CS 201 Lecture 11
Programmable Computers and Instruction Set Architectures

Spring 2014
Plan for today

• Fetch, Decode, Execute
• Instruction Set Architecture Interface
• Programmable Datapath Design
A computer’s finite state machine

Fetch → Decode → Execute 1 → Execute 2 → Execute 3
Fetch & Decode

• The next thing the computer needs to do is out in memory
  – Go get a copy for the datapath to work with
• Okay, we’ve got a copy, what does it mean?
  – Determine the particular sequence of datapath operations needed to complete the operation
Execute

- Different states based on the operations required
- ALU operations often just a single cycle/state
- Memory accesses or more complex operations may take multiple cycles
- Different kinds of operations need different control signals
Instruction interface goals

• Reusable
  – Would like to build a better, faster processor, and still have all of the old programs work
  – Can’t include too many details about the current design

• Easy to write or generate
  – Another reason not to include too many details: microprogramming is a lot of work

• Easy to store in memory
  – Everyone uses 8-bit memory words, so instructions should be some multiple of one Byte
Typical instruction format

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Operands</th>
<th>Result Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Some bits</td>
<td>Some more bits</td>
<td>A few more bits</td>
</tr>
<tr>
<td>Describes operation</td>
<td>Inputs</td>
<td>Output location</td>
</tr>
</tbody>
</table>
# MIPS example instruction

<table>
<thead>
<tr>
<th>Bits</th>
<th>Opcode</th>
<th>Operand 1</th>
<th>Operand 2</th>
<th>Dest.</th>
<th>?</th>
<th>“Opcode” 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>00101</td>
<td>00011</td>
<td>00100</td>
<td>00000</td>
<td>100000</td>
<td></td>
</tr>
<tr>
<td>Means</td>
<td>Arith.</td>
<td>R5</td>
<td>R3</td>
<td>R4</td>
<td>?</td>
<td>Add</td>
</tr>
</tbody>
</table>

\[
R4 \leftarrow R5 + R3
\]
Unified memory

• Most typical computer design today is to have one memory for everything
  – Program data
  – Programs
  – Temporary storage when running out of registers

• Greatly simplifies system design, and is the root issue with many security breaches
  – Send another computer a bunch of “data”, then make it go execute it as code
Implementing the Fetch-Decode-Exec cycle

• Dedicated register to keep track of the current instruction in memory
  – Called the “Instruction Pointer” (Intel) or “Program Counter” (Everybody else)
  – Increments by default after every instruction

• Dedicated register to hold the current instruction
  – “Instruction Register”, or similar name, not typically exposed externally

• Control Unit dedicated to implementing the fetch-decode-execute cycle
What about input?

• Registers can perform arithmetic, but we need other kinds of operands to get data into and out of the datapath

• Other typical operands are
  – Immediates (Imm.): values put into the instruction itself
  – Memory references: directions for how to access memory to load or store data
  – More on these next time
## MIPS basic instruction formats

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R (register)</td>
<td>Opcode (General)</td>
<td></td>
<td>rs (RegA)</td>
<td></td>
<td>rt (RegB)</td>
<td>rd (DestR)</td>
<td>“shamt” Special Imm.</td>
</tr>
<tr>
<td>I (immediate)</td>
<td>Opcode</td>
<td>rs (RegA)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Immediate</td>
</tr>
<tr>
<td>J (memory)</td>
<td>Opcode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Immediate</td>
</tr>
</tbody>
</table>
MIPS Instruction example 1

R12 ← R8 – R2

• Opcode 0 for 2’s complement arithmetic
  – Function code 34
• Write 5-bit unsigned numbers 8, 2, and 12 in the register fields
• Write the binary number for 0 in 5-bit immediate field

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>000000</td>
<td>01000</td>
<td>00010</td>
<td>01100</td>
<td>00000</td>
<td>100010</td>
</tr>
</tbody>
</table>
MIPS Instruction example 2

R4 ← R4 + 1

- Opcode 8 for adding an immediate
  - uses instruction format I
- Use R4 for both the register fields by writing the unsigned binary number 4 in each
- Write the binary number for 1 in 16-bit 2’s complement in the immediate field

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>001000</td>
<td>00100</td>
<td>00100</td>
<td>0000000000000000001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>