CSCI-UA.0201

Computer Systems Organization

C Programming – Preprocessor
Data Representation – Bits and Bytes

Thomas Wies

wies@cs.nyu.edu

https://cs.nyu.edu/wies

Macros

Macros can be a useful way to customize your interface to C and make your code easier to read and less redundant. However, when possible, use a static inline function instead.

Format is very simple:

#define *identifier replacement-text*

Example:

```
#define NUM   10
```

Notes:

- Each occurrence of NUM in your code will be replaced by 10.
- This happens by the preprocessor before compilation.
- In the rest of the code you cannot change NUM.

We can take this idea further. Instead of defining a constant, we define operations.

Macros

- Sophisticated Example

```
#define CIRCLE_AREA(x) (PI * (x) * (x))
area = CIRCLE_AREA(4);
```

    becomes

```
area = (3.14159 * (4) * (4));
```

  - See how parentheses are used. Always enclose parameters in ( ).

- More sophisticated example:

```
#define RECTANGLE_AREA(x, y)  ((x) * (y))
rectArea = RECTANGLE_AREA(a + 4, b + 7);
```

  becomes

```
rectArea = ((a + 4) * (b + 7));
```

Macros: More examples

- `#define forever  for(;;)`

- `#define max(i,j)  ((i) > (j) ? (i) : (j))`

- `#define SWAP(a, b)  {` ...

```
#define SWAP(a, b)  {              \
                      a ^= b;      \
                      b ^= a;      \
                      a ^= b;      \
                    }
```

Other Preprocessor Directives

- ```
  #include <file>
  #include "file"
  ```
  - textually include `file` in current file

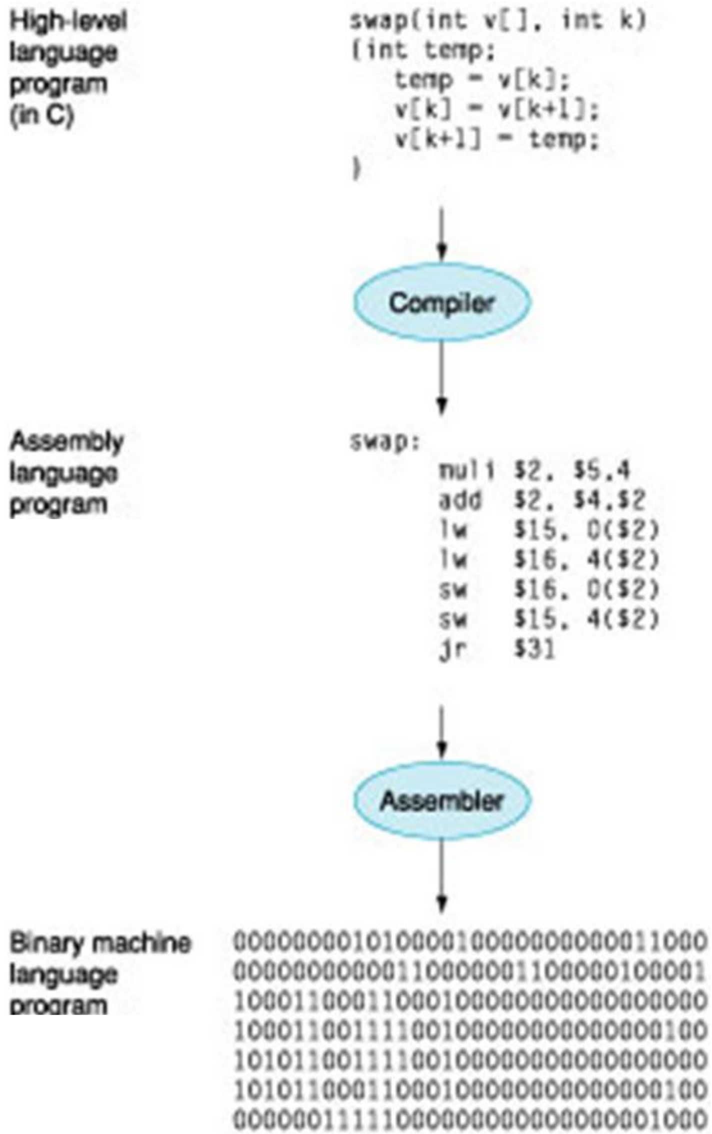
- ```
  #ifdef MACRO
      … // code
  #endif
  ```
  - include code if MACRO is defined

- ```
  #ifndef MACRO
      … // code
  #endif
  ```
  - include code if MACRO is undefined
  ```

Data Representation

Bits and Bytes

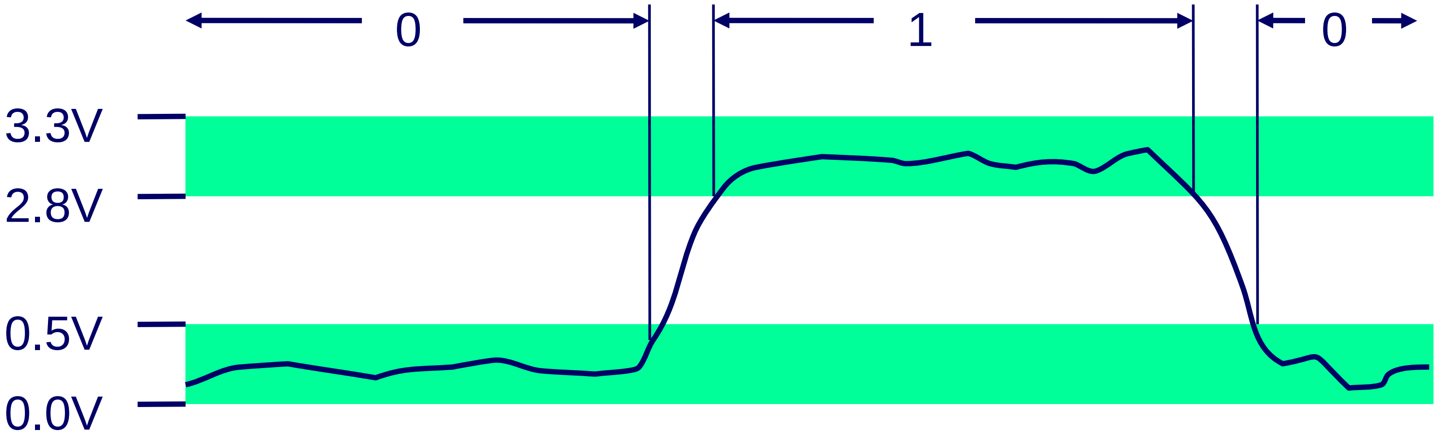
- Representing information as bits
- How are bits manipulated?
- Types of data:
  - Integers
  - Floating points
  - others


High-level language program (in C)
swap(int v[], int k)
{int temp;
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
}
Compiler
Assembly language program
swap:
muli $2, $5,4
add $2, $4,$2
lw $15, 0($2)
lw $16, 4($2)
sw $16, 0($2)
sw $15, 4($2)
jr $31
Assembler
Binary machine language program
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000

Our First Steps…
How do we represent data in a computer?

- How do we represent data using electrical signals?
- At the lowest level, a computer is an electronic machine.
- Easy to recognize two conditions:
  - presence of a voltage – we call this state "1"
  - absence of a voltage – we call this state "0"

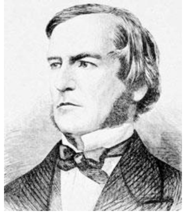
Binary Representations



A Computer is a Binary Digital Machine

- Basic unit of information is the *binary digit*, or *bit*.
- Values with more than two states require multiple bits.
  - A collection of two bits has four possible states: 00, 01, 10, 11
  - A collection of three bits has eight possible states: 000, 001, 010, 011, 100, 101, 110, 111
  - *A collection of n bits has 2^n possible states.*

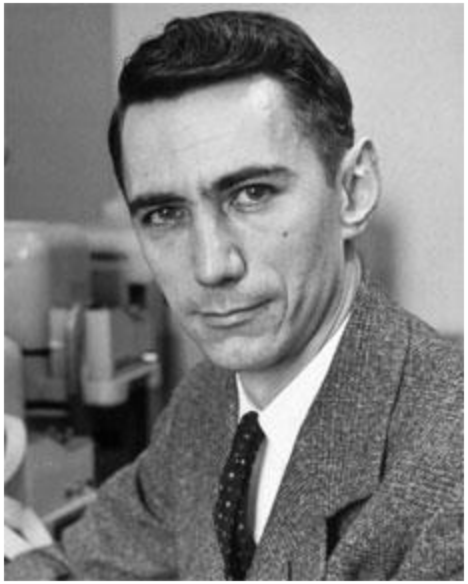
George Boole



- (1815-1864)
- English mathematician and philosopher
- Inventor of Boolean Algebra
- Now we can use things like: AND, OR, NOT, XOR, XNOR, NAND, NOR, ….

Source: http://history-computer.com/ModernComputer/thinkers/Boole.html

Claude Shannon



- (1916–2001)
- American mathematician and electronic engineer
- His work is the foundation for using switches (mainly transistors now), and hence binary numbers, to implement Boolean function.

Source: http://history-computer.com/ModernComputer/thinkers/Shannon.html

So, we use transistors to implement logic gates.

Logic gates manipulate binary numbers to implement Boolean functions.

Boolean functions solve problems.

It's almost that simple... ☺

Encoding Byte Values

- Byte = 8 bits
  - Binary 00000000_2 to 11111111_2
  - Decimal: 0_{10} to 255_{10}
  - Hexadecimal 00_{16} to FF_{16}
    - Base 16 number representation
    - **Every 4 bits → 1 hexadecimal digit**
    - Use characters '0' to '9' and 'A' to 'F'
    - Write $FA1D37B_{16}$ in C language as
      - 0xFA1D37B
      - 0xfa1d37b

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Data Representations

C Data Type	Typical 32-bit	Intel IA32	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	4	8
long long	8	8	8
float	4	4	4
double	8	8	8
pointer	4	4	8

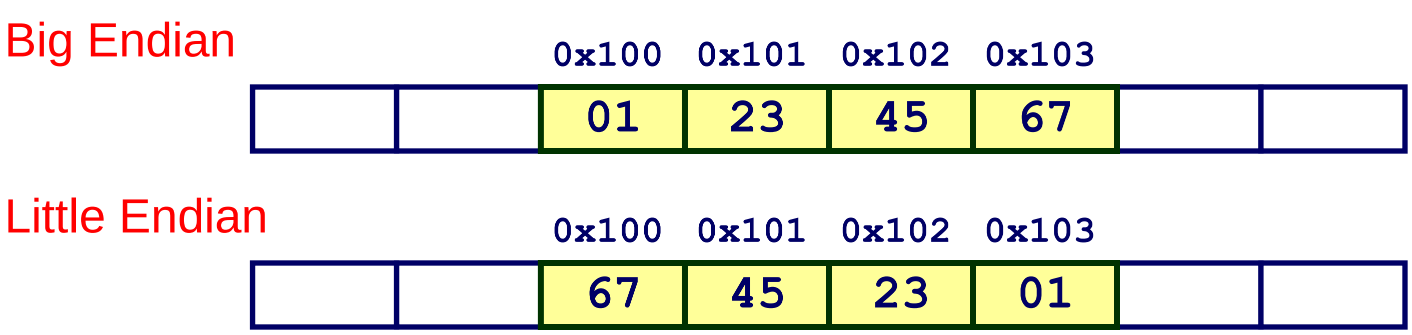
Byte Ordering

- How are bytes within a multi-byte word ordered in memory?
- Conventions
  - Big Endian: Sun, PPC, Internet
    - Most significant byte has lowest address
  - Little Endian: x86
    - Most significant byte has highest address

Byte Ordering Example

- Big Endian
  - Most significant byte has lowest address
- Little Endian
  - Most significant byte has highest address
- Example
  - Variable x has 4-byte representation 0x01234567
  - Address given by &x is 0x100

Most Significant Byte



Examining Data Representations

- Code to print Byte Representation of data

```
void show_bytes(unsigned char * start, int len){
  int i;
  for (i = 0; i < len; i++)
    printf(”%p\t%2x\n",start+i, start[i]);
  printf("\n");
}
```

`printf` directives:
%p: Print pointer
%x: Print integer in hexadecimal

`show_bytes` Execution Example

```
int a = 0x12345678;
printf("int a = 0x12345678;\n");
show_bytes((unsigned char *) &a, sizeof(int));
```

Result (Linux):

```
int a = 0x12345678;
0x11ffffcb8  0x78
0x11ffffcb9  0x56
0x11ffffcba  0x34
0x11ffffcbb  0x12
```

Reading Byte-Reversed Listings

- Disassembly
  - given the binary file, get the assembly
- Example Fragment

Address	Instruction Code	Assembly Rendition	
8048365:	5b	pop	%ebx
8048366:	81 c3 ab 12 00 00	add	$0x12ab,%ebx
804836c:	83 bb 28 00 00 00 00	cmpl	$0x0,0x28(%ebx)

- Deciphering Numbers
  - Value: `0x12ab`
  - Pad to 32 bits (int is 4 bytes): `0x000012ab`
  - Split into bytes: `00 00 12 ab`
  - Reverse (little endian): `ab 12 00 00`