All Together:
Instruction Memory +
Arithmetic Machine

Lab 4 debrief – Timing and control will come back in a big way for exam 3 and when we discuss pipelining. Exam 2 (FSMs) is around the corner

Office hours @ 1pm today

Discussion section timeliness

9/20 Clicker data not recorded...grrrr

Handout has 2 pages, grab both
Today’s lecture

- Instructions **control** the **datapath**
  - Instruction Memory
  - Program Counter (PC) is the **address** unit for instruction memory
  - Adder

- Putting all together
  - Arithmetic unit to work
What will Q[3:0] be during the next clock cycle?

a) 0  b) 1  c) 3  d) 4  e) 5
mod-16 Counter

Adder

a) 0x0
b) 0x2
c) 0x4
d) 0x6
e) 0x8
Previously...

- Register-to-register arithmetic instructions use the **R-type format**.

  \[
  \text{add} \ $5, \ $10, \ $4
  \]

- Instructions with immediates all use the **I-type format**.

  \[
  \text{ori} \ $7, \ $2, \ 0x00ff
  \]
Where are the instructions my program executes?

To look at the assembly code of a.out:

$ objdump -d a.out

The instructions executed by the program are in the .text section:

```
.text
main:
    addi $1, $0, 5
```
Programs require memory structures that are much larger than register files
Programs are stored in an instruction memory

We will read the memory but not modify it

.text
main:
    →addi $1, $0, 5
    →sub  $2, $1, $3
The instruction memory is byte addressable

- Addresses are 32-bits
  - # addresses: $2^{32} = 4 \text{ G}$
- Each address contains 1 byte
  - Instructions occupy four contiguous locations
- Memory stores 4Gbytes

Instructions - 3 2 bits
MIPS instructions start at an address that is divisible by 4

- 0, 4, 8 and 12 are valid instruction addresses.
- 1, 2, 3, 5, 6, 7, 9, 10 and 11 are not valid instruction addresses.

Address

8-bit data

Instruction 1
Instruction 2
Instruction 3
A special register called Program Counter (PC) contains the address of the next instruction to execute.
Use an adder to increment PC to the next instruction
Redrawn to match the MIPS diagram
Why aren’t 2 LSbs provided?
- a) Bug in the slide
- b) Memory is only $2^{30}$ big
- c) Bits [1:0] are always 2’b00
- d) Velociraptors ate them
MIPS datapath with a controlling instruction memory and program counter
Example

My program
$3 = 10$
$5 = -7$
$7 = $3 + $5$

Assembly

What value will be stored in register 7 at the end of the program?

a) -7
b) 3
c) 5
d) 8
e) 10
Example

My program

\[
\begin{align*}
\$3 & = 10 \\
\$5 & = -7 \\
\$7 & = \$3 + \$5
\end{align*}
\]

Assembly

\[
\begin{align*}
\text{Answer A} & : & \text{addi} & \$3, \$0, 0x000A \\
& & \text{subi} & \$5, \$0, 0x0007 \\
& & \text{add} & \$7, \$3, \$5 \\
\text{Answer B} & : & \text{addi} & \$3, \$0, 0x000A \\
& & \text{addi} & \$5, \$0, 0xFFF9 \\
& & \text{add} & \$7, \$3, \$5 \\
\text{Answer C} & : & \text{addi} & \$3, \$0, 0x000A \\
& & \text{addi} & \$5, \$0, 0xFFF8 \\
& & \text{add} & \$7, \$3, \$5 \\
\text{Answer D} & : & \text{add} & \$3, \$0, 0x000A \\
& & \text{sub} & \$5, \$0, 0x0007 \\
& & \text{add} & \$7, \$3, \$5
\end{align*}
\]
Example

My program

$3 = 10$
$5 = -7$
$7 = $3 + $5$

Assembly

addi $3, $0, 0x000A
addi $5, $0, 0xFFF9
add $7, $3, $5

Machine code

\begin{align*}
\text{addi} & \quad \text{rt} \quad \text{rs} \quad \text{imm} \\
& \quad 00 \, 00 \, 00 \, 00 \, 0A \\
\text{op} & \quad \text{rs} \quad \text{rt} \quad \text{imm} \\
\text{add} & \quad \text{rt} \quad \text{rs} \\
& \quad 00 \, 10 \, 00 \, 00 \, 0F \\
\text{add} & \quad \text{rt} \quad \text{rs} \quad \text{imm} \\
& \quad 00 \, 11 \, 10 \, 00 \, 09 \\
\text{add} & \quad \text{rt} \quad \text{rs} \quad \text{rd} \quad \text{sham} \quad \text{funct} \\
& \quad 00 \, 00 \, 00 \, 11 \, 10 \, 00 \, 00 \, 00 \, 09 \, 00 \, 0A \\
\end{align*}
Little Endian - Least significant bits (little end) go first

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>0x0A</td>
</tr>
<tr>
<td>0x00000001</td>
<td>0x00</td>
</tr>
<tr>
<td>0x00000002</td>
<td>0x03</td>
</tr>
<tr>
<td>0x00000003</td>
<td>0x20</td>
</tr>
<tr>
<td>0x00000004</td>
<td>0xFFF9</td>
</tr>
<tr>
<td>0x00000005</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>0x00000006</td>
<td>0x05</td>
</tr>
<tr>
<td>0x00000007</td>
<td>0x20</td>
</tr>
<tr>
<td>0x00000008</td>
<td></td>
</tr>
<tr>
<td>0x00000009</td>
<td></td>
</tr>
<tr>
<td>0x0000000A</td>
<td></td>
</tr>
<tr>
<td>0x0000000B</td>
<td></td>
</tr>
<tr>
<td>0x0000000C</td>
<td></td>
</tr>
<tr>
<td>0x0000000D</td>
<td></td>
</tr>
</tbody>
</table>

Assembly:
- `addi $3, $0, 0x000A` - Machine: 0x2003000A
- `addi $5, $0, 0xFFF9` - Machine: 0x2005FFF9
- `add $7, $3, $5` - Machine: 0x00C53820
Big Endian – Most significant bits (big end) go first

Assembly: addi $3, $0, 0x000A
Machine: 0x2003000A

Assembly: addi $5, $0, 0xFFF9
Machine: 0x2005FFF9

Assembly: add $7, $3, $5
Assembly: 0x00C53820
addi $3, $0, 0x000A
**add**  $7,  $3,  $5

```
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>000000</td>
</tr>
<tr>
<td>rs</td>
<td>00011</td>
</tr>
<tr>
<td>rt</td>
<td>00101</td>
</tr>
<tr>
<td>rd</td>
<td>00111</td>
</tr>
<tr>
<td>shamt</td>
<td>10011</td>
</tr>
<tr>
<td>func</td>
<td>100000</td>
</tr>
</tbody>
</table>
```

What value is on the bus?

- a) -7
- b) 3
- c) 5
- d) 7
- e) 10

**MIPS decoder**

```
opcode[5:0]  | write_enable
rd_src      | rd_writeEnable
alu_src2    | except
alu_op[2:0] |
```

**Sign Extender**

```
in[15:0]  | imm16
out[31:0] |
imm32     | 32
```

**ALU**

```
alu_op[2:0]  | wr_enable
alu_src2     | rd_src
A[31:0]      | Rs
B[31:0]      | Rt
out[31:0]    |
```

overflow  
zero  
negative