Under the Hood: Memory and Bit Operations

CPS 104 Lecture 3

Administrivia

• Homework #1 Due Jan 21
• Memory
• Bitwise operations

Outline
• Review data representations
• Arrays
• Pointers
• Pointer Arithmetic
• Bitwise operations (AND, OR)

Reading
• Chapter 3 (next few lectures)

Review: 2's Complement Negation and Addition

• To negate a number do:
  ➢ Step 1. complement the digits
  ➢ Step 2. add 1

Example

14_{10} = 001110_2
-14_{10} = 110001_2

• To add signed numbers use regular addition but disregard carry out

Example 18_{10} - 14_{10} = 18_{10} + (-14_{10}) = 4_{10}

Review: Floating Point Representation

Numbers are represented by:

\[ X = (-1)^S \times 2^{E-127} \times M \]

\( S \) := 1-bit field; Sign bit
\( E \) := 8-bit field; Exponent: Biased integer, \( 0 \leq E \leq 255. \)
\( M \) := 23-bit field; Mantissa: Normalized fraction with hidden 1.

Single precision floating point number

\[ 31 \quad \begin{array}{c} 22 \quad 6 \end{array} \]

Typical Address Space

Review: Floating Point Representation

Numbers are represented by:

\[ X = (-1)^S \times 2^{E-127} \times M \]

\( S \) := 1-bit field; Sign bit
\( E \) := 8-bit field; Exponent: Biased integer, \( 0 \leq E \leq 255. \)
\( M \) := 23-bit field; Mantissa: Normalized fraction with hidden 1.

Single precision floating point number

\[ 31 \quad \begin{array}{c} 22 \quad 6 \end{array} \]

Typical Address Space

A Program’s View of Memory

• What is Memory? a bunch of bits
• Looks like a large linear array
• Find things by indexing into array
  ➢ unsigned integer
  ➢ Most computers support byte (8-bit) addressing
    ➢ Each byte has a unique address (location).
    ➢ Byte of data at address 0x100 and 0x101
    ➢ Word of data at address 0x100 and 0x104
  ➢ 32-bit v.s. 64-bit addresses
    ➢ we will assume 32-bit for rest of course, unless otherwise stated

Memory Partitions

• Text for instructions
  ➢ add res, src1, src2
  ➢ mem[res] = mem[src1] + mem[src2]

• Data
  ➢ static (constants, globals)
  ➢ dynamic (heap, new allocated)
    ➢ grows up

• Stack
  ➢ local variables
    ➢ grows down

• Variables are names for memory locations
  ➢ int x;
A Simple Program's Memory Layout

int result;
main()
{
    int x;
    ...  
    result = x + result;

    result = mem[0x208] + mem[0x400];
}

Pointers

- A pointer is a memory location that contains the address of another memory location.
- "address of" operator &
  - don't confuse with bitwise AND operator (later today)

Given
- int x; int *p;
- p = &x;
Then
- *p = 2; and x = 2; produce the same result

On 32-bit machine, p is 32-bits

Arrays

- In C++ allocate using array form of new
  - int *a = new int[100];
  - double *b = new double[300];
- new [] returns a pointer to a block of memory
  - how big? where?
- Size of chunk can be set at runtime
- delete [] a; // storage returned
- In C
  - malloc(nbytes);
  - free(ptr);

Vector Class vs. Arrays

- Vector Class
  - insulates programmers
  - array bounds checking
  - automagically growing/shrinking when more items are added/deleted
- How are Vectors implemented?
  - real understanding comes when more levels of abstraction are understood
- Programming close to HW
  - (e.g., operating system, device drivers, etc.)
- Arrays can be more efficient
  - but be leery of claims that C-style arrays required for efficiency
- Can talk about memory easier in terms of arrays
  - pointer to a vector?

Address Calculation

- x is a pointer, what is x+33?
- A pointer, but where?
  - what does calculation depend on?
- Result of adding an int to a pointer depends on size of object pointed to
- Result of subtracting two pointers is an int
  - (d + 3) - d = _______

Arrays

- int *a = new int[100];
- double *b = new double[200];
- *(d+33) is the same as d[33]
  - if d is 0x00a0, then a+1 is 0x00a4, a+2 is 0x00a8
    - (decimal 160, 164, 168)
- char * a = new char[44];
- char * begin = a;
- char * end = a + 44;
- while (begin < end)
  - *begin = 'z';
  - begin++;

More Pointer Arithmetic

- address one past the end of an array is ok for pointer comparison only
- what's at *(begin+44)?
- what does begin++ mean?
  - how are pointers compared using < and using == ?
- what is value of end - begin?
More Pointers & Arrays

int * a = new int[100];

- *a is a pointer
- *a is an int
- a[0] is an int (same as *a)
- a[1] is a pointer
- *(a+1) is an int (same as a[1])
- *(a+99) is an int
- *(a+100) is trouble

Array Example

```cpp
#include <iostream.h>

main()
{
    int *a = new int[100];
    int *p = a;
    int k;
    for (k = 0; k < 100; k++)
    {
        *p = k;
        p++;
    }
    cout << "entry 3 = " << a[3] << endl;
}
```

Array of Classes (Linked List)

```cpp
#include <iostream.h>

class node {
public:
    int me;
    node *next;
};

main()
{
    node *ar = new node[10];
    node *p = ar;
    for (k = 0; k < 9; k++)
    {
        p->me = k;
        p->number = 104;
        p->score = k*20;
        p->average = 0.96;
        p->next = &ar[k+1];
        p++;
    }
    p->me = 9;
    p->number = 104;
    p->score = k*20;
    p->average = 0.96;
    p->next = NULL;
    cout << p->me << " " << *(p->me) << " " << p->number << " "
        << *(p->number) << endl;
}
```

Array of Classes with Inheritance

```cpp
#include <iostream.h>

class course {
public:
    int number;
    float average;
};

class node : public course {
public:
    int me;
    node *next;
};

main()
{
    node *ar = new node[10];
    node *p = ar;
    for (k = 0; k < 9; k++)
    {
        p->me = k;
        p->number = 104;
        p->score = k*20;
        p->average = 0.96;
        p->next = &ar[k+1];
        p++;
    }
    p->me = 9;
    p->number = 104;
    p->score = k*20;
    p->average = 0.96;
    p->next = NULL;
    cout << p->me << " " << *(p->me) << " " << p->number << " "
        << *(p->number) << endl;
}
```

Memory Layout

Output

<table>
<thead>
<tr>
<th>Me</th>
<th>p</th>
<th>p-&gt;next</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x26ca8</td>
<td>0x26cb0</td>
<td></td>
</tr>
<tr>
<td>0x26cb0</td>
<td>0x26cb8</td>
<td></td>
</tr>
<tr>
<td>0x26cb8</td>
<td>0x26cc0</td>
<td></td>
</tr>
<tr>
<td>0x26cc0</td>
<td>0x26cc8</td>
<td></td>
</tr>
<tr>
<td>0x26cc8</td>
<td>0x26cd0</td>
<td></td>
</tr>
<tr>
<td>0x26cd0</td>
<td>0x26cd8</td>
<td></td>
</tr>
<tr>
<td>0x26cd8</td>
<td>0x26ce0</td>
<td></td>
</tr>
<tr>
<td>0x26ce0</td>
<td>0x26ce8</td>
<td></td>
</tr>
<tr>
<td>0x26ce8</td>
<td>0x26cf0</td>
<td></td>
</tr>
<tr>
<td>0x26cf0</td>
<td>0x0</td>
<td></td>
</tr>
</tbody>
</table>

Memory Address  Memory Contents  Symbol
0x26ca8  0x00000000  me[0]
0x26cb0  0x00000000  me[1]
0x26cb8  0x00000000  me[2]
0x26cc0  0x00000000  me[3]
0x26cc8  0x00000000  me[4]
0x26cd0  0x00000000  me[5]
0x26cd8  0x00000000  me[6]
0x26ce0  0x00000000  me[7]
0x26ce8  0x00000000  me[8]
0x26cf0  0x00000000  me[9]
Strings as Arrays

- A string is an array of characters with '0' at the end
- Each element is one byte, ASCII code
- '0' is null (ASCII code 0)

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>12</td>
<td>16</td>
</tr>
</tbody>
</table>
```

Strlen()

- strlen() returns the # of characters in a string
  - same as # elements in char array?

```c
int strlen(char * s)
// pre: '\0' terminated
// post: returns # chars
{
    int count=0;
    while (*s++)
        count++;
    return count;
}
```

Outline

- Memory
- Bit Manipulations

Bit Manipulations

Problem

- 32-bit word contains many values
  - e.g., input device, sensors, etc.
  - current x,y position of mouse and which button (left, mid, right)
- Assume x, y position is 0-255
- How many bits for position?
- How many for button?

Goal

- Extract position and button from 32-bit word
- Need operations on individual bits of binary number

Bitwise AND / OR

- & operator performs bitwise AND
- | operator performs bitwise OR
- Per bit
  - 0 & 0 = 0
  - 0 & 1 = 0
  - 1 & 0 = 0
  - 1 & 1 = 1
- For multiple bits, apply operation to individual bits in same position

```
<table>
<thead>
<tr>
<th>XOR</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>011010</td>
<td>011010</td>
</tr>
<tr>
<td>101110</td>
<td>101110</td>
</tr>
<tr>
<td>001010</td>
<td>111110</td>
</tr>
</tbody>
</table>
```

Mouse Example

- 32-bit word with x,y and button fields
  - bits 0-7 contain x position
  - bits 8-15 contain y position
  - bits 16-17 contain button (0 = left, 1 = middle, 2 = right)
- To extract value need to clear all other bits
- How do I use bitwise operations to do this?

```
button  y  x
0x1a34c = 01 1010 0011 0100 1100
```
Mouse Solution

- **AND with a bit mask**
  - Specific values that clear some bits, but pass others through
- **To extract x position use mask 0x000ff**
  \[
  \text{xpos} = 0x1a34c \& 0x000ff
  \]
- **Button**
  \[
  0x1a34c = 01 \ 1010 \ 0011 \ 0100 \ 1100 \\
  0x000ff = 00 \ 0000 \ 0000 \ 1111 \ 1111 \\
  0x0004c = 00 \ 0000 \ 0000 \ 0100 \ 1100
  \]

More of the Mouse Solution

- **To extract y position use mask 0x0ff00**
  \[
  \text{ypos} = 0x1a34c \& 0x0ff00
  \]
- **Similarly, button is extracted with mask 0x30000**
  \[
  \text{button} = 0x1a34c \& 0x30000
  \]
- **Not quite done...why?**

The SHIFT operator

- **>> is shift right, << is shift left**, operands are int and number of positions to shift
- \[(1 << 3) \rightarrow \ldots 000001 \rightarrow \ldots 000100 \ldots \text{(it's 2^3)}\]
- **0xff00 is 0xff << 8**, and **0xff is 0xff00 >> 8**
- **So, true ypos value is**
  \[
  \text{ypos} = (0x1a34c \& 0x0ff00) >> 8 \\
  \text{button} = (0x1a34c \& 0x30000) >> 16
  \]

Extracting Parts of Floating Point Number

- **See web page for full code**
  \[
  \text{x is 32-bit word}
  \]

  ```
  \#define EXP_BITS 8 \\
  \#define MANTISSA_BITS 23 \\
  \#define SIGN_MASK 0x80000000 \\
  \#define EXP_MASK 0x7f800000 \\
  \#define MANTISSA_MASK 0x007fffff \\
  
  class myfloat {
    public:
      int sign; \\
      unsigned int exp; \\
      unsigned int mantissa;
  }; \\
  
  num->sign = (x & SIGN_MASK) >> (EXP_BITS + MANTISSA_BITS); \\
  num->exp = (x & EXP_MASK) >> MANTISSA_BITS; \\
  num->mantissa = x & MANTISSA_MASK;
  ```

Summary

- **Homework #1 Jan 21**
- **Computer memory is linear array of bytes**
- **Pointer is memory location that contains address of another memory location**
- **Bitwise operations**
- **Code examples are linked to course web page**
- **We'll visit these topics again throughout semester**
- **Next Time**
  - Instruction set architecture (ISA)
- **Reading**
  - Chapter 3