CS 2630
Computer Organization

Meeting 2: Bits, bytes, and memory

Brandon Myers
University of Iowa
Where are we?

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

How the software talks To the hardware

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
Arithmetic on numbers

Number to track the current line in the program

Storing numbers
Arithmetic on numbers

Instruction memory

Execution engine

Data memory

Number to track the current line in the program

Storing numbers
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
Instruction memory

Execution engine

Data memory

can do bitwise operations (and other stuff)

organizes its bits as bytes and words
1. What is $-11_{10}$ in binary (using two’s complement encoding for integers)?

2. $A1_{16} = ______{10}$
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
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

Analogy in 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.
Organize bytes into machine words

(note address in hex)

0000
0001
0002
0003
0004
0005
0006
0007
0008
0009
000A
000B
000C
000D
000E
000F

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

<table>
<thead>
<tr>
<th>0000</th>
<th>0004</th>
<th>0008</th>
<th>000C</th>
<th>0010</th>
<th>0014</th>
<th>0018</th>
<th>001C</th>
<th>0020</th>
<th>0024</th>
<th>0028</th>
</tr>
</thead>
<tbody>
<tr>
<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>
Storing values

- store the **value** integer $15_{10}$ at **address** 0x0008

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0008</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>OF</td>
</tr>
<tr>
<td>000C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0014</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0018</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0020</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0024</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0028</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Storing values

• store the value integer $15_{10}$ at address 0x0014

• store the value integer $20_{10}$ at 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 \(15\) at address \(0x0014\)

- store the value integer \(20\) 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
int[] arr = new int[3];
How arrays look in memory

```java
int[] arr = new int[3];
arr[0] = 13;
```
int[] arr = new int[3];
arr[0] = 13;
arr[1] = 10;

How arrays look in memory
How arrays look in memory

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

![Memory layout of arrays](image)
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
## 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>
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 = _______
Bits store whatever you want, including integers. How do we manipulate sequences of bits?

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\}

• A is the set \{1,0\} \rightarrow 0011
• B is the set \{2,0\} \rightarrow 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**: $5 \ll 2$

- **Right shift**: $5 \gg 1$
Peer instruction

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

(7 << 2) & 15
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?
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