Under the Hood: Data Representations, Memory and Bit Operations

Computer Science 104

Lecture 3

Administrivia

- **Homework #1 Due Feb 1**

Outline

- **Review/finish data representations**
- **Arrays**
- **Pointers**
- **Pointer Arithmetic**
- **Bitwise operations (AND, OR)**

Reading

Chapter 2 (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

$14_{10} = 001110_2$

$-14_{10} = 110001_2$

$+ \quad 1$

110010_2

- To add signed numbers use regular addition but disregard carry out
  - Example $18_{10} - 14_{10} = 18_{10} + (-14_{10}) = 4_{10}$

$$\begin{array}{r} 010010_2 \\ +110010_2 \\ \hline 000100_2 \end{array}$$

Review: Floating Point Representation

Numbers are represented by:

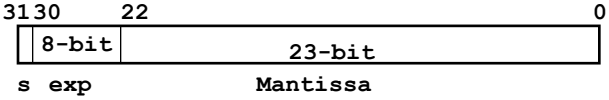
$$X = (-1)^s \times 2^{E-127} \times 1.M$$

S := 1-bit field; Sign bit

E := 8-bit field; Exponent: Biased integer, $0 \le E \le 255$.

M:= 23-bit field; Mantissa: Normalized fraction with hidden 1.

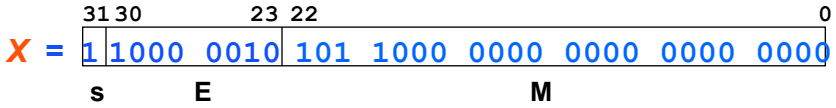
Single precision floating point number



Review

What floating-point number is 0xC1580000?

1100 0001 0101 1000 0000 0000 0000 0000



- E = 128+2-127 = 3 M = 1011

$-1.1011 \times 2^3 = -1101.1 = -13.5$

Review: ASCII Character Representation

Oct.	Chr.	Oct.	Chr.	Oct.	Chr.	Oct.	Chr.	Oct.	Chr.	Oct.	Chr.	Oct.	Chr.	Oct.	Chr.
000	nul	001	soh	002	stx	003	etx	004	eot	005	enq	006	ack	007	bel
010	bs	011	ht	012	nl	013	vt	014	np	015	cr	016	so	017	si
020	dle	021	dc1	022	dc2	023	dc3	024	dc4	025	nak	026	syn	027	etb
030	can	031	em	032	sub	033	esc	034	fs	035	gs	036	rs	037	us
040	sp	041	!	042	"	043	#	044	$	045	%	046	&	047	'
050	(	051	)	052	*	053	+	054	,	055	-	056	.	057	/
060	0	061	1	062	2	063	3	064	4	065	5	066	6	067	7
070	8	071	9	072	:	073	;	074	<	075	=	076	>	077	?
100	@	101	A	102	B	103	C	104	D	105	E	106	F	107	G
110	H	111	I	112	J	113	K	114	L	115	M	116	N	117	O
120	P	121	Q	122	R	123	S	124	T	125	U	126	V	127	W
130	X	131	Y	132	Z	133	[	134	\	135	]	136	^	137	_
140	`	141	a	142	b	143	c	144	d	145	e	146	f	147	g
150	h	151	i	152	j	153	k	154	l	155	m	156	n	157	o
160	p	161	q	162	r	163	s	164	t	165	u	166	v	167	w
170	x	171	y	172	z	173	{	174	|	175	}	176	~	177	del

- Each character is represented by a 7-bit ASCII code.
- It is packed into 8-bits

Summary of Data Representations

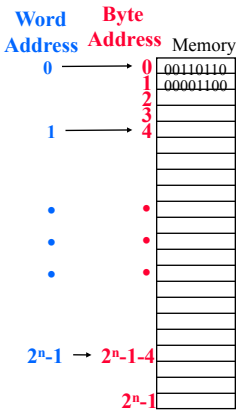
- Computers operate on binary numbers (0s and 1s)
- Conversion to/from binary, oct, hex
- Signed binary numbers
  - 2's complement
  - arithmetic, negation
- Floating point representation
  - hidden 1
  - biased exponent
  - single precision, double precision
- ASCII code for characters

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

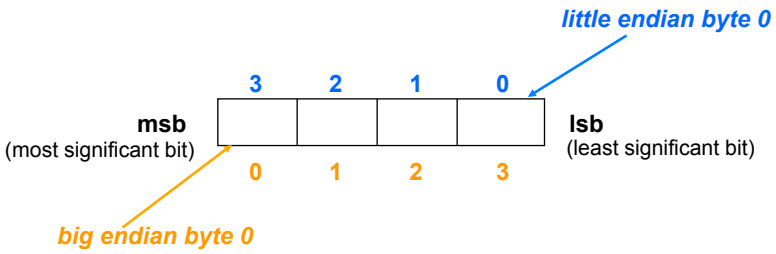
- **What is Memory?** a bunch of bits
- **Looks like** a large linear array
- **Find things by** indexing into array
  - unsigned integer
- Most computers support byte (8-bit) addressing
  - Each byte has a unique address (location).
  - Byte of data at address 0x100 and 0x101
  - Word of data at address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



Buzz Word Definition: Endianess

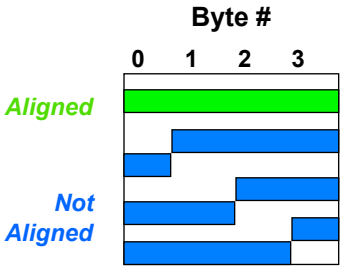
Byte Order

- **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



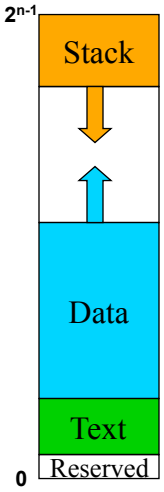
Buzz Word Definition: Alignment

- **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 ?



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;

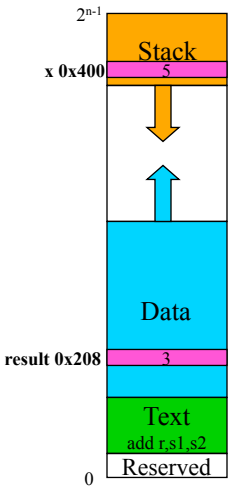


Typical Address Space

A Simple Program's Memory Layout

```
...
int result; // global var
main()
{
  int x;
  ...
  result = x + result;
  ...
}
```

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



Reference (handle) vs. Pointer

Java

- "The value of a reference type variable, in contrast to that of a primitive type, is a reference to (an address of) the value or set of values represented by the variable" http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html
- Cannot manipulate the value of the reference

C or C++

- A pointer is a memory location that contains the address of another memory location
- Can manipulate the value of pointer (double edge sword)

Pointers

- "address of" operator &
  - don't confuse with bitwise AND operator (later today)

Given

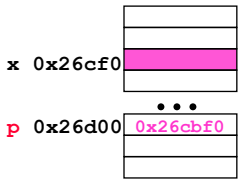
```
int x; int *p;
p = &x;
```

Then

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

- What happens for `p = 2?;`

On 32-bit machine, p is 32-bits



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 (the ridiculous…)
- 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 block can be set at runtime
- `delete [] a;` // storage returned
- In C

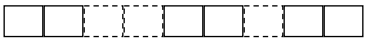
```
malloc(nbytes);
free(ptr);
```


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 = _____
```

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



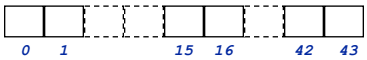

```
a[33] is the same as *(a+33)
if a is 0x00a0, then a+1 is
0x00a4, a+2 is 0x00a8
(decimal 160, 164, 168)
```

```
double * d = new double[200];
```


```
*(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`?

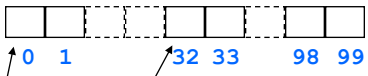



```
char * a = new char[44];
char * begin = a;
char * end = a + 44;

while (begin < end)
{
   *begin = 'z';
   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 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

```
#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)

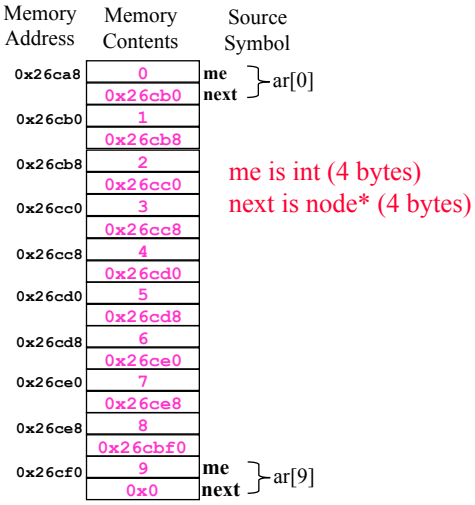
```
#include <iostream.h>
class node {
public:
  int me;
  node *next;
};
main()
{
  node *ar = new node[10];
  node *p = ar;
  int k;
  for (k = 0; k < 9; k++)
    {
      p->me = k;
      p->next = &ar[k+1];
      p++;
    }
  p->me = 9;
  p->next = NULL;
  p = &ar[0];
  while (p != NULL) {
    cout << p->me << " " <<
hex << p << " " << p->next <<
endl;
    p = p->next;
  }
}
```

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

Memory Layout

Output

```
Me   p        p->next
0 0x26ca8 0x26cb0
1 0x26cb0 0x26cb8
2 0x26cb8 0x26cc0
3 0x26cc0 0x26cc8
4 0x26cc8 0x26cd0
5 0x26cd0 0x26cd8
6 0x26cd8 0x26ce0
7 0x26ce0 0x26ce8
8 0x26ce8 0x26cf0
9 0x26cf0 0x0
```

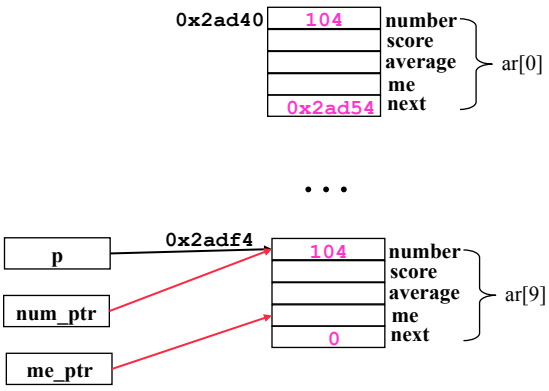


Array of Classes with Inheritence

```
#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;
```

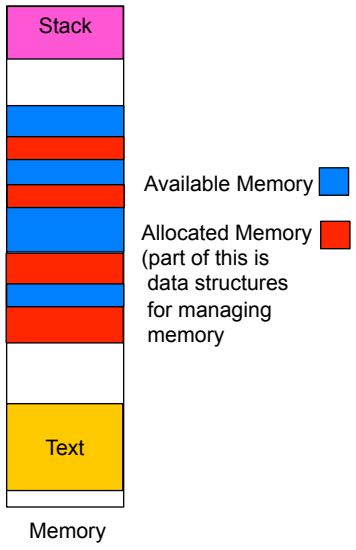
```
for (k = 0; k < 9; k++)
    {
      p->me = k;
      p->number = 104;
      p->score = k*20;
      p->average = 0.96;
      p->next = &ar[k+1];
      p++;
    }
  p->me = 9;
  p->number = 104;
  p->score = k*20;
  p->average = 0.96;
  p->next = NULL;
  num_ptr = &p->number;
  me_ptr = &p->me;
  cout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

Memory Layout

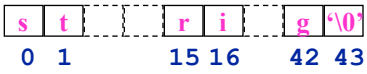


Memory Manager (Heap Manager)

- malloc / new (C/C++)
- Library routines that handle memory management for data segment (allocation / deallocation)
- Java has garbage collection (reclaim memory of unreferenced objects)
- C/C++ must use free/ delete, else memory leak



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?

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

Outline

- Memory
  - Important concept: bits in memory can represent anything
  - Data (char, int, float, int*, char*, etc.)
  - Instructions (the commands of the machine)
- 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 word

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

AND	OR
011010	011010
101110	101110
001010	111110

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

```
xpos = 0x1a34c & 0x000ff
```

```
                button      y           x
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

```
ypos = 0x1a34c & 0x0ff00
```

- Similarly, button is extracted with mask 0x30000

```
button = 0x1a34c & 0x30000
```

- Not quite done…why?

```
                button      y           x
0x1a34c = 01 1010 0011 0100 1100
0x000ff = 00 1111 1111 0000 0000
0x0a300 = 00 1010 0011 0000 0000
```

The SHIFT operator

- >> is shift right, << is shift left, operands are int and number of positions to shift
- (1 << 3) is …000001 -> …0001000 (it's 2^3)
- 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

x is 32-bit word

```
#define EXP_BITS 8
#define FRACTION_BITS 23
#define SIGN_MASK 0x80000000
#define EXP_MASK  0x7f800000
#define FRACTION_MASK  0x007fffff
class myfloat {
public:
  int sign;
  unsigned int exp;
  unsigned int fraction;
};
float x;

  num->sign = (x & SIGN_MASK) >> (EXP_BITS + FRACTION_BITS);
  num->exp  = (x & EXP_MASK) >> FRACTION_BITS;
  num->fraction = x & FRACTION_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 2**