The University of Texas at Arlington

Lecture 6
PIC Programming in C

CSE@UTA

CSE 3442/5442

Embedded Systems 1

Based heavily on slides by Dr. Gergely Záruba and Dr. Roger Walker

Code Space Limitations

- On a general purpose PC, we don't usually care about our program's size
- 2MB max in PIC18's Program ROM
- GB or TB range for general purpose PCs
  - Ex: 1300 line .C file 50 KB → 40 KB .hex file

Why C over ASM?

- While Assembly Language produces a much smaller .HEX file than C…
  - More human-readable in C
    - Easier to write and less time consuming
  - C is easier to modify and update
    - Don’t care about absolute ROM locations
  - Access to many C function libraries
  - C code is portable and can be used on other microcontrollers with little or no modification

C Integer Data Types (Generic)

Type	Explanation	Format Specifier
char	Smallest addressable unit of the machine that can contain basic character set. It is an integer type. Actual type can be either signed or unsigned depending on the implementation. It contains `CHAR_BIT` bits.[3]	%c
signed char	Of the same size as `char`, but guaranteed to be signed. Capable of containing **at least** the [−127, +127] range;[3][4]	%c (or %hhi for numerical output)
unsigned char	Of the same size as `char`, but guaranteed to be unsigned. It is represented in binary notation without padding bits; thus, its range is exactly [0, 2 `CHAR_BIT` − 1].[5]	%c (or %hhu for numerical output)
short short int signed short signed short int	*Short* signed integer type. Capable of containing **at least** the [−32767, +32767] range;[3][4] thus, it is at least 16 bits in size. The negative value is −32767 (not −32768) due to the one's-complement and sign-magnitude representations allowed by the standard, though the two's-complement representation is much more common.[6]	%hi
unsigned short unsigned short int	Similar to `short`, but unsigned.	%hu
int signed signed int	Basic signed integer type. Capable of containing **at least** the [−32767, +32767] range;[3][4] thus, it is at least 16 bits in size.	%i or %d
unsigned unsigned int	Similar to `int`, but unsigned.	%u
long long int signed long signed long int	*Long* signed integer type. Capable of containing **at least** the [−2147483647, +2147483647] range;[3][4] thus, it is at least 32 bits in size.	%li
unsigned long unsigned long int	Similar to `long`, but unsigned.	%lu
long long long long int signed long long signed long long int	*Long long* signed integer type. Capable of containing **at least** the [−9223372036854775807, +9223372036854775807] range;[3][4] thus, it is at least 64 bits in size. Specified since the C99 version of the standard.	%lli
unsigned long long unsigned long long int	Similar to `long long`, but unsigned. Specified since the C99 version of the standard.	%llu

C Integer Data Types (C18 Compiler)

TABLE 2-1: INTEGER DATA TYPE SIZES AND LIMITS

Type	Size	Minimum	Maximum
`char`(1,2)	8 bits	-128	127
`signed char`	8 bits	-128	127
`unsigned char`	8 bits	0	255
`int`	16 bits	-32,768	32,767
`unsigned int`	16 bits	0	65,535
`short`	16 bits	-32,768	32,767
`unsigned short`	16 bits	0	65,535
`short long`	24 bits	-8,388,608	8,388,607
`unsigned short long`	24 bits	0	16,777,215
`long`	32 bits	-2,147,483,648	2,147,483,647
`unsigned long`	32 bits	0	4,294,967,295

C Integer Data Types (XC8 Compiler)

TABLE 5-1: INTEGER DATA TYPES

Type	Size (bits)	Arithmetic Type
bit	1	Unsigned integer
signed char	8	Signed integer
unsigned char	8	Unsigned integer
signed short	16	Signed integer
unsigned short	16	Unsigned integer
signed int	16	Signed integer
unsigned int	16	Unsigned integer
signed short long	24	Signed integer
unsigned short long	24	Unsigned integer
signed long	32	Signed integer
unsigned long	32	Unsigned integer
signed long long	32	Signed integer
unsigned long long	32	Unsigned integer

Unsigned char (0 to 255)

- PIC18 is 8-bit architecture, **char type** (8 bits) is the most natural choice
- C compilers use **signed char** (-128 to +127) by default unless we put "unsigned"
  - **char == signed char**

Write a C18 program to send values 00–FF to Port B.

Solution:

```
#include <P18F458.h>            //for TRISB and PORTB declarations
void main(void)
  {
    unsigned char z;
    TRISB = 0;                  //make Port B an output
    for(z=0;z<=255;z++)
      PORTB = z;
    while(1);                   //NEEDED IF RUNNING IN HARDWARE
  }
```

Unsigned char array (0 to 255)

Write a C18 program to send hex values for ASCII characters of 0, 1, 2, 3, 4, 5, A, B, C, and D to Port B.

Solution:

```
#include <P18F458.h>
void main(void)
  {
    unsigned char mynum[]= "012345ABCD";//data is stored in RAM
    unsigned char z;
    TRISB = 0;                  //make Port B an output
    for(z=0;z<10;z++)
      PORTB = mynum[z];
    while(1);                   //stay here forever
  }
```

Hex	Dec	Char	Hex	Dec	Char
0x20	32	Space	0x40	64	@
0x21	33	!	0x41	65	A
0x22	34	"	0x42	66	B
0x23	35	#	0x43	67	C
0x24	36	$	0x44	68	D
0x25	37	%	0x45	69	E
0x26	38	&	0x46	70	F
0x27	39	'	0x47	71	G
0x28	40	(	0x48	72	H
0x29	41	)	0x49	73	I
0x2A	42	*	0x4A	74	J
0x2B	43	+	0x4B	75	K
0x2C	44	,	0x4C	76	L
0x2D	45	-	0x4D	77	M
0x2E	46	.	0x4E	78	N
0x2F	47	/	0x4F	79	O
0x30	48	0	0x50	80	P
0x31	49	1	0x51	81	Q
0x32	50	2	0x52	82	R
0x33	51	3	0x53	83	S
0x34	52	4	0x54	84	T
0x35	53	5	0x55	85	U
0x36	54	6	0x56	86	V
0x37	55	7	0x57	87	W
0x38	56	8	0x58	88	X
0x39	57	9	0x59	89	Y
0x3A	58	:	0x5A	90	Z
0x3B	59	;	0x5B	91	[
0x3C	60	<	0x5C	92	\
0x3D	61	=	0x5D	93	]
0x3E	62	>	0x5E	94	^
0x3F	63	?	0x5F	95	_

Signed char (-128 to +127)

- Still 8-bit data type but MSB is sign value

Write a C18 program to send values of –4 to +4 to Port B.

Solution:

```
//sign numbers
#include <P18F458.h>
void main(void)
   {
     char mynum[]= {+1,-1,+2,-2,+3,-3,+4,-4};
     unsigned char z;
     TRISB = 0;                    //make Port B an output
     for(z=0;z<8;z++)
       PORTB = mynum[z];
     while(1);                     //stay here forever
   }
```

Unsigned int (0 to 65,535)

- PIC18 is 8-bit architecture, **int type** (16 bits) takes two bytes of RAM (only use when necessary)
- C compilers use **signed int** (-32,768 to +32,767) by default unless we put “unsigned”
  - **int == signed int**

```
#include <P18F458.h>
void main(void)
  {
     unsigned int z;
     TRISB = 0;                  //make Port B an output
     for(z=0;z<=50000;z++)
        {
          PORTB = 0x55;
          PORTB = 0xAA;
        }
    while(1);                    //stay here forever
  }
```

Larger Integer Types (short, long, short long)

Write a C18 program to toggle all bits of Port B 100,000 times.

Solution:

```
//toggle PB 100,00 times
#include <P18F458.h>
void main(void)
  {
    unsigned short long z;
    unsigned int x;
    TRISB = 0;                    //make Port B an output
    for(z=0;z<=100000;z++)
      {
        PORTB = 0x55;
        PORTB = 0xAA;
      }
    while(1);                     //stay here forever
  }
```

Modulus

- In C can use **%** to perform a modulus of two numbers (find the remainder)

- 25 % 5 = 0
- 25 % 7 = 4
- 25 % 10 = 5
- 428 % 100 = 28
- 1568 % 10 = 8

Floating-Point Data Types

- Can store and calculate numbers with decimals (precision)
- Always **signed**, can't be **unsigned**

  2.5, 32.05898, -1.00232, .2600313, 51156.01, etc.

TABLE 5-3: FLOATING-POINT DATA TYPES

Type	Size (bits)	Arithmetic Type
`float`	24 or 32	Real
`double`	24 or 32	Real
`long double`	same as `double`	Real

Casting to Prevent Data Loss

```
int i = 7;
int j = 2;
int k = 0;
float f;

//through variables
k = i / j;          // k =
f = i / j;          // f =
f = (float)i / j; // f =

//direct numbers/literals
k = 7 / 2;          // k =
f = 7 / 2;          // f =
f = 7.0 / 2;        // f =
```

?

?

Casting to Prevent Data Loss

```
int i = 7;
int j = 2;
int k = 0;
float f;

//through variables
k = i / j;          // k = 3
f = i / j;          // f = 3.0
f = (float)i / j; // f = 3.5

//direct numbers/literals
k = 7 / 2;          // k = 3
f = 7 / 2;          // f = 3.0
f = 7.0 / 2;        // f = 3.5
```

Time Delay

- Want to have exact time differences or spacing between certain instructions
- Two methods:
  - Using a simple loop (for >> while)
  - Using PIC18 timer peripheral

Two Factors for Delay Accuracy

1. The crystal's frequency (int. or ext.)
   – Duration of clock period for instruction cycle
2. The compiler used for the C program
   – In ASM, we control the exact instructions
   – Different compilers produce different ASM code

Time Delay Example

Write a C18 program to toggle all the bits of Port B ports continuously with a 250 ms delay. Assume that the system is PIC18F458 with XTAL = 10 MHz.

```
#include <PIC18F452.h>
void MS_Delay(unsigned int);

void main(void)
{
    TRISB = 0;
    while(1)
    {
        PORTB = 0x55;
        MS_Delay(250);
        PORTB = 0xAA;
        MS_Delay(250);
    }
}

void MS_Delay(unsigned int msTime)
{
    unsigned int i;
    unsigned int j;

    for(i=0; i<msTime; i++)
        for(j=0; j<2500; j++);
            //NOP
}
```

F_{OSC} = 10 MHz = 10,000,000 cycles/sec

Each instruction takes 4 clock cycles (ticks)

F_{CY} = Instruction Cycle Frequency

$= \frac{10MHZ}{4}$ = 2.5MHz = 2,500,000 Ins/sec

T_{CY} = Instruction Cycle Time

= 1 / 2.5MHz = 0.0000004 sec per Ins

= 0.0004 ms = 0.4 µs

How many IC (instructions) fit into 1ms?

1ms / 0.0004ms = 2,500

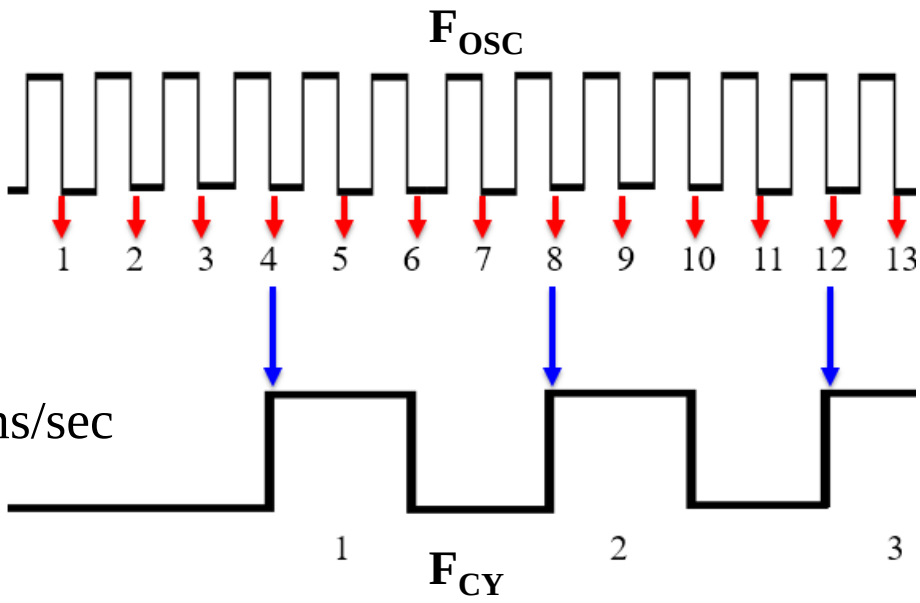
→ 2,500 Instruction Cycles take place in 1ms

→ 2,500 Instructions can complete in 1ms

Instruction Cycle

F_{OSC} = Oscillator Frequency

= 10 MHz = 10,000,000 cycles/sec



Each instruction takes 4 clock cycles (ticks)

F_{CY} = Instruction Cycle Frequency

$= \frac{F_{OSC}}{4} = \frac{10MHZ}{4}$ = 2.5MHz = 2,500,000 Ins/sec

T_{CY} = Instruction Cycle Time

$= \frac{1}{F_{CY}} = \frac{1}{2.5MHz}$ = 0.0000004 sec per Ins

= 0.0004 ms = 0.4 µs

How many IC (instructions) fit into 1ms?

1ms / 0.0004ms = 2,500

→ 2,500 Instruction Cycles take place in 1ms

→ 2,500 Instructions can complete in 1ms (generalizing since most instructions only take 1 Ins. Cycle)

PORT I/O Programming in C

- Btye-Size Register Access
  - Labels still the same
  - PORTA – PORTD
  - TRISA – TRISD
  - INTCON
- Bit-Addressable Register Access
  - PORTBbits.RB3
  - TRISCbits.RC7 or TRISCbits.TRISC7
  - INTCONbits.RBIE

PORT I/O Programming in C

```
TRISB = 0;                    //make Port B an output
TRISC = 0;                    //make Port C an output
PORTB = 00;                   //clear Port B
LED = 0;                      //clear Port C
for(;;)                       //repeat forever
  {
    PORTB++;                  //increment Port B
    LED++;                    //increment Port C
  }
```

```
unsigned char mybyte;
TRISB = 0xFF;                 //Port B as input
TRISC = 0;                    //Port C as output
while(1)
  {
    mybyte = PORTB;           //get a byte from Port B
    MSDelay(500);
    PORTC = mybyte;           //send it to Port C
  }
```

PORTxbits.Rxy

Table 7-2: Single-Bit Addresses of PIC18F458/4580 Ports

PORTA	PORTB	PORTC	PORTD	PORTE	Port's Bit
RA0	RB0	RC0	RD0	RE0	D0
RA1	RB1	RC1	RD1	RE1	D1
RA2	RB2	RC2	RD2	RE2	D2
RA3	RB3	RC3	RD3		D3
RA4	RB4	RC4	RD4		D4
RA5	RB5	RC5	RD5		D5
	RB6	RC6	RD6		D6
	RB7	RC7	RD7		D7

PORT I/O Programming in C

```
#include <P18F458.h>
void MSDelay(unsigned int);
#define Dsensor PORTBbits.RB1
#define buzzer PORTCbits.RC7
void main(void)
  {
    TRISBbits.TRISB1 = 1;          //PORTB.1 as an input
    TRISCbits.TRISC7 = 0;          //make PORTC.7 an output

    while(Dsensor == 1)
      {
        buzzer = 0;
        MSDelay(200);
        buzzer = 1;
        MSDelay(200);
      }
    while(1);                      //stay here forever
  }
```

Write a C18 program to get the status of bit RB0, and send it to RC7 continuously.

Solution:

```
#include <P18F458.h>
#define inbit PORTBbits.RB0
#define outbit PORTCbits.RC7
void main(void)
  {
    TRISBbits.TRISB0 = 1;          //make RB0 an input
    TRISCbits.TRISC7 = 0;          //make RC7 an output
    while(1)
      {
      outbit = inbit;              //get a bit from RB0
                                   //and send it to RC7
      }
  }
```

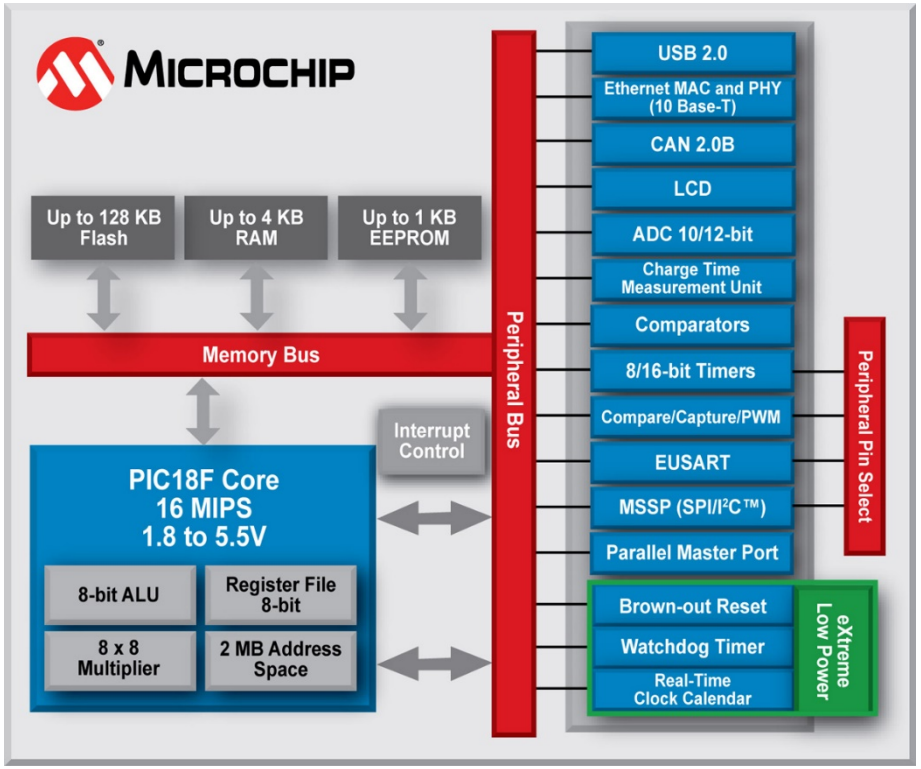
```
1:              #include <P18F458.h>
2:              #define inbit   PORTBbits.RB0
3:              #define outbit   PORTCbits.RC7
4:              void main(void)
5:                 {
6:                   TRISBbits.TRISB0 = 1;          //make RB0 an input
0000E2   8093        BSF 0xf93, 0, ACCESS
7:                   TRISCbits.TRISC7 = 0;          //make RC7 an output
0000E4   9E94        BCF 0xf94, 0x7, ACCESS
8:                   while(1)
0000F2   D7F9        BRA 0xe6
9:                      {
10:                        outbit = inbit;          //get bit from RB0
0000E6   5081              MOVF 0xf81, W, ACCESS
0000E8   0B01              ANDLW 0x1
0000EA   E002              BZ 0xf0
0000EC   8E82              BSF 0xf82, 0x7, ACCESS
0000EE   D001              BRA 0xf2
0000F0   9E82              BCF 0xf82, 0x7, ACCESS
11:                                                 //and send it to RC7
12:                     }
13:                  }
0000F4   0012    RETURN 0
```

Header Files

- Remember that certain register/variable names are not native C keywords
- They are PIC-specific
  - PORTB, TRISA, TMR0H, PRODL, etc.
- Defined and mapped in header file
  - Using regular data types (char, int, struct, etc.)
- Regular P18Fxxx.h (device) header files
  - C:\Program Files (x86)\Microchip\xc8\v1.20\include

Header Files

- Other functional headers are available
  - adc.h
  - delays.h
  - i2c.h
  - pwm.h
  - timers.h
  - usart.h



- Peripheral library Header Files
  - C:\Program Files (x86)\Microchip\xc8\v1.20\include\plib
  - C:\Program Files (x86)\Microchip\xc8\v1.20\sources\pic18\plib

Logic Operations in C

- Bit-Wise Operators

Table 7-3: Bit-wise Logic Operators for C

		AND	OR	EX-OR	Inverter
A	B	A&B	A|B	A^B	Y=~B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

- Bit-Wise Shift Operators
  - Can shift right/left by X bits

    Shift right >>

    Shift left <<

Logic Operations in C

```
TRISB = 0;                  //make Ports B, C,
TRISC = 0;                  //and D output ports
TRISD = 0;
PORTB = 0x35 & 0x0F;        //ANDing
PORTC = 0x04 | 0x68;        //ORing
PORTD = 0x54 ^ 0x78;        //XORing
PORTB = ~0x55;              //inverting
PORTC = 0x9A >> 3;          //shifting right 3 times
PORTD = 0x77 >> 4;          //shifting right 4 times
PORTB = 0x6 << 4;           //shifting left 4 times
while(1);                   //stay here forever
```

```
while(1)
   {
      PORTB = ~PORTB;
      PORTC = ~PORTC;
      MSDelay(250);
   }
```

Binary (hex) to Decimal and ASCII Conversion

- Sometimes we can't handle multiple-digit decimals natively in C for display purposes
- printf() is standard for generic C but requires more memory space than a PIC18 is willing to sacrifice
- Best to build your own "custom" print or display functions in C

Extract Single Decimal Digits

- Want each digit of 253 (0b11111101, 0xFD) and convert to ASCII for displaying

Extract Single Decimal Digits

- Want each digit of 253 (0b11111101, 0xFD) and convert to ASCII for displaying

```
1  unsigned char whole, part, d1, d2, d3;
2
3  whole = 253; //whole == d3_d2_d1
4
```

Hex	Dec	Char	Hex	Dec	Char
0x20	32	Space	0x40	64	@
0x21	33	!	0x41	65	A
0x22	34	"	0x42	66	B
0x23	35	#	0x43	67	C
0x24	36	$	0x44	68	D
0x25	37	%	0x45	69	E
0x26	38	&	0x46	70	F
0x27	39	'	0x47	71	G
0x28	40	(	0x48	72	H
0x29	41	)	0x49	73	I
0x2A	42	*	0x4A	74	J
0x2B	43	+	0x4B	75	K
0x2C	44	,	0x4C	76	L
0x2D	45	-	0x4D	77	M
0x2E	46	.	0x4E	78	N
0x2F	47	/	0x4F	79	O
0x30	48	0	0x50	80	P
0x31	49	1	0x51	81	Q
0x32	50	2	0x52	82	R
0x33	51	3	0x53	83	S
0x34	52	4	0x54	84	T
0x35	53	5	0x55	85	U
0x36	54	6	0x56	86	V
0x37	55	7	0x57	87	W
0x38	56	8	0x58	88	X
0x39	57	9	0x59	89	Y
0x3A	58	:	0x5A	90	Z

Extract Single Decimal Digits

- Want each digit of 253 (0b11111101, 0xFD) and convert to ASCII for displaying

```
1  unsigned char whole, part, d1, d2, d3;
2
3  whole = 253; //whole == d3_d2_d1
4
5  part = whole / 10;   //part = 253 / 10 = 25
6  d1 = whole % 10;     //d1   = 253 % 10 = 3
7  d2 = part % 10;      //d2   = 25  % 10 = 5
8  d3 = part / 10;      //d3   = 25  / 10 = 2
```

Hex	Dec	Char	Hex	Dec	Char
0x20	32	Space	0x40	64	@
0x21	33	!	0x41	65	A
0x22	34	"	0x42	66	B
0x23	35	#	0x43	67	C
0x24	36	$	0x44	68	D
0x25	37	%	0x45	69	E
0x26	38	&	0x46	70	F
0x27	39	'	0x47	71	G
0x28	40	(	0x48	72	H
0x29	41	)	0x49	73	I
0x2A	42	*	0x4A	74	J
0x2B	43	+	0x4B	75	K
0x2C	44	,	0x4C	76	L
0x2D	45	-	0x4D	77	M
0x2E	46	.	0x4E	78	N
0x2F	47	/	0x4F	79	O
0x30	48	0	0x50	80	P
0x31	49	1	0x51	81	Q
0x32	50	2	0x52	82	R
0x33	51	3	0x53	83	S
0x34	52	4	0x54	84	T
0x35	53	5	0x55	85	U
0x36	54	6	0x56	86	V
0x37	55	7	0x57	87	W
0x38	56	8	0x58	88	X
0x39	57	9	0x59	89	Y
0x3A	58	:	0x5A	90	Z

Extract Single Decimal Digits

- Want each digit of 253 (0b11111101, 0xFD) and convert to ASCII for displaying

```
1  unsigned char whole, part, d1, d2, d3;
2
3  whole = 253; //whole == d3_d2_d1
4
5  part = whole / 10;   //part = 253 / 10 = 25
6  d1 = whole % 10;     //d1   = 253 % 10 = 3
7  d2 = part % 10;      //d2   = 25  % 10 = 5
8  d3 = part / 10;      //d3   = 25  / 10 = 2
9
10 d1 = d1 + 48;        //or + 0x30
11 d2 = d2 + 48;        //or + 0x30
12 d3 = d3 + 48;        //or + 0x30
```

Hex	Dec	Char	Hex	Dec	Char
0x20	32	Space	0x40	64	@
0x21	33	!	0x41	65	A
0x22	34	"	0x42	66	B
0x23	35	#	0x43	67	C
0x24	36	$	0x44	68	D
0x25	37	%	0x45	69	E
0x26	38	&	0x46	70	F
0x27	39	'	0x47	71	G
0x28	40	(	0x48	72	H
0x29	41	)	0x49	73	I
0x2A	42	*	0x4A	74	J
0x2B	43	+	0x4B	75	K
0x2C	44	,	0x4C	76	L
0x2D	45	-	0x4D	77	M
0x2E	46	.	0x4E	78	N
0x2F	47	/	0x4F	79	O
0x30	48	0	0x50	80	P
0x31	49	1	0x51	81	Q
0x32	50	2	0x52	82	R
0x33	51	3	0x53	83	S
0x34	52	4	0x54	84	T
0x35	53	5	0x55	85	U
0x36	54	6	0x56	86	V
0x37	55	7	0x57	87	W
0x38	56	8	0x58	88	X
0x39	57	9	0x59	89	Y
0x3A	58	:	0x5A	90	Z

#define Directive

- Can associate labels with numbers or registers as a constant

```
#define LED_OUTPUT    PORTBbits.RB2
#define MAX_USERS     50
```