CS 61C:
Great Ideas in Computer Architecture
Single Cycle MIPS CPU—Part II

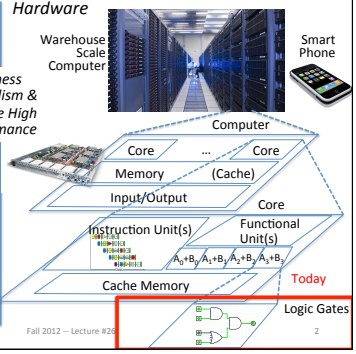
Instructors:
Krste Asanovic, Randy H. Katz
http://inst.eecs.Berkeley.edu/~cs61c/fa12

You are Here!

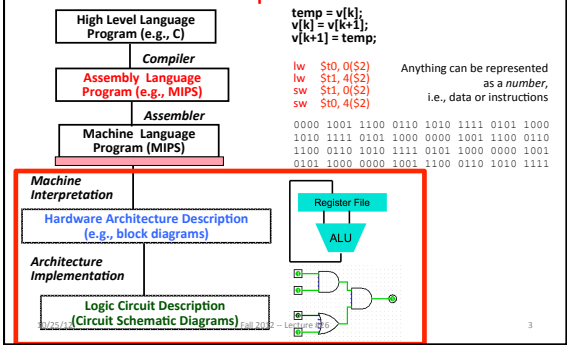
Software / *Hardware*

- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time
- Programming Languages

Harness Parallelism & Achieve High Performance

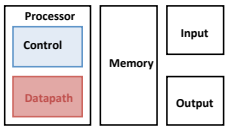


Levels of Representation/ Interpretation



Processor Design Process

- Five steps to design a processor:
  1. Analyze instruction set → datapath requirements
  2. Select set of datapath components & establish clock methodology
  3. Assemble datapath meeting the requirements
  4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
  5. Assemble the control logic
     - Formulate Logic Equations
     - Design Circuits



Agenda

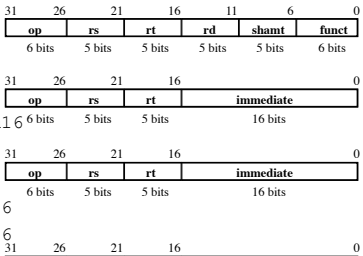
- Datapath Control
- Administrivia
- Control Implementation

Agenda

- Datapath Control
- Administrivia
- Control Implementation

The MIPS-lite Subset

- ADDU and SUBU
  - `addu rd,rs,rt`
  - `subu rd,rs,rt`
- OR Immediate:
  - `ori rt,rs,imm16`
- LOAD and STORE Word
  - `lw rt,rs,imm16`
  - `sw rt,rs,imm16`
- BRANCH:
  - `beq rs,rt,imm16`



Register Transfer Language (RTL)

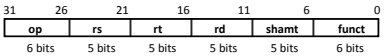
- RTL gives the meaning of the instructions

```
{op , rs , rt , rd , shamt , funct} ← MEM[ PC ]
{op , rs , rt ,   Imm16} ← MEM[ PC ]
```

- All start by fetching the instruction

```
Inst   Register Transfers
ADDU   R[rd] ← R[rs] + R[rt]; PC ← PC + 4
SUBU   R[rd] ← R[rs] – R[rt]; PC ← PC + 4
ORI    R[rt] ← R[rs] | zero_ext(Imm16); PC ← PC + 4
LOAD   R[rt] ← MEM[ R[rs] + sign_ext(Imm16)]; PC ← PC + 4
STORE  MEM[ R[rs] + sign_ext(Imm16) ] ← R[rt]; PC ← PC + 4
BEQ    if ( R[rs] == R[rt] )
           then PC ← PC + 4 + (sign_ext(Imm16) || 00)
           else PC ← PC + 4
```


RTL: The Add Instruction

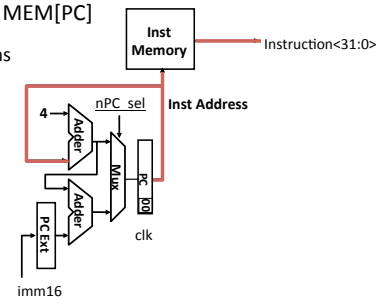


`add rd, rs, rt`

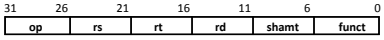
- MEM[PC] Fetch the instruction from memory
- R[rd] = R[rs] + R[rt] The actual operation
- PC = PC + 4 Calculate the next instruction's address

Instruction Fetch Unit at the Beginning of Add

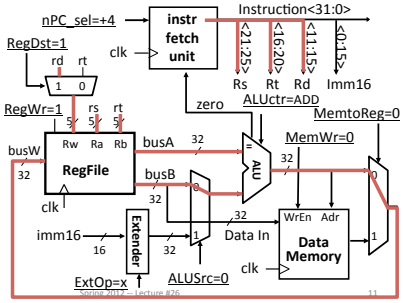
- Fetch the instruction from Instruction memory: Instruction = MEM[PC]
  - same for all instructions



Single Cycle Datapath during Add

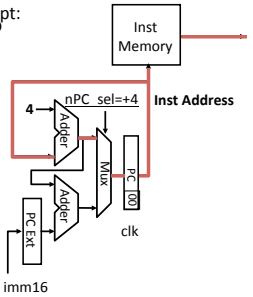


R[rd] = R[rs] + R[rt]

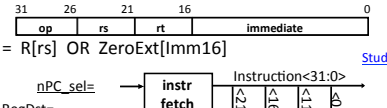


Instruction Fetch Unit at End of Add

- PC = PC + 4
  - Same for all instructions except: Branch and Jump

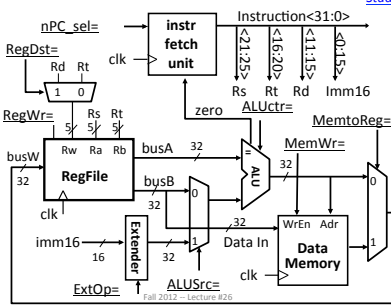


Single Cycle Datapath during Or Immediate

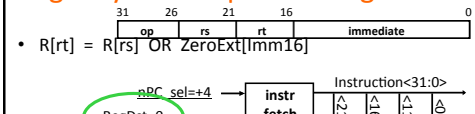


- R[rt] = R[rs] OR ZeroExt[Imm16]

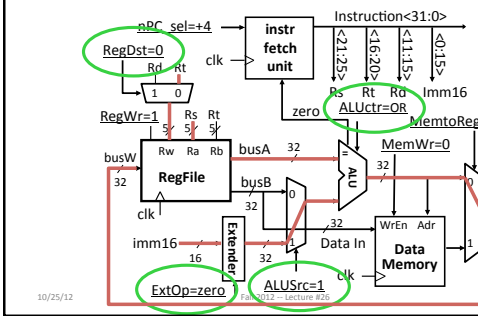
Student Roulette



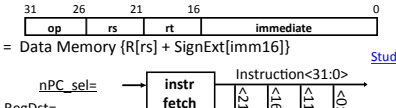
Single Cycle Datapath during Or Immediate



- R[rt] = R[rs] OR ZeroExt[Imm16]

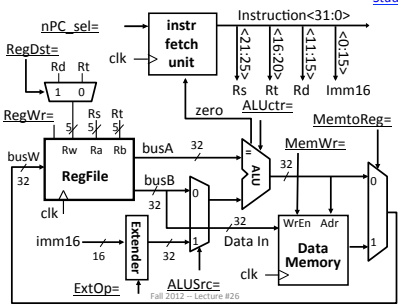


Single Cycle Datapath during Load

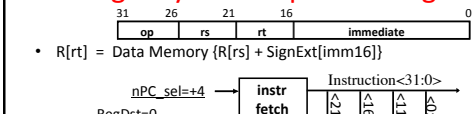


- R[rt] = Data Memory {R[rs] + SignExt[imm16]}

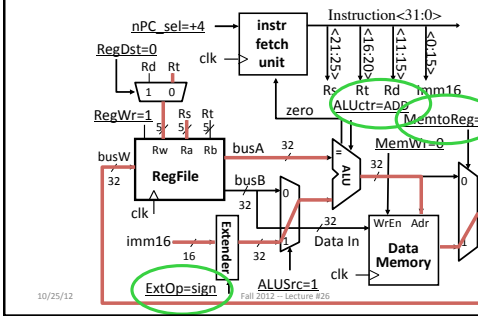
Student Roulette



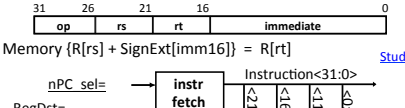
Single Cycle Datapath during Load



- R[rt] = Data Memory {R[rs] + SignExt[imm16]}

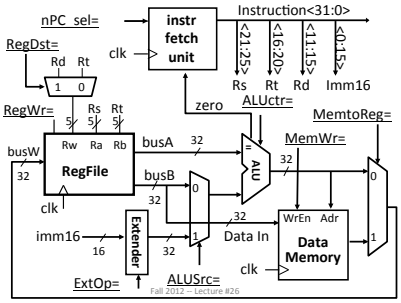


Single Cycle Datapath during Store

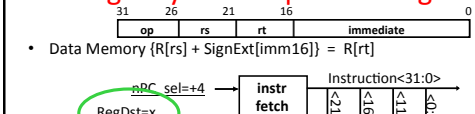


- Data Memory {R[rs] + SignExt[imm16]} = R[rt]

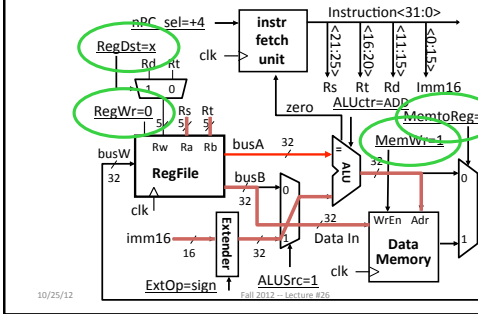
Student Roulette



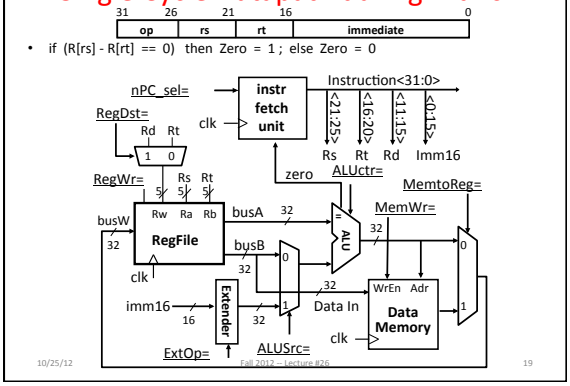
Single Cycle Datapath during Store



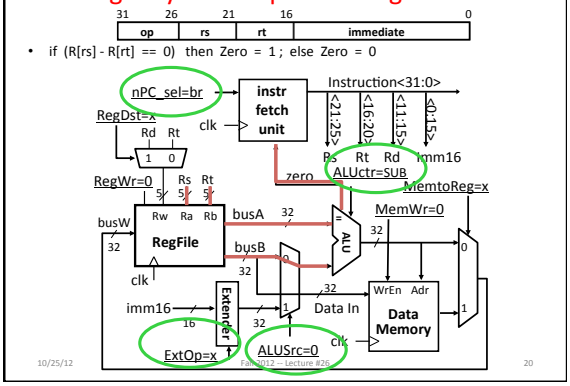
- Data Memory {R[rs] + SignExt[imm16]} = R[rt]



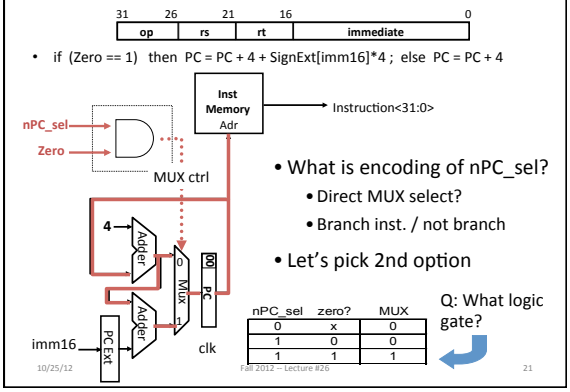
Single Cycle Datapath during Branch



Single Cycle Datapath during Branch

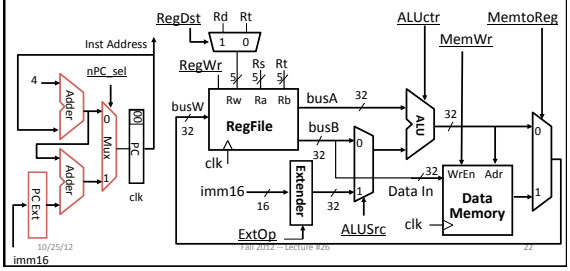


Instruction Fetch Unit at the End of Branch



Summary: Datapath's Control Signals

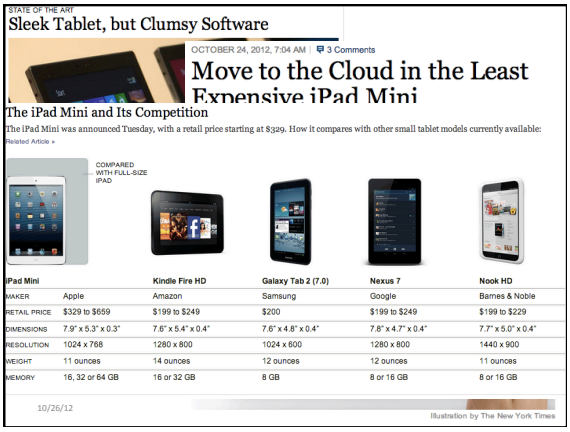
- ExtOp: "zero", "sign"
- ALUsrc: 0 ⇒ regB; 1 ⇒ immed
- ALUctr: "ADD", "SUB", "OR"
- MemWr: 1 ⇒ write memory
- MemtoReg: 0 ⇒ ALU; 1 ⇒ Mem
- RegDst: 0 ⇒ "rt"; 1 ⇒ "rd"
- RegWr: 1 ⇒ write register



Agenda

- Datapath Control
- Administrivia
- Control Implementation

10/25/12 Fall 2012 -- Lecture #26 23



STATE OF THE ART

Sleek Tablet, but Clumsy Software

OCTOBER 24, 2012, 7:04 AM | 3 Comments

Move to the Cloud in the Least Expensive iPad Mini

The iPad Mini and Its Competition

The iPad Mini was announced Tuesday, with a retail price starting at $329. How it compares with other small tablet models currently available:

Related Article »

COMPARED WITH FULL-SIZE IPAD

	iPad Mini	Kindle Fire HD	Galaxy Tab 2 (7.0)	Nexus 7	Nook HD
MAKER	Apple	Amazon	Samsung	Google	Barnes & Noble
RETAIL PRICE	$329 to $659	$199 to $249	$200	$199 to $249	$199 to $229
DIMENSIONS	7.9" x 5.3" x 0.3"	7.6" x 5.4" x 0.4"	7.6" x 4.8" x 0.4"	7.8" x 4.7" x 0.4"	7.7" x 5.0" x 0.4"
RESOLUTION	1024 x 768	1280 x 800	1024 x 600	1280 x 800	1440 x 900
WEIGHT	11 ounces	14 ounces	12 ounces	12 ounces	11 ounces
MEMORY	16, 32 or 64 GB	16 or 32 GB	8 GB	8 or 16 GB	8 or 16 GB

Illustration by The New York Times

10/26/12

OCTOBER 22, 2012, 3:58 PM | 24 Comments

Amazon Cloud Service Goes Down and Takes Popular Sites With It

By NICOLE PERLROTH

FACEBOOK TWITTER GOOGLE+ E-MAIL SHARE PRINT

7:11 p.m. | Updated *Updated throughout.*

Some services at Amazon.com's data centers went down Monday afternoon, taking with them a number of popular Web sites and services, including Flipboard and Foursquare.

Amazon reported problems at data centers in Northern Virginia that appeared to have had a ripple effect across the Internet, as many companies depend on the company's cloud service to run their businesses. Several frustrated customers took to Twitter to complain.

In June, an electrical storm caused problems at the same Northern Virginia data centers and took down sites including Netflix, Pinterest and Instagram for a weekend.

The companies that were affected by the latest shutdown scrambled to respond.

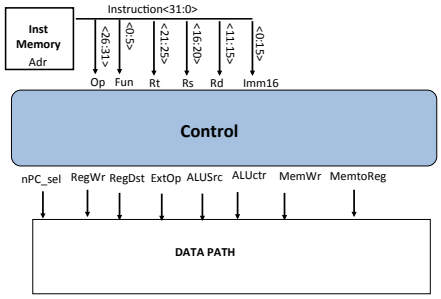
"Like many other services, we've been taken down by the outage," said Erin Gleason, a spokeswoman for Foursquare, the mobile check-in service. "Both the site and the app are inaccessible right now."

Ms. Gleason said the company was still awaiting information from Amazon about when its service might be restored.

Agenda

- Datapath Control
- Administrivia
- Control Implementation

Given Datapath: RTL → Control



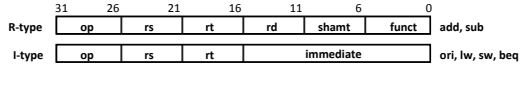
Summary of the Control Signals (1/2)

inst	Register Transfer
add	R[rd] ← R[rs] + R[rt]; PC ← PC + 4
	ALUsrc=RegB, ALUctr="ADD", RegDst=rd, RegWr, nPC_sel="+4"
sub	R[rd] ← R[rs] – R[rt]; PC ← PC + 4
	ALUsrc=RegB, ALUctr="SUB", RegDst=rd, RegWr, nPC_sel="+4"
ori	R[rt] ← R[rs] + zero_ext(Imm16); PC ← PC + 4
	ALUsrc=Im, Extop="Z", ALUctr="OR", RegDst=rt,RegWr, nPC_sel="+4"
lw	R[rt] ← MEM[R[rs] + sign_ext(Imm16)]; PC ← PC + 4
	ALUsrc=Im, Extop="sn", ALUctr="ADD", MemtoReg, RegDst=rt, RegWr, nPC_sel = "+4"
sw	MEM[R[rs] + sign_ext(Imm16)] ← R[rs]; PC ← PC + 4
	ALUsrc=Im, Extop="sn", ALUctr = "ADD", MemWr, nPC_sel = "+4"
beq	if (R[rs] == R[rt]) then PC ← PC + sign_ext(Imm16)] || 00 else PC ← PC + 4
	nPC_sel = "br", ALUctr = "SUB"

Summary of the Control Signals (2/2)

See Appendix A

	func	10 0000	10 0010	We Don't Care :-)			
	op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100
		add	sub	ori	lw	sw	beq
RegDst		1	1	0	0	x	x
ALUSrc		0	0	1	1	1	0
MemtoReg		0	0	0	1	x	x
RegWrite		1	1	1	1	0	0
MemWrite		0	0	0	0	1	0
nPCsel		0	0	0	0	0	1
Jump		0	0	0	0	0	0
ExtOp		x	x	0	1	1	x
ALUctr<2:0>		Add	Subtract	Or	Add	Add	Subtract



Boolean Expressions for Controller

```
RegDst    = add + sub
ALUSrc    = ori + lw + sw
MemtoReg  = lw
RegWrite  = add + sub + ori + lw
MemWrite  = sw
nPCsel    = beq
Jump      = jump
ExtOp     = lw + sw
ALUctr[0] = sub + beq   (assume ALUctr is  00 ADD,  01: SUB,  10: OR)
ALUctr[1] = or
```

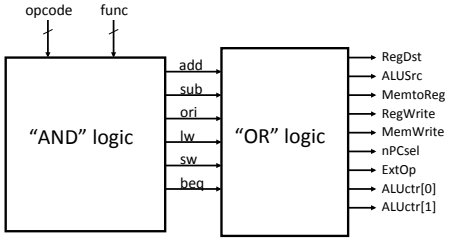
Where:

$rtype = \sim op_5 \cdot \sim op_4 \cdot \sim op_3 \cdot \sim op_2 \cdot \sim op_1 \cdot \sim op_0$,
$ori = \sim op_5 \cdot \sim op_4 \cdot op_3 \cdot \sim op_2 \cdot \sim op_1 \cdot op_0$
$lw = op_5 \cdot \sim op_4 \cdot \sim op_3 \cdot \sim op_2 \cdot op_1 \cdot op_0$
$sw = op_5 \cdot \sim op_4 \cdot op_3 \cdot \sim op_2 \cdot op_1 \cdot op_0$
$beq = \sim op_5 \cdot \sim op_4 \cdot \sim op_3 \cdot op_2 \cdot \sim op_1 \cdot \sim op_0$
$jump = \sim op_5 \cdot \sim op_4 \cdot \sim op_3 \cdot \sim op_2 \cdot op_1 \cdot \sim op_0$

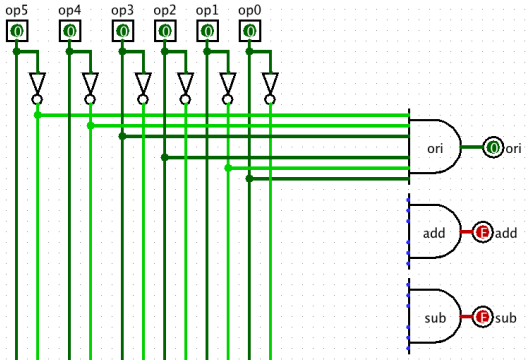
How do we implement this in gates?

$add = rtype \cdot func_5 \cdot \sim func_4 \cdot \sim func_3 \cdot \sim func_2 \cdot \sim func_1 \cdot \sim func_0$
$sub = rtype \cdot func_5 \cdot \sim func_4 \cdot \sim func_3 \cdot \sim func_2 \cdot func_1 \cdot \sim func_0$

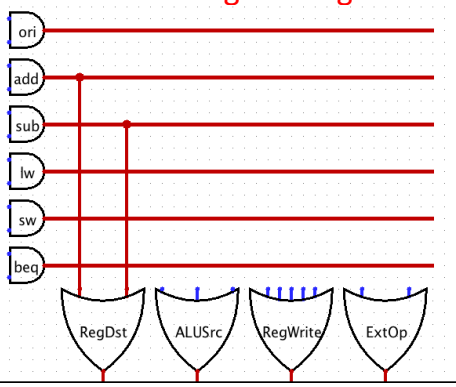
Controller Implementation



AND Control in Logisim



OR Control Logic in Logisim



Single Cycle Performance

- Assume time for actions are
  - 100ps for register read or write; 200ps for other events
- Clock rate is?

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

- What can we do to improve clock rate?
- Will this improve performance as well?
  Want increased clock rate to mean faster programs

And in Conclusion, … Single-Cycle Processor

- Five steps to design a processor:
  1. Analyze instruction set → datapath requirements
  2. Select set of datapath components & establish clock methodology
  3. Assemble datapath meeting the requirements
  4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
  5. Assemble the control logic
     - Formulate Logic Equations
     - Design Circuits