Where are we?

- Instruction set architecture (e.g., MIPS)
  - Compiler: translating source code (C or Java) Programs to assembly language And linking your code to Library code
  - Instruction set architecture (e.g., MIPS)
   - Memory system
   - Processor
   - I/O system
   - Datapath & Control
   - Digital logic

How the software talks To the hardware

How a processor runs MIPS Programs!

How switches (1 or 0) can be used to build Interesting functions: from integer arithmetic to programmable computers
Meeting 2: Bits, bytes, and memory
Brandon Myers
University of Iowa
Arithmetic on numbers

Number to track the current line in the program

Storing numbers
Caveat...we will build everything from switches (on or off)

• A binary digit (or, bit) has two possible values 0 or 1
• Just like a decimal digit has 10 possible values 0-9

Hundreds | tens | ones

193₁₀ = 1  9  3  digit is a power of 10

fours | twos | ones

101₂ = 1  0  1  digit is a power of 2
Peer instruction

• Translate the following binary numbers to decimal
  • $10_2 = \ldots_{10}$
  • $111_2 = \ldots_{10}$
  • $10101_2 = \ldots_{10}$

• Bonus, translate this fractional number
  • $11.01_2$
Bits: for more than just numbers

Bits can be used to encode (represent) ANY kind of data, yes any data

Sample analog signal (displacement of microphone diaphragm) once every 23us (44.1KHz) – store it digitally as an 8-bit number
Peer instruction

How many unique values can you represent with 3 bits?
Peer instruction

• How many bits are needed to represent the number $\pi$?
  a) 1 bit
  b) infinite bits
  c) 3 bits
  d) 8 bits
  e) Can’t be done
Counting with N bits (N=3)
What about negative integers?

Positive integers only ("unsigned")

Make some space for them!

Positive & negative integers ("signed")
Two’s complement just works!

• $11001 = (-1) \times 16 + (1) \times 8 + (0) \times 4 + (0) \times 2 + 1 \times 1$
  
  $= -16 + 9$
  
  $= -7$

• $1001 = (-1) \times 8 + (0) \times 4 + (0) \times 2 + 1 \times 1$
  
  $= -7$

Or the invert and add 1 trick

$-7 = 11001 \rightarrow 00110 \rightarrow 00111 = 7$

invert +1
Peer instructions:
Two’s complement and Hex

1. What is $-11_{10}$ in binary (using two’s complement encoding for integers)?

2. $A1_{16} = ____10$
Another common base: 16

<table>
<thead>
<tr>
<th>Binary (2)</th>
<th>Decimal (10)</th>
<th>Hexadecimal (16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>00001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>00010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>00011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>00100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>00101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>00110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>00111</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>01000</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>01001</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>01010</td>
<td>10</td>
<td>A</td>
</tr>
<tr>
<td>01011</td>
<td>11</td>
<td>B</td>
</tr>
<tr>
<td>01100</td>
<td>12</td>
<td>C</td>
</tr>
<tr>
<td>01101</td>
<td>13</td>
<td>D</td>
</tr>
<tr>
<td>01110</td>
<td>14</td>
<td>E</td>
</tr>
<tr>
<td>01111</td>
<td>15</td>
<td>F</td>
</tr>
<tr>
<td>10000</td>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>10001</td>
<td>17</td>
<td>11</td>
</tr>
</tbody>
</table>
A note on conventions

• Sometimes we indicate a binary number with prefix 0b
  • $10101_2$ or $0b10101$

• Sometimes we indicate a hex number with prefix 0x
  • $FA41_{16}$ or $0xFA41$
Where we are going next

Bits store whatever you want, including integers. How do we manipulate sequences of bits?

10110000

Q: How do we arrange bits in the memory of the computer? (why do we care? we want the computer to store many individual numbers)
A: *bytes* and *words*

Q: How do we name or refer to all those individual numbers in memory?
A: *addresses* and *pointers*
Arithmetic on numbers

Number to track the current line in the program

Storing numbers
Instruction memory → Execution engine → Data memory

Arithmetic on numbers

Number to track the current line in the program

Storing numbers
organizes its bits as **bytes** and **words**
can do bitwise operations (and other stuff)
Back to bits soon, but first...memory!
Organizing bits

- **Byte**: a unit of data
- In the MIPS architecture, bytes are 8-bits long and are the smallest unit of data the architecture gives a name to (more on names soon)
  - True of most other modern architectures, too

```
00011010  1 byte, perhaps storing the integer 26_{10}
```

```
1A
We also like to write the value of a byte as two hex digits
```
Memory organization

We think of memory as one big array of bytes

```
byte[] memory = new byte[NUM_BYTES];
```

- Just like every element in an array has an index, every byte in memory has an index, called its **address**.
- Use the address to find the byte to read or write it.
What to do now

• HW 1 and Quiz 1
• New readings are up
• Buy your clicker license (if you don’t yet have one) and go through getting started doc
  • linked from the syllabus
  • We’ll try it Monday/Wed and start keeping track Friday
• Take a look at syllabus for office hours / Debug Your Brain
Organize bytes into machine words

(note address in hex)

<table>
<thead>
<tr>
<th>0000</th>
<th>0001</th>
<th>0002</th>
<th>0003</th>
<th>0004</th>
<th>0005</th>
<th>0006</th>
<th>0007</th>
<th>0008</th>
<th>0009</th>
<th>000A</th>
<th>000B</th>
<th>000C</th>
<th>000D</th>
<th>000E</th>
<th>000F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>1A</strong></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>

32-bit words

64-bit words
MIPS has 32-bit addresses

What is the largest number of bytes the memory of a MIPS computer can have?
Looking at memory as words
Storing values

• store the value integer $15_{10}$ at address $0x0008$
Storing values

- store the \textbf{value} integer $15_{10}$ at \textbf{address} 0x0014
- store the \textbf{value} integer $20_{10}$ at \textbf{address} 0x0000
Addresses can be values, too!

- store the value integer $15_{10}$ at address $0x0014$
- store the value integer $20_{10}$ at address $0x0000$
- store the value address $0x0014$ at address $0x001C$
Addresses can be values, too!

- store the **value** integer \(15_{10}\) at **address** 0x0014
- store the **value** integer \(20_{10}\) at **address** 0x0000
- store the **value** **address** 0x0014 at **address** 0x001C
  - we say the word at **address** 0x001C is a **pointer** to the integer at **address** 0x0014
Addresses can be values, too!

- store the value integer $15_{10}$ at address $0x0014$
- store the value integer $20_{10}$ at address $0x0000$
- store the value address $0x0014$ at address $0x001C$
  - we say the word at address $0x001C$ is a **pointer** to the integer at address $0x0014$
- store the value address $0x001C$ at address $0x0028$
  - we say the word at address $0x0028$ is a **pointer** to a **pointer** to an integer
How arrays look in memory

```java
int[] arr = new int[3];
```
How arrays look in memory

```java
int[] arr = new int[3];
arr[0] = 13;
```
How arrays look in memory

```java
int[] arr = new int[3];
arr[0] = 13;
arr[1] = 10;
```
How arrays look in memory

```java
int[] arr = new int[3];
arr[0] = 13;
arr[1] = 10;
arr[2] = 16;
```
Peer instruction

• Suppose we allocate this array:
  ```java
  int[] arr = new int[7];
  ```
  and Java decided to put the first byte of the array (first byte of arr[0]) at address 0x04

What is the address of arr[5]?

a. 0x04
b. 0x14
c. 0x09

address = base address + index * element_size
d. 0x54
e. 0x05
f. 0x18
## Size of data types (in bytes)

<table>
<thead>
<tr>
<th>Java data type</th>
<th>size in 32-bit architecture</th>
<th>size in 64-bit architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>byte</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>float (later!)</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>reference (stores a memory address)</td>
<td><strong>4</strong></td>
<td><strong>8</strong></td>
</tr>
</tbody>
</table>

slide inspired by UW CSE351
Administrivia: More office hours!

- Xin’s office hours now scheduled!
- Monday 4:30pm - 6:00
- Friday 9:30am – 11
- 101N in Maclean Hall (MLH)
Aside: naming powers of two

• One of our favorite formulas: how many unique things can you represent with N bits? $2^N$ things

• Naming conventions

<table>
<thead>
<tr>
<th>N = ?</th>
<th>name</th>
<th>Close (but not equal!) to power of 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Kibi (Ki)</td>
<td>$10^3$ or Kilo (K)</td>
</tr>
<tr>
<td>20</td>
<td>Mebi (Mi)</td>
<td>$10^6$ or Mega (M)</td>
</tr>
<tr>
<td>30</td>
<td>Gibi (Gi)</td>
<td>$10^9$ or Giga (G)</td>
</tr>
<tr>
<td>40</td>
<td>Tebi (Ti)</td>
<td>$10^{12}$ or Tera (T)</td>
</tr>
<tr>
<td>50</td>
<td>Pebi (Pi)</td>
<td>$10^{15}$ or Peta (P)</td>
</tr>
<tr>
<td>60</td>
<td>Exbi (Ei)</td>
<td>$10^{18}$ or Exa (E)</td>
</tr>
</tbody>
</table>

Name these numbers of bytes:

$2^{11}$ bytes = _______

$2^{32}$ bytes = _______

$2^{59}$ bytes = _______
Where we are going (today)

Bits store whatever you want, including integers. How do we manipulate sequences of bits?

10110000

Q: How do we arrange bits in the memory of the computer? (why do we care? we want the computer to store many individual numbers)
A: bytes and words

Q: How do we name or refer to all those individual numbers in memory?
A: addresses and pointers

DONE (for now)
Changing the number of bits

• Often we need to change the number of bits we are using to store a number (why?)

• $12_{10}$ using 4 bits is $1100_2$
  
  • If we store it in 8 bits: $00001100_2$

• $-5_{10}$ using 4 bits is: $1011$
  
  • If we store it in 8 bits: _______________ $2$

We call this operation **sign extension**: copy the leftmost bit in 4-bit number to the new 4 leftmost bits in the 8-bit number
Representing sets with bits

• 4-bit vector represents subsets of \{3,2,1,0\}
• 1 means in the set, 0 means not in the set

• A is the set \{1,0\}, encoded as 0011
• B is the set \{2,0\}, encoded as 0101

• Set operations using bitwise operators
  • A&B Intersection 0001 (1 iff both are 1)
  • A|B Union 0111 (1 iff at least one 1)
  • A^B symmetric difference 0110 (1 iff exactly one 1)
  • ~B complement 1010 (1 iff 0)

slide inspired by UW CSE351
Shift

• shift: move bits left or right

• Left shift: 5 $<<$ 2

• Right shift: 5 $>>$ 1
Peer instruction

What is the integer result from evaluating this expression? (assume integers are 32 bits)

\[(7 \ll 2) \& 15\]
Let’s play a card game

• Come up with a binary encoding for a 52-card deck

Slide inspired by CSE351 Spring 2016
Operations on a pair of cards

• We want the following operations to be easy to implement
  • Compare two cards, which is higher value?
  • Compare two cards, are they the same suit?
Where we are headed next

Instruction memory

Execution engine

Data memory

can do bitwise operations (and other stuff)

Stores programs!

organizes its bits as **bytes** and **words**
What to do now

• HW 1 is out today: **due next Thursday**
• Buy your clicker license (if you don’t yet have one) and go through getting started doc
• Vote on Debug Your Brain availability if you haven’t yet
• < 30 posts on the “Introduce yourself...” discussion thread; go do it