Designing MIPS Processor (Single-Cycle)

Dr. Arjan Durresi
Louisiana State University
Baton Rouge, LA 70810
Durresi@Csc.LSU.Edu

These slides are available at:
http://www.csc.lsu.edu/~durresi/CSC3501_07/

Overview

Datapath
Control Unit
Problems with single cycle Datapath

Performance

Instruction Count
Clock cycle time
Clock cycle per Instruction

Implementation of Processor

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 (sequential)

Building the Datapath
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 lead to indeterminate data.

Single Cycle Design

- We shall first design a simpler processor that executes each instruction in only one clock cycle.
- 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
  - 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.

Elements for Datapath Design

- Include the functional units we need for each instruction
- Why do we need this stuff?

Abstract /Simplified View (1st look)

- 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.

Abstract /Simplified View (2nd look)

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

Incrementing PC & Fetching Instruction

- Figure 5.6
Two elements needed to implement R-formant ALU operations

\[ \text{add } \$t1, \$t2, \$t3 \]

Complete Datapath for R-type Instructions

Datapath for R-type, LW & SW Instructions

The datapath for a branch

Compute the branch target address by adding the sign-extended offset of the instruction to PC.

Two elements needed to implement loads and stores

Compute a memory address by adding the base register (\$t2) to the 16-bit signed offset field contained in the instruction.

Datapath for LW and SW Instructions

Control Unit sets:
- ALU control = 4010 (add) for address calculation for both lw and sw
- MemRead=1, MemWrite=0 and RegWrite=1 for lw
- MemRead=1, MemWrite=0 and RegWrite=1 for sw
Datapath for R-type, LW, SW & BEQ

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

```
add $8, $17, $18
```

Instruction Format:

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>shamt</td>
</tr>
</tbody>
</table>
```

ALU's operation based on instruction type and function code

Load, store and branch instructions

- Load or store instruction
- Branch instruction

Truth Table for (Main) Control Unit

Generate the 4-bit ALU control input using a small control unit that has as inputs the function field of the instruction and a 2-bit control field ALUOp
Truth Table of ALU Control Unit

<table>
<thead>
<tr>
<th>Input</th>
<th>ALUOp</th>
<th>Func Field</th>
<th>ALU Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Design of (Main) Control Unit

<table>
<thead>
<tr>
<th>Function</th>
<th>ALUOp</th>
<th>Func Field</th>
<th>ALU Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Design of Control Unit (J included)

Design of 7-Function 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>Instruction</th>
<th>fetch</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>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td>7 nsec</td>
</tr>
<tr>
<td>lw</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td>10 nsec</td>
</tr>
<tr>
<td>sw</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td>9 nsec</td>
</tr>
<tr>
<td>branch</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td>6 nsec</td>
</tr>
<tr>
<td>jump</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3 nsec</td>
</tr>
</tbody>
</table>

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

Single Cycle Implementation

- Calculate cycle time assuming negligible delays except:
  - Memory (200ps)
  - ALU and adders (100ps)
  - register file access (50ps)
Performance of Single cycle machines

- Memory (200ps), ALU and adders (100ps), register file access (50ps)
- Instructions mix: 25% loads, 10% stores, 45% ALU, 15% branches, 5% jumps
- Which of the following implementations would be faster?
  - Every instruction operates in a 1 clock cycle of a fixed length
  - Every instruction operates in a 1 clock cycle of a variable length
- CPU execution time = Instruction count × CPI × Clock cycle time
- Since CPI = 1
- CPU execution time = Instruction count × Clock cycle time
- Using the critical paths we can compute the required length for each class:
  - R-type 400ps, Load word 600ps, Store word 550ps, Branch 350ps, Jump 200ps
- In case 1 the clock has to be 600ps depending on the longest instruction
- The average CPU clock cycle = 600 × 25% + 550 × 10% + 400 × 45% + 350 × 15% + 200 × 5% = 447.5ps
- So the variable clock machine is faster 1.34 times

Example

<table>
<thead>
<tr>
<th>Instruction class</th>
<th>Instruction memory</th>
<th>Register read</th>
<th>ALU</th>
<th>Data memory</th>
<th>Register write</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>0</td>
<td>50</td>
<td>400ps</td>
</tr>
<tr>
<td>Load word</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>200</td>
<td>50</td>
<td>600ps</td>
</tr>
<tr>
<td>Store word</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>200</td>
<td>0</td>
<td>550ps</td>
</tr>
<tr>
<td>Branch</td>
<td>200</td>
<td>50</td>
<td>100</td>
<td>0</td>
<td></td>
<td>350ps</td>
</tr>
<tr>
<td>Jump</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>200ps</td>
</tr>
</tbody>
</table>

Abstract View of our single cycle process:

- Looks like a FSM with PC as state

Where we are headed

- Single Cycle Problems:
  - What if we had a more complicated instruction like floating point?
  - Wasteful of area
- One Solution:
  - Use a "smaller" cycle time
  - Have different instructions take different numbers of cycles
  - A "multicycle" datapath

What's wrong with our CPI=1 processor?

- Arithmetic & Logical
  - PC, Inst Memory, Reg File, ALU
  - Load
    - PC, Inst Memory, Reg File, ALU, Data Mem
  - Store
    - PC, Inst Memory, Reg File, ALU, Data Mem
  - Branch
    - PC, Inst Memory, Reg File, cmp, Join

- Long Cycle Time
- All instructions take as much time as the slowest
- Real memory is not as nice as our idealized memory
  - Cannot always get the job done in one (short) cycle
### 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.