All Together:
Instruction Memory + Arithmetic Unit
Today’s lecture

- Instructions
  - Instruction Memory
  - Program Counter (PC)
  - Adder

- Putting all together
  - Arithmetic unit to work
Friday’s lecture

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

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

  \[
  \begin{array}{cccccc}
  \text{op} & \text{rs} & \text{rt} & \text{rd} & \text{shamt} & \text{func} \\
  6 \text{ bits} & 5 \text{ bits} & 5 \text{ bits} & 5 \text{ bits} & 5 \text{ bits} & 6 \text{ bits}
  \end{array}
  \]

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

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

  \[
  \begin{array}{cccc}
  \text{op} & \text{rs} & \text{rt} & \text{address} \\
  6 \text{ bits} & 5 \text{ bits} & 5 \text{ bits} & 16 \text{ bits}
  \end{array}
  \]
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
```

```
void main() {
    int a = 0;
    int b = a+5;
}
```
Where are the instructions my program executes?

- The program is stored in Memory
- Assume our program is stored in a Read Only Memory (ROM)
  - We can read its contents, but cannot modify them
  - A 32-bit address serves as an array index
    - # addresses: \(2^{32} = 4\) G
    - Each address contains 1 byte:
      - 4Gbytes
  - MIPS memory is byte-addressable, so a 32-bit instruction actually occupies four contiguous locations (bytes) of memory
Memory alignment

- 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.

<table>
<thead>
<tr>
<th>Address</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit data</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>

Instruction 1  Instruction 2  Instruction 3
How do we know which instruction to execute?

We have a register called Program Counter (PC) that contains the address of the next instruction to execute.
What do you get if you connect a register to an adder?
Implementing counters
Instruction Memory + PC + Adder
Putting all together
Example

My program

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

Assembly
Example

My program

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

Assembly

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

Machine code

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

opcode
funct
add 0x00 0x20
addi 0x08
Program Counter (PC)

Adder

nextaddr [31:0]

clk

reset

enable

Instruction Memory

data[31:0]

addr[31:0]

0x0

0x4

0x8

0xC

addr[31:0]

0

0

...
Putting all together

```
addi $3, $0, 0x000A
```

**MIPS decoder**
- **opcode[5:0]**
- **alu_op[2:0]**
- **write_enable**
- **wr_enable**

**Sign Extender**
- **in[15:0]**
- **out[31:0]**
- **imm16**
- **imm32**

**Register File**
- **reset**
- **clk**
- **wr_enable**
- **Rdest**
- **Rt**
- **Rs**

**ALU**
- **A[31:0]**
- **B[31:0]**
- **out[31:0]**
- **alu_op[2:0]**

**Zero, Negative, Overflow**
- **zero**
- **negative**
- **overflow**

**alu_op**
- **alu_op[2:0]**

**inst**
- **inst[31:26]**
- **inst[25:21]**
- **inst[20:16]**
- **inst[15:11]**
- **inst[10:6]**
- **inst[5:0]**

**rs, rt, imm**
- **rs, rt**
- **imm**

**alu_op**
- **alu_op[2:0]**

**write_enable**
- **write_enable**

**itype**
- **itype**
Putting all together

add $7, $3, $5