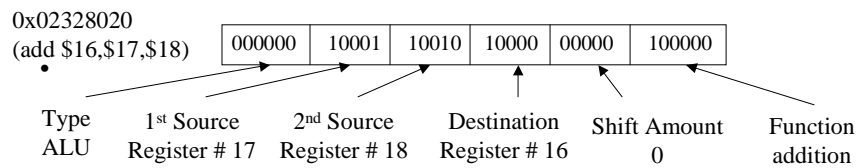


Control Design

- 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:



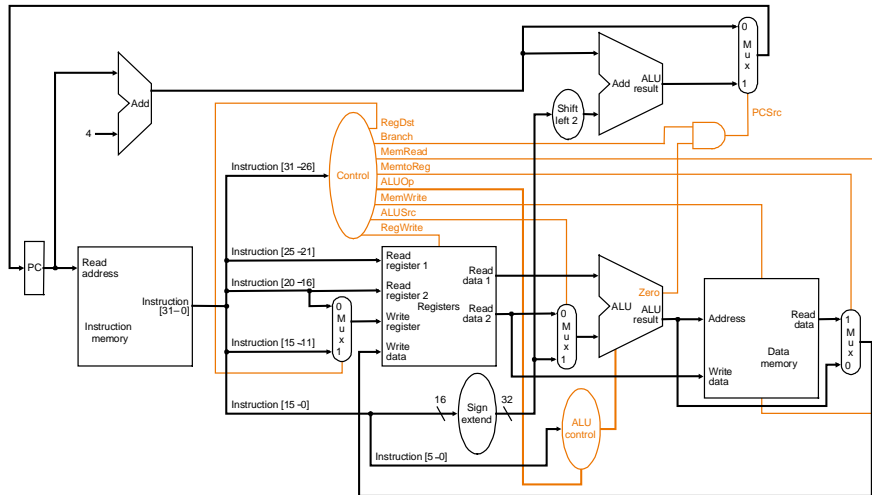
- ALU's operation based on instruction type and function code

ALU Control

- Must describe hardware to compute 3-bit ALU control input
 - given instruction type
 - 00 = lw, sw
 - 01 = beq,
 - 11 = arithmetic
 - function code for arithmetic
- Describe it using a truth table (can turn into gates):

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
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

Data-path with control unit



2003-3-19

Computer Architecture and Design

3

Control Signals

Signal Name	Effects when de-asserted	Effects when asserted
RegDst	Dst. Reg. # = instr[20:16]	Dst. Reg. # = instr[15:11]
RegWrite	none	Reg. indexed by "write register" is written by "write data"
ALUSrc	2nd ALU operand from reg.	2nd ALU operand from instr[15:0]
PCSrc	execute next instr.	branch is taken
MemRead	None	Read data = mem[address]
MemWrite	None	Mem[address]=write data
MemtoReg	"Write data" of reg file from ALU	"Write data" of reg file from mem

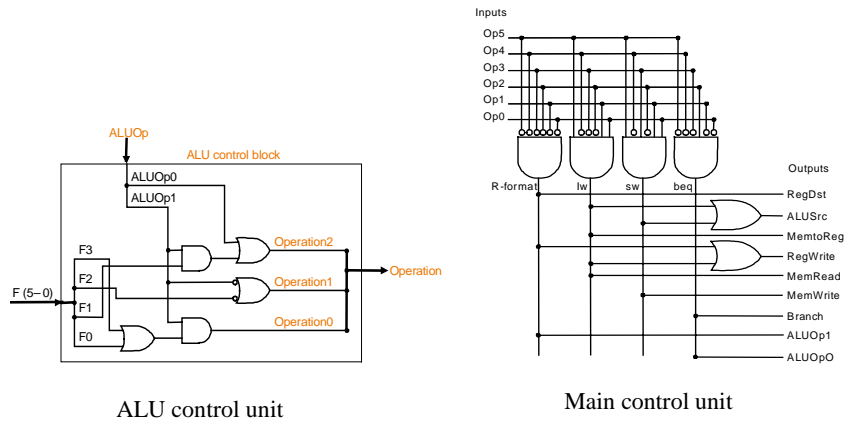
Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

2003-3-19

Computer Architecture and Design

4

Mapping Control Logic to Gates

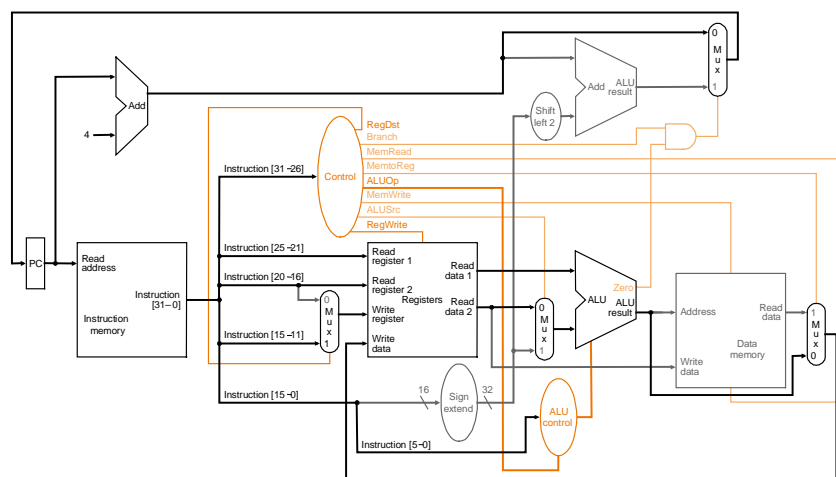


2003-3-19

Computer Architecture and Design

5

Operation of the Data-path: R-type



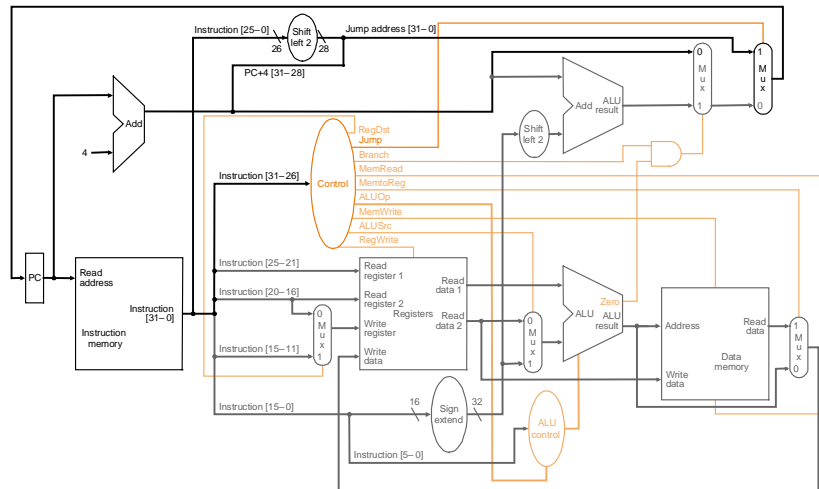
Note: Signals asserted are highlighted

2003-3-19

Computer Architecture and Design

6

Supporting Jump Instruction



Note: Signals asserted are highlighted

2003-3-19

Computer Architecture and Design

9

Summary of Single Cycle Implementation

- 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
- Single Cycle Problems:
 - what if we had a more complicated instruction like floating point?
 - wasteful of area

2003-3-19

Computer Architecture and Design

10

Performance of Single Cycle Machine

Assume operation time for the major functional units are the following:

Memory Access (MA): 2 ns

ALU and Adders (ALU): 2 ns

Register-file Access (RA): 1 ns

Assume other units have no delay.

Machine 1: every instruction operates in 1 clock cycle of a fixed length.

Machine 2: every instruction executes in 1 clock cycle but using a variable-length clock.

Instruction mix:

Loads 24% store 12%

R-format 44%

branch 18% jump 2%

Performance of Single Cycle Machine (cont.)

Instruction class	Functional units used by the instruction class					Total time
	IF	RA	ALU	MA	RA	
R-format	IF	RA	ALU		RA	6
Load	IF	RA	ALU	MA	RA	8
Store	IF	RA	ALU	MA		7
Branch	IF	RA	ALU			5
jump	IF					2

Performance of Single Cycle Machine (cont.)

Machine 1: $ClockTime_{CPU} = Time_{longest_datapath} = 8ns$

Machine 2: $AverageClockTime_{CPU} = \sum (Time_{datapath_i} \times f_i)$
 $= (8 \times 24\% + 7 \times 12\% + 6 \times 44\% + 5 \times 18\% + 2 \times 2\%)$
 $= 6.3ns$

$$\frac{Performance_{machine_2}}{Performance_{machine_1}} = \frac{ExecutionTime_{machine_1}}{ExecutionTime_{machine_2}}$$

$$= \frac{IC \times CPI \times ClockCycle_{machine_1}}{IC \times CPI \times ClockCycle_{machine_2}} = \frac{8}{6.3} = 1.27$$

Performance of floating point processor

Assume operation time for the major functional units are the following:

Memory Access (MA): 2 ns

Integer ALU and Adders (ALU): 2 ns

Floating point addition, subtraction: 8 ns

Floating point multiplication, division: 16ns

Register-file Access (RA): 1 ns

Assume other units have no delay.

Machine 1: every instruction operates in 1 clock cycle of a fixed length.

Machine 2: every instruction executes in 1 clock cycle but using a variable-length clock.

Instruction mix:

Loads 31% store 21%

R-format 27%

Add/sub 7% mul/div 7%

branch 5% jump 2%

Performance of floating point processor (cont.)

Instruction class	Functional units used by the instruction class					Total time
	IF	RA	ALU		RA	
R-format	IF	RA	ALU		RA	6
Load	IF	RA	ALU	MA	RA	8
Store	IF	RA	ALU	MA		7
Branch	IF	RA	ALU			5
jump	IF					2
add/sub	IF	RA	add/sub		RA	12
mul/div	IF	RA	mul/div		RA	20

2003-3-19

Computer Architecture and Design

15

Performance of floating point processor (cont.)

Machine 1:

$$ClockTime_{CPU} = Time_{longest_datapath} = 20ns$$

Machine 2:

$$\begin{aligned}
 AverageClockTime_{CPU} &= \sum (Time_{datapath_i} \times f_i) \\
 &= (8 \times 31\% + 7 \times 21\% + 6 \times 27\% + 5 \times 5\% + 2 \times 2\% + 20 \times 7\% + 12 \times 7\%) \\
 &= 7.0ns
 \end{aligned}$$

$$\begin{aligned}
 \frac{Performance_{machine_2}}{Performance_{machine_1}} &= \frac{ExecutionTime_{machine_1}}{ExecutionTime_{machine_2}} \\
 &= \frac{IC \times CPI \times ClockCycle_{machine_1}}{IC \times CPI \times ClockCycle_{machine_2}} = \frac{20}{7.0} = 2.9
 \end{aligned}$$

2003-3-19

Computer Architecture and Design

16

How to improve single cycle machine

- Single Cycle Problems:
 - what if we had a more complicated instruction like floating point?
 - wasteful of area
- One Solution:
 - use a smaller clock cycle time
 - have different instructions take different numbers of cycles
 - a multi-cycle data-path:

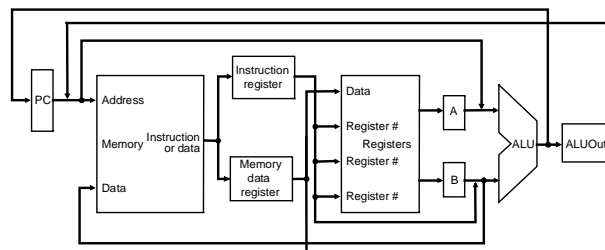
2003-3-19

Computer Architecture and Design

17

Multi-cycle approach

- We will be reusing functional units
 - ALU used to compute address and to increment PC
 - Memory used for instruction and data
- Our control signals will not be determined solely by instruction
 - e.g., what should the ALU do for a subtraction instruction?
- We will use a finite state machine for control



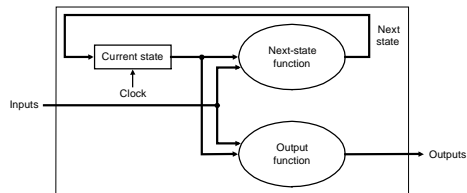
2003-3-19

Computer Architecture and Design

18

Review: finite state machines

- Finite state machines:
 - a set of states and
 - next state function (determined by current state and the input)
 - output function (determined by current state and possibly input)

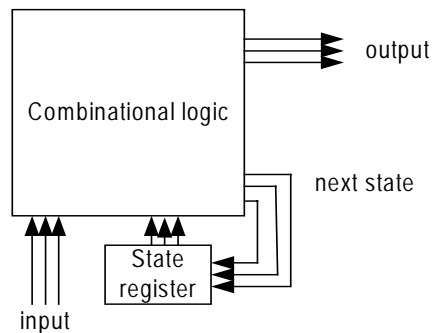


2003-3-19

Computer Architecture and Design

19

General implementation of State Machine

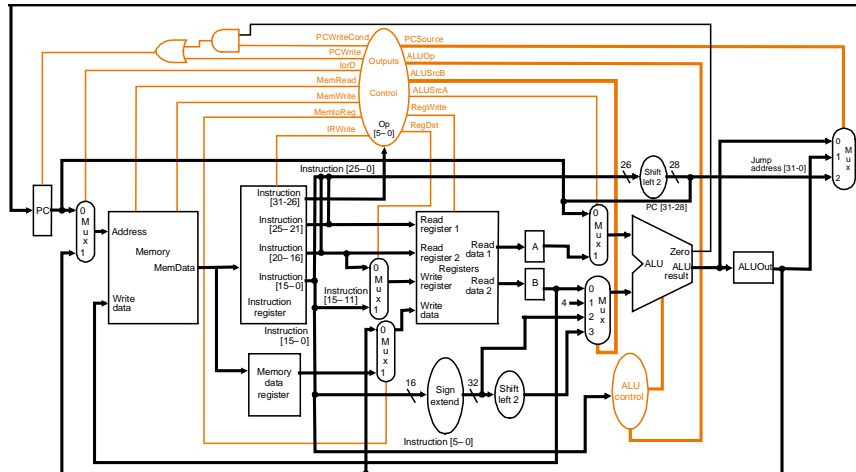


2003-3-19

Computer Architecture and Design

20

Multi-cycle approach (completed)



2003-3-19

Computer Architecture and Design

21

1-bit Control Signals

Signal Name	Effect when de-asserted	Effect when asserted
Regdst	Dst. Reg. # = IR[20:16]	Dst. Reg. # = IR[15:11]
Regwrite	None	Reg. indexed by "write register" is written by "write data"
ALUSrcA	First ALU operand is PC	First ALU operand comes from A register
MemRead	None	Memdata = mem[address]
MemWrite	None	Mem[address]=write data
MemtoReg	ALUOut -> "write data" of registers	MDR -> "Write data" of registers
IorD	Mem Access use PC as address	Mem Access use ALUout as address
IRWrite	None	Memdata -> IR
PCWrite	None	Write PC with source controlled by PCSource
PCWriteCond	None	PC is written if the "Zero of ALU" is active

2003-3-19

Computer Architecture and Design

22

2-bit Control Signals

Signal Name	Value	Effect
ALUOp	00	Add
	01	Sub
	10	Use funct. Field to determine the operation
ALUSrcB	00	2nd operand of ALU from B register
	01	2nd operand of ALU is 4
	10	2nd operand of ALU is sign_extend(IR[15:0])
	11	2nd operand of ALU is sign_extend(IR[15:0]<<2)
PCSource	00	Output of ALU (PC+4) is sent to PC
	01	ALUout->PC
	10	PC+4[31:28] # IR[25:0] # 00 ->PC

Five Execution Steps

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

INSTRUCTIONS TAKE FROM 3 - 5 CYCLES!

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;
```

control signals:

<i>MemRead</i>	<i>IorD=0</i>	<i>IRWrite</i>
<i>ALUSrcA = 0</i>	<i>ALUSrcB = 01</i>	<i>ALUOp = 00</i>
<i>PCWrite</i>	<i>PCSource = 00</i>	

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:
 $A = \text{Reg}[\text{IR}[25-21]];$
 $B = \text{Reg}[\text{IR}[20-16]];$
 $\text{ALUOut} = \text{PC} + (\text{sign-extend}(\text{IR}[15-0]) \ll 2);$
- by this step, We aren't setting any control lines based on the instruction type, we are busy "decoding" it in our control logic

Control signals:

```
ALUSrcA = 0  
ALUSrcB = 11  
ALUOp = 00
```

Step 3 (instruction dependent)

ALU is performing functions based on instruction type

- Memory Reference: $ALUOut = A + \text{sign-extend}(IR[15:0]);$
ALUSrcA = 1 ALUSrcB = 10 ALUOp = 00
- R-type: $ALUOut = A \text{ op } B;$
ALUSrcA = 1 ALUSrcB = 00 ALUOp = 10
- Branch: if (A==B) PC = ALUOut;
ALUSrcA = 1 ALUSrcB = 00 ALUOp = 01
PCWriteCond PCSource = 01
- Jump: $PC = PC[31:28] \# IR[25:0] \# 00$
PCWrite PCSource = 10

Step 4 (R-type or memory-access)

- Loads and stores access memory
 $MDR = \text{Memory}[ALUOut];$
MemRead IorD = 1
- Memory[ALUOut] = B;
MemWrite IorD = 1
- R-type instructions finish
 $\text{Reg}[IR[15:11]] = ALUOut;$
RegDst = 1 RegWrite MemtoReg = 0
- *The write actually takes place at the end of the cycle on the edge*

Step 5 Write back

Only “lw” instruction

RTL:

$\text{Reg}[\text{IR}[20-16]] = \text{MDR};$
RegDst=0 *RegWrite* *MemtoReg=1*

summary

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

Implementing the Control

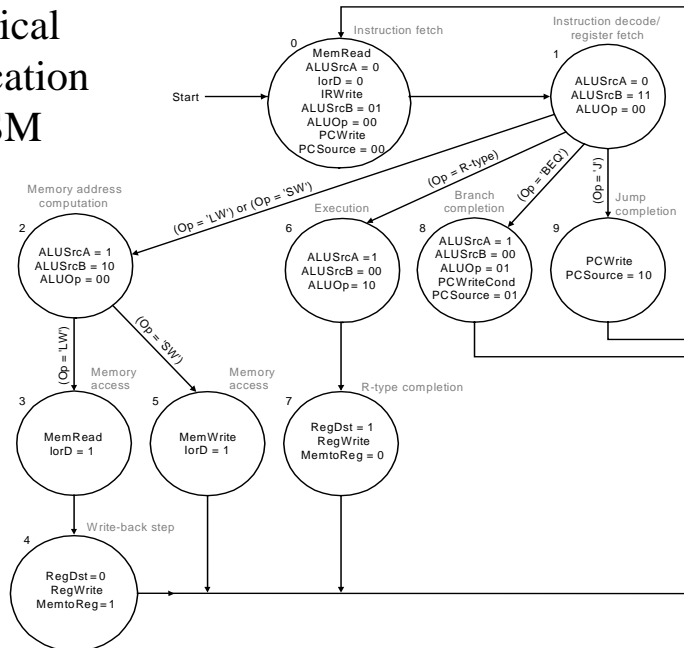
- Value of control signals is dependent upon:
 - what instruction is being executed
 - which step is being performed
- Use the information we have accumulated to specify a finite state machine
 - specify the finite state machine graphically, or
 - use microprogramming
- Implementation can be derived from specification

2003-3-19

Computer Architecture and Design

31

Graphical Specification of FSM



2003-3-19

Computer Architecture and Design

32