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:

\[14_{10} = 001110_2\]
\[-14_{10} = 110001_2\]
\[+1\]
\[110010_2\]

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

Example:

\[18_{10} - 14_{10} = 18_{10} + (-14_{10}) = 4_{10}\]
\[010010_2\]
\[+110010_2\]
\[000100_2\]

Review: Floating Point Representation

Numbers are represented by:

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

\[S := 1\text{-bit field; Sign bit}\]
\[E := 8\text{-bit field; Exponent: Biased integer, } 0 \leq E \leq 255.\]
\[M := 23\text{-bit field; Mantissa: Normalized fraction with hidden 1.}\]

Single precision floating point number

```
31 30 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
\hline\hline
s & e & M
```

\[s\text{ exp Mantissa}\]
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;
    ...
}

mem[0x208] = mem[0x400] + mem[0x208]

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
Vector Class vs. Arrays

• Vector Class
  ➢ insulates programmers
  ➢ array bounds checking
  ➢ \textit{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?

Arrays

• In C++ allocate using array form of \texttt{new}
  \begin{verbatim}
  int *a = new int[100];
  double *b = new double[300];
  \end{verbatim}

• \texttt{new []} returns a pointer to a block of memory
  ➢ how big? where?

• Size of chunk can be set at runtime
• \texttt{delete []} \texttt{a}; // storage returned

• In C
  \begin{verbatim}
  malloc(nbytes);
  free(ptr);
  \end{verbatim}
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

  \((d + 3) - d = \) _______

int * a = new int[100]

\[
\begin{array}{cccccccc}
0 & 1 & 32 & 33 & 98 & 99 \\
\end{array}
\]

a[33] is the same as \(*\(a+33\)\)
if a is 0x00a0, then a+1 is 0x00a4, a+2 is 0x00a8
(decimal 160, 164, 168)

double * d = new double[200];

\[
\begin{array}{cccccccc}
0 & 2 & 32 & 33 & 199 \\
\end{array}
\]

\(*\(d+33\)\) is the same as \(d[33]\)
if d is 0x00b0, then d+1 is 0x00b8, d+2 is 0x00c0
(decimal 176, 184, 192)

More Pointer Arithmetic

• address one past the end of an array is ok for pointer comparison only

• what’s at \(*\(\text{begin}+44\)\)?

• what does \text{begin++} mean?

• how are pointers compared using < and using == ?

• what is value of \text{end} - \text{begin}?

\[
\begin{array}{cccccccc}
0 & 1 & 15 & 16 & 42 & 43 \\
\end{array}
\]

char * a = new char[44];

char * begin = a;

char * end = a + 44;

while (begin < end)
{
  \*begin = ‘z’;
  begin++;
}

© Alvin R. Lebeck

CPS 104
More Pointers & Arrays

```cpp
int * a = new int[100];

// 0 1 32 33 98 99

a is a pointer
*a is an int
a[0] is an int (same as *a)
a[1] is an int
a+1 is a pointer
a+32 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;
    int k;
    for (k = 0; k < 9; k++)
    {
        p->me = k;
        p->next = &ar[k+1];
        p++;
    }
    p->me = 9;
    p->next = NULL;
    p = &ar[0];
    while (p != NULL) {
        cout << p->me << " " <<
        hex << p << " " << p->next << endl;
        p = p->next;
    }
}
```

- Given ar = 0x10000, what does memory layout look like?

Memory Layout

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

- me is int (4 bytes)
- next is node* (4 bytes)
Array of Classes with Inheritance

```cpp
#include <iostream.h>
class course {
public:
    int number;
    int score;
    float average;
};
class node : public course {
public:
    int me;
    node *next;
};
main()
{
    node *ar = new node[10];
    node *p = ar;
    int *num_ptr, *me_ptr;
    int k;
    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;
    num_ptr = &p->number;
    me_ptr = &p->me;
    cout << p->me << " " << *me_ptr 
<< " " << p->number << " " << *num_ptr << endl;
}
```

Memory Layout

```
0x2ad40        104  number
    score
    average
    me
    next

0x2ad54  ar[0]

```

```
0x2adf4  104  number
    score
    average
    me
    next

```

```
0x2af4  num_ptr
  me_ptr

```

...
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)

```
0 1 15 16 42 43
```

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 | 0 = 0
  0 & 1 = 0 0 | 1 = 1
  1 & 0 = 0 1 | 0 = 1
  1 & 1 = 1 1 | 1 = 1
• For multiple bits, apply operation to individual bits in same position

<table>
<thead>
<tr>
<th>AND</th>
<th>OR</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`
  
  \[
  \text{xpos} = 0\times1a34c \ & \ 0\times000ff
  \]

<table>
<thead>
<tr>
<th>button</th>
<th>y</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0\times1a34c = 01 1010 0011 0100 1100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0\times000ff = 00 0000 0000 1111 1111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0\times0004c = 00 0000 0000 0100 1100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

More of the Mouse Solution

- To extract y position use mask `0x0ff00`
  
  \[
  \text{ypos} = 0\times1a34c \ & \ 0\times0ff00
  \]

- Similarly, button is extracted with mask `0x30000`
  
  \[
  \text{button} = 0\times1a34c \ & \ 0\times30000
  \]

- Not quite done...why?

<table>
<thead>
<tr>
<th>button</th>
<th>y</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0\times1a34c = 01 1010 0011 0100 1100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0\times000ff = 00 1111 1111 0000 0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0\times0a300 = 00 1010 0011 0000 0000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The SHIFT operator

- >> is shift right, << is shift left, operands are int and number of positions to shift
- (1 << 3) is ...00001 -> ...0001000 (it’s 2³)
- 0xff00 is 0xff << 8, and 0xff is 0xff00 >> 8
- So, true ypos value is
  ypos = (0x1a34c & 0x0ff00) >> 8
  button = (0x1a34c & 0x30000) >> 16

Extracting Parts of Floating Point Number

- See web page for full code

```cpp
#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 lineary 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