Week 2 Lecture

# B&O Chapter 2, Part 1 – Information Storage

## Memory

1. Most computers use block of 8 **bits** (1 **byte**) as the smallest addressable unit of memory
2. Viewed by machine-level programs as huge array of bytes (i.e. **virtual memory**)
3. Each byte of memory identified by a unique number (i.e. **address**)
4. Set of all possible addresses known as **virtual address space**, which is
   1. Virtual address space just a conceptual image presented to the machine-level program
   2. Actual implementation discussed later and in CIS 3207

## Hexadecimal Notation

1. Reason it is used
   1. Binary notation = too verbose
   2. Decimal notation = too tedious to convert from binary to decimal and back
   3. Hexadecimal notation (or simply “hex”) = much less verbose and straightforward to convert to and from binary
      1. 24 = 16 🡪 1 hex digit used to represent 4 bits (so 2 hex digits used to represent 1 byte vs. 8 binary digits)
         1. Range of values for one byte = 0x00 to 0Xff

| Hex Digit | Decimal Value | Binary Value |
| --- | --- | --- |
| 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 |

## Base Conversions

1. **Converting from hex to binary**: Simply convert each hex digit.
   1. ***Example***: Convert **0x39A7F8** to binary.
      1. ***Answer***: 0011 1001 1010 0111 1111 1000

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Hex: | 3 | 9 | A | 7 | F | 8 |
| Binary: | 0011 | 1001 | 1010 | 0111 | 1111 | 1000 |

1. **Converting from binary to hex**: Split into groups of 4 bits (4 bits since 24 = 16, and note that you should start with 4 bits on right end and work right to left), and then translate to corresponding hex value.
   1. ***Example***: Convert **10 0110 1110 0111 1011 0101** to hex.
      1. ***Answer***: 0x26E7B5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Binary: | 10 | 0110 | 1110 | 0111 | 1011 | 0101 |
| Hex: | 2 | 6 | E | 7 | B | 5 |

1. **Converting from decimal to hex**: Repeatedly divide by 16 giving a quotient and a remainder such that , use the hex representation of as the least significant digit, and then generate the remaining digits (right to left) by repeating the process on until .
   1. Same method can be used to **convert decimal to binary** but instead .
      1. Alternatively, you could use repeated subtraction to do this conversion (both for conversion from decimal to binary or to hex).
   2. Example: Convert 188 to both binary and hex.
      1. Answer:
         1. **Binary**: 1011 1100
            1. **Method 1**: Repeated division

|  |
| --- |
|  |

* + - * 1. **Method 2**: Repeated Subtraction

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Power of 2: |  |  |  |  |  |  |  |  |
| Binary: | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

* + - 1. **Hex**: 0xBC

|  |
| --- |
|  |

1. **Converting from hex to decimal**: Multiply each hex digit by the appropriate power of 16.
   1. Same method can be used to **convert from binary to decimal** but instead multiply each binary digit by the appropriate power of 2.
   2. Example: Convert 0x7AF to decimal.
      1. Answer: 1967

|  |  |  |  |
| --- | --- | --- | --- |
| Hex: | 7 | A | F |
| Power of 16: |  |  |  |
| Product: |  |  |  |
| Sum: |  | | |

* 1. Example: Convert binary 110111 to decimal.
     1. Answer: 55

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Binary: | 1 | 1 | 0 | 1 | 1 | 1 |
| Power of 16: |  |  |  |  |  |  |
| Product: |  |  |  |  |  |  |
| Sum: |  | | | | | |

1. Conversion between octal and binary (same as working with hex except working with three bits at a time instead of four since )

## Addition in bases other than 10 (binary, octal, hexadecimal, etc.)

1. Recall addition in decimal
   1. Example:
2. Addition in other bases follows the same process, except instead of a single digit consisting of 0-9, the range of what is a single digit is dependent on the base.
   1. **Binary digits**: 0,1
      1. Example: 10110 + 00111 = 11101
   2. **Hex digits**: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
      1. Example: 0x503C + 0x8 = 0x5044
   3. **Octal digits**: 0,1,2,3,4,5,6,7,8

## Data Sizes

1. **Word** **size**
   1. Every computer has one
   2. Indicates the normal size of pointer data
   3. For a machine with a -bit word size, the virtual address space can range from to
      1. Gives the program access to at most bytes
   4. Recent years there has been a huge shift from 32-bit word sizes to 64-bit word sizes
2. **sizeof(**type**)**
   1. Can be used if you need to programmatically determine the size of a specific data type.
      1. Examples:

## Addressing and Byte Ordering

1. **Little endian** vs. **Big endian**
   1. Example: Suppose int x at address 0x100 has a hex value of 0x01234567 (4 bytes).
      1. The ordering of the bytes within range 0x100 through 0x103 depends on type of machine.
         1. **Big endian**:
            1. Most significant byte first

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | 0x100 | 0x101 | 0x102 | 0x103 |
| Value | 01 | 23 | 45 | 67 |

* + - 1. **Little endian**:
         1. Least significant byte first

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address | 0x100 | 0x101 | 0x102 | 0x103 |
| Value | 67 | 45 | 23 | 01 |

## Bitwise operations in C

1. Boolean operations

|  |  |  |  |
| --- | --- | --- | --- |
| ~ is bitwise NOT | & is bitwise AND | | is bitwise OR | ^ is bitwise XOR |
| |  |  | | --- | --- | | A | ~A | | 0 | **1** | | 1 | **0** | | |  |  |  | | --- | --- | --- | | A | B | A&B | | 0 | 0 | **0** | | 0 | 1 | **0** | | 1 | 0 | **0** | | 1 | 1 | **1** | | |  |  |  | | --- | --- | --- | | A | B | A|B | | 0 | 0 | **0** | | 0 | 1 | **1** | | 1 | 0 | **1** | | 1 | 1 | **1** | | |  |  |  | | --- | --- | --- | | A | B | A^B | | 0 | 0 | **0** | | 0 | 1 | **1** | | 1 | 0 | **1** | | 1 | 1 | **0** | |

* 1. More on XOR
     1. a XOR a = 0
     2. a XOR b XOR b = a
     3. an interesting trick – neither is more efficient than the other, just an intellectual curiosity

|  |
| --- |
| void swap1(int \*a, int \*b) {  int tmp = \*a;  \*a=\*b;  \*b=tmp;  } |

|  |
| --- |
| void swap2(int \*a, int \*b) {  \*a^=\*b; /\* a = a XOR b \*/  \*b^=\*a;  \*a^=\*b;  } |

* 1. Example:

int x=9, y=5;

printf("x=%d, y=%d, x&y=%d\n", x, y, x&y);

printf("x=%d, y=%d, x|y=%d\n", x, y, x|y);

printf("x=%d, y=%d, x^y=%d\n", x, y, x^y);

* + 1. Output:

x&y=1  
x|y=13  
x^y=12

## Logical operations in C

1. && is logical AND
2. || is logical OR
3. ! is logical NOT

# C Programming

## I/O

1. int getchar(); – reads the next char from STDIN
2. int putchar(int c); – writes the character c to STDOUT

### I/O Redirection

1. Redirect STDIN Only Example: read\_file.c

|  |
| --- |
| #include <stdio.h>  int main(int argc, char \*\*argv) {  int c;  c=getchar();  while (c!=EOF) {  putchar(c);  c=getchar();  }  return 0;  } |

$> gcc –o read\_file read\_file.c

$> ./read\_file < tacos.txt

1. Redirect both STDIN and STDOUT Example:

$> ./read\_file < tacos.txt > out.txt

$> cat out.txt

|  |
| --- |
| It's raining tacos  From out of the sky  Tacos  No need to ask why  Just open your mouth and close your eyes  It's raining tacos  It's raining tacos  Out in the street  Tacos  All you can eat  Lettuce and shells  Cheese and meat  It's raining tacos |

## Programming Example: word\_count.c

### Deterministic Finite Automata (DFA)

**non-space**

**space**

**space**

**non-space**

**EOF**

**start**

**EOF**

1. DFAs are digraphs (i.e. **di**rected **graphs**)
   1. Key
      1. states
         1. In this case, we have two states (inside of word and outside of word) plus the final state
      2. arrows – transition from one state to the same or another state depending on value of input (called the transition function)
         1. transition function of example
            1. **current state**: outside of word

**input**: space, **next state**: outside of word

**input**: non-space, **next state**: inside of word

**input**: EOF, **next state**: exit program

* + - * 1. **current state**: inside of word

**input**: space, **next state**: outside of word

**input**: non-space, **next state**: inside of word

**input**: EOF, **next state**: exit program

1. Now that we have DFA, how do we write the program?

|  |
| --- |
| #include <stdio.h>  #include <ctype.h>  #define IN\_WORD 0  #define OUT\_WORD 1  int main(int argc, char \*\*argv) {  int cur; /\* current character \*/  int cc, /\* char count \*/  lc, /\* line count \*/  wc; /\* word count \*/  int state=OUT\_WORD;    cc=lc=wc=0;  while ((cur=getchar())!=EOF) {  cc++;  if (cur=='\n')  lc++;  if (state==IN\_WORD) {  if (isspace(cur)) {  state=OUT\_WORD;  }  } else { /\* state is OUT\_WORD \*/  if (!isspace(cur)) {  state=IN\_WORD;  wc++;  }  }  }    printf("%d %d %d\n", cc, wc, lc);  return 0;  } |

# Key Terms and Concepts

* bits
* byte
* virtual memory
* address
* virtual address space
* non-decimal representation of numbers
  + hexadecimal (base 16)
  + binary (base 2)
  + octal (base 8)
    - Note: Conversion of a number to and from binary should be simple and follow the same processes for any base that equals a power of 2
* base conversion
* addition (and subtraction) in non-decimal bases
* word size
* sizeof(type)
* byte ordering
  + little endian
  + big endian
* bitwise operations
  + bitwise logic
  + bitwise data manipulation
* DFA