CMPE-013/L

Introduction to “C” Programming

Gabriel Hugh Elkaim
Winter 2015

Bit manipulation

Bit masking
Bit flags
Bit fields

— sprint about —
Bit manipulation

Bit packing

C1CTRL1 - dsPIC33EP256MC502

Bit manipulation

Bit masks

Example

```
// Abort the current CAN message transmission
C1CTRL1 = C1CTRL1 | 0x1000;

C1CTRL1 = 0x1000;  // SET bit

c1ctrl1 |= 0x1000;
```
Bit manipulation

Bit masks

Example

// Disable CAN message timestamping
C1CTRL1 = C1CTRL1 & 0xFFF7;

CLEAR BIT

Example

// Disable CAN message timestamping
C1CTRL1 &= ~(1 << 3);
Bit manipulation

- A constant that indicates which bits are relevant for a given variable
- One bits indicate significant bits
- Zero bits indicate ignore bits

Example

```c
#define CxCTRL1_MASK_CANCAP (1 << 3)

// Disable CAN message timestamping
C1CTRL1 &= CxCTRL1_MASK_CANCAP; // forces bit to 0.
```
Bit manipulation

- **Setting a bit**
  - ORing with 1: \( C1CTRL1 |= CxCTRL1\_MASK\_CANCAP; \)
- **Clearing a bit**
  - ANDing with 0: \( C1CTRL1 &= \sim CxCTRL1\_MASK\_CANCAP; \)
- **Toggling a bit**
  - XORing with 1: \( C1CTRL1 ^= CxCTRL1\_MASK\_CANCAP; \)

\[ \sim 125\ \text{msec.} \]

Bit manipulation

- Setting a bit can OR multiple masks together

```c
enum {
    BUTTON_EVENT_1UP = 0x01,
    BUTTON_EVENT_2UP = 0x04,
}

{ uint8_t event = BUTTON_EVENT_1UP | BUTTON_EVENT_2UP;
}```
Bit manipulation

Bit masking

• Getting a bit
  – ANDing with 1

Example

```c
#define CxCTRL1_MASK_CANCAP (1 << 3)

// If CAN message timestamping is enabled
if (C1CTRL1 & CxCTRL1_MASK_CANCAP == CxCTRL1_MASK_CANCAP) {
    ...
}
```

---

Bit manipulation

Bit masking

• Getting a bit
  – ANDing with 1

Example

```c
#define CxCTRL1_MASK_CANCAP (1 << 3)

// If CAN message timestamping is enabled
if (C1CTRL1 & CxCTRL1_MASK_CANCAP) {
    ...
}
```
Bit manipulation

Bit masking

Example

// Retrieve the operating mode of the CAN hardware
int opmode = (C1CTRL1 & 0xE0) >> 5;

Bit Fields

Definition

Bit Fields are (unsigned) int members of structures that occupy a specified number of adjacent bits from one to sizeof(int). They may be used as an ordinary int variable in arithmetic and logical operations.

• Bit Fields:
  – Are ordinary members of a structure
  – Have a specified bit width
  – Provide bit access to a variable without masking operations
Bit Fields

- Bit Fields:
  - May only be integers (short, long, __, long long)
    - No larger than the base type
  - Unsigned by default, but may be signed
  - Non-portable across architectures/compilers!
    - Just like regular structs

Syntax

```
struct StructName {
    ((un)signed) int memberName1: bitWidth;
    ...
    ((un)signed) int memberName_n: bitWidth;
}
```

Example

```
struct ByteBits {
    unsigned int a: 1;
    long b: 1;
    short c: 2;
    unsigned d: 1;
    long long e: 3;
};
```
Bit Fields
How to Use a Bit Field

Example

typedef struct {
    unsigned int  a: 1;
    long          b: 1;
    short         c: 2;
    unsigned     d: 1;
    long long     e: 3;
} ByteBits;

ByteBits x;

x = 0x101

bitfield struct may be declared normally or as a typedef

Bit Fields
How to Use a Bit Field

Example

struct ByteBits {
    unsigned a: 1;
    unsigned b: 1;
    unsigned c: 2;
    unsigned d: 1;
    unsigned e: 3;
} x;

int main(void)
{
    x.a = 1;     // x.a may contain values from 0 to 1
    x.b = 0;     // x.b may contain values from 0 to 1
    x.c = 0b10;  // x.c may contain values from 0 to 3
    x.d = 0x10;  // x.d may contain values from 0 to 1
    x.e = 7;     // x.e may contain values from 0 to 7
}
Bit Fields

Microchip's SFRs

Example

```c
extern volatile unsigned int C1CTRL1 __attribute__((__sfr__));

// SFR bitfield declaration
typedef struct {
    unsigned WIN : 1;
    unsigned CANCAP : 1;
    unsigned OPMODE : 3;
    unsigned REOOP : 3;
    unsigned CANCKS : 1;
    unsigned ABAT : 1;
    unsigned CSIIDL : 1;
} C1CTRL1BITS;

extern volatile C1CTRL1BITS C1CTRL1bits __attribute__((__sfr__));
```

---

Bit Fields

How to Use a Bit Field

Example

```c
int main(void)
{
    // Abort the current CAN message transmission
    C1CTRL1  = 0x1000;
    C1CTRL1.ABAT = 1;

    // Disable CAN message timestamping
    C1CTRL1  &= 0xFFFF;

    // If CAN message timestamping is enabled
    if (C1CTRL1 & 0x0008) {
        ...
    }
}
```
Bit Fields

Example

```c
typedef struct {
    signed int a: 3;
    short b: 2;
    signed short c: 2;
    long long d: 3;
} ByteBits;

ByteBits x;
```

Signed values
Bit Fields
Maximum bitness

Example

typedef struct {
    signed int a: 3;
    short b: 2;
    signed short c: 1;
    long long d: 3;
} ByteBits;

ByteBits x;

Bit Fields
Maximum bitness

Example

typedef struct {
    signed short a: 3;
    short b: 2;
    signed short c: 1;
    short d: 3;
} ByteBits;

ByteBits x;

x.b = 5;

0 1 0 1

0 1 1
Metaprogramming: The C Preprocessor

Directives
Constants/Macros
Conditionals
Debugging

Preprocessor
Preprocessor stage

C Source File

C Compiler

Preprocessor

Compiler

C Header Files

Assembly Source File
Preprocessor

Operation of

- Preprocessor operates on all sources files before they're pass to the compiler
- Processes special **preprocessor directives** specified in the code
- Final text of the source file after all preprocessor directives are processed is then compiler

---

Preprocessor Directives

**Definition**

**Preprocessor Directives** are parts of the code that give special instructions to the compiler. They always begin with a `#` at the beginning of the line, and are used to direct the compiler with a number of specific commands.

- Groups:
  - `#defines`: constants, macros
  - Conditionals
- Usage:
  - Code organization
  - Debugging
Preprocessor Directives

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define</td>
<td>Define a preprocessor macro.</td>
</tr>
<tr>
<td>#elif</td>
<td>Alternatively include some text based on the value of another expression, if the previous #if, #ifdef, #ifndef, or #elif test failed.</td>
</tr>
<tr>
<td>#else</td>
<td>Alternatively include some text, if the previous #if, #ifdef, #ifndef, or #elif test failed.</td>
</tr>
<tr>
<td>#endif</td>
<td>Terminate conditional text.</td>
</tr>
<tr>
<td>#error</td>
<td>Produce a compile-time error with a designated message.</td>
</tr>
<tr>
<td>#if</td>
<td>Conditionally include text, based on the value of an expression.</td>
</tr>
<tr>
<td>#ifdef</td>
<td>Conditionally include text, based on whether a macro name is defined.</td>
</tr>
<tr>
<td>#ifndef</td>
<td>Conditionally include text, based on if a name is not a defined macro.</td>
</tr>
<tr>
<td>#include</td>
<td>Insert text from another source file.</td>
</tr>
<tr>
<td>#line</td>
<td>Reset the line number for compiler output</td>
</tr>
<tr>
<td>#pragma</td>
<td>Allows for extending preprocessor directives beyond what's in the standard</td>
</tr>
<tr>
<td>#</td>
<td>Null directive</td>
</tr>
<tr>
<td>#warning</td>
<td>Emits a warning described by the rest of the line</td>
</tr>
</tbody>
</table>

---

Preprocessor Directives

*Text substitution using #define*

- Defines a text substitution label

**Syntax**

```plaintext
#define label text
```

- Each instance of `label` will be replaced with `text` by the preprocessor unless `label` is inside a string
- `text` is optional
- Uses no memory

**Example**

```plaintext
#define PI 3.14159
#define MOL 6.02E23
#define MCU "PIC32MX320F128H"
#define PI_2 2 * PI
#define STDIO_H
```
Preprocessor Directives

Text substitution using `#define`

- Labels must be valid identifiers

```
#define 0 1
#define _WRONG
#define __WRONG
#define RIGHT
```

Preprocessor Directives

Text substitution using `#define`

- Text goes until the end of the line
  - Unless newline is escaped with a `\`

```
#define true false
#define true \false
```

- Constants can be nested

```
#define OLED_NUM_LINES (OLED_DRIVER_PIXEL_ROWS / ASCII_FONT_HEIGHT)
```
**Preprocessor Directives**

**Predefined constants**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FILE</strong></td>
<td>Full path of current file</td>
</tr>
<tr>
<td><strong>LINE</strong></td>
<td>The current line in the file</td>
</tr>
<tr>
<td><strong>DATE</strong></td>
<td>The current date as a string, like &quot;Jan 27 2014&quot;</td>
</tr>
<tr>
<td><strong>TIME</strong></td>
<td>The current time as a string, like &quot;17:20:50&quot;</td>
</tr>
<tr>
<td><strong>func</strong></td>
<td>The current function as a string, like &quot;main&quot;</td>
</tr>
<tr>
<td>DEBUG</td>
<td>When debugging is specified in MPLAB X, not part of the standard!</td>
</tr>
</tbody>
</table>

---

**Preprocessor Directives**

`#undef`  

**Syntax**  

```
#undef LABEL
```

- Deletes a macro definition  
- Allows you to change a macro  
  - Error when macros are redefined otherwise

**Example**

```
#define M_PI 3.14
#undef M_PI
#define M_PI 3.141592653589793238462643383279502884197
```
Preprocessor Directives

Argument Macros

• Create a function-like macro

**Syntax**

```c
#define LABEL(arg₁, ..., argₙ) code
```

- The `code` must fit on a single line or use `\` to split lines
- Text substitution used to insert arguments into `code`
- Each instance of `LABEL()` will be expanded into `code`
- This is not the same as a C function! No stack allocation.

**Example**

```c
#define MIN(x, y) ((x) < (y) ? (x) : (y))
#define SQUARE(x) ((x) * (x))
#define SWAP(x, y) { (x) ^= (y); (y) ^= (x); (x) ^= (y); }
```

---

Preprocessor Directives

Argument Macros – Side Effects

**Example**

```c
#define SQUARE(x) ((x) * (x))
```

Extreme care must be exercised when using macros. Consider the following use of the above macro:

```c
i = 5;
a = SQUARE(i + 3);
```

```
a = \underline{(i+3)} * (i+3) \quad \Downarrow
```

```
a = SQUARE (i++)
```

```
\[ a = (i+3) * (i+3) = 23 \]
```

```
\[ a = SQUARE (i++) \quad i++ \leftrightarrow i++ \]
```
Preprocessor Directives

Argument Macros – Side Effects

Example

```c
#define SQUARE(x) ((x)*(x))

Extreme care must be exercised when using macros.
Consider the following use of the above macro:

```c
i = 5;
a = SQUARE(i++);
```

Macros with `#define`

Argument Macros – Side Effects

Example

```c
#define ABS(x) (((x) > 0) ? (x) : (-x))
#define NORM1(x, y) (ABS((x)) + ABS((y)))

int x = NORM1(5, 6.6);
```

```
int x = (((5 > 0)?(5):(-5)) + (((6.6 > 0)?(6.6):(-6.6)));
```

```c
#define BTN1 PORTFbits.RF1
```
Macros with `#define`

**Emulating functions**

- Functions provide useful features:
  - **Encapsulation**
  - Evaluate as an expression
  - Return values

---

**Preprocessor Directives**

**Emulating functions**

- For encapsulation

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>
| `#define LABEL(arg_1, ..., arg_n) { ...

- Code blocks forces all code in the macro to execute in the same context
- Also allows for temporary variables within the macros
Preprocessor Directives
Emulating functions

Example

```c
#define INIT() TRISA = 5; LATA = 5;

if (beginStartup)
    INIT();
```

Preprocessor Directives
Emulating functions

Example

```c
#define INIT() {TRISA = 5; LATA = 5;};

if (beginStartup)
    INIT();
else
    ...
```

BAD
Preprocessor Directives

Emulating functions

• For encapsulation with expression-ness

Example

#define LABEL(arg₁, ..., argₙ) do {
    ... 
} while (0)

• Code blocks forces all code in the macro to execute in the same context
• Also allows for temporary variables within the macros
• `while`-statement allows for semi-colon termination
• Generates a single statement

Preprocessor Directives

Emulating functions

• To "return" values, just have the statement evaluate to a value

Example

#define LABEL(arg₁, ..., argₙ) VALUE
Preprocessor Directives
Stringification of macro values

Example

```c
#define VERSION 6.3
#define TEXTIFY(x) #x

printf("%s", TEXTIFY(VERSION));
```

6.3

Preprocessor Directives
Stringification of macro values

- You need another layer of indirection

Example

```c
#define TEXTIFY(x) TEXTIFY_HELPER(x)
#define TEXTIFY_HELPER(x) #x
#define MAJOR_VER 1
#define MINOR_VER 3
#define VERSION_STRING TEXTIFY(MAJOR_VER) \ 
    "." \ 
    TEXTIFY(MINOR_VER)

printf("%s", TEXTIFY(VERSION));
```

1.3
Preprocessor Directives

Token concatenation

- To combine argument with existing token to generate identifiers

```
#define DEBUGIFY(x) x ## _DEBUG
printf("%s", DEBUGIFY(asdf));
```

Preprocessor Directives

Conditional compilation

- Control what code actually gets compiled
  - Already seen this with header guards

```
#ifndef BUTTONS_H
#define BUTTONS_H

... // code goes here
#endif
```
Preprocessor Directives

Conditional compilation

- Family of if-statements
  - `#if`
  - `#ifdef` — `if defined`
  - `#ifndef` — `if not defined`
- Ended with `#endif`
- `#if` is the general case
  - `#ifdef/#ifndef` only check if a macro has been defined

Example

```c
#if INIT

#if 0

#if defined(_WIN32)

#if defined(__unix__) && !defined(__APPLE__)

#if __STDC_VERSION__ > 199409L
```
Preprocessor Directives

Conditional compilation

- `#ifdef text`
  - Same as `#if defined(...)`
- `#ifndef text`
  - Same as `#if !defined(...)`
- `#elif text`
  - Else-if, follows same rules as `#if`
- `#else`
- `#endif`

Preprocessor Directives

Unit testing

- Conditionally compile in test code

```
Example

int main(void)
{
    // Initialization code

    #if 0
        // Test code
    #endif
    // Main program
}
```
Preprocessor Directives

Fatal errors

- Output location of failure and stop running

```c
#define FATAL_ERROR()
    do {
        printf("FATAL ERROR at \%s:\%s() : \%ld\n", __FILE__, __func__, __LINE__);
        TRISE = 0;
        LATE = 0xFF;
    } while (1);
```

Preprocessor Directives

Forcing compilation errors/warnings

- `#warning text`
  - Outputs compilation warning
- `#error text`
  - Outputs compilation error

```c
#if __STDC_VERSION__ < 199901
#error "Must be compiled with C99 or greater"
#endif
```
Switch statements

switch Statement

Syntax

```java
switch (expression)
{
    case const-expr\_1: statements\_1
    :
    case const-expr\_n: statements\_n
    default: statements\_n+1
}
```

- `expression` is evaluated and tested for a match with the `const-expr` in each `case` clause
- The `statements` in the matching `case` clause is executed
**switch Statement**

Flow Diagram (default)

1. **START**
   - **Const-expr = expression?**
     - **YES** → **statement_1**
     - **NO** → **Const-expr = expression?**
       - **YES** → **statement_2**
       - **NO** → ...
       - **YES** → **statement_n**
       - **NO** → **statement_{n+1}**

Notice that each statement falls through to the next.

This is the default behavior of the `switch` statement.

**switch Statement**

Flow Diagram (modified)

1. **START**
   - **Const-expr = expression?**
     - **YES** → **statement_1**; **break**;
     - **NO** → **Const-expr = expression?**
       - **YES** → **statement_2**; **break**;
       - **NO** → ...
       - **YES** → **statement_n**; **break**;
       - **NO** → **statement_{n+1}**

Adding a `break` statement to each statement block will eliminate fall through, allowing only one case clause's statement block to be executed.
switch Statement

Simple example

**switch Example 1**

```c
switch (channel) {
  case 2:  puts("WBBM Chicago"); break;
  case 3:  puts("DVD Player"); break;
  case 4:  puts("WTMJ Milwaukee"); break;
  case 5:  puts("WMAQ Chicago"); break;
  case 6:  puts("WITI Milwaukee"); break;
  case 7:  puts("WLS Chicago"); break;
  case 9:  puts("WGN Chicago"); break;
  case 10: puts("WMVS Milwaukee"); break;
  case 11: puts("WTTW Chicago"); break;
  case 12: puts("WSN Milwaukee"); break;
  default: puts("No Signal Available");
}
```

switch Statement

Styling

**switch Example 1**

```c
switch (channel) {
  case 2:
    puts("WBBM Chicago");
    break;
  case 3:
    puts("DVD Player");
    break;
  case 4:
    puts("WTMJ Milwaukee");
    break;
  case 9:
    puts("WGN Chicago"); break;
  case 10:
    puts("WMVS Milwaukee"); break;
  case 11:
    puts("WTTW Chicago"); break;
  case 12:
    puts("WSN Milwaukee"); break;
  default:
    puts("No Signal Available");
}
```
switch Statement

With ASCII

switch Example 2

switch (letter) {
    case 'a':
        puts("Letter 'a' found.");
        break;
    case 'b':
        puts("Letter 'b' found.");
        break;
    case 'c':
        puts("Letter 'c' found.");
        break;
    default:
        puts("Letter not in list.");
}

switch Statement

Fall-through

switch Example 3

switch (channel) {
    case 4:
    case 5:
    case 6:
    case 7:
        puts("VHF Station");
        break;
    case 9:
    case 10:
    case 11:
    case 12:
        puts("VHF Station");
        break;
    default:
        puts("No Signal Available");
}
### switch Statement

**Range syntax**

```c
switch(channel)
{
    case 4 ... 7:
        puts("VHF Station");
        break;
    case 9 ... 12:
        puts("VHF Station");
        break;
    default:
        puts("No Signal Available");
}
```

### switch Statement

**Real-world example**

```c
bool IsHex(char character)
{
    switch (character)
    {
        case 'a' ... 'f':
        case 'A' ... 'F':
        case '0' ... '9':
            return true;
        default:
            return false;
    }
}
```