CS 61C: Great Ideas in Computer Architecture (Machine Structures)

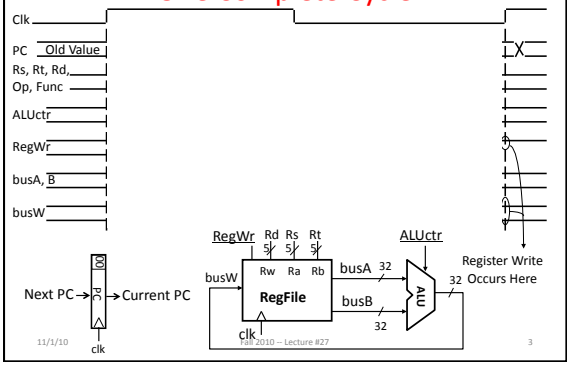
Control Implementation

Instructors:
Randy H. Katz
David A. Patterson
http://inst.eecs.Berkeley.edu/~cs61c/fa10

Agenda

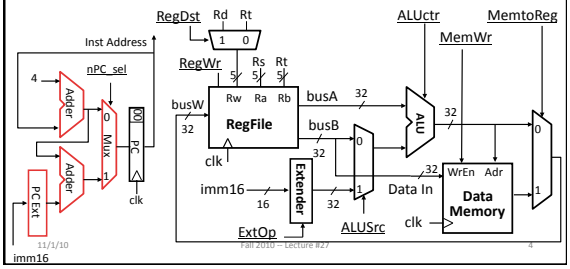
- Datapath Control
- Administrivia
- Technology Break
- Controller Implementation

Register-Register Timing: One Complete Cycle



Datapath's Control Signals

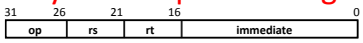
- ExtOp: "zero", "sign"
- ALUsrc: 0 ⇒ regB; 1 ⇒ immed
- ALUctr: "ADD", "SUB", "OR"
- MemWr: 1 ⇒ write memory
- MemtoReg: 0 ⇒ ALU; 1 ⇒ Mem
- RegDst: 0 ⇒ "rt"; 1 ⇒ "rd"
- RegWr: 1 ⇒ write register



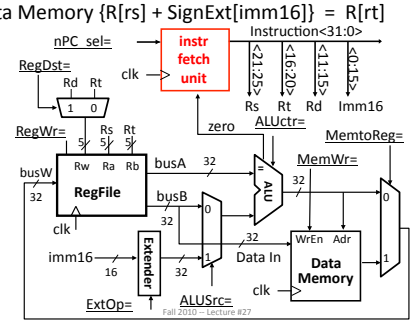
Agenda

- Datapath Control
- Administrivia
- Technology Break
- Controller Implementation

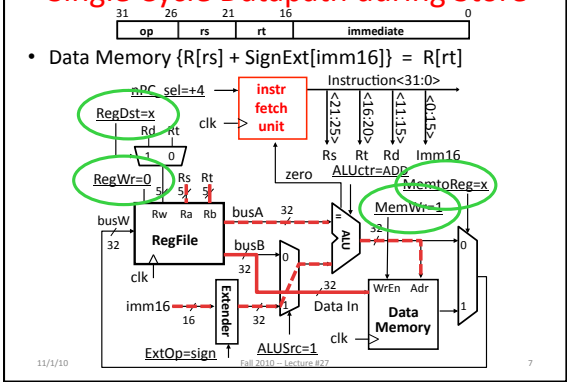
Single Cycle Datapath during Store



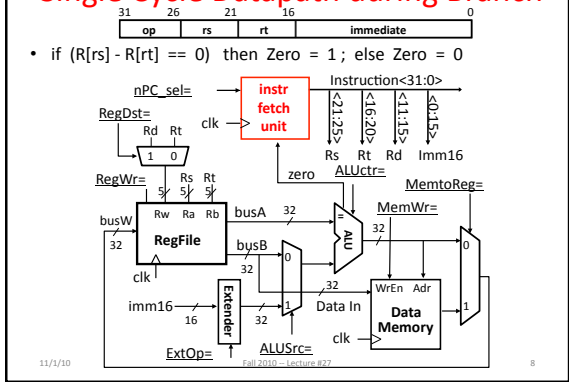
- Data Memory {R[rs] + SignExt[imm16]} = R[rt]



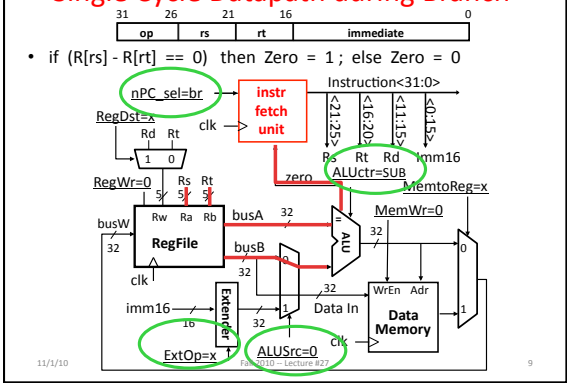
Single Cycle Datapath during Store



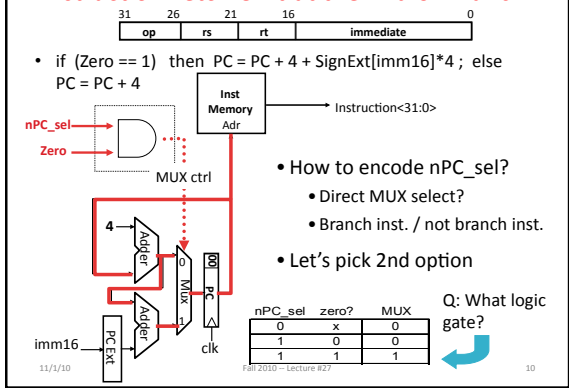
Single Cycle Datapath during Branch



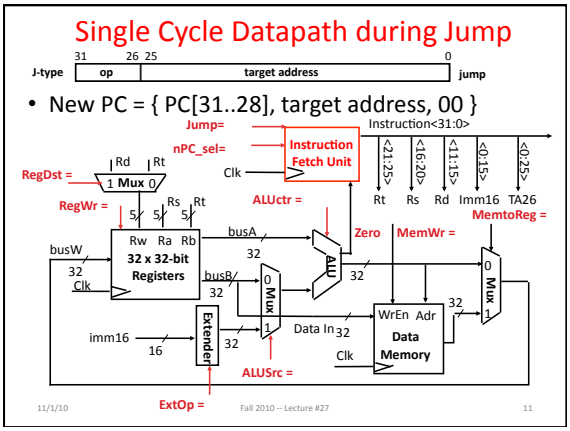
Single Cycle Datapath during Branch



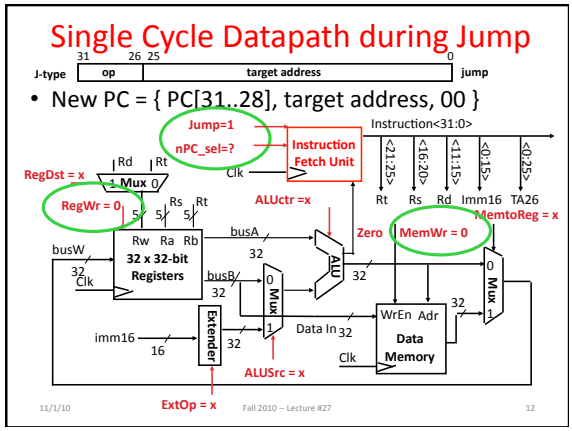
Instruction Fetch Unit at the End of Branch



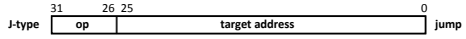
Single Cycle Datapath during Jump



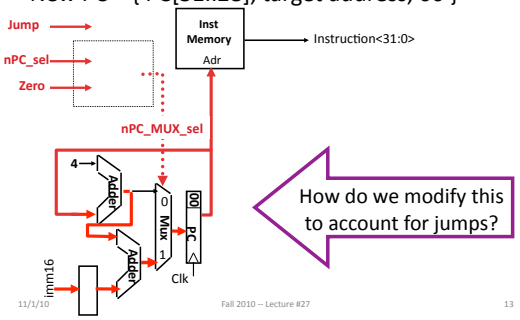
Single Cycle Datapath during Jump



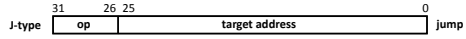
Instruction Fetch Unit at the End of Jump



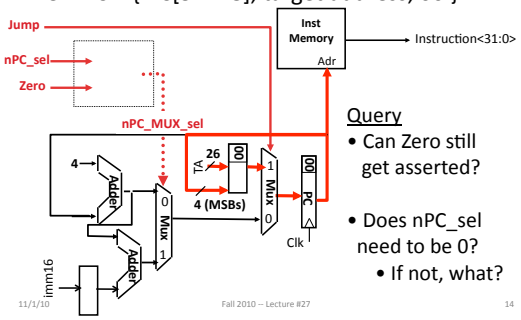
- New PC = { PC[31..28], target address, 00 }



Instruction Fetch Unit at the End of Jump



- New PC = { PC[31..28], target address, 00 }



Query

- Can Zero still get asserted?
- Does nPC_sel need to be 0?
  - If not, what?

Agenda

- Datapath Control
- Administrivia
- Technology Break
- Controller Implementation

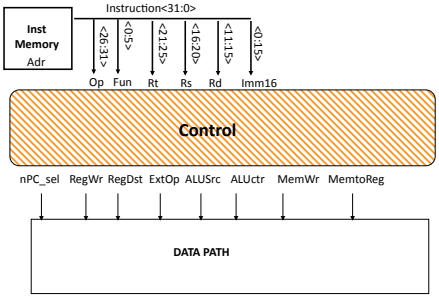
Agenda

- Datapath Control
- Administrivia
- Technology Break
- Controller Implementation

Agenda

- Datapath Control
- Administrivia
- Technology Break
- Controller Implementation

Given Datapath: RTL → Control



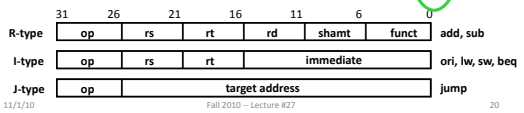
Summary of the Control Signals (1/2)

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 / op

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



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
```

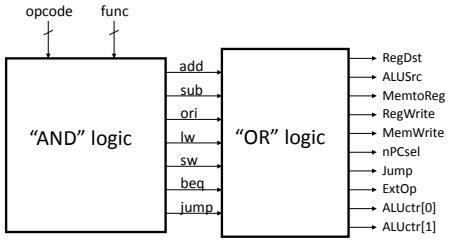
Where:

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

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



Summary: 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