CS201- Midterm Review

RAOUL RIVAS

PORTLAND STATE UNIVERSITY

Midterm Overview

- In-class Midterm
- Closed book. Bring a cheap calculator
- 70 Minutes, 4 Questions
- Covers Lectures 1 to 6 and Homeworks 1 and 2
- Show all your work

4 Broad Topics

- Hex/Binary/Decimal Arithmetic
  - Conversion, Two’s Complement, Addition, Subtraction, Multiplication, Logic Operations

- Compilation and Linking
  - Object Files, Linking Process, Endianness

- C Language
  - Dynamic Memory, Arrays, Pointers, Strings

- IEEE Floating Point
  - IEEE Representation, Fractional Binary

C Strings

- Implemented as static arrays of characters

  char mystr [length];

- Strings are not a type in C. They are an array!
- Last character must be NULL (zero) also written ‘\0’.
  - So if you need to store words of 5 letters you need an array of characters of length 6.

```
char one[6] = “Hello”;
char two[6] = {‘H’,’e’,’l’,’l’,’o’,‘\0’};
```

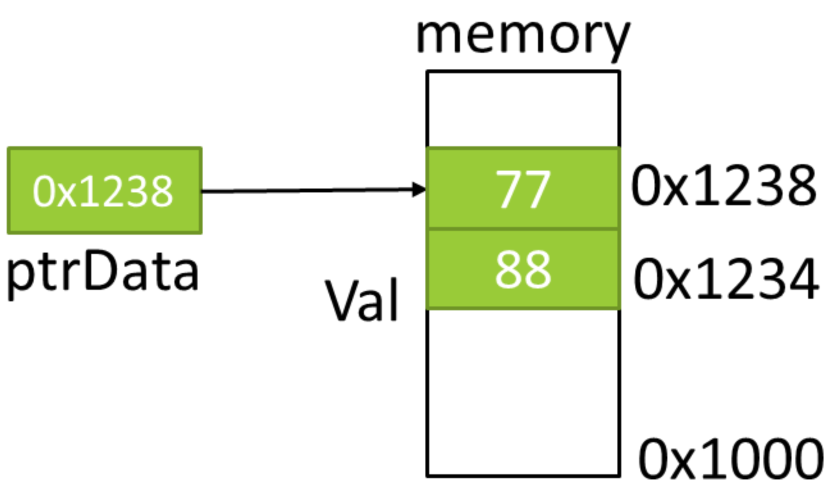
H	e	l	l	o	\0
72	101	108	108	111	0

Memory (ASCII)

Pointers

- A variable that stores the address of a region in memory
- Use arithmetic operators to manipulate pointers
- Dereference operator * to access what the pointer "points to"
- Address-of operator & to get the address of a variable

```
int *ptrData;
int Val[2] = {55, 66};

ptrData=&Val;
*ptrData=88;
ptrData++;
*ptrData=77;
```



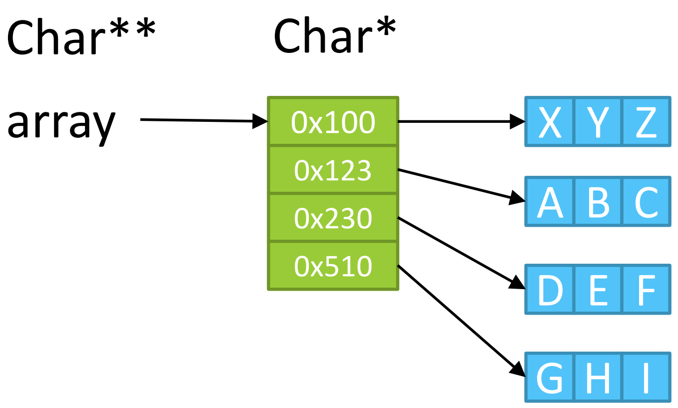
Pointers to Functions

- C also allows to create pointers to functions
  - Change the execution of a program at runtime
  - Create plugins and extensions

```
void print_even(int i) {printf(“Even: %d\n”, i);}
void print_odd(int i) {printf(“Odd: %d\n”, i);}

int main(int argc, char *argv[])
{
  void (*fp)(int);

  fp=(argc%2) ? print_even : print_odd;
  fp(argc);
  return 0;
}
```

Multidimensional Dynamically Allocated Arrays

```
char** array;
int i;

array = (char**)malloc(sizeof(char*)*4);

for(i=0; i < 4; i++){
  array[i] = (char*)malloc(sizeof(char)*3);
}

array[2][0]='D';

for(i=0; i < 4; i++) {
  free (array[i]);
}

free(array);
```



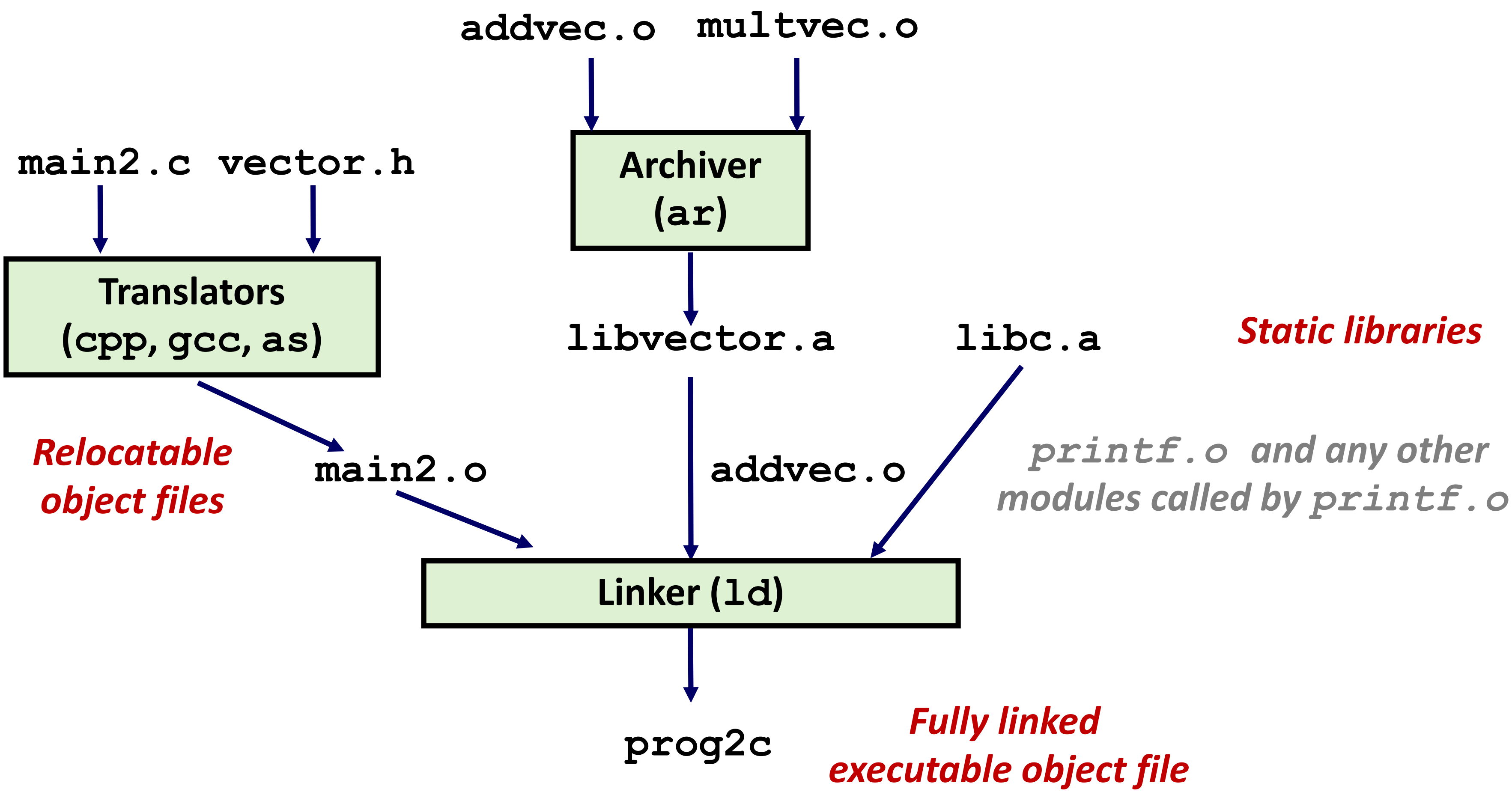
Object Files

- Relocatable object file (`.o` file)
  - Contains code and data in a form that can be combined with other relocatable object files to form executable object file.
    - Each `.o` file is produced from exactly one source (`.c`) file

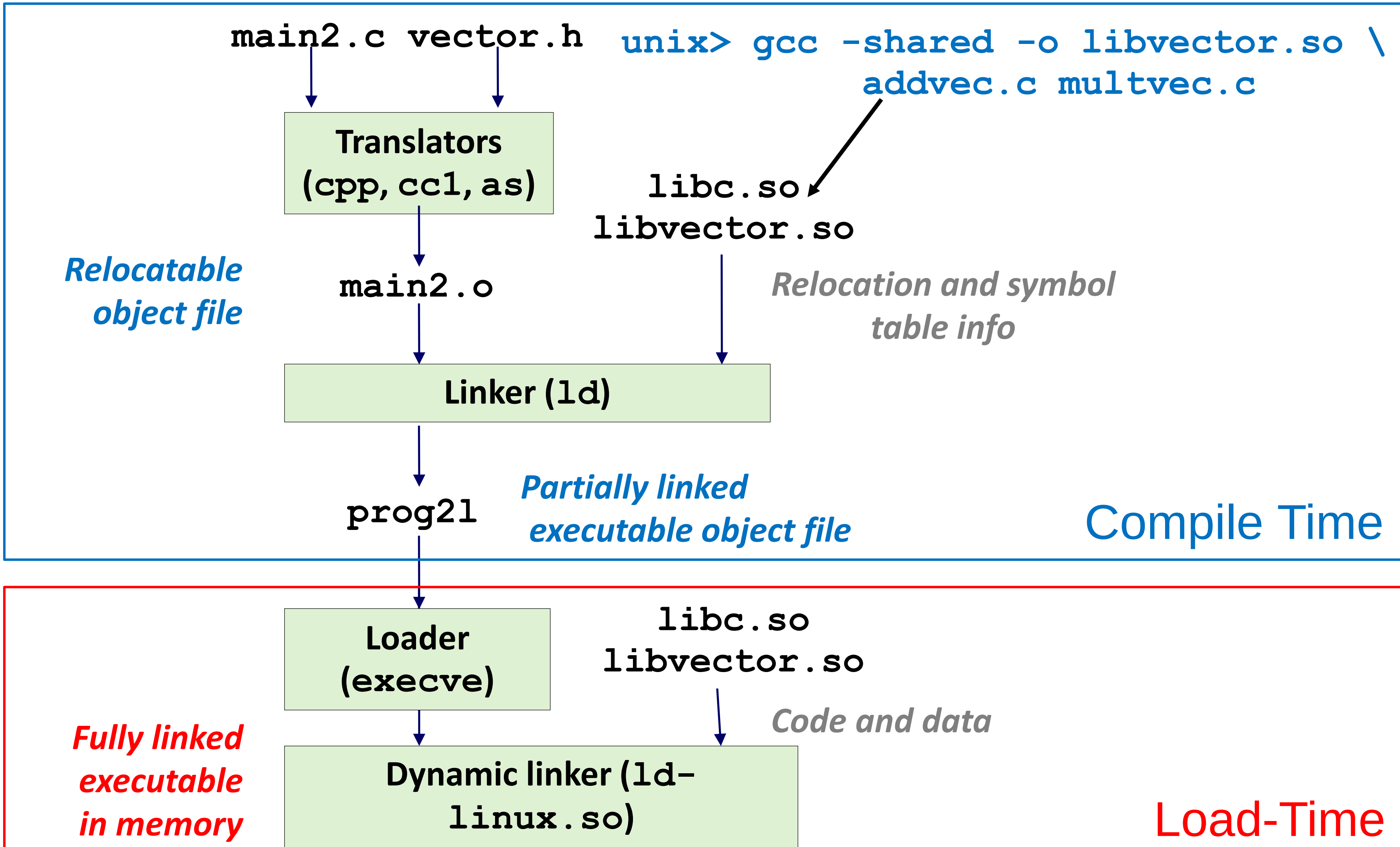
- Executable object file (`a.out` file)
  - Contains code and data in a form that can be copied directly into memory and then executed.
  - This are *.EXE and *.COM files in Windows
  - Non Relocatable!

- Shared object file (`.so` file)
  - Special type of relocatable object file that can be loaded into memory and linked dynamically, at either load time or run-time.
  - Called *Dynamic Link Libraries* (DLLs) by Windows

Linking Static Libraries

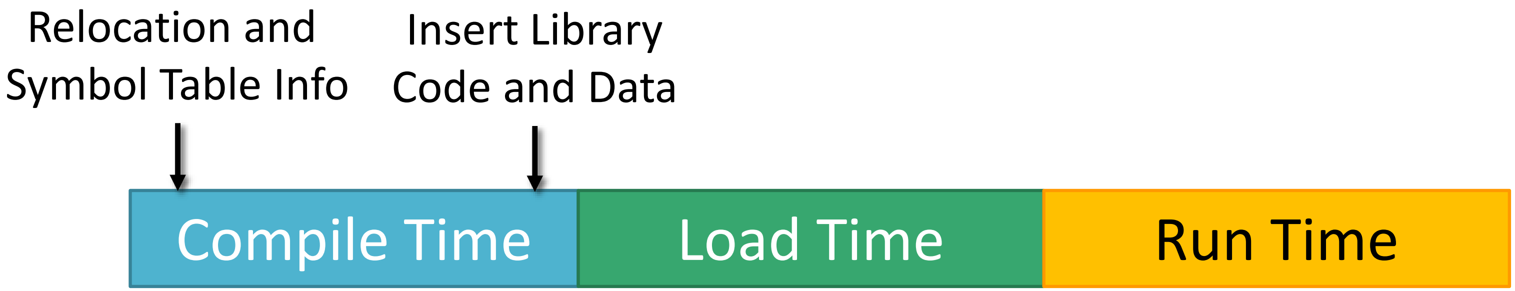


Dynamic Linking at Load-time

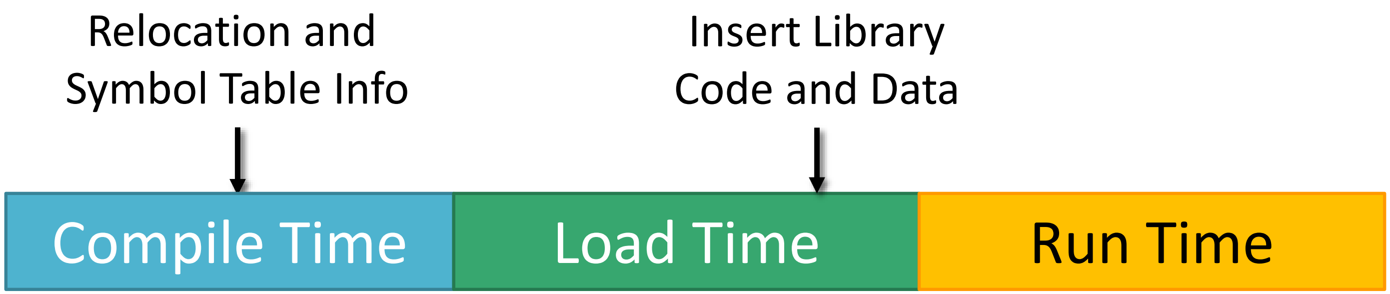


Library Linking Timeline

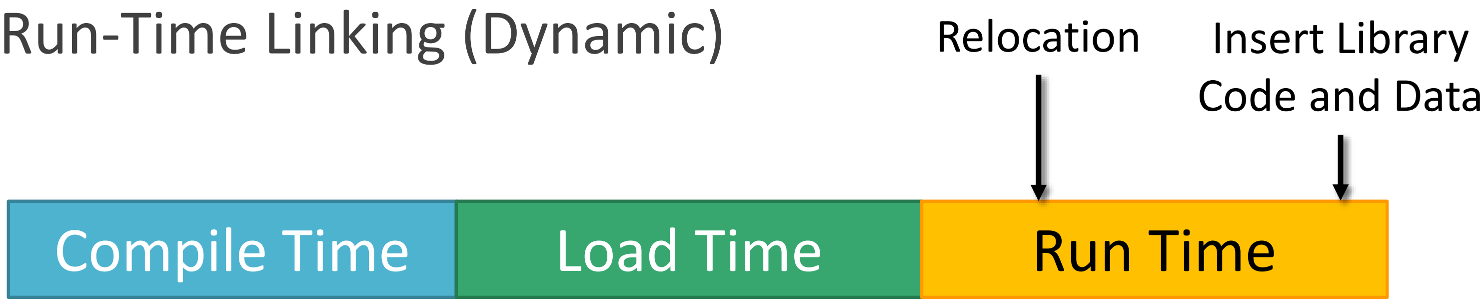
- Compile-Time Linking (Static)



- Load-Time Linking (Dynamic)



- Run-Time Linking (Dynamic)



Binary Numbers

- Base 2 Number Representation
  - Represent 15213_{10} in Binary
  - To convert we use a sequence of divisions by powers of 2:
    - 15213 – 8192 = 7021
    - 7021 – 4096 = 2925
    - 2925 – 2048 = 877
    - 877 – 512 = 365
    - 365 – 256 = 109
    - 109 – 64 = 45
    - 45 – 32 = 13
    - 13 – 8 = 5
    - 5 – 4 = 1
    - 1 – 1 = 0
    - 11101101101101_2

1	2^0
2	2^1
4	2^2
8	2^3
16	2^4
32	2^5
64	2^6
128	2^7
256	2^8
512	2^9
1024	2^{10}
2048	2^{11}
4096	2^{12}
8192	2^{13}
16384	2^{14}

Basic Binary Arithmetic - Addition

- Binary addition by hand is similar to its base-10 addition (“grade-school algorithm”)

```
  1     1
  01101001
+ 01010101
----------
  10111110
```

```
  1  1111
   11011111
+  10000110
-----------
  101100101
```

Basic Binary Arithmetic - Multiplication

- Binary multiplication by hand is similar to its base-10 multiplication (“grade-school algorithm”)

```
    1101001
×       101
-----------
    1101001
+  0000000
  1101001
-----------
 1000001101
```

```
       11011111
×         10000
---------------
   110111110000
```

The same trick of shifting left applies ☺

Two's Complement Representation

- Signed Integer representation in modern computers
  - Suggested by Von Neumann in 1945
- Positive Integers are represented by themselves
- Negative Integers are represented by its Two's complement
- The two's complement TC(n) of an N-bit number n is defined as the complement with respect to 2^N:

$$\mathrm{TC(n)} = 2^N - n$$

- For a 16-bit Integer:
  - **15213 = $00111011\ 01101101_2$**
  - **-15213 = TC(15213) = 2^{17} – 15213 = 1100010010010011_2**

```
short int x =  15213;
short int y = -15213;
```

	Decimal	Hex	Binary
x	15213	**3B 6D**	00111011 01101101
y	-15213	**C4 93**	11000100 10010011

Integer Binary Subtraction

- Binary subtraction is done as an addition of the minuend plus the two's complement of the subtrahend
  - Ignore the carry over at the end! (Modular arithmetic)

```
  01101001
- 01010101
```

Two's Complement →

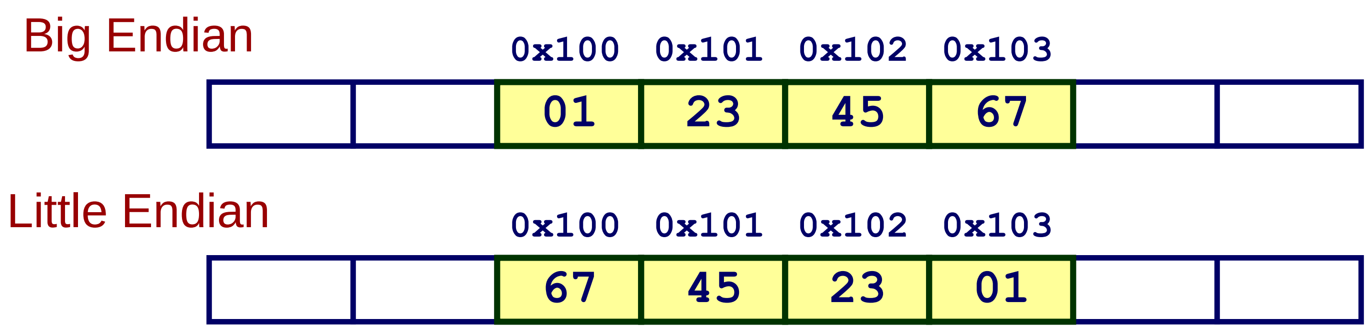
```
 111 1 11
  01101001
+ 10101011
 100010100
```

105 – 85 = 276 mod 256 = 20

$$s \quad = \quad \mathrm{USub}_w(u\,,\,v) \quad = \quad u + (\sim v + 1) \ \bmod 2^w$$

Byte Ordering Example

- Example
  - Variable x has 4-byte value of 0x01234567
  - Address given by &x is 0x100



This is important when writing files or connecting to the network

Fractional Binary Numbers

- **Value** **Representation**

Value	Representation
5 3/4	101.11_2
2 7/8	10.111_2
1 7/16	1.0111_2

- **Observations**
  - Divide by 2 by shifting right (unsigned)
  - Multiply by 2 by shifting left
  - Numbers of form $0.111111\ldots_2$ are just below 1.0
    - $1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0$
    - Use notation $1.0 - \varepsilon$

Normalized Encoding

$v = (-1)^s \, M \, 2^E$
$Exp = E + Bias$

- Value: `float F = 15213.0;`
  - $15213_{10} = 11101101101101_2$
    $= 1.1101101101101_2 \times 2^{13}$ NORMALIZE

- Significand
  M = `1.`$\underline{1101101101101}_2$
  `frac` = $\underline{1101101101101}0000000000_2$

- Exponent
  E = 13
  $Bias$ = 127 (because we are encoding a single precision number)
  Exp = 140 = 10001100_2

- Result:

s	exp	frac
0	10001100	11011011011010000000000