Recap: The MIPS Subset

- **ADD and subtract**
  - add rd, rs, rt
  - sub rd, rs, rt

- **OR Imm:**
  - ori rt, rs, imm16

- **LOAD and STORE**
  - lw rt, rs, imm16
  - sw rt, rs, imm16

- **BRANCH:**
  - beq rs, rt, imm16

- **JUMP:**
  - j target

```
31  26  21  16  11  6  0
op  rs  rt  rd  shamt  funct
```

- **ADD and subtract**:
  - 6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

- **OR Imm:**
  - 6 bits 5 bits 5 bits 16 bits

- **LOAD and STORE**
  - 6 bits 5 bits 5 bits 16 bits

- **BRANCH:**
  - 6 bits 26 bits

- **JUMP:**
  - 6 bits 26 bits

**Recap: A Single Cycle Datapath**

- We have everything except control signals (underline)
  - Today’s lecture will show you how to generate the control signals

**The Big Picture: Where are We Now?**

- **The Five Classic Components of a Computer**

- **Today’s Topic: Designing the Control for the Single Cycle Datapath**
Outline of Today’s Lecture

° Recap and Introduction
° Control for Register-Register & Or Immediate instructions
° Control signals for Load, Store, Branch, & Jump
° Building a local controller: ALU Control
° The main controller
° Summary

RTL: The ADD Instruction

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

° add rd, rs, rt

- mem[PC] Fetch the instruction from memory
- PC <- PC + 4 Calculate the next instruction’s address

The Single Cycle Datapath during Add and Subtract
The Single Cycle Datapath during Add and Subtract

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

\* \text{R[rd]} \gets \text{R[rs]} + / - \text{R[rt]}

Instruction-fetch Unit at the End of Add and Subtract

\* \text{PC} \gets \text{PC + 4}

- This is the same for all instructions except: Branch and Jump

The Single Cycle Datapath during Or Immediate

<table>
<thead>
<tr>
<th></th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>31</td>
<td>26</td>
<td>21</td>
<td>16</td>
</tr>
</tbody>
</table>

\* \text{R[rt]} \gets \text{R[rs]} \text{ or } \text{ZeroExt}[\text{imm16}]

Instruction-fetch Unit at the End of Or Immediate

<table>
<thead>
<tr>
<th></th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>31</td>
<td>26</td>
<td>21</td>
<td>16</td>
</tr>
</tbody>
</table>
The Single Cycle Datapath during Load

- **R[rt] <- Data Memory (R[rs] + SignExt[imm16])**

- **R[rt] <- Data Memory (R[rs] + SignExt[imm16])**

The Single Cycle Datapath during Store

- **Data Memory (R[rs] + SignExt[imm16]) <- R[rt]**

- **Data Memory (R[rs] + 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** \( (Zero = 1) \) **then** \( PC = PC + 4 + \text{SignExt}[imm16] 	imes 4 \); **else** \( PC = PC + 4 \)

### The Single Cycle Datapath during Jump

- **Nothing to do!** Make sure control signals are set correctly!

### Instruction Fetch Unit at the End of Jump

- \( \text{PC} \leftarrow \text{PC}<31:29> \text{concat} \text{target}<25:0> \text{concat} \text{"00"} \)

---

**Note:** The above diagrams and text illustrate the behavior of the Single Cycle Datapath during Branch and Jump instructions. The diagrams show the flow of data and control signals through the Instruction Fetch Unit. The text provides the necessary conditions and equations for determining the values of the control signals such as RegWr, ALUctr, and MemWr during Branch and Jump operations.
### Step 4: Given Datapath: RTL -> Control

#### Instruction<31:0>

<table>
<thead>
<tr>
<th>Inst Memory</th>
<th>Adr</th>
<th>Op</th>
<th>Fun</th>
<th>Rd</th>
<th>Rs</th>
<th>Rd</th>
<th>Imm16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Control

<table>
<thead>
<tr>
<th>nPCselRegWr</th>
<th>RegDst</th>
<th>ExtOp</th>
<th>ALUSrc</th>
<th>ALUctr</th>
<th>MemtoReg</th>
<th>Equal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### DATA PATH

### A Summary of Control Signals

<table>
<thead>
<tr>
<th>Instruction</th>
<th>BUS</th>
<th>ALUop</th>
<th>ExtOp</th>
<th>Branch</th>
<th>Jump</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td></td>
<td>add</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sub</td>
<td></td>
<td>sub</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Or</td>
<td></td>
<td>or</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td></td>
<td>add</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subtract</td>
<td></td>
<td>sub</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load</td>
<td></td>
<td>load</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Store</td>
<td></td>
<td>store</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### The Concept of Local Decoding

<table>
<thead>
<tr>
<th>op</th>
<th>00 0000</th>
<th>00 0100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00 0000</td>
<td>00 0100</td>
</tr>
<tr>
<td></td>
<td>00 0100</td>
<td>00 0100</td>
</tr>
<tr>
<td></td>
<td>00 0100</td>
<td>00 0100</td>
</tr>
</tbody>
</table>

### Register Transfer

- **ADD**: \( R[rd] \leftarrow R[rs] + R[rt] \); \( \text{PC} \leftarrow \text{PC} + 4 \)
  - \( \text{ALUsrc} = \text{RegB} \), \( \text{ALUctr} = \text{"add"} \), \( \text{RegDst} = \text{rd} \), \( \text{RegWr}, \text{nPCsel} = \text{"+4"} \)
- **SUB**: \( R[rd] \leftarrow R[rs] - R[rt] \); \( \text{PC} \leftarrow \text{PC} + 4 \)
  - \( \text{ALUsrc} = \text{RegB} \), \( \text{ALUctr} = \text{"sub"} \), \( \text{RegDst} = \text{rd} \), \( \text{RegWr}, \text{nPCsel} = \text{"+4"} \)
- **ORI**: \( R[rt] \leftarrow R[rs] + \text{zero_ext}(\text{Imm16}); \) \( \text{PC} \leftarrow \text{PC} + 4 \)
  - \( \text{ALUsrc} = \text{Im} \), \( \text{ExtOp} = \text{"Z"} \), \( \text{ALUctr} = \text{"or"} \), \( \text{RegDst} = \text{rt} \), \( \text{RegWr}, \text{nPCsel} = \text{"+4"} \)
- **LOAD**: \( R[rt] \leftarrow \text{MEM}[R[rs] + \text{sign_ext}(\text{Imm16})]; \) \( \text{PC} \leftarrow \text{PC} + 4 \)
  - \( \text{ALUsrc} = \text{Im} \), \( \text{ExtOp} = \text{"Sn"} \), \( \text{ALUctr} = \text{"add"} \), \( \text{MemtoReg}, \text{RegDst} = \text{rt} \), \( \text{RegWr}, \text{nPCsel} = \text{"+4"} \)
- **STORE**: \( \text{MEM}[R[rs] + \text{sign_ext}(\text{Imm16})] \leftarrow R[rs]; \) \( \text{PC} \leftarrow \text{PC} + 4 \)
  - \( \text{ALUsrc} = \text{Im} \), \( \text{ExtOp} = \text{"St"} \), \( \text{ALUctr} = \text{"add"} \), \( \text{MemWr}, \text{nPCsel} = \text{"+4"} \)
- **BEQ**: if \( \text{R[rs]} == \text{R[rt]} \) then \( \text{PC} \leftarrow \text{PC} + \text{sign_ext}(\text{Imm16}) \) \( \| \) 00 else \( \text{PC} \leftarrow \text{PC} + 4 \)
  - \( \text{nPCsel} = \text{"Br"} , \text{ALUctr} = \text{"sub"} \)
In this exercise, ALUop has to be 2 bits wide to represent:
- (1) "R-type" instructions
- "I-type" instructions that require the ALU to perform:
  - (2) Or, (3) Add, and (4) Subtract

To implement the full MIPS ISA, ALUop has to be 3 bits to represent:
- (1) "R-type" instructions
- "I-type" instructions that require the ALU to perform:
  - (2) Or, (3) Add, (4) Subtract, and (5) And (Example: andi)

The Truth Table for ALUctr

The Logic Equation for ALUctr<2>
The Logic Equation for ALUctr<1>

<table>
<thead>
<tr>
<th>ALUop</th>
<th>func</th>
<th>ALUctr&lt;1&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit&lt;2&gt;</td>
<td>bit&lt;1&gt;</td>
<td>bit&lt;0&gt;</td>
</tr>
<tr>
<td>0 0 0</td>
<td>x x x x</td>
<td>1</td>
</tr>
<tr>
<td>0 x 1</td>
<td>x x x x</td>
<td>1</td>
</tr>
<tr>
<td>1 x x</td>
<td>0 0 1 0</td>
<td>1</td>
</tr>
</tbody>
</table>

ALUctr<1> = !ALUop<2> & !ALUop<1> + ALUop<2> & !func<2> & func<0>

The Logic Equation for ALUctr<0>

<table>
<thead>
<tr>
<th>ALUop</th>
<th>func</th>
<th>ALUctr&lt;0&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit&lt;2&gt;</td>
<td>bit&lt;1&gt;</td>
<td>bit&lt;0&gt;</td>
</tr>
<tr>
<td>0 1 x</td>
<td>x x x x</td>
<td>1</td>
</tr>
<tr>
<td>1 x x</td>
<td>0 1 0 1</td>
<td>1</td>
</tr>
</tbody>
</table>

ALUctr<0> = !ALUop<2> & ALUop<1> + ALUop<2> & func<3> & func<2> & func<1> & func<0>

The ALU Control Block

```
  ALUop
  --------------
  |       |       |
  |       |       |
  |  func |
  |       |
  |       |
  | ALUctr |
  |       |
  |       |
  |       |
```

ALUctr<2> = !ALUop<2> & ALUop<0> + ALUop<2> & func<2> & func<1> & func<0>

ALUctr<1> = !ALUop<2> & !ALUop<1> + ALUop<2> & !func<2> & func<1> & func<0>

ALUctr<0> = !ALUop<2> & ALUop<1> + ALUop<2> & !func<3> & func<2> & func<1> & func<0>

ALUctr<0> = !ALUop<2> & ALUop<1> + ALUop<2> & func<3> & !func<2> & func<1> & func<0>

Step 5: Logic for each control signal

nPC_sel <= if (OP == BEQ) then EQUAL else 0

ALUsrc <= if (OP == "Rtype") then "regB" else "immed"

ALUctr <= if (OP == "Rtype") then func elseif (OP == ORi) then "OR" elseif (OP == BEQ) then "sub" else "add"

ExtOp <= __________

MemWr <= __________

MemtoReg <= __________

RegWr: <= __________

RegDst: <= __________
Step 5: Logic for each control signal

° nPC_sel <= if (OP == BEQ) then EQUAL else 0
° ALUSrc <= if (OP == "Rtype") then "regB" else "immed"
° ALUctr <= if (OP == "Rtype") then funct
  
elseif (OP == ORi) then
  
elseif (OP == BEQ) then
  
else
    x
° ExtOp <= if (OP == ORi) then "zero" else "sign"
° MemWr <= (OP == Store)
° MemtoReg <= (OP == Load)
° RegWr: <= if ((OP == Store) || (OP == BEQ)) then 0 else 1
° RegDst: <= if ((OP == Load) || (OP == ORi)) then 0 else 1

The “Truth Table” for RegWrite

<table>
<thead>
<tr>
<th>op</th>
<th>00 0000</th>
<th>00 1101</th>
<th>10 0011</th>
<th>10 1011</th>
<th>00 0100</th>
<th>00 0010</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>ori</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>lw</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>sw</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>beq</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>jump</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

° RegWrite = R-type + ori + lw

  = lop<5> & lop<4> & lop<3> & lop<2> & lop<1> & lop<0>  \hspace{1cm} \text{(R-type)}

  + lop<5> & lop<4> & op<3> & op<2> & lop<1> & op<0>  \hspace{1cm} \text{(ori)}

  + op<5> & lop<4> & lop<3> & lop<2> & op<1> & op<0>  \hspace{1cm} \text{(lw)}

The “Truth Table” for the Main Control

<table>
<thead>
<tr>
<th>op</th>
<th>00 0000</th>
<th>00 1101</th>
<th>10 0011</th>
<th>10 1011</th>
<th>00 0100</th>
<th>00 0010</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ori</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lw</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>sw</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>beq</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>jump</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

PLA Implementation of the Main Control
Putting it All Together: A Single Cycle Processor

Drawback of this Single Cycle Processor

- Long cycle time:
  - Cycle time must be long enough for the load instruction:
    - PC’s Hold +
    - Instruction Memory Access Time +
    - Register File Access Time +
    - ALU Delay (address calculation) +
    - Data Memory Access Time +
    - Register File Setup Time +
    - Clock Skew
  - Cycle time is much longer than needed for all other instructions

Worst Case Timing (Load)

Summary

- Single cycle datapath => clocks per instruction=1, clock cycle time => long

- 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 effects the register transfer.
  - 5. Assemble the control logic

- Control is the hard part
  - MIPS makes control easier
    - Instructions same size
    - immediates have same size & location
    - Source registers always in same place
    - Operations always on registers/immediates