All Together:
Instruction Memory +
Arithmetic Machine

\[ x = x + 1, \]

CS 233
prof. Zilles (subbing for prof. Herman)
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
We use finite state machines to control datapaths, how do we create a programmable finite state machine?
Use indirection to keep track of where things are stored and not the actual things

“Any problem in CS can be solved by adding a level of indirection”

Without Indirection

With Indirection
Design a FSM that uses addresses to tell us where our instructions are stored

Finite State Machine

Instruction memory

State Storage

State Manipulations (ALU)
Program instructions are stored in a new, larger RAM

myprogram.c

void main() {
    int a = 0;
    int b = a+5;
}

gcc myprogram.c

a.out

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.

```assembly
.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** is associated with **1 byte**
  - Instructions occupy four contiguous locations
- **Memory stores 4Gbytes**

$$2^b = 64$$

$$2^x = 1024$$

$$2^y = 4$$

$$2^30 \approx 10^9$$
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.
a) 0x1
b) 0x2
c) 0x3
d) 0x4
e) 0x5
Our FSM is an adder that increments 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

\[ \$3 = 10 \]
\[ \$5 = -7 \]
\[ \$7 = \$3 + \$5 \]

Assembly

Answer A

add $3, $0, 0x000A
subi $5, $0, 0x0007
add $7, $3, $5

Answer B

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

Answer C

addi $3, $0, 0x000A
addi $5, $0, 0xFFF8
add $7, $3, $5

Answer D

add $3, $0, 0x000A
sub $5, $0, 0x0007
add $7, $3, $5

\[ \text{in} \ a = \text{char} \ b \ b \ c \ c \ \text{df} \]
\[ \text{char} \ c = \ast (\text{char} + \text{df}) \]
**Example**

**My program**

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

**Assembly**

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

<table>
<thead>
<tr>
<th>opcode</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>0x00</td>
</tr>
<tr>
<td>addi</td>
<td>0x08</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Machine code</th>
</tr>
</thead>
<tbody>
<tr>
<td>$7 = 3 + 5$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>imm</th>
</tr>
</thead>
<tbody>
<tr>
<td>addi</td>
<td>$3$</td>
<td>$0$</td>
<td>0x000A</td>
</tr>
<tr>
<td>addi</td>
<td>$5$</td>
<td>$0$</td>
<td>0xFFF9</td>
</tr>
<tr>
<td>add</td>
<td>$7$</td>
<td>$3$</td>
<td>$5$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>$7$</td>
<td>$3$</td>
<td>$5$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

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
What decimal value is on the bus?
a) -7  
b) 3  
c) 5  
d) 7  
e) 10