
Write

COMPUTER SCIENCE ENGINEERING

©1998 Morgan Kaufmann Publishers 11

Register File

□ Note: we still use the real clock to determine when to write

Write

0
1
...
n-1
n

Register number

Register 0
Register 1
...
Register n-1
Register n

C
D
C
D
C
D
C
D

Register data

D flip-flop

COMPUTER SCIENCE ENGINEERING

©1998 Morgan Kaufmann Publishers 12

Simple Implementation

□ Include the functional units we need for each instruction

Instruction address
Instruction memory

PC

Add Sum

a. Instruction memory

b. Program counter

c. Adder

Address
Read data
Write data
Data memory

MemWrite

MemRead

a. Data memory unit

b. Sign-extension unit

16 Sign extend 32

Register functions
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2

Registers

ALU control
Zero
ALU result

a. Registers

b. ALU

Why do we need this stuff?

COMPUTER SCIENCE ENGINEERING

©1998 Morgan Kaufmann Publishers 13

Building the Datapath

□ Use multiplexers to stitch them together

PC

Instruction memory

Read address
Write data
RegWrite

Registers

Read register 1
Read data 1
Read register 2
Read data 2

MUX

ALUSrc

Shift left 2

Add

ALU operation

Zero

ALU result

MemWrite

MemRead

MemtoReg

PCSrc

Fig. 5.13 pp. 354

COMPUTER SCIENCE ENGINEERING

©1998 Morgan Kaufmann Publishers 14

R-type Instructions

PC

Instruction memory

Read address
Instruction [31-0]

Instruction [25-21]
Instruction [20-16]
Instruction [15-11]
Instruction [10-6]
Instruction [5-0]

PCSrc

Add

4

RegWrite

Shift left 2

Add

ALU result

MemtoReg

MemWrite

MemRead

ALUSrc

Zero

ALU result

ALU control

ALUOp

Sign extend

16

32

COMPUTER SCIENCE ENGINEERING

©1998 Morgan Kaufmann Publishers 15

Lw Instruction

PC

Instruction memory

Read address
Instruction [31-0]

Instruction [25-21]
Instruction [20-16]
Instruction [15-11]
Instruction [10-6]
Instruction [5-0]

PCSrc

Add

4

RegWrite

Shift left 2

Add

ALU result

MemtoReg

MemWrite

MemRead

ALUSrc

Zero

ALU result

ALU control

ALUOp

Sign extend

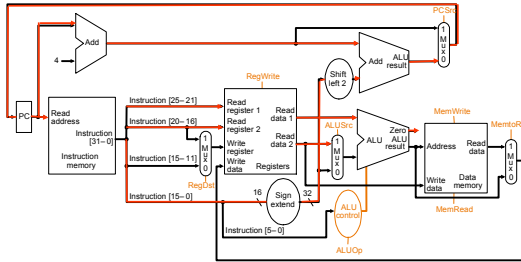
16

32

COMPUTER SCIENCE ENGINEERING

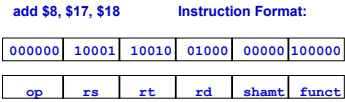
©1998 Morgan Kaufmann Publishers 16

Branch Instructions



Control

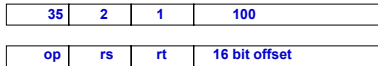
- Selecting the operations to perform (ALU, read/write, etc.)
- Controlling the flow of data (multiplexor inputs)
- Information comes from the 32 bits of the instruction
- Example:



- ALU's operation based on instruction type and function code

Control

- e.g., what should the ALU do with this instruction
- Example: lw \$1, 100(\$2)



- ALU control input

000 AND
001 OR
010 add
110 subtract
111 set-on-less-than

- Multi-level control is common: main control unit generates ALUOp (load/store, branch, arithmetic) bits, which are then used as input to the ALU control that generates the actual signals (3 bits shown above) to control the ALU unit

Control

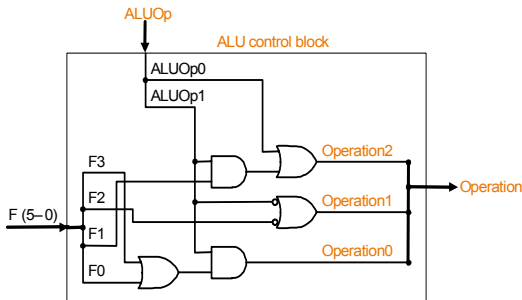
- Must describe hardware to compute 3-bit ALU control input

- given instruction type
 - 00 = lw, sw
 - 01 = beq,
 - 10 = arithmetic
 - function code for arithmetic
- ALUOp computed from instruction type

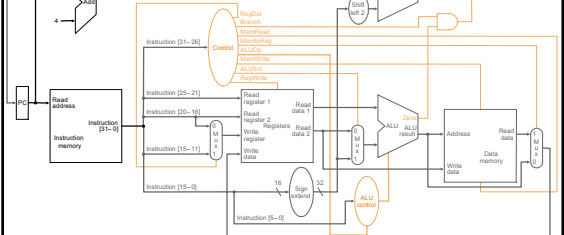
- Describe it using a truth table (can turn into gates):

ALUOp	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

ALU Control Schematic Diagram

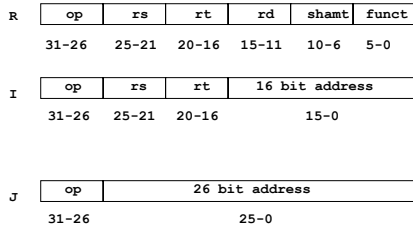


Control

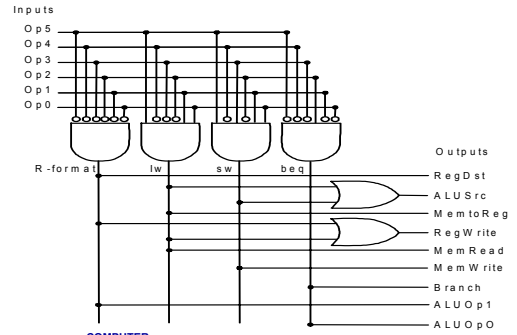


Instruction	MemRead	MemWrite	Branch	ALUOp0	ALUOp1	ALUOp2
R-format	1	0	0	1	0	0
lw	0	1	1	1	1	0
sw	X	1	X	0	0	1
beq	X	0	X	0	0	0

Recall Instruction Format

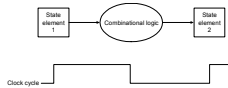


Main Control Schematic Diagram



Our Simple Control Structure

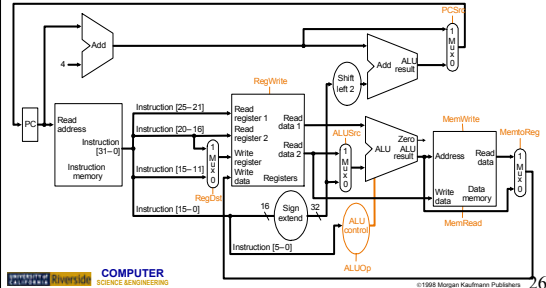
- All of the logic is combinational
- We wait for everything to settle down, and the right thing to be done
 - ALU might not produce “right answer” right away
 - we use write signals along with clock to determine when to write
- Cycle time determined by length of the longest path



We are ignoring some details like setup and hold times

Single Cycle Implementation

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



Single Cycle – Steps of each instruction

Inst. Type	Functional Units Used				
	Instruction fetch	Register read	ALU	Register write	Memory access
R-type	Instruction fetch	Register read	ALU	Register write	
Load	Instruction fetch	Register read	ALU	Memory access	Register write
Store	Instruction fetch	Register read	ALU	Memory access	
Branch	Instruction fetch	Register read	ALU		
Jump	Instruction fetch				

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 = $6 \cdot 44\% + 8 \cdot 24\% + 7 \cdot 12\% + 5 \cdot 18\% + 2 \cdot 2\% = 6.3$

How much faster would this machine be?