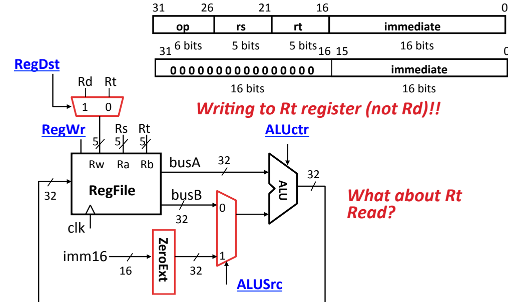
CS 61C: Great Ideas in Computer Architecture (Machine Structures)
Lecture 30: Single-Cycle CPU
Datapath Control Part 2

Instructor: Dan Garcia
http://inst.eecs.berkeley.edu/~cs61c

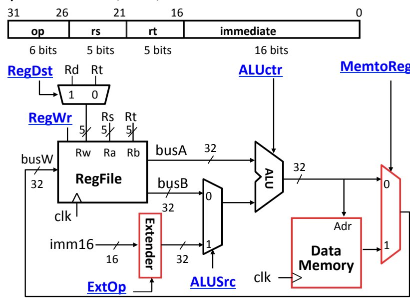
3c: Logical Op (or) with Immediate

- R[rt] = R[rs] op ZeroExt[imm16]



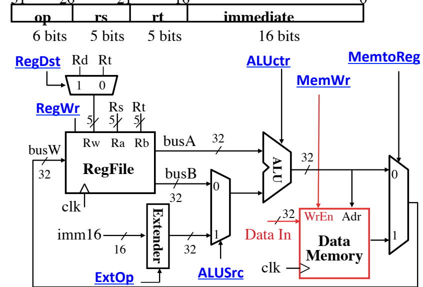
3d: Load Operations

- R[rt] = Mem[R[rs] + SignExt[imm16]]
  Example: `lw rt,rs,imm16`



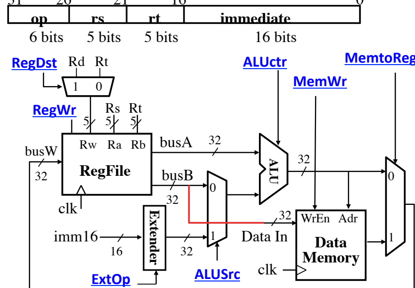
3e: Store Operations

- Mem[R[rs] + SignExt[imm16]] = R[rt]
  Ex.: `sw rt, rs, imm16`



3e: Store Operations

- Mem[R[rs] + SignExt[imm16]] = R[rt]
  Ex.: `sw rt, rs, imm16`



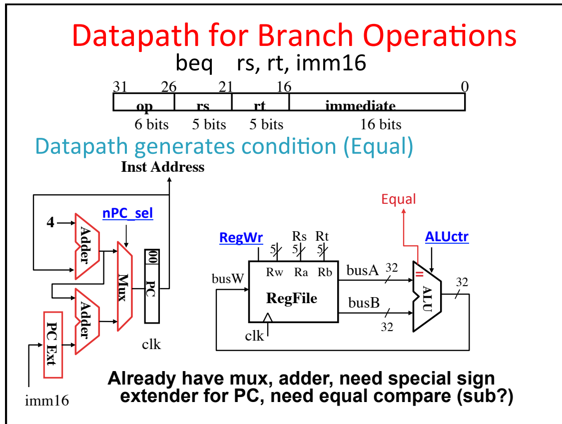
3f: The Branch Instruction

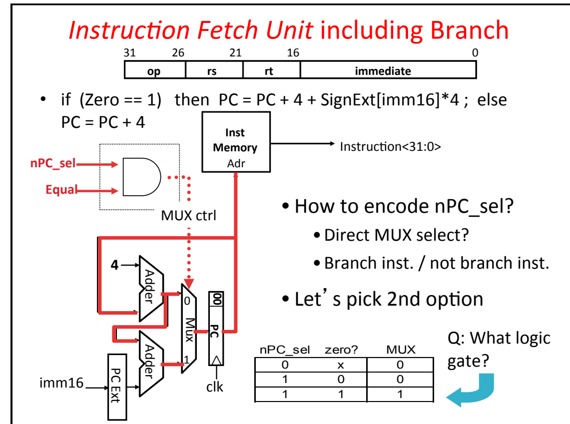
31 op	26 rs	21 rt	16 immediate 0
6 bits	5 bits	5 bits	16 bits

`beq rs, rt, imm16`

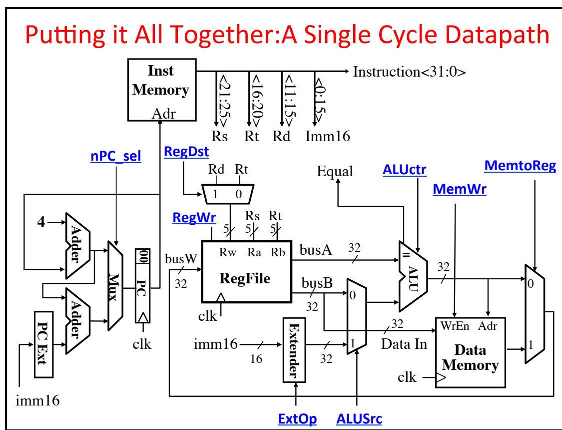
- mem[PC] Fetch the instruction from memory
- Equal = R[rs] == R[rt] Calculate branch condition
- if (Equal) Calculate the next instruction's address
  - PC = PC + 4 + (SignExt(imm16) x 4)

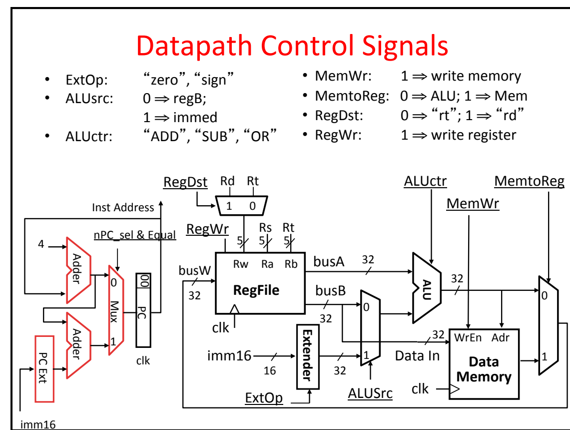
  else
  - PC = PC + 4

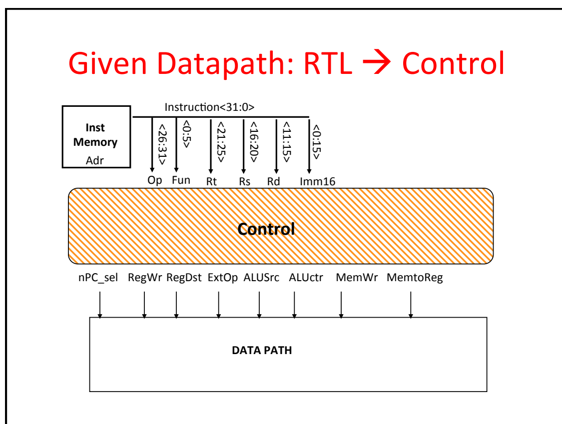


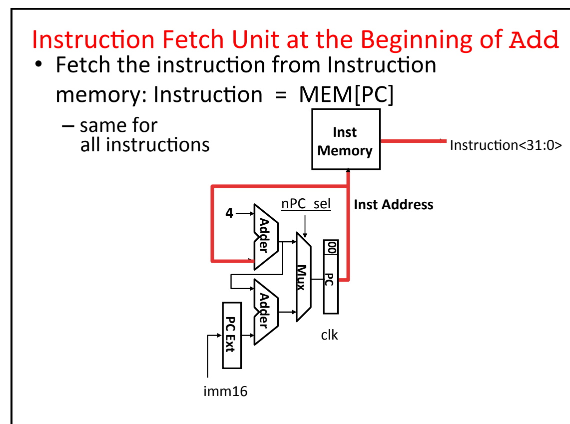


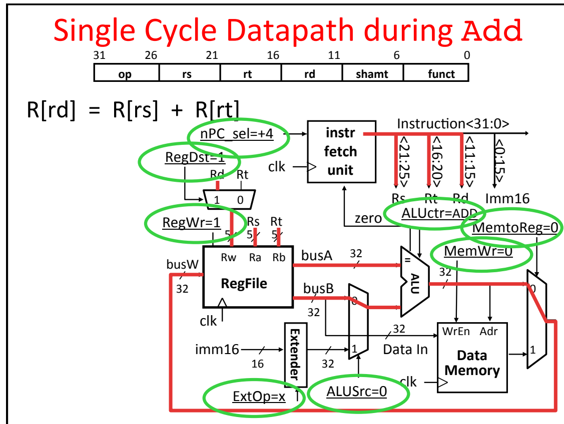
nPC_sel	zero?	MUX
0	x	0
1	0	0
1	1	1

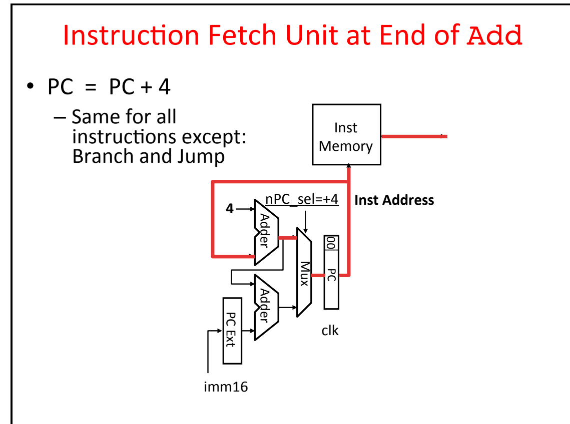


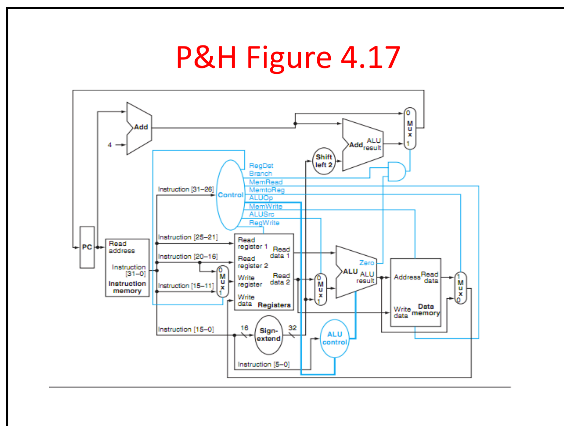












Summary of the Control Signals (1/2)

```
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	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

	31–26	25–21	20–16	15–11	10–6	5–0	
R-type	op	rs	rt	rd	shamt	funct	add, sub
I-type	op	rs	rt	immediate			ori, lw, sw, beq
J-type	op	target address					jump

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:

$rtype = \sim op_5 \bullet \sim op_4 \bullet \sim op_3 \bullet \sim op_2 \bullet \sim op_1 \bullet \sim op_0$,

$ori = \sim op_5 \bullet \sim op_4 \bullet op_3 \bullet op_2 \bullet \sim op_1 \bullet op_0$

$lw = op_5 \bullet \sim op_4 \bullet \sim op_3 \bullet \sim op_2 \bullet op_1 \bullet op_0$

$sw = op_5 \bullet \sim op_4 \bullet op_3 \bullet \sim op_2 \bullet op_1 \bullet op_0$

$beq = \sim op_5 \bullet \sim op_4 \bullet \sim op_3 \bullet op_2 \bullet \sim op_1 \bullet \sim op_0$

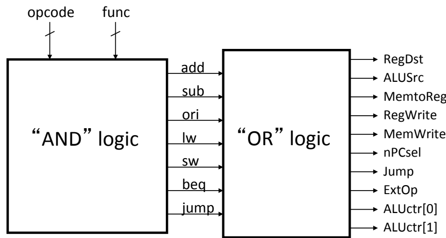
$jump = \sim op_5 \bullet \sim op_4 \bullet \sim op_3 \bullet \sim op_2 \bullet op_1 \bullet \sim op_0$

How do we implement this in gates?

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

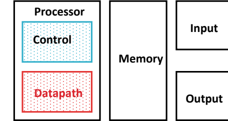
$sub = rtype \bullet func_5 \bullet \sim func_4 \bullet \sim func_3 \bullet \sim func_2 \bullet func_1 \bullet \sim func_0$

Controller Implementation



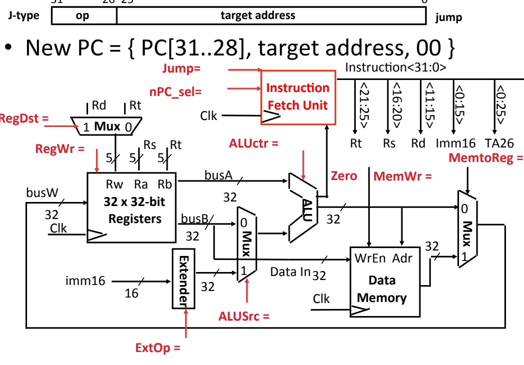
Summary: Single-cycle Processor

- 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



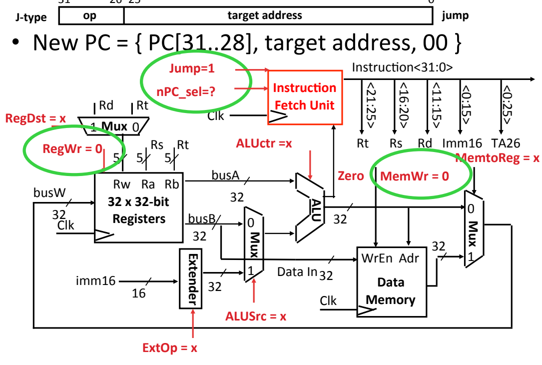
Single Cycle Datapath during Jump

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



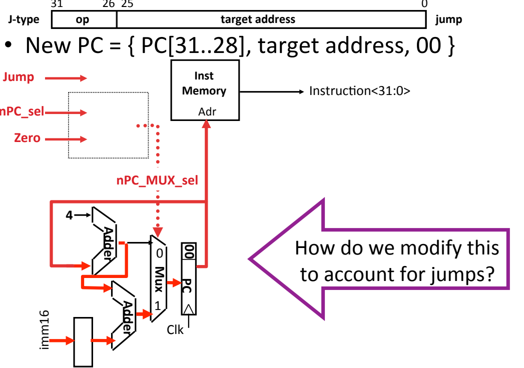
Single Cycle Datapath during Jump

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



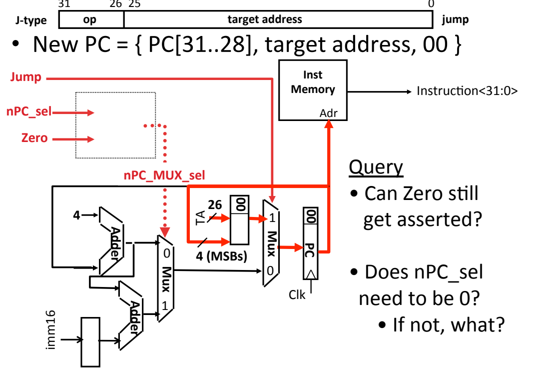
Instruction Fetch Unit at the End of Jump

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



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?