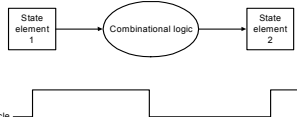


Our Simple Control Structure

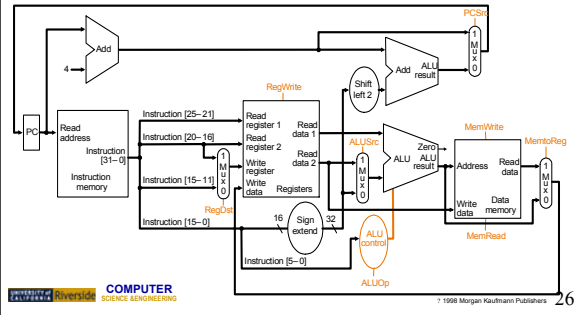
- All of the logic is combinational
- We wait for everything to settle down, and the right thing to be done
 - ALU might not produce “right answer” right away
 - we use write signals along with clock to determine when to write
- Cycle time determined by length of the longest path



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

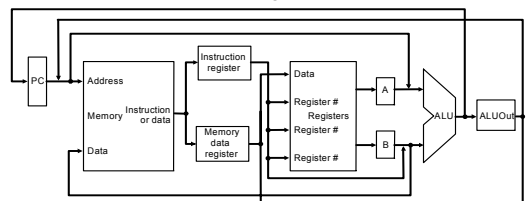
How much faster would this machine be?

Where we are headed

- Single Cycle Problems:
 - what if we had a more complicated instruction like floating point?
 - wasteful of area
- One Solution:
 - use a “smaller” cycle time
 - have different instructions take different numbers of cycles
 - a “multicycle” datapath:
- We will be reusing functional units
 - ALU used to compute address and to increment PC
 - Memory used for instruction and data

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
- At the end of a cycle
 - store values for use in later cycles (easiest thing to do)
 - introduce additional “internal” registers



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;

The write actually takes place at the end of the cycle on the edge

Write-back step

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

What about all the other instructions?

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) then PC = ALUOut	PC = PC [31-28] (IR[25-0] << 2)
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		