MIPS-lite processor

- Want to build a processor for a subset of MIPS instruction set ("MIPS-lite")
  - just enough to illustrate key ideas
  - instruction set subset (3 groups):
    - arithmetic-logical: `add`, `sub`, `and`, `or`, `slt`
    - memory reference: `lw`, `sw`
    - control flow: `j`, `beq`
    - can we write real programs with just these?
- Need up to 5 steps to execute any instruction in our subset
Instruction Execution Steps

1. Read IM[PC]
2. Instruction Decode, PC = PC + 4, Register read
3. ALU operation, Branch address computation
4. LW/STORE in Data memory
5. Register Write

Building a Datapath for MIPS (step 1)

Step 1: instruction fetch

Flow of execution

PC  Instruction Memory

add $t0,$t0,$t0
add $t0,$s1,$t0
lw $t1,20($s0)
sw $t1,4($t0)
Datapath Step 1: Any Instruction

- PC
- Instruction Memory (IMem)
- Address
- Instruction
- Clock

32-bit adder or ALU wired only for add

Once program is loaded, IMem is read-only

Building a Datapath for MIPS (step 2)

- PC
- Instruction Memory
- Registers
- Step 1
- Step 2: Decode and Read Registers

Instruction format:

```
op  rs  rt  rd  shamt  funct
0    17  8   8   0     32 R
```

Example instruction:

```
add $t0, $s1, $t0
```
Datapath Step 2: Any Instruction

![Datapath Diagram]

- **Instruction**: In the Datapath Step 2, any instruction involves reading registers and writing data.
- **Control**: The control unit decides the flow based on the instruction.
- **Datapath Control Points**: These are critical points in the datapath where data is processed.

Remaining Steps in Executing Instructions

- **3rd step onwards depends on instruction class**
- **EX**: For ALU instructions: \texttt{add $t0, $t1, $t2}
  - Outputs from registers t1 and t2 will be sent to the ALU input.
- For Memory-reference instruction: \texttt{lw $t0,20($s0)}
  - Address $\leftarrow$ Base + offset

![ALU Diagram]
Building a Datapath for MIPS (lw step 3)

Step 1
I      op    rs    rt    address

lw $t0, 20($s0)

Step 2

Step 3

Datapath Step 3-4: R-format Instructions

add, sub, and, or

Instruction

[$t3] ← [$t1] ⊕ [$t2]

( +, -, AND, OR, etc.)

ALU control

Zero

Result

RegWrite

32

[$t1] ⊕ [$t2]
Datapath Step 3: Branch

beq $t0,$t1,loop

PC + 4 from step 1 datapath

ALU control

Add

Branch target

Branch control logic

Steps 4,5 in Executing lw,sw

- 4th step depends on instruction class
- Ex: for lw: Fetch Data from Memory
  \[ \text{Data} \leftarrow \text{Mem}[\text{Address}] \]
- For sw: Put the contents of a register in Memory

lw $t1,20($s0)
sw $t1,4($t0)

5th step only for lw; rest are done
for lw: Write Result
\[ \text{Reg}[rt] \leftarrow \text{Data} \]
Datapath Step 3-5: Load/Store

```
lw $t0,24($s3)
```

Compose Datapath: R-form + Load/Store

```
Add muxes
```

```
1 = Load/Store
0 = R-form
MemTo-Reg
```

Datapath: Register fields

- Destination registers may differ across instruction formats:
  - R-format: \([rd] \leftarrow [rs] \ op \ [rt]\)
    
    \[
    \text{add } $t0,$s0,$s1
    \]
    
    For this instruction, bits 11-15 are the destination (t0), which should be connected to the write reg. inputs

  - I-format: \([rt] \leftarrow \text{mem}\[\[rs\] + \text{imm16}]\)
    
    \[
    \text{lw } $t0,24($s3)\]
    
    For this instruction, bits 16-20 should go to the write reg. port. Bits 0-15 go to the ALU as address
    - Connection to the write reg. port changes!
    - Solution? mux to the rescue!
Datapath: Determine next PC

- What if instruction is a conditional branch (`beq`)?
  - if operands equal, take branch (PC gets PC+4+offset)
  - else PC gets PC+4
- Therefore, set control point `PCSrc` = 1 if and only if `beq` and Zero asserted

Datapath (add Branch control point)
Adding Control

- CPU = Datapath + Control
- Single-Cycle Design:
  - Instruction takes exactly one clock cycle
  - Datapath units used only once per cycle
  - Writable state updated at end of cycle

What must be “controlled”?
- Multiplexors (Muxes)
- Writable components: Register File, Data Memory (DMem)
  - what about PC? IMem?
  - ALU (which operation?)

Processor = Datapath + Control

- **Single-Cycle Design**: everything happens in one clock cycle until next falling edge of clock, processor is just one big combinational circuit!!!
- Control is a combinational circuit where the output is a function of the inputs
  - outputs? control points in datapath
  - inputs? the current instruction! (opcode, funct control everything)

![Control Logic Diagram](image)
Defining Control

- Note that \texttt{funct} field only present in R-format instruction - \texttt{funt} controls ALU only
- To simplify control, define Main control, ALU control separately – using multiple levels will also increase speed – important optimization technique
- ALUop inputs will be defined

\begin{center}
\begin{tikzpicture}
    \node (control) [draw] {Control Logic};
    \node (main) [draw, right of=control] {Main Control};
    \node (alu) [draw, below of=main] {ALU control};
    \node (aluop) [draw, above of=alu] {ALUop};
    \node (funct) [draw, left of=aluop] {\texttt{funt}};
    \node (op) [draw, above of=control] {\texttt{op}};
    \draw[->] (control) -- (main);
    \draw[->] (main) -- (alu);
    \draw[->] (alu) -- (aluop);
    \draw[->] (aluop) -- (funct);
    \draw[->] (aluop) -- (op);
\end{tikzpicture}
\end{center}

Defining ALU Control

\begin{center}
\begin{tabular}{c|c|c}
\texttt{ALUcon} & \texttt{ALU function} & Instruction(s) supported \\
0000 & AND & R-format (and) \\
0001 & OR & R-format (or) \\
0010 & add & R-format (add), lw, sw \\
0110 & subtract & R-format (sub), beq \\
0111 & set on less than & R-format (slt) \\
1100 & NOR & R-format (nor) \\
\end{tabular}
\end{center}
# Defining ALU Control

## Instruction Descriptions

<table>
<thead>
<tr>
<th>Instruction Opcode</th>
<th>Desired ALU Action</th>
<th>ALUOp</th>
<th>funct</th>
<th>ALUcon</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>add</td>
<td>00</td>
<td>xxxxxx</td>
<td>0010</td>
</tr>
<tr>
<td>sw</td>
<td>add</td>
<td>00</td>
<td>xxxxxx</td>
<td>0010</td>
</tr>
<tr>
<td>beq</td>
<td>subtract</td>
<td>01</td>
<td>xxxxxx</td>
<td>0110</td>
</tr>
<tr>
<td>R-type</td>
<td>add</td>
<td>10</td>
<td>100000 (add)</td>
<td>0010</td>
</tr>
<tr>
<td>R-type</td>
<td>subtract</td>
<td>10</td>
<td>100010 (sub)</td>
<td>0110</td>
</tr>
<tr>
<td>R-type</td>
<td>logical AND</td>
<td>10</td>
<td>100100 (and)</td>
<td>0000</td>
</tr>
<tr>
<td>R-type</td>
<td>logical OR</td>
<td>10</td>
<td>100101 (or)</td>
<td>0001</td>
</tr>
<tr>
<td>R-type</td>
<td>set on less</td>
<td>10</td>
<td>101010 (slt)</td>
<td>0111</td>
</tr>
</tbody>
</table>

## ALUOp, Func Field, and ALUcon

<table>
<thead>
<tr>
<th>ALUOp</th>
<th>ALUcon</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>xxxxxx</td>
</tr>
<tr>
<td>sub</td>
<td>xxxxxx</td>
</tr>
<tr>
<td>add</td>
<td>100000 (add)</td>
</tr>
<tr>
<td>sub</td>
<td>100010 (sub)</td>
</tr>
<tr>
<td>and</td>
<td>100100 (and)</td>
</tr>
<tr>
<td>or</td>
<td>100101 (or)</td>
</tr>
<tr>
<td>slt</td>
<td>101010 (slt)</td>
</tr>
</tbody>
</table>

## Don’t Care

<table>
<thead>
<tr>
<th>c3</th>
<th>c2</th>
<th>c1</th>
<th>c0</th>
<th>ALUcon</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0010</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>0110</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>0011</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>0111</td>
</tr>
</tbody>
</table>

## OpCode field

<table>
<thead>
<tr>
<th>$O_{28}, O_{27}, O_{26}$</th>
<th>$O_{31}, O_{30}, O_{29}$</th>
<th>$O_{28}, O_{27}, O_{26}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>001</td>
</tr>
<tr>
<td>010</td>
<td>010</td>
<td>010</td>
</tr>
<tr>
<td>011</td>
<td>011</td>
<td>011</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>101</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>110</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>111</td>
<td>111</td>
<td>111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$O_{31}, O_{30}, O_{29}$</th>
<th>$O_{28}, O_{27}, O_{26}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-format</td>
<td>j</td>
</tr>
<tr>
<td>add</td>
<td>addiu</td>
</tr>
<tr>
<td>slti</td>
<td>sltiu</td>
</tr>
<tr>
<td>andi</td>
<td>ori</td>
</tr>
<tr>
<td>xor</td>
<td></td>
</tr>
<tr>
<td>llo</td>
<td>lhi</td>
</tr>
<tr>
<td>trap</td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>lhu</td>
</tr>
<tr>
<td>lbu</td>
<td>lhu</td>
</tr>
<tr>
<td>sb</td>
<td>sh</td>
</tr>
<tr>
<td>sw</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Function field  \( f_5, f_4 \ldots f_1, f_0 \)

<table>
<thead>
<tr>
<th>( f_2, f_1, f_0 )</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_5, f_4, f_3 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>sll</td>
<td>srl</td>
<td>sra</td>
<td>slv</td>
<td>srlv</td>
<td>srav</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>jr</td>
<td>jalr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>mfhi</td>
<td>mthi</td>
<td>mflo</td>
<td>mtlo</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>mult</td>
<td>multu</td>
<td>div</td>
<td>divu</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>add</td>
<td>addu</td>
<td>sub</td>
<td>subu</td>
<td>and</td>
<td>or</td>
<td>xor</td>
<td>nor</td>
</tr>
<tr>
<td>101</td>
<td>slt</td>
<td>sltu</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fully Minimized ALU Control

- From the truth table, output signals can be easily derived because of don’t cares
- \( c_2 = a_0 \text{ OR } (a_1 \text{ AND } f_1) \)
- \( c_1 = (\text{Not } a_1) \text{ OR } (\text{Not } f_2) \)
- \( c_0 \text{ (lsb)} = a_1 \text{ AND } (f_3 \text{ OR } f_0) \)
- \( \text{ALUOp is supplied by the main control unit (to be designed)} \)

ALUOp

\[ \begin{align*}
\text{a0, f5,4,3,2,1,0 ALUcon} \\
\text{X0 X1 X2 X3 X4 X5} \\
\text{0010 0110 0010 0110 0000 0001 0111}
\end{align*} \]
Adding Jumps

- Update PC with concatenation of
  - Top 4 bits of old PC
  - 26-bit jump address
  - 00
- Need an extra control signal decoded from opcode