MIPS64

- Registers
  - 32 64-bit GPRs (int registers)
  - 32 64-bit fpu registers (dual use)
  - R0=0
  - Additional special purpose registers
- Data types
  - 8-bit byte
  - 2 bytes = half word
  - 4 bytes = word
  - 8 bytes = dword
- Addressing modes
  - Immediate and displacement
  - Register indirect and absolute are easily represented
  - Byte addressable 64bit address
  - Big or little endian
  - Load/Store architecture

I-type instructions

- I-Type instruction:
  - Encodes: Loads and stores of bytes, half words, words, dwords
    - All immediate (rt ← rs op immediate)
    - Ex: Add base register rs to 16 bit offset

<table>
<thead>
<tr>
<th>Op code</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

R-type instructions

- R-Type instruction:
  - Register-Register ALU operations: rd ← rs func rt
    - Function encodes data path operation: add, sub, silt, and, or
    - Read/write special registers and moves

<table>
<thead>
<tr>
<th>Op code</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>func</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>11 bits</td>
</tr>
</tbody>
</table>
J-type instructions

- J-Type instruction:
  - 6 bits for op code
  - 26 bits for immediate

- Encodes:
  - Jump and jump & link
  - Trap and return from exception

In summary...

- Pitfalls
  - Designing “high-level” language instructions
  - Not considering compiler design when targeting code size

- Fallacies
  - There are “typical” programs
  - An architecture with flaws cannot be successful
  - You can design a flawless architecture

Data Path and Control Review

Kirk W. Cameron, Ph.D.
Assistant Professor
Department of Computer Science and Engineering
University of South Carolina

ALU components

1-bit logical unit for AND and OR

3/2 adder

Carry_in

Carry_out

Carry_in

Carry_out
1-bit Simplified ALU

1-bit Simplified ALU w/ subtraction

32 bit ALU (ripple carry)

Edge-triggered Design

Why use edge-triggering?

*input values must be stable when active clock edge returns

\[
a + b + 1 = a + (b + 1) = a + (-b) = a - b
\]
6 bit register using D flip flops

Register Files
- Set of registers that can be read and written by supplying a register number to be accessed
  - One set of registers operated on by a port
- Multiple-read ports: not too difficult
- Multiple-write ports: problems arise

Read ports
- Single Read Port
- Multi Read Port

Write Ports
- Write register
- Write (clock)

Data Path for I-type instruction
- Assuming 32, 64-bit registers (R0..R31)
- Use same register file as R-type

Register Files
- Example: 2 read ports, 1 write port, high-level view

Why not multiple writes?
What if read and write at same time? Who goes first?
Data Path for R-type instruction

- Assuming 32, 64-bit registers (R0..R31)
- Using 2 read ports and 1 write port
  - Inputs: 3 register #s (each 5 bits wide), data for write (64 bits)
  - Outputs: readdata1 and readdata2 (both 64 bits)