Where we are going (today)

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*

```
first
10110000 00001110 01000010 11110001
```

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

```
second
0 1 2 3
10110000 00001110 01000010 11110001
```

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

```
10110000
Lab1
```
Where we are going

Compiler
- translating source code (C or Java)
- Programs to assembly language
- And linking your code to
- Library code

Instruction set architecture (e.g., MIPS)

How the software talks
- To the hardware

Memory system

Processor

I/O system

Datapath & Control

How a processor runs MIPS
- Programs!

Digital logic

How switches (1 or 0) can be used
- to build Interesting functions:
- from integer arithmetic to
- programmable computers
CS 2630
Computer Organization

Meeting 3: memory organization and addresses
Brandon Myers
University of Iowa
Processor

Instruction memory → Execution engine ← Data memory

organizes its bits as **bytes** and **words**
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

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00011010</td>
<td>1A</td>
<td>1 byte, perhaps storing the integer $26_{10}$</td>
</tr>
<tr>
<td>1A</td>
<td></td>
<td>We also like to write the value of a byte as two hex digits</td>
</tr>
</tbody>
</table>
Memory organization

We think of memory as one big array of bytes

| 1A |  |  |  |  |  |  |

Analogy in Java?

```java
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.
• 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

---

Peer instruction:
A “MIPS computer” has addresses that are 32-bit integers

**What is the largest number of bytes the memory of a MIPS computer can have?**
How do I refer to an individual cow? Name it; like farmed cows, our bytes will have numbers as names.
Organize bytes into machine **words**

(in this diagram, we are writing the address in hex)

<table>
<thead>
<tr>
<th>bytes (or, 8 bits)</th>
<th>4-byte (or, 32 bits) <strong>words</strong></th>
<th>8-byte (or, 64 bits) <strong>words</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0002</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0003</td>
<td>1A</td>
<td></td>
</tr>
<tr>
<td>0x0004</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x000F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

```
10110000
00001110
01000010
11110001
```
done (for now)

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

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>10110000</td>
<td>00001110</td>
<td>01000010</td>
<td>11110001</td>
</tr>
</tbody>
</table>
```

next
Looking at memory as words
Storing values

- store the value \(15_{10}\) at address \(0x00000014\)
Storing values

• store the value integer $15_{10}$ at address $0x00000014$

• store the value integer $20_{10}$ at address $0x00000000$
Addresses can be values, too!

• store the value integer $15_{10}$ at address 0x00000014

• store the value integer $20_{10}$ at address 0x00000000

• store the value address 0x0014 at address 0x0000001C
Addresses can be values, too!

- store the value integer $15_{10}$ at address $0x00000014$
- store the value integer $20_{10}$ at address $0x00000000$
- store the value address $0x0014$ at address $0x0000001C$
  - 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 $0x00000014$
- store the value integer $20_{10}$ at address $0x00000000$
- store the value address $0x0014$ at address $0x0000001C$
  - we say the word at address $0x0000001C$ is a pointer to the integer at address $0x00000014$
- store the value address $0x001C$ at address $0x00000028$
  - we say the word at address $0x00000028$ is a pointer to a pointer to an integer
Peer instruction

1. Draw this memory on the board

2. Fill in the memory so that the word at address $0x00000010$ holds the integer $4096_{10}$.

3. Fill in the memory so that the word at address $0x00000000$ holds a pointer to the integer from question 2.
Waitlist

• bring your Registrar form to me @ today, OH, DebugYourBrain if you haven’t already
• I will return some signed on Wednesday in class if seats are available
How arrays look in memory

```java
int[] arr = new int[3];
```

```
0x0000
0x0004
0x0008
0x000C
0x0010
0x0014
0x0018
0x001C
0x0020
0x0024
0x0028
```
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  
d. 0x54  
e. 0x05  
f. 0x18  

address = base address + index * element_size

https://b.socrative.com/login/student/
room name: CS2630
name is optional
### 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>4</td>
<td>8</td>
</tr>
</tbody>
</table>

*slide inspired by UW CSE351*
Administrivia:

- Survey results...?
Slip days PER ASSIGNMENT

result: 2

<table>
<thead>
<tr>
<th>Minimum</th>
<th>Maximum</th>
<th>Mean</th>
<th>Std Deviation</th>
<th>Variance</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>7.00</td>
<td>2.38</td>
<td>1.27</td>
<td>1.61</td>
<td>32</td>
</tr>
</tbody>
</table>
Slip days TOTAL

<table>
<thead>
<tr>
<th>Minimum</th>
<th>Maximum</th>
<th>Mean</th>
<th>Std Deviation</th>
<th>Variance</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.00</td>
<td>21.00</td>
<td>6.09</td>
<td>3.99</td>
<td>15.90</td>
<td>32</td>
</tr>
</tbody>
</table>
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

NEXT, Lab1

DONE (for now)
What to do now

• Readings
• finish HW 1
• finish Quiz 1
• Pre-lab for Lab 1 on Wednesday
Instruction memory → Execution engine → Data memory

can do bitwise operations (and other stuff)
Op 1. 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
Op 2. Bitwise operations

<table>
<thead>
<tr>
<th>Logical AND (&amp;)</th>
<th>Logical OR (|)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011</td>
<td>0011</td>
</tr>
<tr>
<td>1010</td>
<td>1010</td>
</tr>
<tr>
<td><strong>0010</strong></td>
<td><strong>1011</strong></td>
</tr>
<tr>
<td>XOR (^)</td>
<td>negate (~)</td>
</tr>
<tr>
<td>0011</td>
<td>0110</td>
</tr>
<tr>
<td>1010</td>
<td>0110</td>
</tr>
<tr>
<td><strong>1001</strong></td>
<td><strong>1001</strong></td>
</tr>
</tbody>
</table>
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
Op 3. Shift

• shift: move bits left or right

• Left shift: $6 \ll 2$

• Right shift: $6 \gg 1$

(diagrams show that I am storing the integers using 5 bits)
Peer instruction

What is the integer result from evaluating each of the expressions? (assume integers are 32 bits)

(7 << 2) & 15

(15 >> 2) | 8 \hspace{1cm} \text{assume} \gg \text{is right logical shift}
Let’s play a card game

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

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?
Summary and what’s next

TODAY can do bitwise operations (and other stuff)

TODAY organizes its bits as bytes and words

NEXT Stores programs!

Instruction memory  Execution engine  Data memory