The Processor: Datapath & Control

- We're ready to look at an implementation of the MIPS
- Simplified to contain only:
  - memory-reference instructions: `lw, sw`
  - arithmetic-logical instructions: `add, sub, and, or, slt`
  - control flow instructions: `beq, j`

- **Generic Implementation:**
  - use the program counter (PC) to supply instruction address
  - get the instruction from memory
  - read registers
  - use the instruction to decide exactly what to do

- All instructions use the ALU after reading the registers
More Implementation Details

- Abstract / Simplified View:

- Two types of functional units:
  - elements that operate on data values (combinational)
  - elements that contain state (state elements)
Our Implementation

• Typical execution:
  – read contents of some state elements,
  – send values through some combinational logic
  – write results to one or more state elements
• Using a clock signal for synchronization
• Edge triggered methodology
State elements needed to access instructions

- Instruction memory
- Program counter
- Adder
A portion of the datapath used for fetching instructions
Elements needed to implement R-type instructions

a. Registers

b. ALU

RegWrite

Read register 1
Read data 1
Write register
Write data

Read register 2

Read data 2

RegWrite

ALU control

ALU result

Zero
The datapath for R-type instructions
Elements needed to implement loads and stores

a. Data memory unit

b. Sign-extension unit
The datapath for load and store instructions
The datapath for branch instructions

PC + 4 from instruction datapath

Add Sum Branch target

Shift left 2

ALU operation

To branch control logic

Instruction

Read register 1
Read register 2
Write register
Write data

RegWrite

Read data 1

Read data 2

Sign extend

32

16
Simple Implementation

- Include the functional units we need for each instruction
Building the complete datapath

- Use multiplexors to stitch them together
Control

- Selecting the operations to perform (ALU, read/write, etc.)
- Controlling the flow of data (multiplexor inputs)
- Information comes from the 32 bits of the instruction
- Example:

  add $8, $17, $18

  **Instruction Format:**

  |
  |---|---|---|---|---|---|
  |000000|10001|10010|01000|00000|100000|
  |op | rs | rt | rd | shamt | funct |

- ALU's operation based on instruction type and function code
Control

• e.g., what should the ALU do with this instruction
• Example: lw $1, 100($2)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>2</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>16 bit offset</th>
</tr>
</thead>
</table>

• ALU control input

  000  AND
  001  OR
  010  add
  110  subtract
  111  set-on-less-than

• Why is the code for subtract 110 and not 011?
Control

- Must describe hardware to compute 3-bit ALU control input
  - given instruction type
    - 00 = lw, sw
    - 01 = beq,
    - 11 = arithmetic
  - function code for arithmetic

- Describe it using a truth table (can turn into gates):

<table>
<thead>
<tr>
<th>ALUOp</th>
<th>ALUOp0</th>
<th>F5</th>
<th>F4</th>
<th>F3</th>
<th>F2</th>
<th>F1</th>
<th>F0</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>010</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>110</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>010</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>110</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>111</td>
</tr>
</tbody>
</table>
Control

<table>
<thead>
<tr>
<th>ALUOp</th>
<th>ALUOp0</th>
<th>Funct field</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X X X X X X X</td>
<td>010</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>X X X X X X 0 0 0</td>
<td>110</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X X 0 0 0 1 0</td>
<td>110</td>
</tr>
<tr>
<td>1</td>
<td>X X 0 1 0 0</td>
<td>000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X X 0 1 0 1</td>
<td>001</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X X 1 0 1 0</td>
<td>111</td>
<td></td>
</tr>
</tbody>
</table>

**Instruction Format**
- **R format**: 1 0 0 1 0 0 0 0 0 1 0 0
- **Iw**: 0 1 1 1 1 0 0 0 0 0 0 0
- **Sw**: X 1 X 0 0 1 0 0 0 0
- **beq**: X 0 X 0 0 0 1 0 1 0 0 1

©1998 Morgan Kaufmann Publishers
Our Simple Control Structure

- All of the logic is combinational
- We wait for everything to settle down, and the right thing to be done
  - ALU might not produce “right answer” right away
  - we use write signals along with clock to determine when to write
- Cycle time determined by length of the longest path

We are ignoring some details like setup and hold times
Single Cycle Implementation

- Calculate cycle time assuming negligible delays except:
  - memory (2ns), ALU and adders (2ns), register file access (1ns)