Multiple Cycle Data Path

CS 365 Lecture 7
Prof. Yih Huang

Multicycle Approach

- Break up the instructions into steps, each step takes a cycle
  - balance the amount of work to be done
  - restrict each cycle to use only one major functional unit
- At the end of a cycle
  - store values for use in later cycles
  - introduce additional “internal” registers
Multicycle Datapath and Control

Highlights in the New Datapath

- Instruction and data memory merged.
- An Instruction Register (IR) is introduced to store the current instruction.
- A Memory Data Register (MDR) is introduced to store data read from memory.
- New registers, simply called A and B, store the output of register read port 1 and 2.
- A new register, ALUout, stores the output of ALU.
- PC-only adders removed.
New Control Signals

- **IorD**: Determine the input to the memory address port
  - 0: PC (for instruction fetches)
  - 1: ALU output (for data accesses)
- **RegDst**: select rt or rd as the destination register
- **Write signals for IR, MDR, A, B, PC, and ALUout.**

- **ALUsrcA**: determines the first operand to ALU
  - 0: PC
  - 1: A
- **ALUsrcB (2 bits)**: determines the first operand to ALU
  - 00: B
  - 01: 4
  - 10: Sign-extended immediate
  - 11: shifted immediate
Five Execution Steps

- Instruction Fetch
- Instruction Decode and Register Fetch
- Execution, Memory Address Computation, or Branch Completion
- Memory Access or R-type instruction completion
- Write back

*INSTRUCTIONS TAKE FROM 3 - 5 CYCLES!*

---

Step 1: Instruction Fetch

- Actions

\[
IR = \text{Memory}[PC]; \\
PC = PC + 4;
\]

*Can we figure out the values of the control signals?*

- Parallel or sequential actions?
Step 2: Instruction Decode and Register Fetch

- Read registers rs and rt in case we need them
- Compute the branch address in case the instruction is a branch
- Actions

\[ \begin{align*}
A &= \text{Reg}[\text{IR}[25-21]]; \\
B &= \text{Reg}[\text{IR}[20-16]]; \\
\text{ALUOut} &= \text{PC} + \text{Offset}
\end{align*} \]

Step 3: instruction dependent

- ALU is performing one of three functions, based on instruction type
- Memory Reference:
  \[ \text{ALUOut} = A + \text{sign-extend}(\text{IR}[15-0]); \]
- R-type:
  \[ \text{ALUOut} = A \text{ op } B; \]
- Branch:
  \[ \text{if (A==B) PC} = \text{ALUOut}; \]
Step 4: R-type or memory-access

MDR = Memory[ALUOut]; \quad \text{(lw)}

or

Memory[ALUOut] = B; \quad \text{(sw)}

or

Reg[IR[15-11]] = ALUOut; \quad \text{(R type)}

Step 5: Write-back

Reg[IR[20-16]] = MDR;

Summary

<table>
<thead>
<tr>
<th>Step name</th>
<th>Action for R-type instructions</th>
<th>Action for memory-reference instructions</th>
<th>Action for branches</th>
<th>Action for jumps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction fetch</td>
<td>(IR = Memory[PC])</td>
<td>PC = PC + 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instruction decode/register fetch</td>
<td>A = Reg[IR[25-21]]</td>
<td>B = Reg[IR[20-16]]</td>
<td>ALUOut = PC + (sign-extend IR[15-0]) &lt;&lt; 2</td>
<td></td>
</tr>
<tr>
<td>Execution, address computation, branch/ jump completion</td>
<td>ALUOut = A op B</td>
<td>ALUOut = A + sign-extend IR[15-0]</td>
<td>If (A == B) then PC = ALUOut</td>
<td>PC = PC[31-28] \text{II} (IR[25-0] &lt;&lt; 2)</td>
</tr>
<tr>
<td>Memory access or R-type completion</td>
<td>Reg[IR[15-11]] = ALUOut</td>
<td>Load: MDR = Memory[ALUOut] \text{ or}</td>
<td>Store: Memory[ALUOut] = B</td>
<td></td>
</tr>
<tr>
<td>Memory read completion</td>
<td>Load: Reg[IR[20-16]] = MDR</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Control Signals for ADD**

<table>
<thead>
<tr>
<th>ALU Op</th>
<th>RegWrite</th>
<th>MemWrite</th>
<th>Mem2Reg</th>
<th>IRWrite</th>
<th>MDRWr</th>
<th>AWrite</th>
<th>BWrite</th>
<th>ALUoutWr</th>
<th>PCWrite</th>
<th>RegDst</th>
<th>IorD</th>
<th>IorS</th>
<th>ALUsrcA</th>
<th>ALUsrcB</th>
<th>PCsrc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Sub, and, or, slt are similar
<table>
<thead>
<tr>
<th>Control Signals for SW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td><strong>ALU Op</strong></td>
</tr>
<tr>
<td>PCsrc</td>
</tr>
<tr>
<td>ALUsrcB</td>
</tr>
<tr>
<td>ALUsrcA</td>
</tr>
<tr>
<td>IorD</td>
</tr>
<tr>
<td>RegDst</td>
</tr>
<tr>
<td>PCWrite</td>
</tr>
<tr>
<td>ALUoutWr</td>
</tr>
<tr>
<td>BWrite</td>
</tr>
<tr>
<td>AWrite</td>
</tr>
<tr>
<td>MDRWr</td>
</tr>
<tr>
<td>IRWrite</td>
</tr>
<tr>
<td>Mem2Reg</td>
</tr>
<tr>
<td>MemWrite</td>
</tr>
<tr>
<td>MemRead</td>
</tr>
<tr>
<td>RegWrite</td>
</tr>
</tbody>
</table>

| **ALU Op**             |
| PCsrc                  |
| ALUsrcB                |
| ALUsrcA                |
| IorD                   |
| RegDst                 |
| PCWrite                |
| ALUoutWr               |
| BWrite                 |
| AWrite                 |
| MDRWr                  |
| IRWrite                |
| Mem2Reg                |
| MemWrite               |
| MemRead                |
| RegWrite               |
Control Signals for BEQ

<table>
<thead>
<tr>
<th>ALU Op</th>
<th>RegWrite</th>
<th>MemRead</th>
<th>Mem2Reg</th>
<th>IRWrite</th>
<th>MDRWr</th>
<th>AWrite</th>
<th>BWrite</th>
<th>ALUoutWR</th>
<th>PCWrite</th>
<th>RegDst</th>
<th>IoT</th>
<th>IALuseA</th>
<th>IALuseB</th>
<th>PCsrc</th>
</tr>
</thead>
</table>

Review: finite state machines

- Finite state machines:
  - a set of states and
  - next state function (determined by current state and the input)
  - output function (determined by current state and possibly input)
Review: finite state machines

Example:
A fake security device consists of three lights lined up in a row, controlled by the outputs Left, Middle, and Right, which, if asserted, indicate that a light should be on. Only one light is on at a time, and the light “moves” from left to right and then from right to left, thus scaring away thieves who believe that the device is monitoring their activity. The rate of the eye’s movement is controlled by the clock speed (which should not be too great) and that there are essentially no inputs.
Solution

- State Transition diagram

- Next/output functions

- Implementation

FSM-based Control Unit

- 3-bit state register keeps track of the current instruction step.

- FSM Inputs:
  - opcode (IR[31:26]),
  - funct (IR[5:0]),
  - Zero (that is rs==rt)

- Outputs: all control signals
State Transition Diagram

1  2  3  4  5

FSM Control

Next State Function

Output Function

opcode  6
funct  6
zero

NS₀  S₀
NS₁  S₁
NS₂  S₂

ALUop  3
ALUsrcA  2
ALUsrcB  2
IR-Write
The Output Function