Week 9 Lecture 1

NWEN 241
Systems Programming

Alvin C. Valera

alvin.valera@ecs.vuw.ac.nz

Content

Low-level Systems Programming

Lecture 1:

- Conditional inclusion/compilation
- Standard integer types
- Bit-wise operators

Lecture 2:

- Bit-fields
- Memory alignment
- Structure padding and packing

Conditional Inclusion / Compilation

Recall from Week 1:

- Pre-processor directives (begins with #) are instructions to the pre-processor for performing
  - File inclusion
  - Macro substitution
  - *Conditional inclusion*

Conditional Inclusion (Conditional Compilation):
Capability provided by the preprocessor for selecting lines of code that will be compiled and ignored

Conditional Inclusion Directives

- Six directives can be used to control conditional compilation
  - Beginning of block:

```
#if
```

```
#ifdef
```

```
#ifndef
```

  - Optional, alternative block:

```
#else
```

```
#elif
```

  - End of block:

```
#endif
```

Conditional Inclusion Operator

- In addition to the six directives, an operator is also available

```
defined name
```

```
defined (name)
```

  - Evaluates to 1 if ***name*** is defined, otherwise it evaluates to 0

Example using #ifdef and #endif

```
#ifdef MACRO

/**
 * Code here will be compiled
 * if MACRO is defined previously
 */

#endif /* MACRO */
```

Example using #ifndef and #endif

```
#ifndef MACRO

/**
 * Code here will be compiled
 * if MACRO is not defined previously
 */

#endif /* MACRO */
```

Example using #if, #elif, #else and #endif

```
#if defined(linux)

/* Code here will be compiled if linux is defined */

#elif defined(_WIN32)

/* Code here will be compiled if _WIN32 is defined */

#else

/* Code here will be compiled if neither linux or _WIN32
   is defined */

#endif
```

Where to define a macro

- Within a header or source file
  - Using `#define` directive
- Within a `Makefile`
  - To be discussed in “Writing Large Program” Tutorial on Friday
- As a command-line option to `gcc` or `g++`
  - `gcc –DMACRO_NAME source.c`: `MACRO_NAME` will be defined in `source.c`
  - `g++ -DMACRO_NAME source.cc`: `MACRO_NAME` will be defined in `source.cc`

Example

hello.c

```c
#include <stdio.h>

int main(void)
{
#ifdef HELLO
    printf("hello\n");
#else
    printf("world\n");
#endif
    return 0;
}
```

```
$ gcc -DHELLO hello.c
$ ./a.out
hello

$ gcc hello.c
$ ./a.out
world
```

Recall: Basic C/C++ Data Types

Data Type	Size (bytes)
~~boolean~~	~~1~~
~~byte~~	~~1~~
char	~~2~~ 1
short (short int)	~~2~~ Machine-dependent
int	~~4~~ Machine-dependent
long (long int)	~~8~~ Machine-dependent
long long (long long int)	Machine-dependent
float	~~4~~ Machine-dependent
double	~~8~~ Machine-dependent
long double	10

Integral types

Float types

Integer Types

- A major problem in systems programming is the uncertainty of integer types
  - Size of an integer depends on the CPU architecture

- There are situations where you will need to specify the size precisely

Fortunately, C99 defines integer types with precise sizes

Fixed Size Integer Types

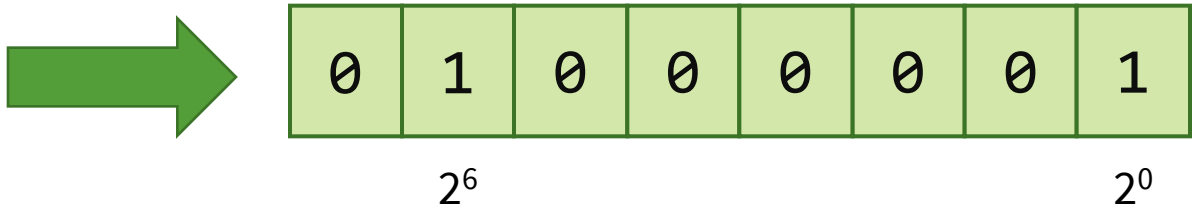
- Defined in `stdint.h` (C) and `cstdint` (C++)

Type	Sign and size
`int8_t`	8-bit signed integer
`int16_t`	16-bit signed integer
`int32_t`	32-bit signed integer
`int64_t`	64-bit signed integer
`uint8_t`	8-bit unsigned integer
`uint16_t`	16-bit unsigned integer
`uint32_t`	32-bit unsigned integer
`uint64_t`	64-bit unsigned integer

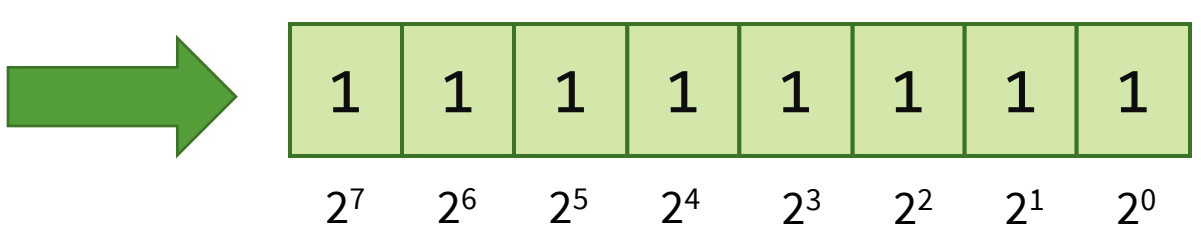
Binary Number Representation

- How are numbers actually stored in variables?
  - Numbers are stored using binary representation
  - See https://www.bottomupcs.com/chapter01.xhtml for details

```
char c = 'A'; //65
```




```
uint8_t b = 255;
```



Binary Number Representation

How are negative numbers represented?

- Use **two's complement**

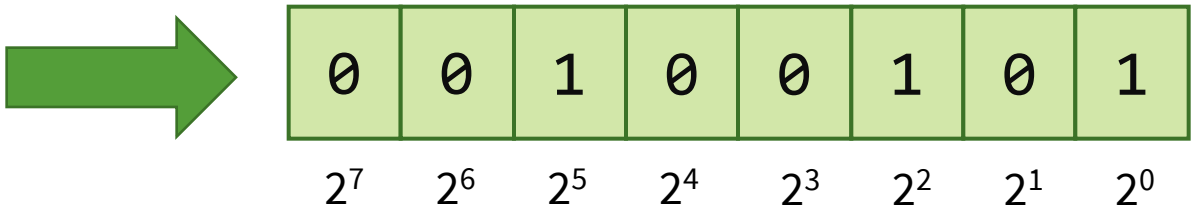
Two's Complement Operation:

1. Obtain binary representation disregarding the sign
2. If sign is positive, done
3. Otherwise:
    1. Invert all bits of the binary representation
    2. Add 1 to the binary representation

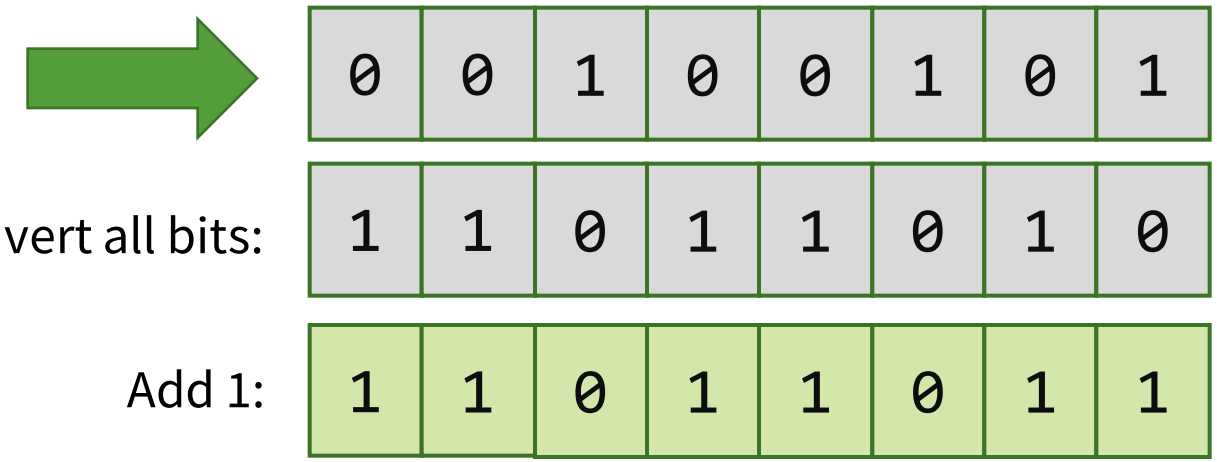
Binary Number Representation

- Example:

```
int8_t a = 37;
```

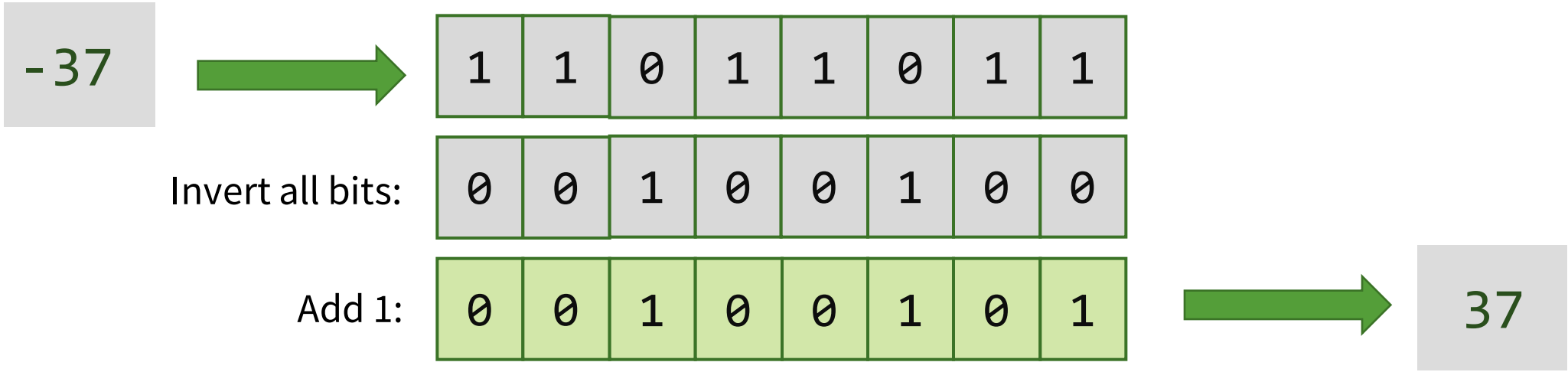



```
int8_t b = -37;
```



Binary Number Representation

- When you get the two's complement of a negative number, you will obtain the positive number



When to use unsigned types?

- Passing and comparing signed and unsigned values is a common pitfall in C/C++ programming

```
#include <stdio.h>
int array[] = {1, 2, 3};
void dump(unsigned int len)
{
        for(unsigned int i=0; i<len; i++)
                printf("array[%d]: %d\n", i, array[i]);
}
int main(void)
{
        dump(-1);
        return 0;
}
```

Will be treated as a very large positive integer by dump()

When to use unsigned types?

- If possible, avoid using unsigned types

- Areas where unsigned types are used:
  - When dealing with bit values
  - Performing bit-level operations

Endianness

- Consider a 32-bit (4-byte) integer:

```
uint32_t a = 305419896;
```

```
00010010001101000101011001111000
```

- How is it actually stored in computer memory?

There are two ways of storing multi-byte numbers in memory:

- **Big Endian**
- **Little Endian**

Endianness

- Big Endian – Store MSB in lower address
- Little Endian – Store LSB in lower address

Endianness

- Consider a 32-bit (4-byte) integer:

```
uint32_t a = 305419896;
```

```
00010010 00110100 01010110 01111000
```

- Suppose a is at address `100`

Big Endian:

100	00010010
101	00110100
102	01010110
103	01111000

Little Endian:

100	01111000
101	01010110
102	00110100
103	00010010

Bit-wise Operators

& – AND

- Result is **1** if both operand bits are **1**

| – OR

- Result is **1** if either operand bit is **1**

^ – Exclusive OR

- Result is **1** if operand bits are different

~ – Complement

- Each bit is reversed

<< – Shift left

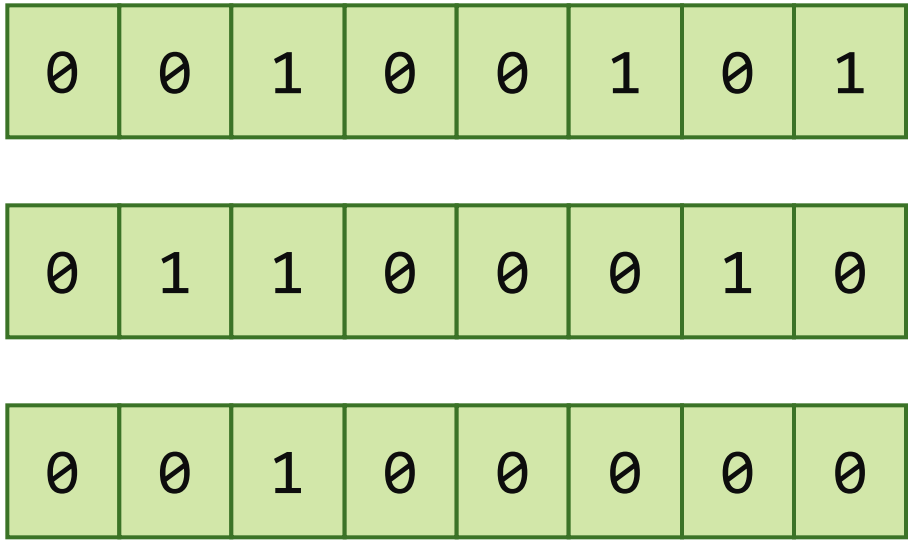
- Multiply by 2

>> – Shift right

- Divide by 2

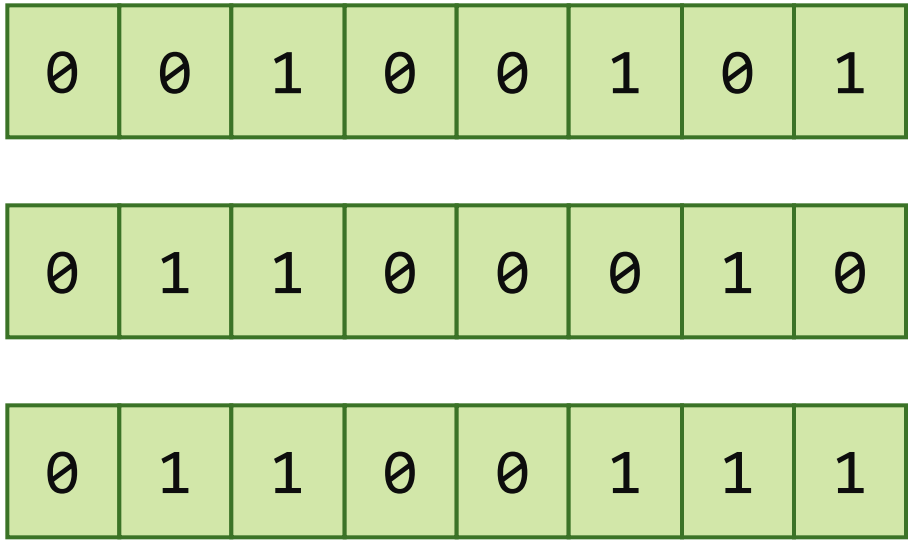
Example: AND

```
uint8_t a = 37;

uint8_t b = 98;

uint8_t c = a & b;
```



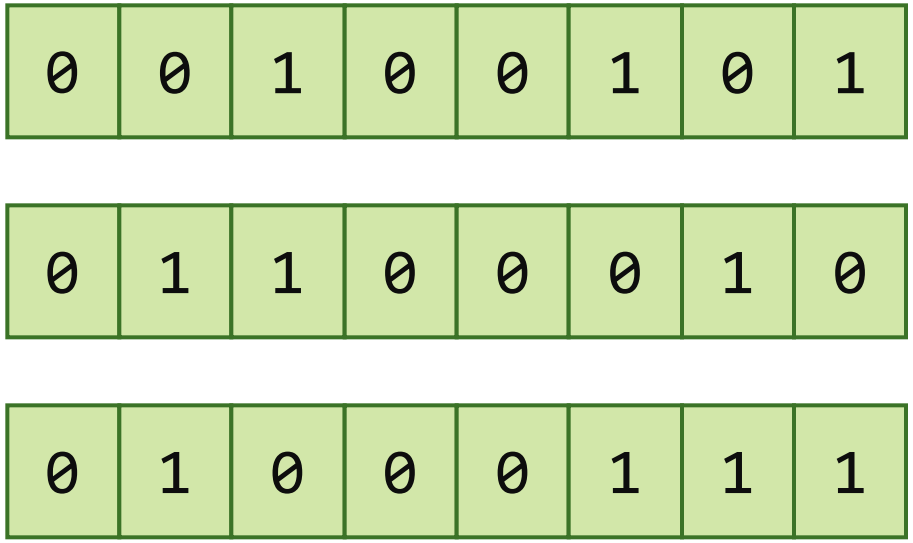
Example: OR

```
uint8_t a = 37;

uint8_t b = 98;

uint8_t c = a | b;
```



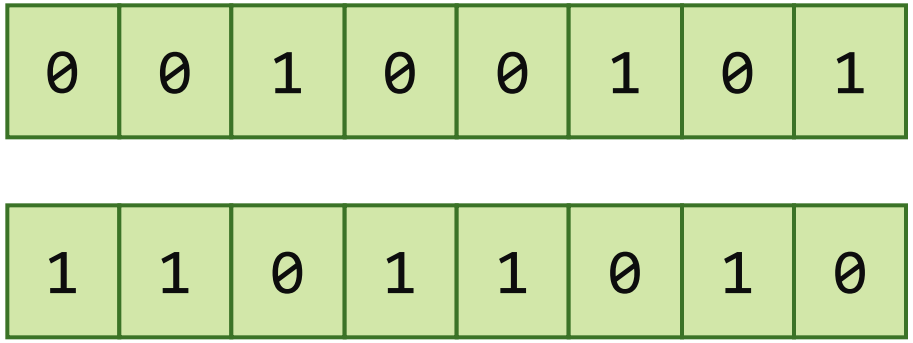
Example: Exclusive OR

```
uint8_t a = 37;

uint8_t b = 98;

uint8_t c = a | b;
```



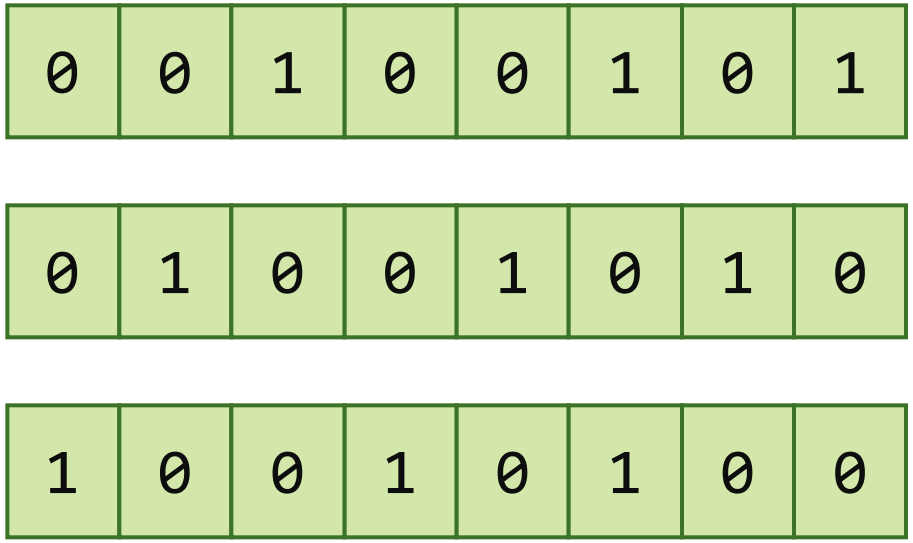
Example: Complement

```
uint8_t a = 37;

uint8_t b = ~a;
```



Example: Shift Left

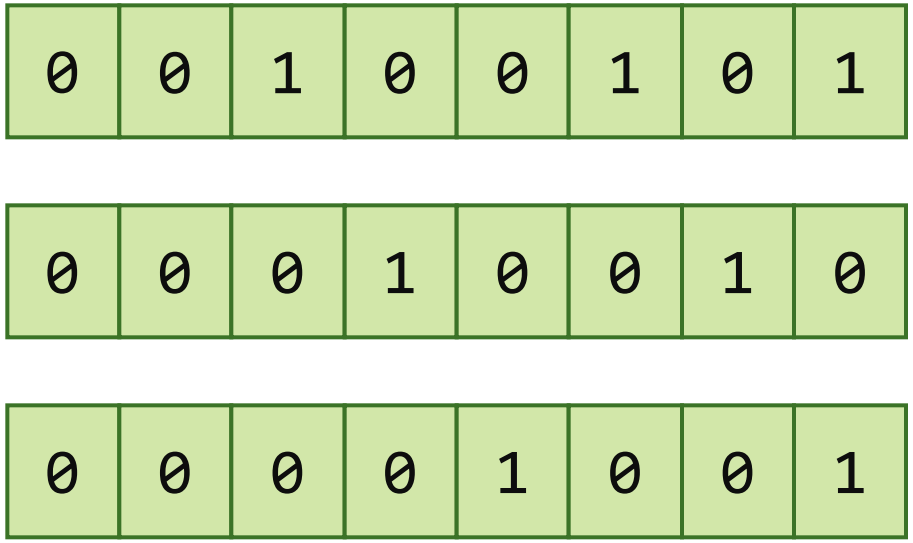
```
uint8_t a = 37;

uint8_t b = a << 1;

uint8_t c = a << 2;
```



What is b and c in decimal?

Example: Shift Right

```
uint8_t a = 37;

uint8_t b = a >> 1;

uint8_t c = a >> 2;
```



What is b and c in decimal?

Content

Next Lecture

- Bit-fields
- Memory alignment
- Structure padding and packing