Designing MIPS Processor (Single-Cycle)

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, ori, xor, xori, jal, jr, jalr, bne, beqz, bgtz, bltz, nop, mfhi, mflo, mfepc, mfco, lwc1, swc1, etc.

Single Cycle Design

- We shall 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 unnecessary long cycle.
- Additionally, no resource in the design may be used more than once per instruction, thus some resources will be duplicated.
- Because of that, the single cycle design will require:
  - two memories (instruction and data),
  - two additional adders.
• 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.

Figure 5.1

• PC is incremented by 4, by most instructions, and by \(4 + 4 \times \text{offset}\),
  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.

• An edge triggered methodology allows a state element to be read and written in the same clock cycle without creating a race that could to indeterminate data.
Datapath for R-type Instructions

R-type          000000       rs rt rd        00000     funct

31              26 25            21 20           16  15         11  10              6  5               0

add = 32
sub = 34
slt = 42
and = 36
or = 37
nor = 39

In s t r u c t i o n
R e g i s t e r s
W r i t e
re g is te r
R e a d
data 1
R e a d
data 2
R e a d
re g is te r 1
R e a d
re g is te r 2
W r i t e
data
ALU
result
ALU
Zer o
Reg W r i t e
4

Complete 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

sw or lw opcode rs rt offset

31 26 25 21 20 16 15 0

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

Datapath for R-type, LW & SW Instructions

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

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>25</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>beq</td>
<td>rs</td>
<td>rt</td>
<td>offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Branch target = [PC] + 4 + 4×offset

![Figure 5.9 with additions in red](image)

---

**Datapath for R-type, LW, SW & BEQ**

![Figure 5.15 with additions in red](image)

---

**Control Unit and Datapath**

![Figure 5.17 with additions in red](image)

---

**Truth Table for (Main) Control Unit**

- **ALUOp[1-0] = 00** → 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** → 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** → signal to ALU Control unit to look at bits I[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

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

---
Truth Table of ALU Control Unit

<table>
<thead>
<tr>
<th>ALUOp</th>
<th>Funct field</th>
<th>ALU Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>dddddd</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>100000</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>100010</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>100100</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>100101</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>101010</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>100111</td>
</tr>
</tbody>
</table>

Input: ALUOp, Funct field
Output: ALU Control

Design of (Main) Control Unit

<table>
<thead>
<tr>
<th>Op-code bits</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>0</td>
<td>0</td>
<td>1</td>
<td>0</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>0</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>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Jump = Op_5 Op_4 Op_3 Op_2 Op_1 Op_0

No changes in ALU Control unit

Datapath for R-type, LW, SW, BEQ & J

Design of Control Unit (J included)

Jump = Op_5 Op_4 Op_3 Op_2 Op_1 Op_0
Design of 7-Function ALU Control Unit

<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>ALU Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>010</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>010</td>
</tr>
<tr>
<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>110</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>110</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>000</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>001</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>111</td>
</tr>
</tbody>
</table>

Figure C.2.3 with improvements

ALU Control Lines (Binvert & Operation)

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 assumption here are instruction execution times:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Reg</th>
<th>ALU</th>
<th>Data</th>
<th>Reg</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>3</td>
<td>+</td>
<td>1</td>
<td>+</td>
<td>2</td>
</tr>
<tr>
<td>lw</td>
<td>3</td>
<td>+</td>
<td>1</td>
<td>+</td>
<td>2</td>
</tr>
<tr>
<td>sw</td>
<td>3</td>
<td>+</td>
<td>1</td>
<td>+</td>
<td>2</td>
</tr>
<tr>
<td>branch</td>
<td>3</td>
<td>+</td>
<td>1</td>
<td>+</td>
<td>2</td>
</tr>
<tr>
<td>jump</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Thus a clock cycle time has to be 10nsec, 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.
## Control Unit Truth Table and Design

<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>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>000100</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### R-format Inputs
- \(O_5\)
- \(O_4\)
- \(O_3\)
- \(O_2\)
- \(O_1\)

### R-format Outputs
- RegDst
- ALUSrc
- MemtoReg
- RegWrite
- MemRead
- MemWrite
- Branch
- ALUOp1
- ALUOp0
- Jump