Paralyzed people can now control artificial limbs! There are two brain connection techniques, implants requiring significant surgery (& brain inflammation) and the non-invasive “swimming cap”. You adapt to either.

Summary: A Single Cycle Datapath
- Rs, Rt, Rd, imm16 connected to datapath
- We have everything except control signals

Recap: Meaning of the Control Signals
- nPC_MUX_sel: 0 ⇒ PC ← PC + 4
  “n”=next
  1 ⇒ PC ← PC + 4 + {SignExt(Imm16), 00}
- Later in lecture: higher-level connection between mux and branch cond

Administrivia
- Steven & Andy moved today’s OH to 11-noon before lecture
**RTL: The `add` Instruction**

```
31 26 21 16 11 6 0
<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>6 bits</td>
</tr>
</tbody>
</table>
```

`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

---

**The Single Cycle Datapath during `add`**

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

---

**Instruction Fetch Unit at the End of `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] OR ZeroExt[Imm16]**

---

**Single Cycle Datapath during Or Immediate?**

- **R[rt] = R[rs] OR ZeroExt[Imm16]**
The Single Cycle Datapath during Load?

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

The Single Cycle Datapath during Store?

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

The Single Cycle Datapath during Branch?

- If \( (R[rs] - R[rt] = 0) \) then Zero = 1; else Zero = 0
**Instruction Fetch Unit at the End of Branch**

- If \( \text{Zero} = 1 \), then PC = PC + 4 + SignExt(imm16) 
- else PC = PC + 4

- What is encoding of nPC_sel?
  - Direct MUX select?
  - Branch / not branch?

**A Summary of the Control Signals (1/2)**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Register Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>( R[i] \leftarrow R[i] + R[j] )</td>
</tr>
<tr>
<td>ALUsrc</td>
<td>( R[i] )</td>
</tr>
<tr>
<td>ALUctr</td>
<td>( R[i] )</td>
</tr>
<tr>
<td>STORE</td>
<td>( \text{MEM}[R[i]] \leftarrow \text{sign_ext}(\text{imm16}) )</td>
</tr>
</tbody>
</table>

**A Summary of the Control Signals (2/2)**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB</td>
<td>( R[i] \leftarrow R[j] - R[i] )</td>
</tr>
<tr>
<td>ORI</td>
<td>( R[i] \leftarrow R[i] )</td>
</tr>
<tr>
<td>ORI</td>
<td>( \text{sign_ext}(\text{imm16}) )</td>
</tr>
<tr>
<td>LOAD</td>
<td>( \text{MEM}[R[i]] \leftarrow \text{sign_ext}(\text{imm16}) )</td>
</tr>
</tbody>
</table>

**Step 4: Given Datapath: RTL -> Control**

**Summary: Single cycle datapath**

- 5 steps to design a processor
  1. Analyze instruction set \( \Rightarrow \) 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

**Control is the hard part**

MIPS makes that easier
- Instructions same size
- Source registers always in same place
- IMMediates same size, location
- Operations always on registers/immediates

**Peer Instruction**

A. Our **ALU** is a synchronous device
B. We **could have used tri-state devices** instead of a MUX to feed busW, the register write data line
C. The **ALU** is inactive for memory reads or writes.