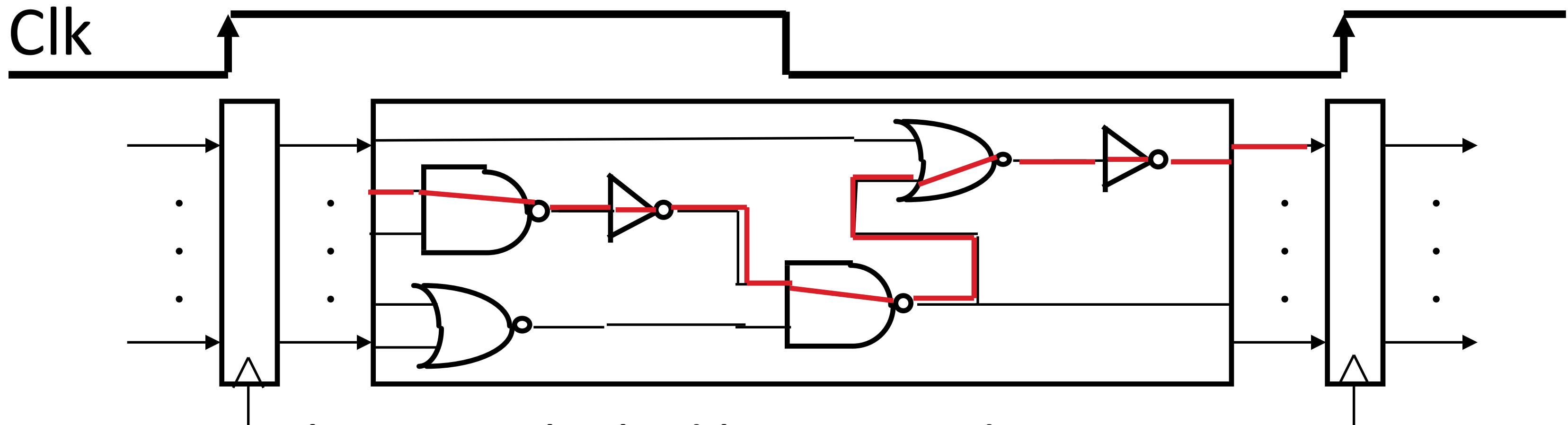
CS 61C:
Great Ideas in Computer Architecture
Single-Cycle CPU
Datapath and Control II

Instructors:

Nicholas Weaver & Vladimir Stojanovic

http://inst.eecs.Berkeley.edu/~cs61c/sp16

Clocking Methodology



- Storage elements clocked by same edge
- Flip-flops (FFs) and combinational logic have some delays
  - Gates: delay from input change to output change
  - Signals at FF D input must be stable before active clock edge to allow signal to travel within the FF (set-up time), and we have the usual clock-to-Q delay
- "Critical path" (longest path through logic) determines length of clock period

Register-Register Timing: One Complete Cycle (Add/Sub)

Clk

PC Old Value

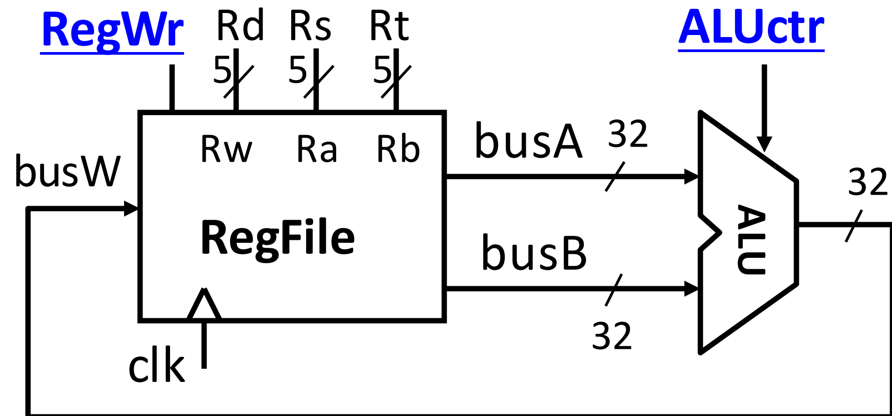
Rs, Rt, Rd,
Op, Func

ALUctr

RegWr

busA, B

busW



Register Write Occurs Here

Register-Register Timing: One Complete Cycle (Add/Sub)

Clk

PC Old Value New Value

Instruction Memory Access Time

Rs, Rt, Rd, Op, Func Old Value New Value

Delay through Control Logic

ALUctr Old Value New Value

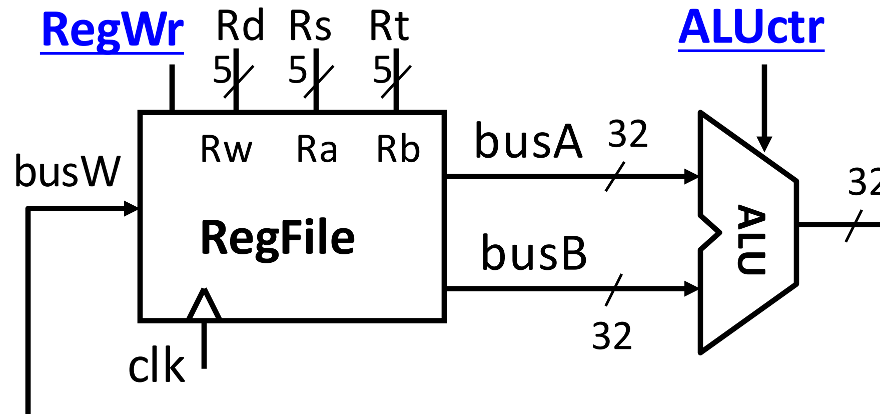
RegWr Old Value New Value

Register File Access Time

busA, B Old Value New Value

ALU Delay

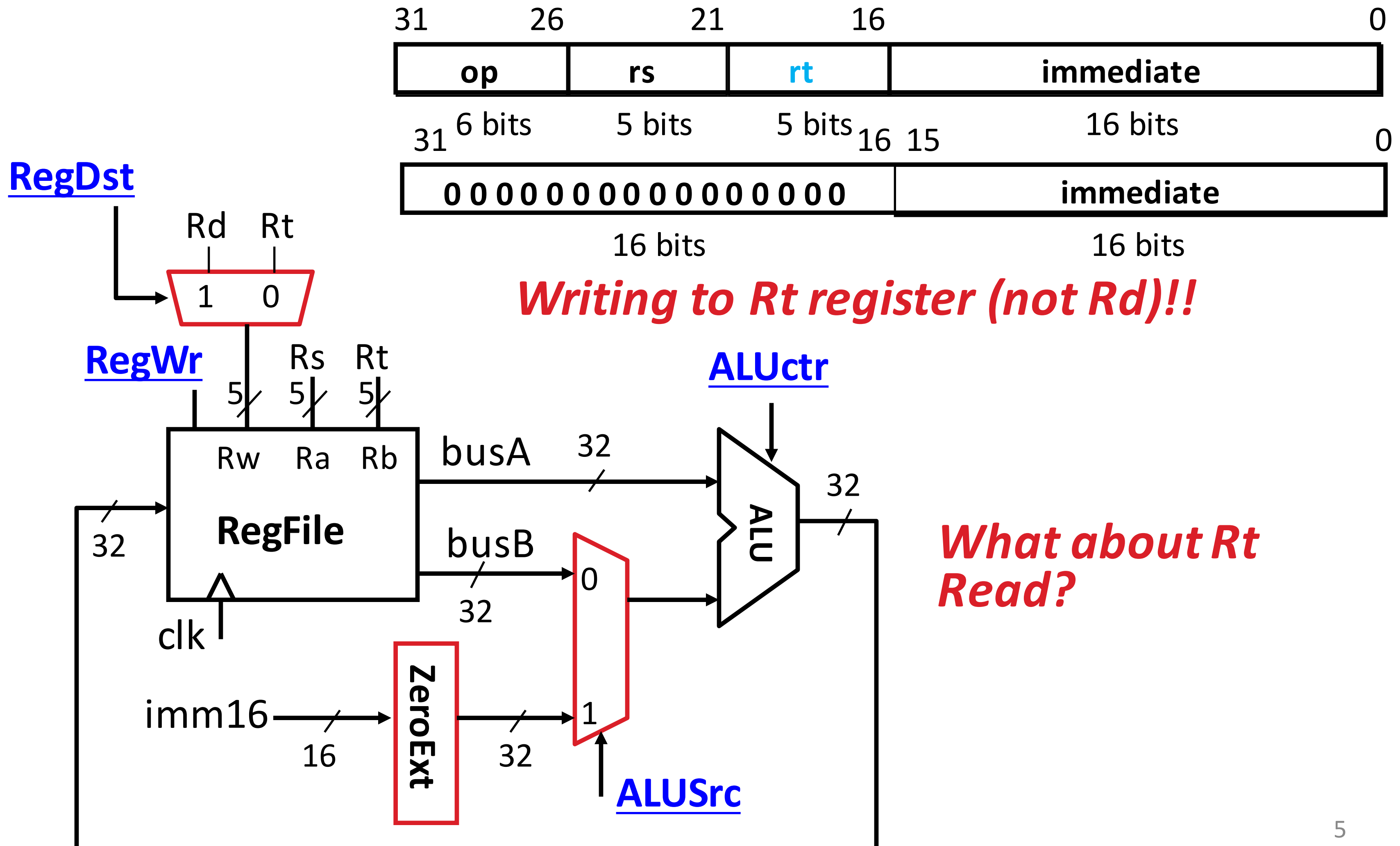
busW Old Value New Value



Register Write Occurs Here

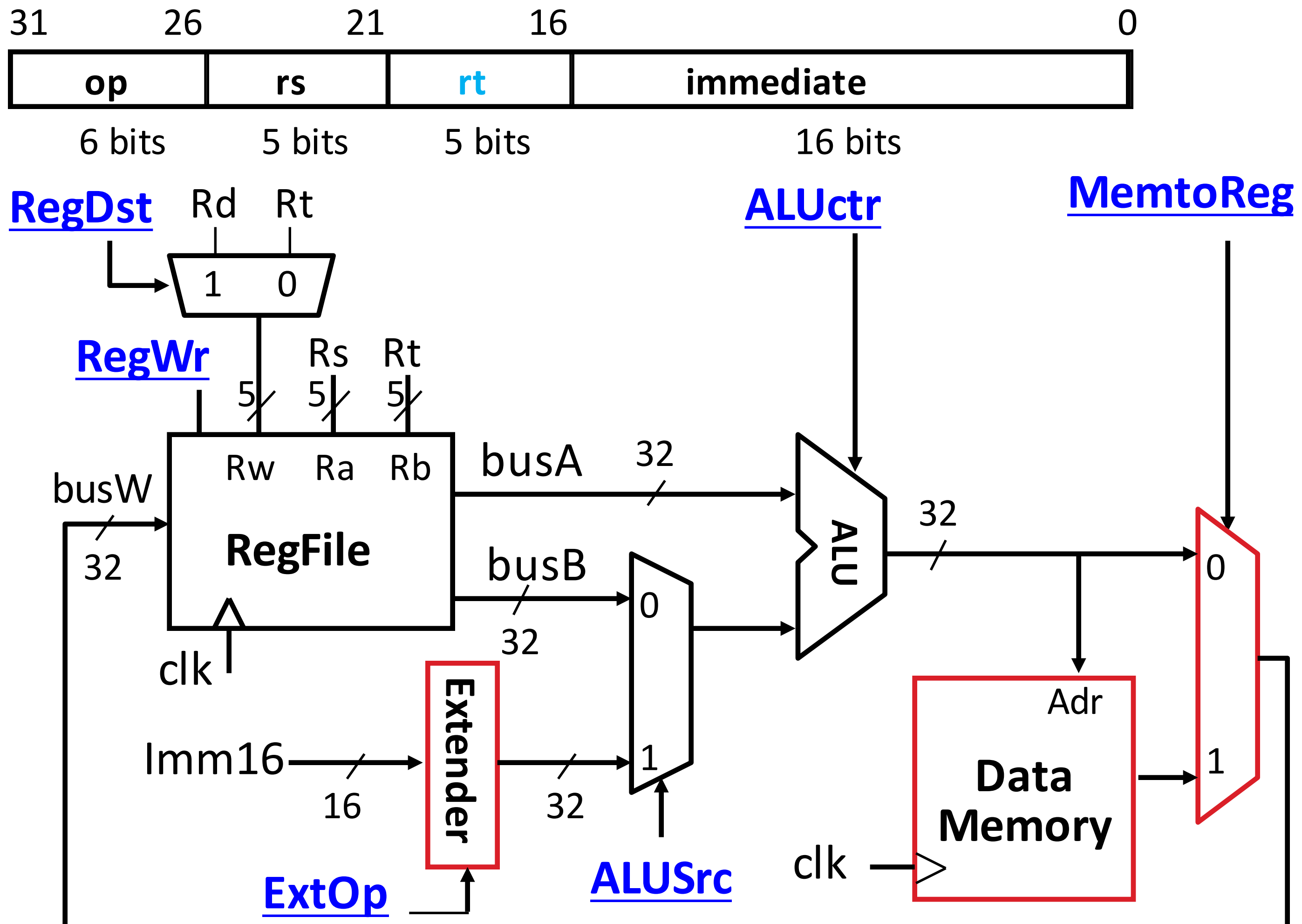
3c: Logical Op (or) with Immediate

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



3d: Load Operations

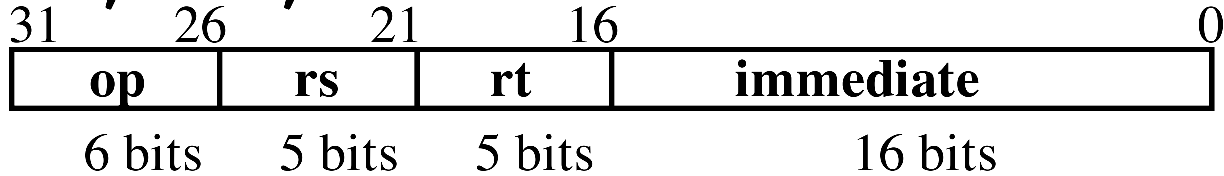
- R[rt] = Mem[R[rs] + SignExt[imm16]]
  Example: `lw rt,rs,imm16`



3e: Store Operations

- Mem[R[rs] + SignExt[imm16]] = R[rt]

  Ex.: `sw rt, rs, imm16`

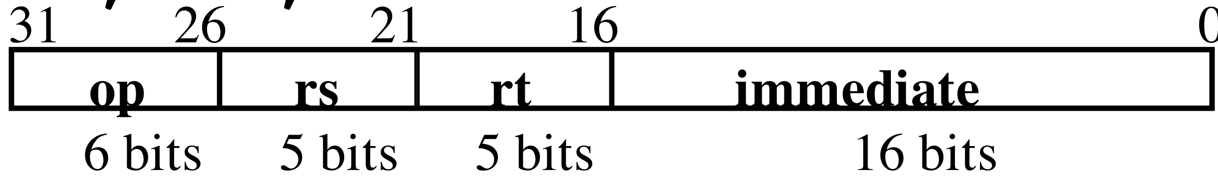


RegDst
Rd
Rt
1
0
RegWr
Rs
Rt
5
5
5
Rw
Ra
Rb
busW
32
RegFile
clk
busA
32
busB
32
ALUctr
ALU
32
MemWr
MemtoReg
0
1
0
1
Imm16
16
Extender
32
ExtOp
ALUSrc
32
Data_In
WrEn
Adr
Data Memory
clk

3e: Store Operations

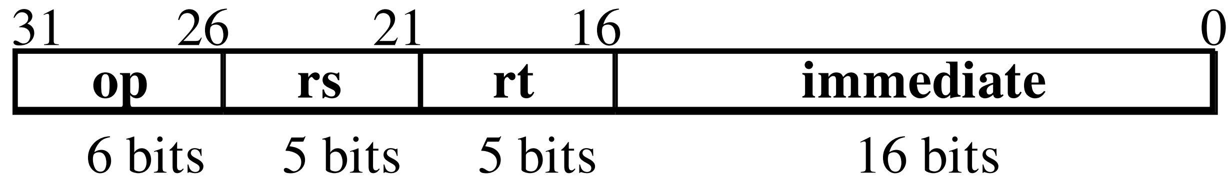
- Mem[R[rs] + SignExt[imm16]] = R[rt]

Ex.: `sw rt, rs, imm16`



RegDst Rd Rt 1 0
RegWr Rs Rt 5 5 5
Rw Ra Rb
RegFile
busW 32
clk
busA 32
busB 32
ALUctr
ALU
32
MemWr
MemtoReg
0 1
imm16 16
Extender 32
ExtOp
ALUSrc
32
Data In
WrEn Adr
Data Memory
clk

3f: The Branch Instruction




```
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

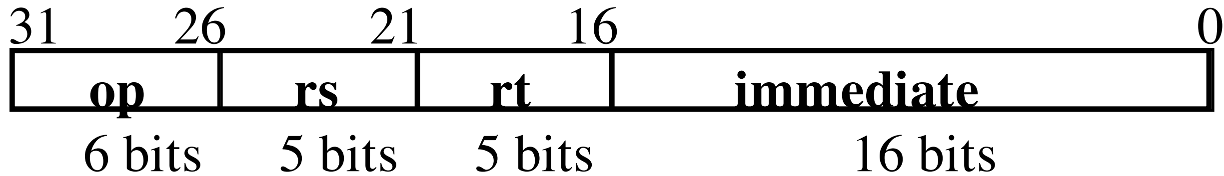
- PC = PC + 4 + (SignExt(imm16) x 4)

else

- PC = PC + 4

Datapath for Branch Operations

beq   rs, rt, imm16

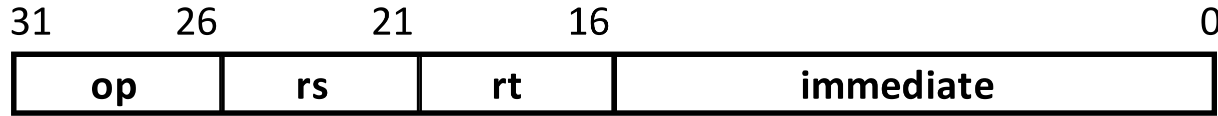


Datapath generates condition (Equal)

Inst Address
4
Adder
nPC_sel
Mux
00
PC
Adder
PC Ext
clk
Imm16
RegWr
Rs
Rt
5
5
5
Rw
Ra
Rb
busW
RegFile
clk
busA
32
busB
32
Equal
ALUctr
=
ALU
32

Already have mux, adder, need special sign extender for PC, need equal compare (sub?)

Instruction Fetch Unit including Branch



- if (Equal == 1) then PC = PC + 4 + SignExt[imm16]*4 ; else PC = PC + 4

Inst Memory Adr

Instruction<31:0>

nPC_sel

Equal

MUX_ctrl

4

Adder

Adder

Mux

0

1

PC

00

clk

imm16

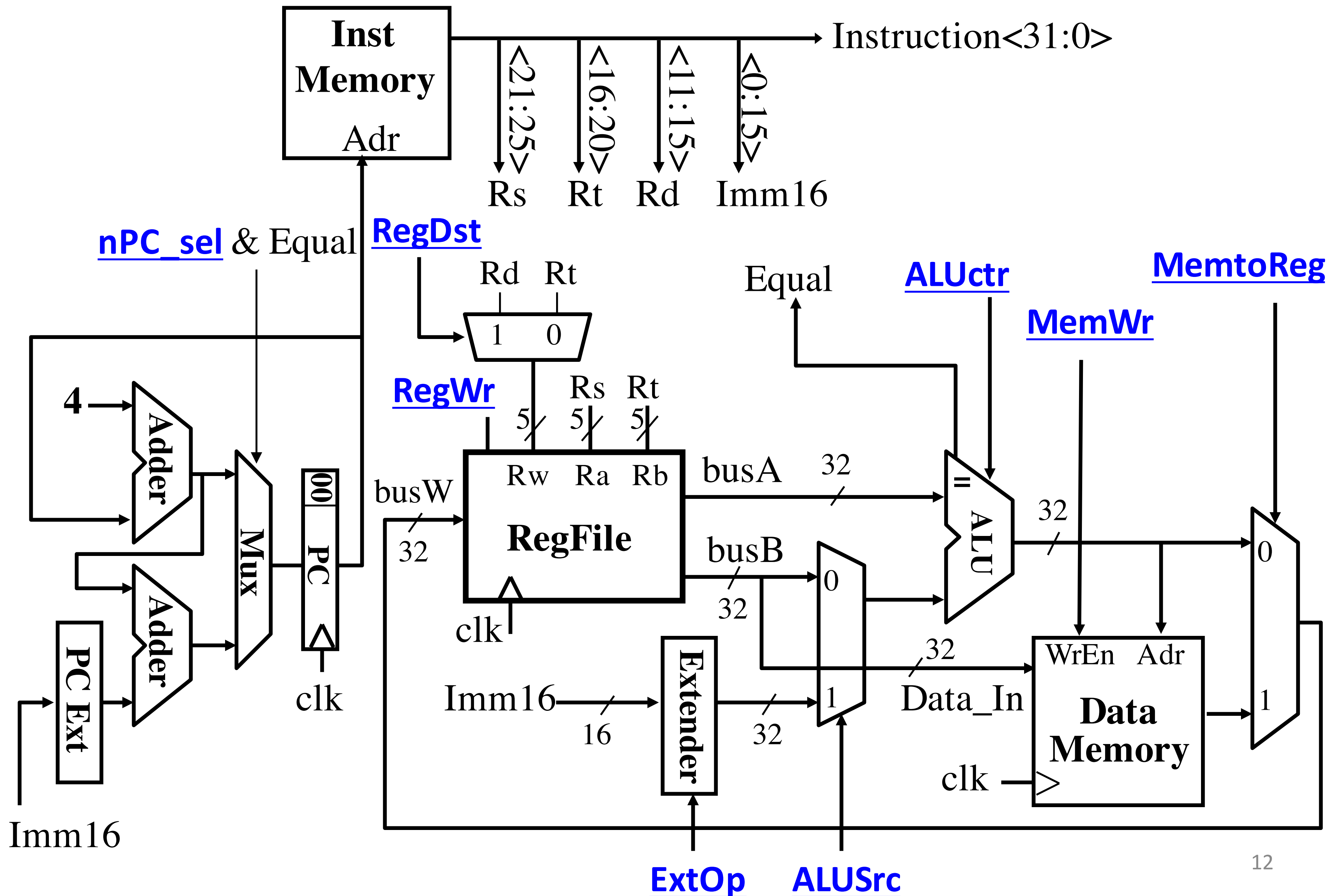
PC Ext

- How to encode nPC_sel?
  - Direct MUX select?
  - Branch inst. / not branch inst.
- Let's pick 2nd option

nPC_sel	Equal	MUX_ctrl
0	x	0
1	0	0
1	1	1

Q: What logic gate?

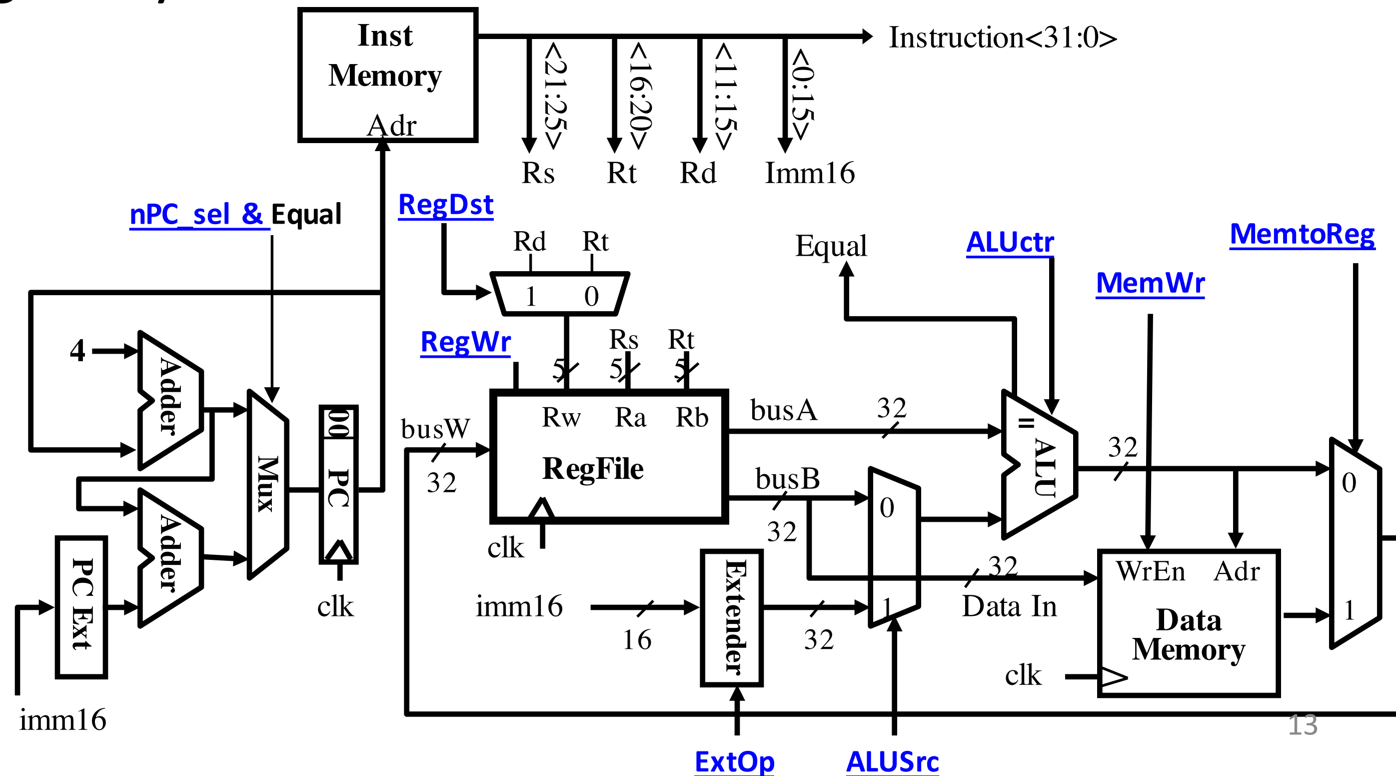
Putting it All Together:A Single Cycle Datapath



Clickers/Peer Instruction

What new instruction would need no new datapath hardware?

- A: branch if reg==immediate
- B: add two registers and branch if result zero
- C: store with auto-increment of base address:
  - sw rt, rs, offset // rs incremented by offset after store
- D: shift left logical by two bits



Administrivia

- Project 2-2 released, due 3/08 (a Tuesday!)
- Guerrilla Session:
  - Synchronous Digital Systems
  - Sat 3/05 1 - 3 PM @ 521 Cory

Processor Design: 5 steps

Step 1: Analyze instruction set to determine datapath requirements

- Meaning of each instruction is given by register transfers
- Datapath must include storage element for ISA registers
- Datapath must support each register transfer

Step 2: Select set of datapath components & establish clock methodology

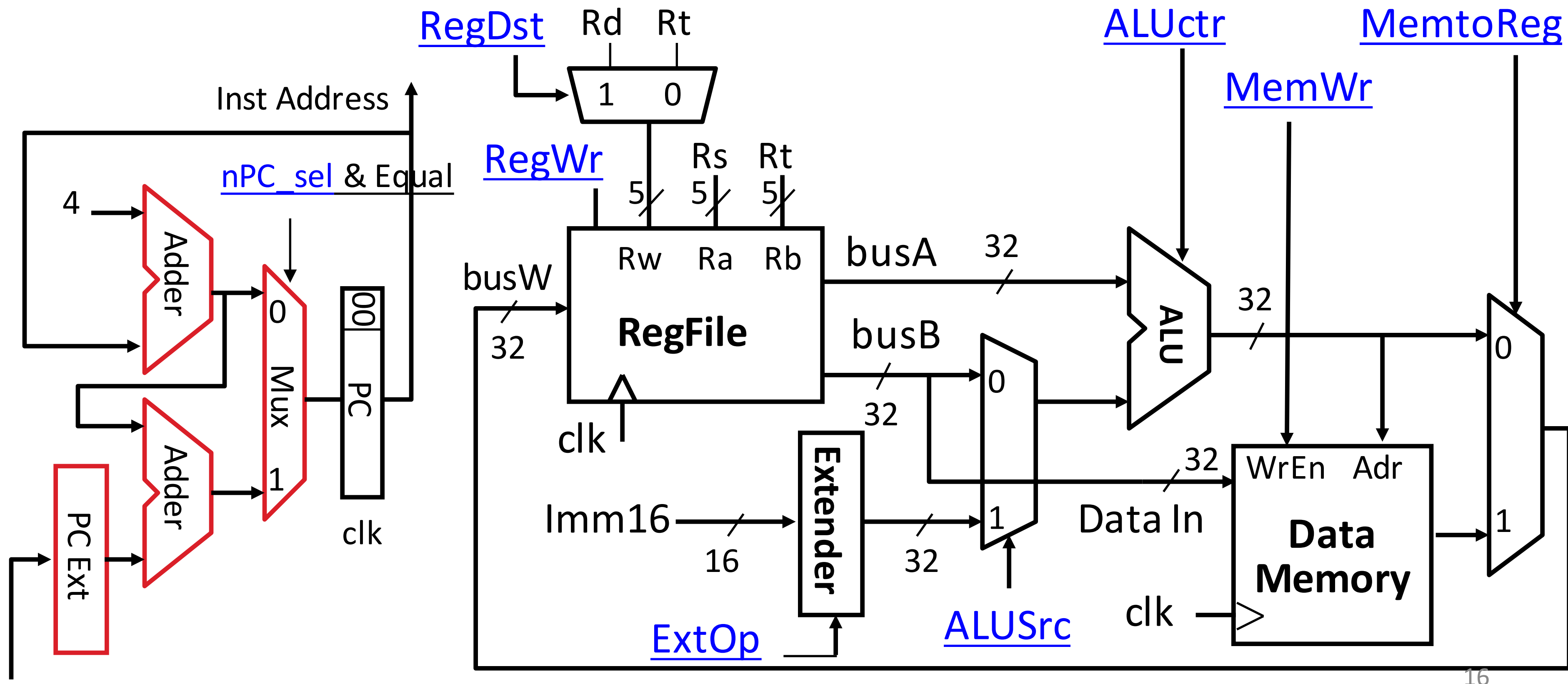
Step 3: Assemble datapath components that meet the requirements

Step 4: Analyze implementation of each instruction to determine setting of control points that realizes the register transfer

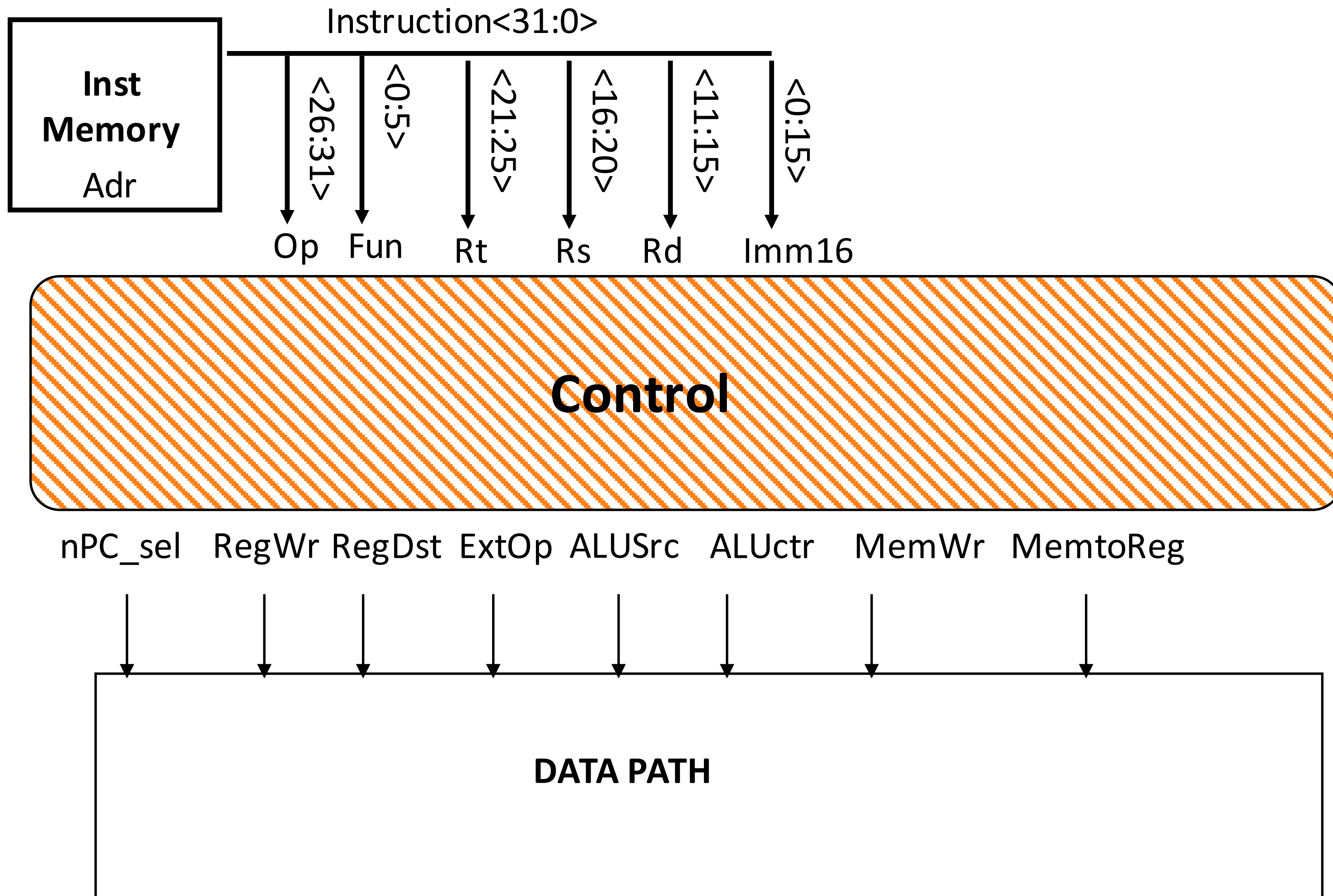
Step 5: Assemble the control logic

Datapath Control Signals

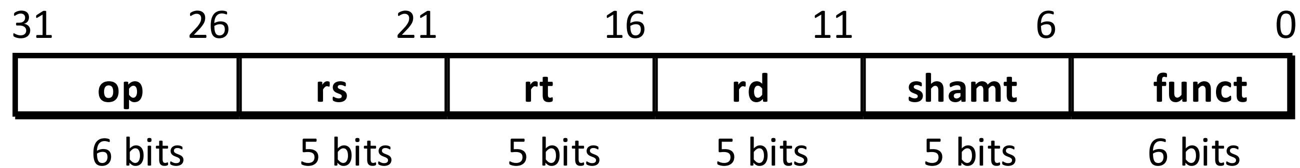
- ExtOp: "zero", "sign"
- ALUsrc: 0 ⇒ regB; 1 ⇒ immed
- ALUctr: "ADD", "SUB", "OR"
- MemWr: 1 ⇒ write memory
- MemtoReg: 0 ⇒ ALU; 1 ⇒ Mem
- RegDst: 0 ⇒ "rt"; 1 ⇒ "rd"
- RegWr: 1 ⇒ write register



Given Datapath: RTL → Control



RTL: The Add Instruction

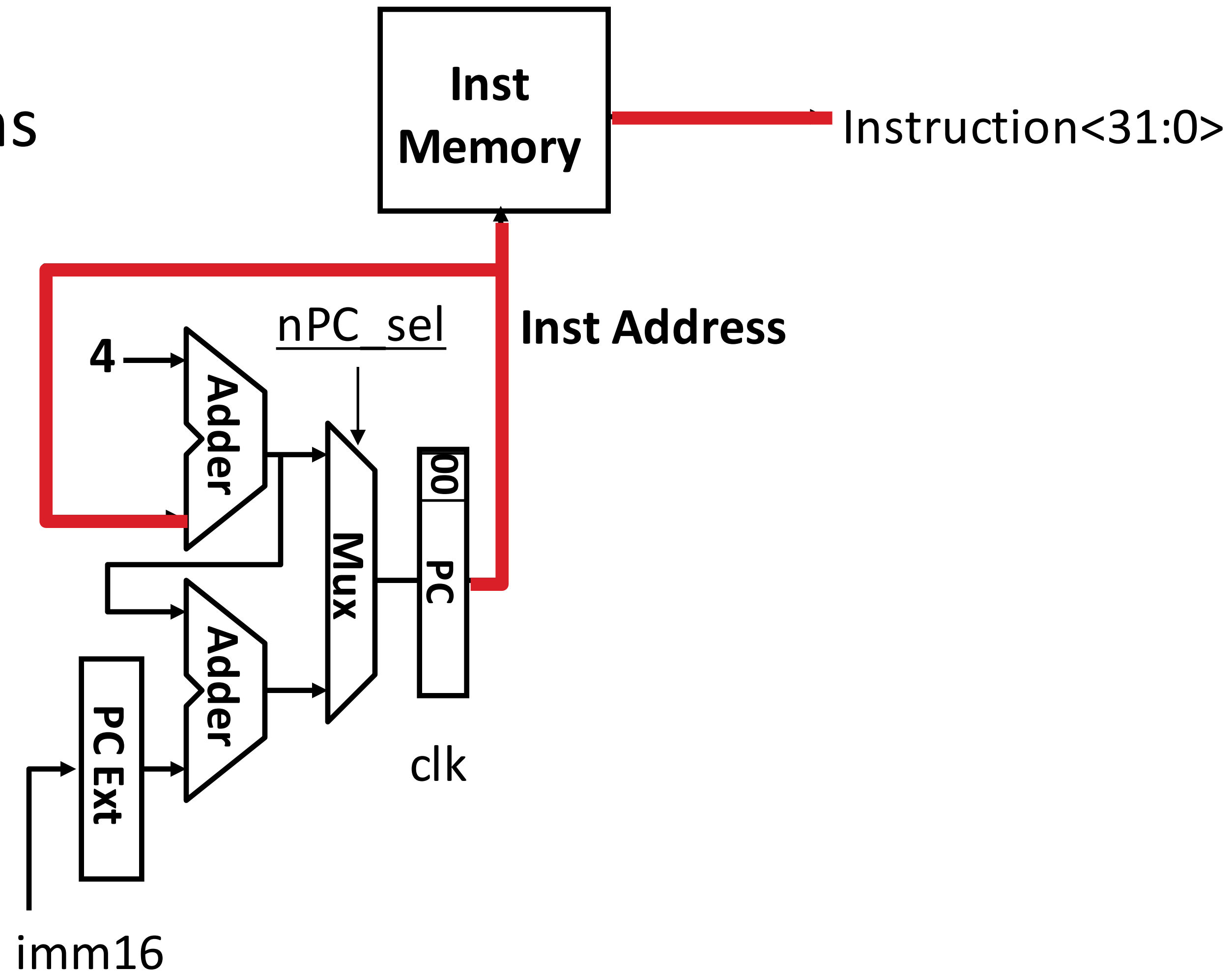



```
add rd, rs, rt
```

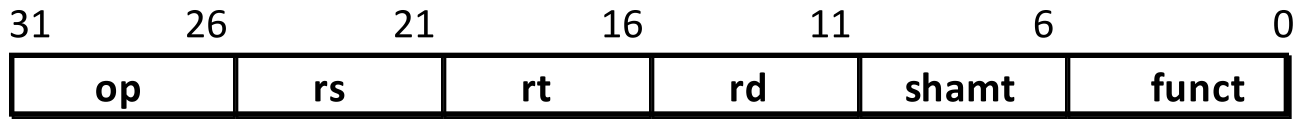
- MEM[PC] Fetch the instruction from memory
- R[rd] = R[rs] + R[rt] The actual operation
- PC = PC + 4 Calculate the next instruction's address

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

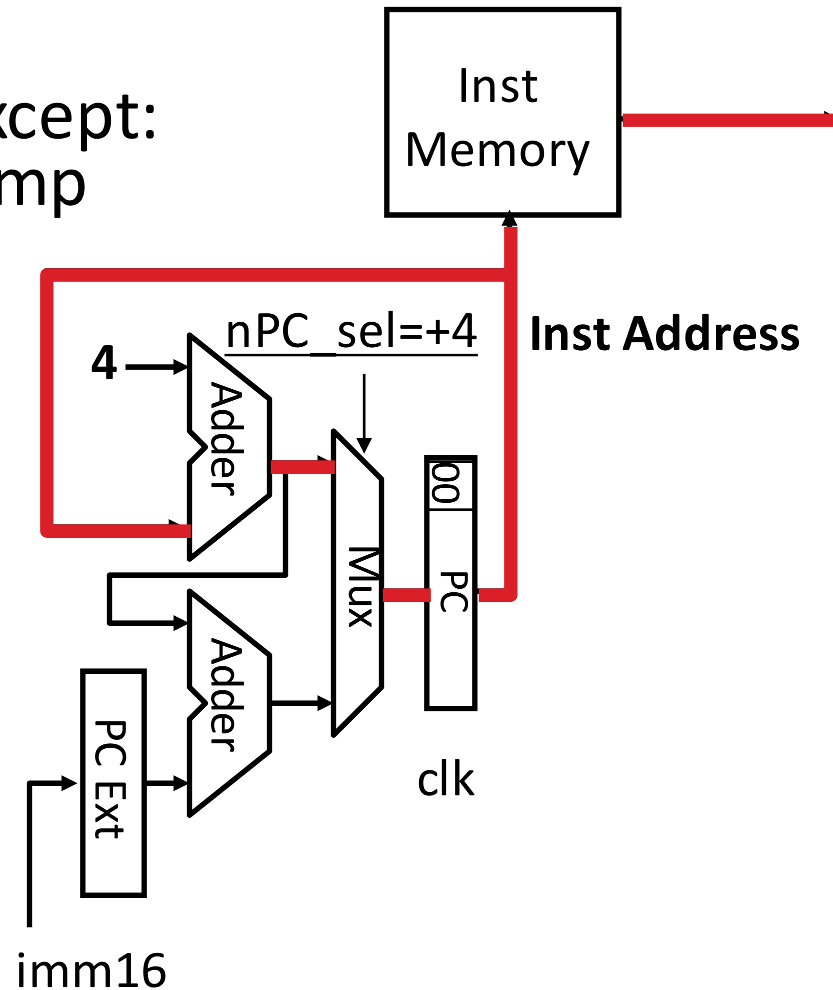


R[rd] = R[rs] + R[rt]

nPC_sel=+4
instr fetch unit
Instruction<31:0>
clk
RegDst=1
Rd Rt
1 0
<21:25>
<16:20>
<11:15>
<0:15>
Rs Rt Rd Imm16
RegWr=1
Rs Rt
5 5 5
zero
ALUctr=ADD
MemtoReg=0
MemWr=0
busW
Rw Ra Rb
RegFile
busA
32
busB
32
32
ALU
=
32
0
1
0
1
clk
imm16
16
Extender
32
32
WrEn Adr
Data In
Data Memory
clk
ExtOp=x
ALUSrc=0

Instruction Fetch Unit at End of Add

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



Single Cycle Datapath during Jump

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

Jump=
nPC_sel=
Instruction Fetch Unit
Clk
Instruction<31:0>
<21:25>
<16:20>
<11:15>
<0:15>
<0:25>
Rs
Rt
Rd
Imm16
TA26
RegDst =
Rd
Rt
1 Mux 0
RegWr =
5
5
5
Rs
Rt
ALUctr =
Zero
MemWr =
MemtoReg =
busA
32
busW
Rw Ra Rb
32 x 32-bit Registers
32
Clk
busB
32
0
Mux
1
ALU
32
0
Mux
1
32
WrEn Adr
Data Memory
Data In
32
Clk
imm16
16
Extender
32
ALUSrc =
ExtOp =

Single Cycle Datapath during Jump

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

Jump=1
nPC_sel=?
Instruction Fetch Unit
Clk
Instruction<31:0>
<21:25>
<16:20>
<11:15>
<0:15>
<0:25>
Rs
Rt
Rd
Imm16
TA26
RegDst = x
Rd
Rt
1 Mux 0
RegWr = 0
5
5
5
Rs
Rt
ALUctr =x
Zero
MemWr = 0
MemtoReg = x
busW
32
Clk
Rw
Ra
Rb
32 x 32-bit Registers
busA
32
busB
32
0
Mux
1
ALU
32
0
Mux
1
32
imm16
16
Extender
32
Data In
32
WrEn
Adr
Data Memory
Clk
ALUSrc = x
ExtOp = x

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?

Clicker Question

Which of the following is TRUE?

A. The clock can have a shorter period for instructions that don't use memory

B. The ALU is used to set PC to PC+4 when necessary

C. Worst-delay path in Instruction Fetch unit is Add+mux delay

D. The CPU's control needs only *opcode* to determine the next PC value to select

E. npc_sel affects the next PC address on a *jump*

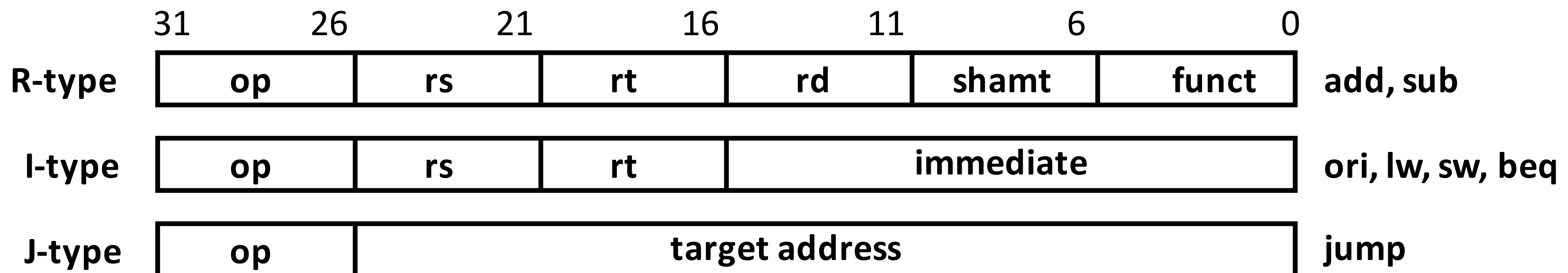
Summary of the Control Signals (1/2)

```
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



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 = $\sim op_5 \cdot \sim op_4 \cdot \sim op_3 \cdot op_2 \cdot \sim op_1 \cdot \sim op_0$

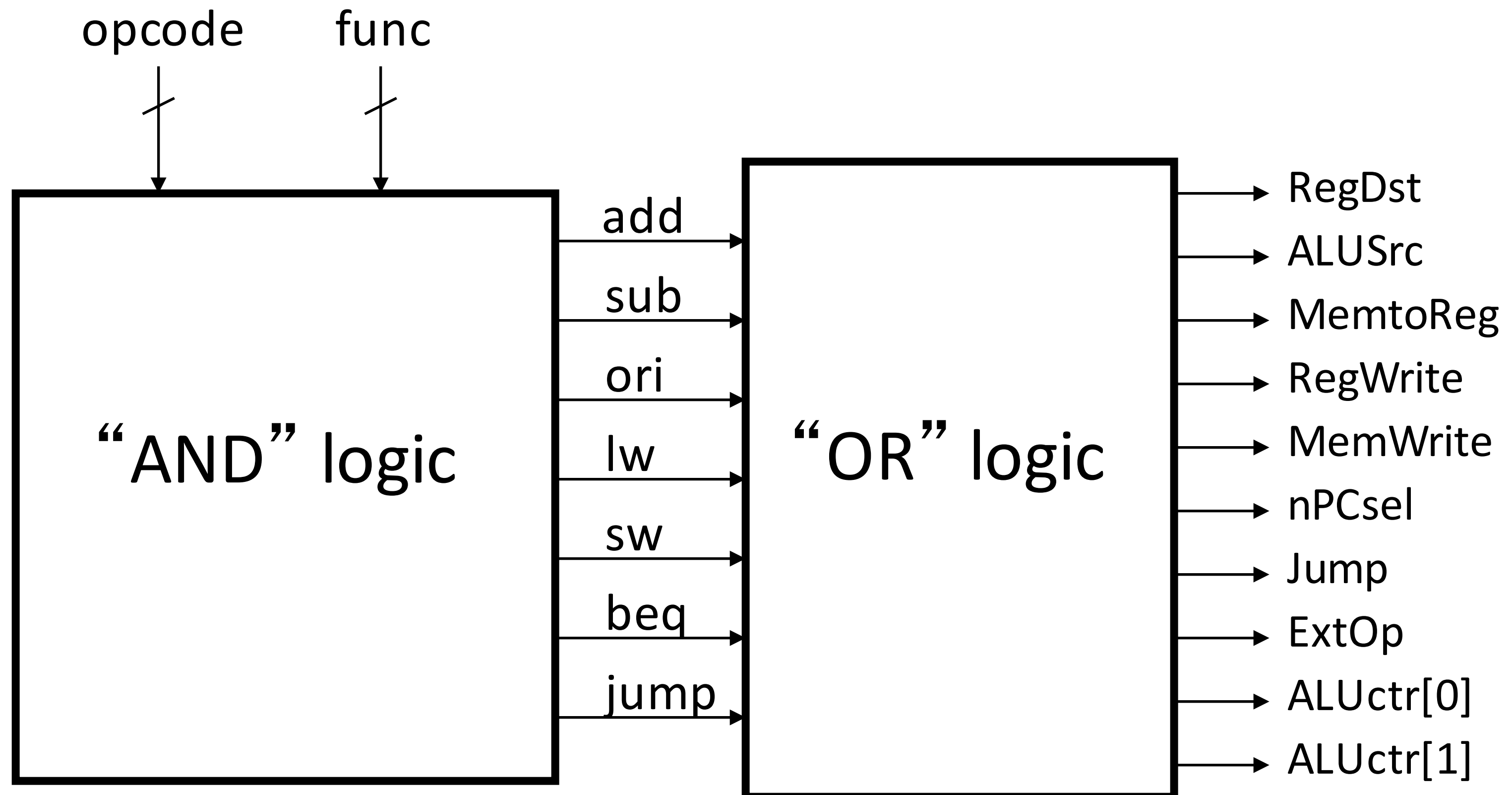
jump = $\sim op_5 \cdot \sim op_4 \cdot \sim op_3 \cdot \sim op_2 \cdot op_1 \cdot \sim op_0$

How do we implement this in gates?

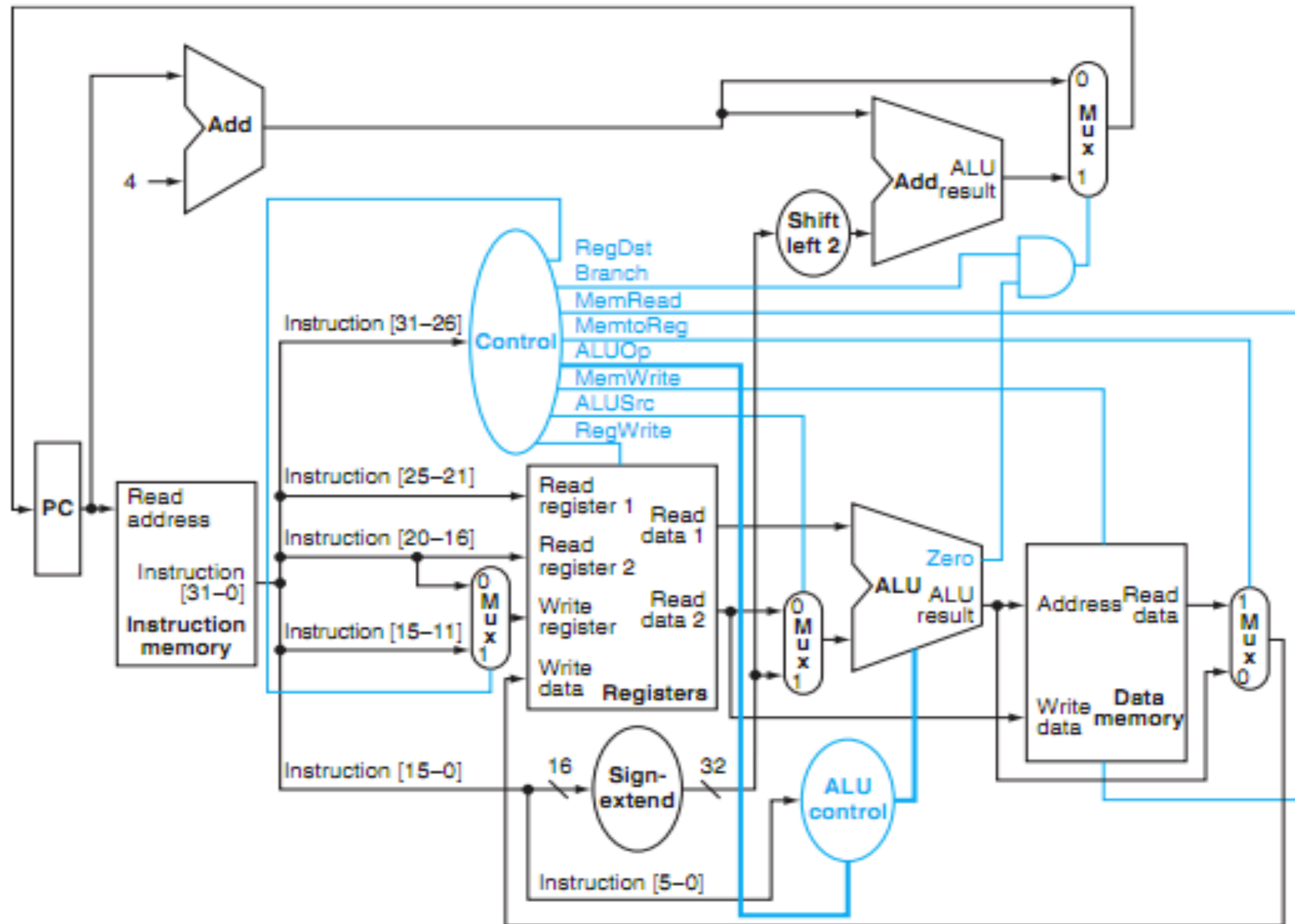
add = $rtype \cdot func_5 \cdot \sim func_4 \cdot \sim func_3 \cdot \sim func_2 \cdot \sim func_1 \cdot \sim func_0$

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

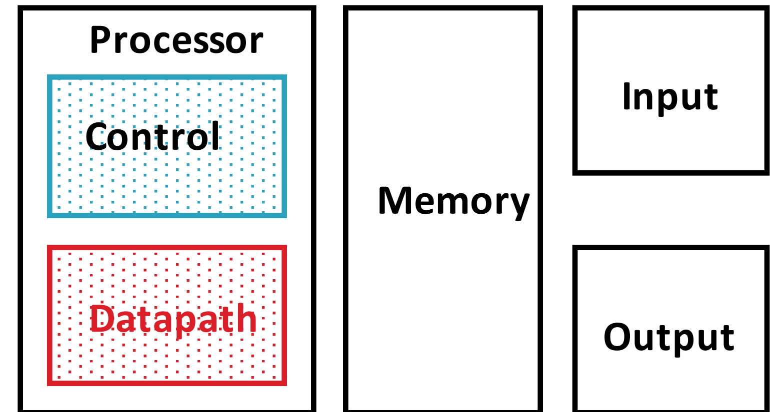


P&H Figure 4.17



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 Performance

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

- Clock rate (cycles/second = Hz) = 1/Period (seconds/cycle)

Single Cycle Performance

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

Levels of Representation/Interpretation

High Level Language Program (e.g., C)

Compiler

Assembly Language Program (e.g., MIPS)

Assembler

Machine Language Program (MIPS)

Machine Interpretation

Hardware Architecture Description (e.g., block diagrams)

Architecture Implementation

Logic Circuit Description (Circuit Schematic Diagrams)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

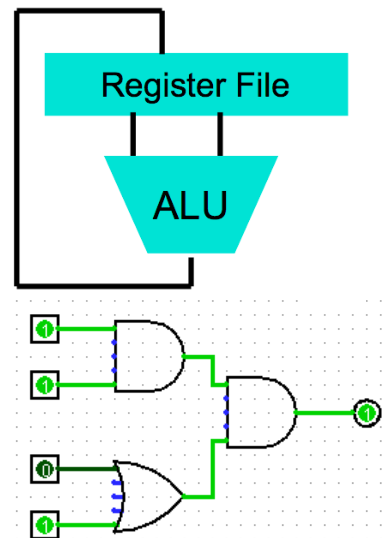
```
lw   $t0, 0($2)
lw   $t1, 4($2)
sw   $t1, 0($2)
sw   $t0, 4($2)
```

Anything can be represented as a *number*, i.e., data or instructions

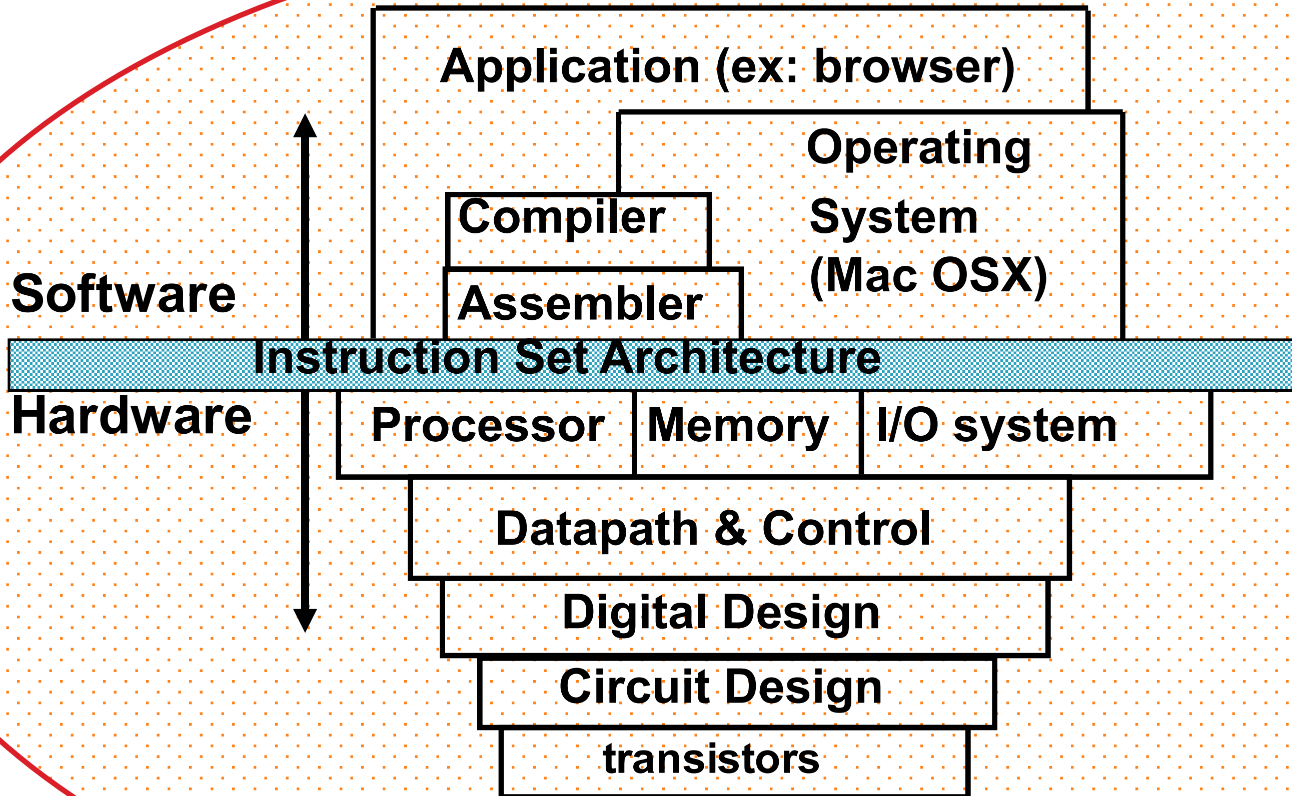
```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Register File

ALU



No More Magic!



CS61A
CS61B
CS61C ✓
CS61C ✓
CS61C ✓
CS61C ←
CS61C ✓
EE40
Phys 7B