CPS 104
Computer Organization and Programming

Lecture-3 : Memory, Bit Operations.

Sep. 3, 1999

Dietolf (Dee) Ramm

http://www.cs.duke.edu/~dr/cps104.html
Administrivia

- Homework #1 on Web
- 2 parts
  - simple submit
  - paper and pencil
  - programs (use submit)
- Part 1: Due in class Friday, 9/10
- Part 2: Due Friday, 9/10, 11:59pm
Computer Memory

- What is Computer Memory?
- What does it “look like” to the program?
- How do we find things in computer memory?
A Program’s View of Memory

- **What is Memory?** a bunch of bits
- **Looks like** a large linear array or vector
- **Find things by** indexing into vector
  - 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
Buzz Word Definition: Endianess

Byte Order

- **Big Endian**: byte 0 is 8 most significant bits IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA
- **Little Endian**: byte 0 is 8 least significant bits Intel 80x86, DEC Vax, DEC Alpha
**Buzz Word Definition: Alignment**

- **Alignment**: require that objects fall on address that is multiple of their size.
- 32-bit integer
  - Aligned if address % 4 = 0
- 64-bit integer?
  - Aligned if ?

![Alignment Diagram]

<table>
<thead>
<tr>
<th>Byte #</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aligned</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Not Aligned</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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;

©GK&DR Fall 1999
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

```c
int x; int *p;
p = &x;
```

Then

```c
*p = 2;  and x = 2; produce the same result
```

On 32-bit machine, p is 32-bits

```
x 0x26cf0
... 
p 0x26d00 0x26cbf0
```
Vector Class v.s. 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?
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);
  ```
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

\[(d + 3) - d == \_\_\_\_\_\_\_\_\_\]

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

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];

\[
\begin{array}{ccccccc}
0 & 1 & \_\_\_\_\_\_\_\_ & 33 & \_\_\_\_\_\_\_\_ & 199 \\
\end{array}
\]

*(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 *(begin+44)?

- what does begin++ mean?

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

- what is value of end - begin?

```c
char * a = new char[44];
char * begin = a;
char * end = a + 44;

while (begin < end)
{
    *begin = 'z';
    begin++;
}
```
More Pointers & Arrays

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

- `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
#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;
};
int 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?
}
```

CPS104 Lec3.24
Memory Layout

<table>
<thead>
<tr>
<th>Source Symbol</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></td>
<td>0x26cb0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0x26cb8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0x26cc0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0x26cc8</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>0x26cd0</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>0x26cd8</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>0x26ce0</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>0x26ce8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>0x26cf0</td>
<td>9</td>
</tr>
</tbody>
</table>

Output

<table>
<thead>
<tr>
<th>me</th>
<th>p</th>
<th>p-&gt;next</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x26ca8</td>
<td>0x26cb0</td>
</tr>
<tr>
<td>1</td>
<td>0x26cb0</td>
<td>0x26cb8</td>
</tr>
<tr>
<td>2</td>
<td>0x26cb8</td>
<td>0x26cc0</td>
</tr>
<tr>
<td>3</td>
<td>0x26cc0</td>
<td>0x26cc8</td>
</tr>
<tr>
<td>4</td>
<td>0x26cc8</td>
<td>0x26cd0</td>
</tr>
<tr>
<td>5</td>
<td>0x26cd0</td>
<td>0x26cd8</td>
</tr>
<tr>
<td>6</td>
<td>0x26cd8</td>
<td>0x26ce0</td>
</tr>
<tr>
<td>7</td>
<td>0x26ce0</td>
<td>0x26ce8</td>
</tr>
<tr>
<td>8</td>
<td>0x26ce8</td>
<td>0x26cf0</td>
</tr>
<tr>
<td>9</td>
<td>0x26cf0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

me is int (4 bytes)  
next is node* (4 bytes)
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)
**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;
}
```
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 numbers
Bitwise AND / OR

- & operator performs bitwise AND
- | operator performs bitwise OR
- Per bit
  
  \[
  \begin{array}{c c c}
  0 & 0 = 0 & 0 | 0 = 0 \\
  0 & 1 = 0 & 0 | 1 = 1 \\
  1 & 0 = 0 & 1 | 0 = 1 \\
  1 & 1 = 1 & 1 | 1 = 1 \\
  \end{array}
  \]

- For multiple bits, apply operation to individual bits in same position

\[
\begin{array}{c c}
\text{AND} & \text{OR} \\
011010 & 011010 \\
101110 & 101110 \\
001010 & 111110 \\
\end{array}
\]
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?

\[
\begin{array}{ccc}
\text{button} & \text{y} & \text{x} \\
0x1a34c = & 01 & 1010 & 0011 & 0100 & 1100
\end{array}
\]
Mouse Solution

- AND with a bit mask
  - specific values that clear some bits, but pass others through
- To extract x position use mask 0x000ff

\[ \text{xpos} = 0x1a34c \& 0x000ff \]

<table>
<thead>
<tr>
<th>button</th>
<th>y</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1a34c</td>
<td>01 1010 0011 0100 1100</td>
<td></td>
</tr>
<tr>
<td>0x000ff</td>
<td>00 0000 0000 1111 1111</td>
<td></td>
</tr>
<tr>
<td>0x0004c</td>
<td>00 0000 0000 0100 1100</td>
<td></td>
</tr>
</tbody>
</table>
More of the Mouse Solution

- To extract y position use mask 0x0ff00
  \[ y_{pos} = 0x1a34c \& 0x0ff00 \]

- Similarly, button is extracted with mask 0x3ffff
  \[ button = 0x1a34c \& 0x30000 \]

- Not quite done...why?

<table>
<thead>
<tr>
<th>button</th>
<th>y</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1a34c</td>
<td>01 1010 0011 0100 1100</td>
<td></td>
</tr>
<tr>
<td>0x000ff</td>
<td>00 1111 1111 0000 0000</td>
<td></td>
</tr>
<tr>
<td>0x0a300</td>
<td>00 1010 0011 0000 0000</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 \ldots 000001 \rightarrow \ldots 0001000 (it’s $2^3$)
- `0xff00` is `0xff << 8`, and `0xff` is `0xff00 >> 8`
- So, true ypos value is:

```
ypos = (0x1a34c & 0xff00) >> 8
```

```
button = (0x1a34c & 0x30000) >> 16
```
Extracting Parts of Floating Point Number

- See web page for full code

```c
#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;
```

x is 32-bit word
Summary

- Homework #1 due Fri 9/10
- Computer memory is a 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