Lecture 27(3-27-26)

Data Representation (C)

Motivating Questions

- How are **integers** represented in memory
  - How are **negative** numbers represented?
  - How are **multi-byte** numbers stored?

Data: Binary Information

- Most modern computers represent information in **binary** format (as a series of **1's** and **0's**)
- **Analog** systems deal with a smooth or continuous **range** of values
- **Digital** systems deal with **discrete** values (only a fixed number of possible values)

Data: Number Representation

- All numbers are encoded by some sort of **base.** For instance, consider the number "123"
- EX:

```
Decimal (base 10):          Hexadecimal (base 16):

  1 x 10^2  = 100             1 x 16^2  = 256
  2 x 10^1  =  20             2 x 16^1  =  32
  3 x 10^0  =   3             3 x 16^0  =   3
            = 123                       = 291
```

- Need to know decimal 0-15 representation in binary and hex for exam03

Data: Bytes

- A sequence of 8 **bits** as a **byte**. We can further group bytes into prefix groups:

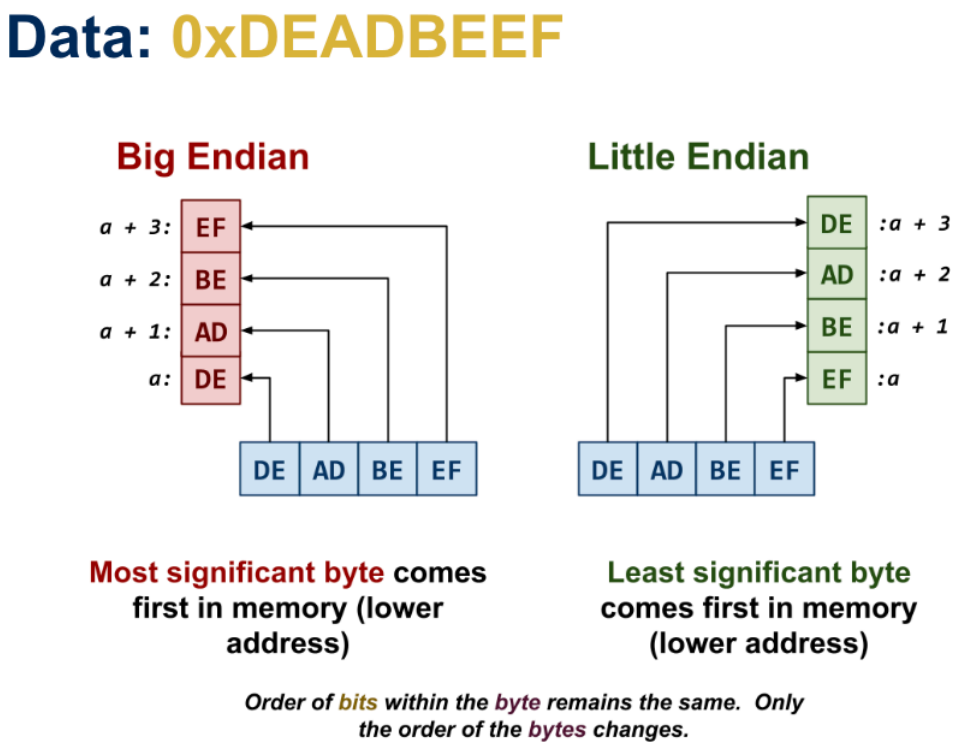
Symbol	Prefix	Multiplier	Approximation
K	Kilo	2^{10}	Thousand
M	Mega	2^{20}	Million
G	Giga	2^{30}	Billion
T	Tera	2^{40}	Trillion

- Our computers can do ~Billion instruction per second
- Number representation matters:
  - Boeing used to (maybe still does) reboot aircraft every 30 days to reset their 32 bit int to 0, to prevent overflow

---

Data: 0xDEADBEEF

- In **Big Endian** systems, the **most significant byte** comes first in memory (lower address)
- In **Little Endian** systems, the **least significant byte** comes first in memory (lower address)
- Order of **bits** within the **byte** remains the same. Only the order of the **bytes** changes.
- Slide:



- today most machines are *little endian*

---

dump_bits.c

```c
/* dump_bits.c */

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

/* Macros */

#define DUMP_BITS(_v) \
    for (int _i = sizeof(_v)*8 - 1; _i >= 0; _i--) {\
        if ((_i + 1) % 8 == 0) putc(' ', stdout); \
        putc(((_v) & 1<<_i) ? '1' : '0', stdout); \
    } \
    putc('\n', stdout);

/* Main Execution */

int main(int argc, char *argv[]) {
    /* Signed and Unsigned are the same underlying bits */
    int         i = 0xDEADBEEF;
    unsigned    u = 0xDEADBEEF;

    DUMP_BITS(i);
    DUMP_BITS(u);
    printf("i = %d, u = %u, i == u ? %d\n", i, u, i == u);  // Formatting

    /* Need to be careful with overflowing integer range */
#if 0
    for (int8_t b = 0; b < (1<<8); b++) {   // Overflows into infinite loop
         DUMP_BITS(b);
    }

    for (int c = 0; c < (1<<8); c++) {      // Fix by using larger int
        int8_t b = c;
         DUMP_BITS(b);
    }
#endif

    /* Intel machines are little endian. */
#if 0
    int8_t *a = (int8_t *)&u;               // Cast into array
    for (int i = sizeof(u) - 1; i >= 0; i--) {
        printf("%lx = %hhx\n", (intptr_t)&a[i], a[i]);
```

```
    }
#endif

    return EXIT_SUCCESS;
}

/* vim: set sts=4 sw=4 ts=8 expandtab ft=c: */
```

---

wtf.c

```
/* wtf.c */

#include <stdio.h>
#include <stdint.h>

/* Main Execution */

int main(int argc, char *argv[]) {
    unsigned pos_one =  1;
    int      neg_one = -1;

    /* When mixing signed and unsigned, the signed int is treated as
     * unsigned.*/
    if (pos_one < neg_one)  // Fix with a cast
        puts("wtf!");
    else
        puts("obv!");

    /* When mixing integers of different sizes, the smaller one gets
promoted
     * to the larger one */
    int16_t count = 0x7ff0;
    printf("count:   %hd\n", count);
    printf("literal: %d\n", 0x8001);
    printf("literal: %hd\n", (short)0x8001);
    while (count < 0x8001) {    // This is promoted to signed int
        count += 1;             // We overflow
        printf("%hd\n", count);
    }
    // Fix 1: Cast (short)0x8001
    // Fix 2: Change int16_t count to uint64_t count
    // Fix 3: Change (count < 0x8001) to (count < 0x8001U)
```

```
    return 0;
}

/* vim: set sts=4 sw=4 ts=8 expandtab ft=c: */
```

- if you have something that is unsigned and signed in comparison, the signed is interpreted as unsigned