MIPS Datapath
(Single Cycle and Multi-Cycle)
Basic MIPS Implementation

• How does hardware implement instructions?
  – Key principles and techniques used in creating data path
  – How the instruction set arch determines many aspects of implementations
• For a limited subset of the MIPS instructions
  – Memory reference: LW and SW
  – Arithmetic-logical: add, sub, and, or, slt
  – Branch: beq
• Hardware components: PC, registers, memory units, ALU, multiplexors, decoders
Single Cycle v. Multi-Cycle

• *Single cycle*: one (long) clock cycle to process each instruction
  – not realistic but useful for understanding the implementation.

• *Multi-cycle*: divide the processing of each instruction into 5 stages and allocate one clock cycle per stage
Fetch instruction and increment PC

PC is a register that holds the current instruction address.
Add Registers, ALU, and Memory

- What is missing?
  - **MUX** to select input data lines.
  - Control lines for the major functional units.
And Multiplexors and Control Lines
Memory Instruction & R-type instructions

* **LW $t1, offset($t2)**, Need register file, Memory, and ALU
* SW is similar
Computing branch condition and target address
Single cycle and control unit
Single cycle plus jump
Example of execution

Instr: add $s0, $a1, $t7
Field: op rs rt rd shamt funct
M. Code: 000000 00101 01111 10000 00000 100000
PC: 0 x 4000
Suggested Exercises

- Repeat the example of execution for:
  - lw $t3, 16($t2)
  - sw $t3, 16($t2)
  - addi $t1, $s2, $v0
  - beq $t3, $s0, gothere
  - j gothere
  - jr $ra
Multicycle Datapath

- Break the operations on an instruction into a series of 5 steps.
- One clock cycle per step
  - Advantage: a functional unit can be used more than once per instruction.
- Instruction Fetch (IF)
- Instruction Decode (ID)
- Execute (EX)
- Memory Access (MEM)
- Write Back (WB)
Multicycle Datapath

• HW changes:
  – Single memory unit for both instructions and data
  – Single ALU for all arithmetic operations
  – Extra registers needed to hold values between each steps
    • Instruction Register (IR) holds the instruction
    • Memory Data Register (MDR) holds the data coming from memory
    • A, B hold operand data coming from the registers
    • ALUOut holds output coming out of the ALU
Extra hardware for multicycle datapath
# Multicycle Datapath: the 5 steps

<table>
<thead>
<tr>
<th>Step Name</th>
<th>R-Type</th>
<th>Memory Reference</th>
<th>Branches</th>
<th>Jumps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Fetch</td>
<td></td>
<td>IR &lt;= Memory[PC]</td>
<td>PC &lt;= PC + 4</td>
<td></td>
</tr>
<tr>
<td>Instruction Decode</td>
<td></td>
<td>A &lt;= Reg[IR[25:21]]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register Fetch</td>
<td></td>
<td>B &lt;= Reg[IR[20:16]]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ALUOut &lt;= PC + (sign-extend(IR[15:0]) &lt;&lt; 2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution</td>
<td>ALUOut &lt;= A op B</td>
<td>ALUOut &lt;= A + sign-extend(IR[15:0])</td>
<td>If (A==B) PC &lt;= ALUOut</td>
<td>PC &lt;= {PC[31:28], IR[25:0], 00}</td>
</tr>
<tr>
<td>Address Computation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Branch/Jump Completion</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R-type completion</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Read Completion</td>
<td></td>
<td>Load: Reg[IR[20:16]] &lt;= MDR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Multicycle datapath
Complete multicycle datapath