In-class
Notes
(Lecture 1)
CS 354

REMEKI ARPADC - DUSSEAU

Machine Org + Programming

How stuff works

Today

Start:
- How computer works
  - right now
@end:
- course logistics

Breaks
Three Parts:

1) C programming language

2) Lower-level programming (Assembly)

3) Beyond processing: input/output, mem systems,

Books:
Processor

Central Processing Unit (CPU)

executes instructions:

add (example)

speed:

billions/sec
**Processing:**
Program is running

**Loop:**
Processor:

1. Fetch instruction from memory
2. Decode (figure out what the inst. does)
3. Execute the instruction
4. Get next instruction

**Memory:**
- DRAM
- 10s or 100s of nano-sec

**Processor:**
- Exec. inst. in 
  ~1 nano sec

**Time:**
- Seconds
- Milli 1/1000
- Micro 1/1,000,000
- Nano 1/1e9
C programming language

Put C code/text on screen, want to understand each character

Bits/Binary/Hex

Inside computers:
Everything is represented by 0's and 1's

Why? real world of circuitry (analog)

=> digital

= high

= low

voltage high 5V

voltage low 2V

reliably distinguish
Represent as 1's and 0's

Numbers:

\[ \Rightarrow 5 \Rightarrow \]

= ENIAC

=) decimal #5 = Bit \Rightarrow Binary Digit

2 bits

00
01
10
11

=) 4 possibilities

\[ 2^2 \]

10 bits \Rightarrow 2^{10} possibilities
Number in memory of computer:

3-bit binary

<table>
<thead>
<tr>
<th>X₂</th>
<th>X₁</th>
<th>X₀</th>
<th>X₁' = 0 or 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>

$100_{10} + 001_{10} = 101_{10}$

N bits $\Rightarrow 2^N$ possibilities
binary: a bit verbose for people

32-bit numbers (64-bit)

\[
\begin{array}{c}
3 \\
0011 \\
9 \\
01001 \\
01111 \\
1001 \hline \\
0000 \\
1011 \\
1010 \\
F \\
\end{array}
\]

Hexadecimal:

- each character: 16 values
- 0 ... 15
- 0 1 2 ... 9 A B C D E F
- 10 11 12 13 14 15

\[\text{0x AAAA A0000} \]

- Zero and X
Memory Abstraction / Model

of Memory
8 bits: 1 byte

each slot has name

address

read (2) => 10
write (4, 21)

max

Operations:

read (address) => returns value/data @ address

write (address, data) => updates memory @ address to hold value
4 GB of memory

address:

how many bits do you need in address to refer to all 4 GB?

4 addresses

2 bits

2^2

11

10

1024 bytes

2^10

10 bits

1 million / MB: 1024 · 1024

2^20 · 2^10 = 2^30

20 bits

1 billion / GB: 1024 · 1024 · 1024

2^30 · 2^2 = 2^32

30 bits + 2 bits

32 bits

=> 32 bits => 4 GB memory
C : programming language

Numbers
5
23
-6
0x10

whole numbers
negative, positive integers

Operators (over integers)
=) divide
add
multiply
subtract
mod/remainder

unary operators
-3 + 6

binary operators
2 entities
3 + 5
operators: binary
+ - * / %

expression:

3 + 5 \text{ first} \Rightarrow 8
\[ (6 + 6) \div 2 \Rightarrow \times 6 \]

order: precedence

higher: multiply, divide, remainder
lower: addition, subtraction

left \rightarrow right

8 \times 3 \div 2 \Rightarrow 12

TIP: when in doubt, parenthesize
Law: C int is not a mathematical pure integer

C int: 4 byte
(32 bits)
positive and negative
and 0 (zero)

3 bits (not 32)

\[ X_2 X_1 X_0 \]

\[ 2^3 \Rightarrow 8 \text{ numbers} \]

\[ -4 \Rightarrow 3 \]
\[ -3 \Rightarrow 4 \]

Positive:
- 001 = 1
- 010 = 2
- 011 = 3

Sign:
- 0 = positive
- 1 = negative

000 = 0

101 = -1
110 = -2
111 = -3

Wrong:

\[ 111 \]
2's complement encoding

3 bits (for example)

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>-4</td>
<td>100</td>
</tr>
<tr>
<td>-3</td>
<td>101</td>
</tr>
<tr>
<td>-2</td>
<td>110</td>
</tr>
<tr>
<td>-1</td>
<td>111</td>
</tr>
</tbody>
</table>

\[\begin{array}{c}
1 + (-2) \\
\hline
-1 \\
\end{array}\]

\[\begin{array}{c}
001 \\
\hline
110 \\
1
\end{array}\]

\[\begin{array}{c}
2 + (-1) \\
\hline
1
\end{array}\]

\[\begin{array}{c}
010 \\
+ 111 \\
\hline
001 \\
\hline
1
\end{array}\]
Assignment: \( \frac{\text{new operator}}{=} \) later

Variable: \( \text{named container} \)

Readability \( \Rightarrow \) names for humans

C: define integer

\[
\text{int } x, y;
\]

Legal names:
- lowercase letters
- uppercase letters
- underscore (\(-\))
- digits (numeric)

\( \text{int } x-y \)
```c
int x;
in+ y;

new operator == (equals) = (lower precedence)

x = 3;
y = 10 + (5/2);

x = (y + 2);

precedence
```

![Memory diagram](image-url-here)
What is a good name?

```c
int x;
int xx;
int y;
int yyyy;
```

\[ x = \left\lfloor \frac{y}{2} \right\rfloor \]

mean = sum / count;

[READABILITY]

- Tension: concise, readable
- The sum of all the numbers, S
Comparisons: integers

>  <  >=  <=

!=  ==

result comparison:

True  False

1  0
(int) (int+)