Introduction

• We're now ready to look at an implementation of the system that includes MIPS processor and memory.
• The design will include support for execution of only:
  – memory-reference instructions: lw & sw,
  – arithmetic-logical instructions: add, sub, and, or, slt & nor,
  – control flow instructions: beq & j,
  – exception handling: illegal instruction & overflow.
• But that design will provide us with principles, so many more instructions could be easily added such as: addu, lb, lbu, lui, addi, adiu, sltu, slti, andi, ori, xor, xori, jal, jr, jalr, bne, beqz, bgtz, bltz, nop, mfhi, mflo, mfepc, mfco, lwc1, swc1, etc.
Single Cycle Design

- We will first design a simpler processor that executes each instruction in only one clock cycle time.
- This is not efficient from performance point of view, since:
  - a clock cycle time (i.e. clock rate) must be chosen such that the longest instruction can be executed in one clock cycle and
  - that makes shorter instructions execute in one unnecessarily long cycle.
- Additionally, no resource in the design may be used more than once per instruction, thus some resources will be duplicated.
- The single cycle design will require:
  - two memories (instruction and data),
  - two additional adders.

Elements for Datapath Design

- Program counter
- ALU
- Address
- Write data
- Data memory
- Sign extension unit
- Read register 1
- Read register 2
- Registers
- Write register
- Write data
- Adder
- Instruction memory
- Shift left 2
Abstract /Simplified View (1st look)

- This generic implementation:
  - uses the program counter (PC) to supply instruction address,
  - gets the instruction from memory,
  - reads registers,
  - uses the instruction opcode to decide exactly what to do.

Abstract /Simplified View (2nd look)

- Figure 5.1

- PC is incremented by 4 by most instructions, and 4 + 4×offset by branch instructions.
- Jump instructions change PC differently (not shown).
Our Implementation

- An edge triggered methodology
- Typical execution:
  - read contents of some state elements at the beginning of the clock cycle,
  - send values through some combinational logic,
  - write results to one or more state elements at the end of the clock cycle.

![Figure 5.5](image)

- An edge triggered methodology allows a state element to be read and written in the same clock cycle.

Incrementing PC & Fetching Instruction

![Figure 5.6](image)
Datapath for R-type Instructions

Based on contents of op-code and funct fields, Control Unit sets ALU control appropriately and asserts RegWrite, i.e. RegWrite = 1.
Datapath for LW and SW Instructions

**Control Unit sets:**
- ALU control = 0010 (add) for address calculation for both lw and sw
- MemRead=0, MemWrite=1 and RegWrite=0 for sw
- MemRead=1, MemWrite=0 and RegWrite=1 for lw

Let us determine setting of control lines for R-type, lw & sw instructions.
Datapath for BEQ Instruction

Branch target = [PC] + 4 + 4×offset

Figure 5.9
with additions in red

 Datapath for R-type, LW, SW & BEQ

Figure 5.15
with additions in red
Figure 5.17
with additions in red

Truth Table for (Main) Control Unit

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op-code</td>
<td>RegDst</td>
</tr>
<tr>
<td>R-type</td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>000000</td>
</tr>
<tr>
<td>sw</td>
<td>100011</td>
</tr>
<tr>
<td>beq</td>
<td>101011</td>
</tr>
<tr>
<td></td>
<td>000100</td>
</tr>
</tbody>
</table>

- ALUOp[1-0] = 00 $\rightarrow$ signal to ALU Control unit for ALU to perform add function, i.e. set Ainvert = 0, Binvert=0 and Operation=10
- ALUOp[1-0] = 01 $\rightarrow$ signal to ALU Control unit for ALU to perform subtract function, i.e. set Ainvert = 0, Binvert=1 and Operation=10
- ALUOp[1-0] = 10 $\rightarrow$ signal to ALU Control unit to look at bits [5-0] and based on its pattern to set Ainvert, Binvert and Operation so that ALU performs appropriate function, i.e. add, sub, slt, and, or & nor
Truth Table of ALU Control Unit

<table>
<thead>
<tr>
<th>ALUOp</th>
<th>funct field</th>
<th>ALU Op1</th>
<th>F5</th>
<th>F4</th>
<th>F3</th>
<th>F2</th>
<th>F1</th>
<th>F0</th>
<th>ALU Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>0</td>
<td>0.10</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>1</td>
<td>0.10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.11</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1.10</td>
</tr>
</tbody>
</table>

Input: ALUOp
Output: ALUControl

Design of (Main) Control Unit

<table>
<thead>
<tr>
<th>Op-code bits</th>
<th>RegDst</th>
<th>ALUSrc</th>
<th>Memto Reg</th>
<th>RegWrite</th>
<th>MemRead</th>
<th>MemWrite</th>
<th>Branch</th>
<th>ALUOp1</th>
<th>ALUOp0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>100011</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>101011</td>
<td>d</td>
<td>0</td>
<td>1</td>
<td>d</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>000100</td>
<td>d</td>
<td>0</td>
<td>0</td>
<td>d</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

RegDst = Op5Op4Op3Op2Op1Op0

Figure C.2.5
Datapath for R-type, LW, SW, BEQ & J

PC ← PC_{31:28} || jump_target || 00

Figure 5.24
with correction in red

Design of Control Unit (J included)

<table>
<thead>
<tr>
<th>Op-code</th>
<th>RegDst</th>
<th>ALUSrc</th>
<th>MemtoReg</th>
<th>RegWrite</th>
<th>MemRead</th>
<th>MemWrite</th>
<th>Branch</th>
<th>ALUOp1</th>
<th>ALUOp0</th>
<th>Jump</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>d</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100011</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>101011</td>
<td>d</td>
<td>1</td>
<td>d</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>000100</td>
<td>d</td>
<td>0</td>
<td>d</td>
<td>0</td>
<td>d</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>000101</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>1</td>
</tr>
</tbody>
</table>

Jump = \overline{Op_5} \overline{Op_4} \overline{Op_3} Op_2 Op_1 Op_0

No changes in ALU Control unit
**Cycle Time Calculation**

- Let us assume that the **only** delays introduced are by the following tasks:
  - Memory access (read and write time = 3 nsec)
  - Register file access (read and write time = 1 nsec)
  - ALU to perform function (= 2 nsec)
- Under those assumptions here are instruction execution times:

<table>
<thead>
<tr>
<th>Instr</th>
<th>Reg</th>
<th>ALU</th>
<th>Data</th>
<th>Reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetch</td>
<td>read</td>
<td>oper</td>
<td>memory</td>
<td>write</td>
</tr>
<tr>
<td>R-type</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>lw</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>sw</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>branch</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>jump</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Thus a clock cycle time has to be 10 nsec, and clock rate = 1/10 nsec = 100MHz

**Single Cycle Processor: Conclusion**

- Single Cycle Problems:
  - what if we had a more complicated instruction like floating point?
  - a clock cycle would be much longer,
  - thus for shorter and more often used instructions, such as add & lw, wasteful of time.
- One Solution:
  - use a “smaller” cycle time, and
  - have different instructions take different numbers of cycles.
- And that is a “multi-cycle” processor.