CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne
Spring 2016
Bit manipulation

Bit masking
Bit flags
Bit fields
Bit manipulation

Bit packing

- Data is commonly packed into larger unsigned integers on embedded systems
- Generally a tie in to hardware or when space is critical
  - Hardware
  - Storage
  - Binary formats
Bit manipulation

Bit packing

C1CTRL1 – dsPIC33EP256MC502
Bit manipulation
Bit masks

Example

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

Bit masks

Example

// Disable CAN message timestamping
C1CTRL1 = C1CTRL1 & 0xFFFF7;
Bit manipulation

Bit masks

Example

// Disable CAN message timestamping
ClCTRL1 &= ~(1 << 3);
Bit manipulation

Bit masks

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

Bit masks

Example

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

// Disable CAN message timestamping
C1CTRL1 &= ~CxCTRL1_MASK_CANCAP;
```
Bit manipulation

Bit masking

- Setting a bit
  - ORing with 1
    
- Clearing a bit
  - ANDing with 0
    
- Toggling a bit
  - XORing with 1
    
C1CTRL1 |= CxCTRL1_MASK_CANCAP;

C1CTRL1 &= ~CxCTRL1_MASK_CANCAP;

C1CTRL1 ^= CxCTRL1_MASK_CANCAP;
Bit manipulation

Bit masking

- Setting a bit can OR multiple masks together

Example

```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) {
    ...
}
```
// 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
Bit Fields
How to Create a Bit Field

Syntax

```c
struct StructName {
    ((un)signed) int memberName_1: bitWidth;
    ...
    ((un)signed) int memberName_n: bitWidth;
}
```

Example

```c
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;

bitfield struct may be declared normally or as a typedef
Bit Fields
How to Use a Bit Field

Example

```c
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 = 0x0; // x.d may contain values from 0 to 1
    x.e = 7; // x.e may contain values from 0 to 7
}
```

Byte in Data Memory (RAM)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

X: 1 1 1 0 1 0 0 1
// SFR register declaration
extern volatile unsigned int C1CTRL1 __attribute__((__sfr__));

// SFR bitfield declaration
typedef struct {
    unsigned WIN : 1;
    unsigned : 2;
    unsigned CANCAP : 1;
    unsigned : 1;
    unsigned OPMODE : 3;
    unsigned REQOP : 3;
    unsigned CANCKS : 1;
    unsigned ABAT : 1;
    unsigned CSIDL : 1;
} C1CTRL1BITS;
extern volatile C1CTRL1BITS C1CTRL1bits __attribute__((__sfr__));
```c
int main(void)
{
    // Abort the current CAN message transmission
    C1CTRL1 |= 0x1000;

    // Disable CAN message timestamping
    C1CTRL1 &= 0xFFFF7;

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

Example

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

ByteBits x;
Bit Fields
Signed values

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 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;
Metaprogramming: The C Preprocessor

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

```c
#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**

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

Text substitution using \texttt{#define}

- Labels must be valid identifiers

\begin{example}
\texttt{#define 0 1}
\texttt{#define \_WRONG}
\texttt{#define \_\_WRONG}
\texttt{#define RIGHT}
\end{example}
Preprocessor Directives

Text substitution using \#define

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

Example

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

- Constants can be nested

Example

```c
#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><code>__FILE__</code></td>
<td>Full path of current file</td>
</tr>
<tr>
<td><code>__LINE__</code></td>
<td>The current line in the file</td>
</tr>
<tr>
<td><code>__DATE__</code></td>
<td>The current date as a string, like &quot;Jan 27 2014&quot;</td>
</tr>
<tr>
<td><code>__TIME__</code></td>
<td>The current time as a string, like &quot;17:20:50&quot;</td>
</tr>
<tr>
<td><code>__func__</code></td>
<td>The current function as a string, like &quot;main&quot;</td>
</tr>
<tr>
<td><code>__DEBUG</code></td>
<td>When debugging is specified in MPLAB X, not part of the standard!</td>
</tr>
</tbody>
</table>
Preprocessor Directives

#define

#undef

- 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); }
```
#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);
```
Preprocessor Directives

Argument Macros – Side Effects

Example

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

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

```c
int x = (((5) > 0)?(5):(-5)) + (((6.6) > 0)?(6.6):(-6.6));
```
Macros with `#define`

Emulating functions

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

Emulating functions

- For encapsulation

**Example**

```c
#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; LATB = 5;};

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

Emulating functions

- For encapsulation with expression-ness

Example

```
#define LABEL(arg_1, ..., arg_n) 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(arg_1, ..., arg_n) 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

```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

Maxwell James Dunne – Spring 2016
Preprocessor Directives

Token concatenation

- To combine argument with existing token to generate identifiers

Example

```
#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

Example

```
#ifndef BUTTONS_H
#define BUTTONS_H

...
#endif
```
Preprocessor Directives
Conditional compilation

• Family of if-statements
  – #if
  – #ifdef
  – #ifndef
• Ended with #endif
• #if is the general case
  – #ifdef/#ifndef only check if a macro has been defined
Preprocessor Directives

Emulating functions

Example

```c
#if INIT
#if 0
#if defined(_WIN32)
#if defined(__unix__) && !defined(__APPLE__)
#if __STDC_VERSION__ > 199409L
```
Preprocessor Directives
Conditional compilation

• #ifdef  
  − Same as #if defined(...)  
• #ifndef  
  − Same as #if !defined(...) 
• #elif  
  − Else-if, follows same rules as #if
• #else
• #endif
Preprocessor Directives

Unit testing

• Conditionally compile in test code

Example

```c
int main(void)
{
    // Initialization code

    #if 0

        // Test code

    #endif

    // Main program
}
```
Preprocessor Directives

Fatal errors

- Output location of failure and stop running

```
#define FATAL_ERROR()  
   do {  
       printf("FATAL ERROR at \%s:\%s():\%ld\n",  
             __FILE__, __func__, __LINE__);  
       TRISE = 0;  
       LATE = 0xFFFF;  
   } while (1);
```
Preprocessor Directives
Forcing compilation errors/warnings

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

Example

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

Syntax

```c
switch (expression)
{
    case const-expr₁: statements₁
    :
    case const-exprₙ: statementsₙ
    default: statementsₙ₊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)

```
START

Const-expr_1 = expression? YES
    - statement_1

    NO

Const-expr_2 = expression? YES
    - statement_2

    NO

...  

Const-expr_n = expression? YES
    - statement_n

    NO

statement_{n+1}

END
```

Notice that each statement falls through to the next

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

Flow Diagram (modified)

START

Const-expr_1 = expression?  YES → statement_1; break;
NO

Const-expr_2 = expression?  YES → statement_2; break;
NO

Const-expr_n = expression?  YES → statement_n; break;
NO

statement_{n+1}

END

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**

```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("WISN 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;
    ...
}
```
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

```c
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

switch Example 2

```c
bool IsHex(char character)
{
    switch (character) {
    case 'a' ... 'f':
    case 'A' ... 'F':
    case '0' ... '9':
        return true;
    default:
        return false;
    }
}
```
CMPE-013/L

Linked Lists

Maxwell James Dunne
Spring 2016
Linked List

Theory

Foo 1016 bar

Foo bar

Sort
typedef struct ListItem {
    struct ListItem *previousItem;
    struct ListItem *nextItem;
    char *data;
} ListItem;
Linked List

ListItem *LinkedListNew(char *data);

LI = malloc(sizeof(ListItem));

C = data;

return LI;
Linked List

ListItem *ListItemCreateAfter(ListItem *item, char *data);

```
Item
  p
  N
  c
```

```
NItem
  p
  N
  c
```

Item -> N = NItem

NItem -> p = Item
 ListItem *LinkedListCreateAfter(ListItem *item, char *data);
Linked List

ListItem *LinkedListCreateAfter(ListItem *item, char *data);
Linked List

char *LinkedListRemove(ListItem *item);

\[ b \rightarrow N \rightarrow p = 0 \]
\[ \text{free}(b) \]

\[ e \rightarrow p \rightarrow n = 0 \]
\[ \text{free}(e) \]
Linked List

char *LinkedListRemove(ListItem *item);

begin

middle

end

m → p → N = m → N
m → N → p = m → p
free(m)
Linked List

`ListItem *LinkedListGetFirst(ListItem *list);`

```
0  ← P
  ↗   ↗
  N   N
  ↖   ↖
  C   C
```

```
if (item -> p == NULL)
```
Linked List

int LinkedListSize(ListItem *list);

int LinkedListPrint(ListItem *list);

Linked List get first

LI = LI \Rightarrow N;

Print float

strcmp

NULL \Rightarrow 0
X = malloc(3)
if(x)