### Recap: Meaning of the Control Signals

- **nPC_sel**: 
  - “+4” 0 \(\Rightarrow\) PC \(\leftarrow\) PC + 4  
  - “br” 1 \(\Rightarrow\) PC \(\leftarrow\) PC + 4 + \{SignExt(Im16), 00\}  
- Later in lecture: higher-level connection between mux and branch condition

### RTL: The Add Instruction

```
add rd, rs, rt
- MEM[PC] Fetch the instruction from memory
- 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

### Review: A Single Cycle Datapath

- We have everything except control signals

```
  instr fetch unit
  Instruction<31:0>
  ALU src
  ExtOp
  ALUsrc
  0 \Rightarrow regB; 1 \Rightarrow immed  
  ALU ctr: “ADD”, “SUB”, “OR”  
  MemWr: 1 \Rightarrow write memory  
  MemtoReg: 0 \Rightarrow ALU; 1 \Rightarrow Mem  
  RegDst: 0 \Rightarrow “rt”; 1 \Rightarrow “rd”  
  RegWr: 1 \Rightarrow write register
```
The Single Cycle Datapath during \texttt{Add}

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

- R[rt] = Data Memory \{ R[rs] + SignExt[imm16] \}
- R[rt] = R[rs] \text{ OR } ZeroExt[Imm16]

Instruction Fetch Unit at the End of \texttt{Add}

- PC = PC + 4
- This is the same for all instructions except: Branch and Jump

Single Cycle Datapath during Or Immediate?

- R[rt] = R[rs] \text{ OR } ZeroExt[Imm16]

The Single Cycle Datapath during Load?

- R[rt] = Data Memory \{ R[rs] + SignExt[Imm16] \}
The Single Cycle Datapath during Store?

- Data Memory \((R[rs] + \text{SignExt}[imm16]) = R[rt]\)

\[ \text{Zero} = \begin{cases} 1 & \text{if } (R[rs] - R[rt] == 0) \\ 0 & \text{otherwise} \end{cases} \]

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

The Single Cycle Datapath during Branch?

- if \((R[rs] - R[rt] == 0)\) then \(\text{Zero} = 1\); else \(\text{Zero} = 0\)

\[ \text{Zero} = \begin{cases} 1 & \text{if } (R[rs] - R[rt] == 0) \\ 0 & \text{otherwise} \end{cases} \]

Instruction Fetch Unit at the End of Branch

- if \((\text{Zero} == 1)\) then \(\text{PC} = \text{PC} + 4 + \text{SignExt}[imm16]*4\); else \(\text{PC} = \text{PC} + 4\)

**What is encoding of nPC_sel?**
- Direct MUX select?
- Branch inst. / not branch
- Let’s pick 2nd option

**Q: What logic gate?**

Administrivia

- Sorry about Proj1 woes
- Grading is rough stuff. Don’t blame Ben, he’s innocent.
- HW6 Due imminently! Students have claimed it takes a very long time
  - Remember MODULAR DESIGN. This could save you a lot of time.
- HW7 Out Now! Get started soon.
- Proj3 is on its way, will be out soon after the weekend.
Appendix A

R-type
J-type
I-type

Controller Implementation

A Summary of the Control Signals (1/2)

inst Register Transfer
add \( R[d] \leftarrow R[s] + R[t] \); \( PC \leftarrow PC + 4 \)
aluUse = RegB, aluCtrl = "ADD", regDst = rd, regWr, nPC_sel = "+4"
sub \( R[d] \leftarrow R[s] - R[t] \); \( PC \leftarrow PC + 4 \)
aluUse = RegB, aluCtrl = "SUB", regDst = rd, regWr, nPC_sel = "+4"
ori \( R[d] \leftarrow R[s] \) zeroExtImm16; \( PC \leftarrow PC + 4 \)
aluUse = Imm, extOp = "OR", regDst = rt, regWr, nPC_sel = "+4"
lor \( R[d] \leftarrow MEM[R[s] + signExtImm16]; \( PC \leftarrow PC + 4 \)
aluUse = Imm, extOp = "LOR", memWr, nPC_sel = "+4"
sW \( R[d] \leftarrow MEM[R[s] + signExtImm16]; \( PC \leftarrow PC + 4 \)
aluUse = Imm, extOp = "SW", memWr, nPC_sel = "+4"
beq if \( R[s] == R[t] \) then \( PC = PC + 4 \); else \( PC = PC + 4 \)
aluCtrl = "BEQ", aluUse = "SW",

Boolean Expressions for Controller

regDst = add | sub
aluSrc = add | & | lw | sw | beq | ori
MmWrite = add | & | lw | sw | beq
aluCtrl = sub + beq
jump = jump
extOp = lw | sw
aluCtrl(bit) = sub + beq (assume aluCtrl is 0 add, 01 sub, 10 or)
aluCtrl[1] = or

where,

- \( rtype \) = \( op \oplus oprs \oplus oprs \oplus oprs \oplus oprs \oplus oprs \)
- \( lw \) = \( oprs \oplus oprs \oplus oprs \oplus oprs \oplus oprs \)
- \( sw \) = \( oprs \oplus oprs \oplus oprs \oplus oprs \oplus oprs \)
- \( beq \) = \( oprs \oplus oprs \oplus oprs \oplus oprs \oplus oprs \)
- \( add \) = \( rtype \oplus func \oplus func \oplus func \oplus func \oplus func \)
- \( or \) = \( rtype \oplus func \oplus func \oplus func \oplus func \oplus func \)
- \( or \) = \( rtype \oplus func \oplus func \oplus func \oplus func \oplus func \)

How do we implement this in gates?

Peer Instruction

A. MemToReg
B. add \( \text{aluCtrl[2]} \). Which 1 signal is different for all 3 of: ADD, LW, & SW? RegDst or ExtOp?
C. “Don’t Care” signals are useful because we can simplify our PLA personality matrix. F / ???
Summary: Single-cycle Processor

- 5 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 affects the register transfer.
  - 5. Assemble the control logic
    - Formulate Logic Equations
    - Design Circuits

Bonus slides

- These are extra slides that used to be included in lecture notes, but have been moved to this, the “bonus” area to serve as a supplement.
- The slides will appear in the order they would have in the normal presentation

The 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 \}

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