CMPE011 Winter 2003

Building a *Single-cycle Central Processing Unit (CPU)*

- Building an Arithmetic-Logic Unit (ALU)
- Register files
- Building a *single-cycle* U'With

- *combinatorial* control

Textbook: 4.5, 5.1 to 5.3
Building an ALU

Building an ALU – step 1: logical operators

We will implement only the AND and OR logical operators

1-bit logical unit

32-bit logical unit
Building an ALU — step 2: adding the adder/subtracter
Building an ALU – step 3: adding comparison instructions

**relative magnitude:** the `slt` instruction writes a 1 (i.e. 00.001) in register `rd` if `rs < rt`.

To decide whether `rs < rt`, we can just subtract `rt` from `rs` and check the sign of the result (Less input and Set output in picture on next page).

**equality/inequality:** the branch instructions (`beq`) test for equality or inequality of `rs` and `rt`.

This comparison is done by subtracting one operand from the other and checking if the result is zero. The **Zero** signal is simply the NOR of all result bits (`Result[3:0]`).
1-bit ALU

The complete 1-bit ALU built so far looks like:

The ALU for the msb is slightly different: the **Overflow** output comes from the overflow detection unit, and the **Set** output is the sign bit used for **slt**.
Building an ALU – step 4: 32-bit ripple carry ALU

The ALU symbol:

<table>
<thead>
<tr>
<th>ALU_Op[2:0]</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>and</td>
</tr>
<tr>
<td>001</td>
<td>or</td>
</tr>
<tr>
<td>010</td>
<td>add</td>
</tr>
<tr>
<td>110</td>
<td>sub</td>
</tr>
<tr>
<td>111</td>
<td>slt</td>
</tr>
</tbody>
</table>
Register files

Single write-dual read register file
Decoders

Symbol and truth table of a 3-to-8 decoder

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00000001</td>
</tr>
<tr>
<td>001</td>
<td>00000010</td>
</tr>
<tr>
<td>010</td>
<td>00000100</td>
</tr>
<tr>
<td>011</td>
<td>00001000</td>
</tr>
<tr>
<td>100</td>
<td>00010000</td>
</tr>
<tr>
<td>101</td>
<td>00100000</td>
</tr>
<tr>
<td>110</td>
<td>01000000</td>
</tr>
<tr>
<td>111</td>
<td>10000000</td>
</tr>
</tbody>
</table>
Building a single-cycle CPU

Step by step:

- Arithmetic-logic instructions
- Load/store instructions
- Instruction fetch and PC increment
- Jumps
- Branches
- ALL TOGETHER
Arithmetic-logic instructions (R-type)

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26</td>
<td>25</td>
<td>21</td>
<td>20</td>
<td>16</td>
</tr>
</tbody>
</table>

Operation[2:0] function

- 000: and
- 001: or
- 010: add
- 110: sub
- 111: slt
Load/store instructions

<table>
<thead>
<tr>
<th>Instr.</th>
<th>Operation [2:0]</th>
<th>MemRead</th>
<th>MemWrite</th>
<th>RegWrite</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
16 to 32 sign extension
Instruction fetch and PC increment
Jumps

PC

Addr[31:0]

Dout[31:0]

Instruction Memory

Ins[31:0]

Jump

<table>
<thead>
<tr>
<th>op</th>
<th>target</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26 25</td>
</tr>
</tbody>
</table>

Instr. Jump

j
Branches

<table>
<thead>
<tr>
<th>op</th>
<th>base/rs</th>
<th>rt</th>
<th>offset/immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26 25</td>
<td>21 20</td>
<td>16 15</td>
</tr>
</tbody>
</table>

Instr. | Operation[2:0] | Mbranch | RegWrite
beq    |               |          |
Left shift by 2
All together
About ALUop[1:0] and Operation[2:0]

<table>
<thead>
<tr>
<th>Instruction</th>
<th>ALUop[1:0]</th>
<th>IR[5:0] (Func)</th>
<th>Operation[2:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD/STORE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRANCH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARITH/L</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
“Don’t cares”

A signal is a “don’t care” if in a specific operation the final outcome does not depend on the value of the signal, that is, if the outcome is the same for every possible value of the signal.
**Single-cycle CPU**

**Combinatorial control**

The setting of the control lines is completely determined by the opcode fields of the instruction.

<table>
<thead>
<tr>
<th>Instr.</th>
<th>and</th>
<th>or</th>
<th>add</th>
<th>sub</th>
<th>slt</th>
<th>lw</th>
<th>sw</th>
<th>beq</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>Funct[5:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALUop[1:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation[2:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALUsrc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MemRead</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MemWrite</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MemtoReg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RegDst</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RegWrite</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Branch</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mbranch</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jump</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performance of single-cycle implementation

The implementation we have just seen can be summarized like:

- what does “single-cycle implementation” mean?
- what is the slowest instruction (\textit{worst case path})?
Example (p. 373)

Let: \( t_{pd}(\text{Instr. Mem}) = t_{pd}(\text{Data Mem.}) = t_{pd}(\text{ALU}) = t_{pd}(\text{Aders}) = 2n^s \)
\( t_{pd}(\text{Register Read}) = t_{pd}(\text{Register Write}) = 1ns \)
\( t_{pd}(\text{everything else}) = 0ns \)

Q.1 what is the highest clock frequency at which we can run the CPU?

<table>
<thead>
<tr>
<th>Instr. class</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>loads</td>
<td>24%</td>
</tr>
<tr>
<td>stores</td>
<td>12%</td>
</tr>
<tr>
<td>ALU</td>
<td>44%</td>
</tr>
<tr>
<td>branches</td>
<td>18%</td>
</tr>
<tr>
<td>jumps</td>
<td>2%</td>
</tr>
</tbody>
</table>

Given the instruction mix:

Q.2 what is the CPI?

Q.3 what would the CPI be with a variable length clock cycle?

Q.4 what would the average time per instruction be with a variable length clock cycle?

Q.5 what is the performance ratio between the fixed clock implementation and the variable clock implementation?
Solution

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>beq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ T_{\text{fix}} = \]

Q.2 and Q.3

Q.4

\[ T_{\text{var}} = \]

Q.5

\[ \frac{\text{Perf}_{\text{var}}}{\text{Perf}_{\text{fix}}} = \]
Homework

Recommended exercises

- Ex. 5.1, 5.2, 5.5 to 5.10, 5.14