All Together: Instruction Memory + Arithmetic Machine
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
Previously...

- Register-to-register arithmetic instructions use the **R-type** format.
  
  \[ \text{add } \$5, \$10, \$4 \]

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</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>5 bits</td>
<td>6 bits</td>
</tr>
</tbody>
</table>

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

  \[ \text{ori } \$7, \$2, 0x00ff \]

<table>
<thead>
<tr>
<th>op</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>
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
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.
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
MIPS datapath with a controlling instruction memory and program counter
Example

My program

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

Assembly
Little Endian - Least significant bits (little end) go first

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

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

Assembly:   add  $7, $3, $5
Assembly: 0x00C53820
Big Endian – Most significant bits (big end) go first

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>0x00000001</td>
<td></td>
</tr>
<tr>
<td>0x00000002</td>
<td></td>
</tr>
<tr>
<td>0x00000003</td>
<td></td>
</tr>
<tr>
<td>0x00000004</td>
<td></td>
</tr>
<tr>
<td>0x00000005</td>
<td></td>
</tr>
<tr>
<td>0x00000006</td>
<td></td>
</tr>
<tr>
<td>0x00000007</td>
<td></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

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

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