

Compilador PICs CC2

Variables.

Se permiten variables de 8-bit y 16-bit . Pueden ser definidas como globales o locales. La variable local sólo puede ser usada en su ambito

Ejemplo:

```
const char a = 10; // esta a es global

char fun( char a )
{
    return a+5;
}

main( )
{
    char b;
    b = 0;
    fun( b ); //after the call b will be equal to 5
}
```

No se aconseja el uso de variables de este modo. Recordar que una variable global puede ser modificada desde cualquier punto del programa.

Ejemplo:

```
void fun1( void )
{
    char a;
    a = 10;
}

void fun2( void )
{
    char b;
    b = 1;
    fun1(); //After the call b may be equal to 10
}
```

Las variables de 8 bits deben ser declaradas como **char**.

Las de 16 bit deben ser declaradas como **short, int** or **long**.

Ejemplo:

```
char x; //8bit variable
short y; //16bit variable
long z; //16bit variable, the same as short z;
```

Se permiten arrays de 1 dimension (vectores). Si se declara como **const** debe ser de tipo **char**.

Direccionamiento Absoluto

Una variable puede ser colocada en una dirección definida por el usuario. Para hacerlo se entepone el caracter '@' y la dirección después de la variable.

Ejemplo para el PIC:

```
char a@20; //Variable 'a' will be placed on address 20
char b@0x20; //Variable 'b' will be placed on address 32
```

Punteros

Solo se permiten punteros a caracter **const char ***, por ejemplo:

```
const char *txt = "This is a text";
output_port_b( txt[1] ); //the 'h' will be put into port b
```

Variables Internas

Para el PIC:

INDF	registro indf (0x00)
TMR0	clock/contador de 8 bits (0x01)
PCL	8 bits de menos peso del PC (0x02, 0x82)
STATUS	Registro de estado (0x03, 0x83)
FSR	Puntero de direccionamiento indirecto (0x04, 0x84)
PORTA	Puerto A(0x05)
PORTB	Puerto B(0x06)
EEDATA	Registro de datos de la EEPROM (0x08)
EEADR	Registro de direcciones de la EEPROM (0x09)
PCLATH	5 bits de mayor peso del PC (0x0A, 0x8A)
INTCON	Registro intcon (0x0b, 0x8b)
OPTION_REG	Registro de opciones (config.)(0x81)
TRISA	Config. del puerto A (0x85)
TRISB	Config. del puerto B (0x86)
EECON1	Registro de control de la EEPROM (0x88)
EECON2	Registro de control de la EEPROM (0x89)

Arrays (Vectores)

Se admiten sólo arrays de 1 dimensión y de tipo **char**:

Ejemplo:

```
char a[35], b[] = { 'A', 'B', 'C', 'D' };
```

Se soportan arrays constantes pero solo de tipo **char** y deben ser inicializados.

Ejemplo:

```
const char z[] = { 2, 'x', 0xFE, 012 };
```

Expresiones

Las expresiones de 8 bits pueden ser de cualquier complejidad

Ejemplo:

```
~b * 4 <= 8 != c++ - fun(5) / 2
```

Las expresiones de 16 bits deben ser **solo** de 2 operandos

Ejemplos:

```
a16 + b16
```

```
a16 & b16++
```

Cualquier expresión produce el resultado en una variable temporal.

Usar expresiones de 16 bits solo si es necesario pues requieren mucho más código que las de 8 bits

Las operaciones *, / y % están contempladas .

Funciones

Las funciones pueden devolver tipos **void, char, short, int o long**, y pueden admitir parámetros de tipo **void, const char*, char, short, int o long**.

Ejemplo:

```
char fun1( char p1, short p2, char p3 );
short fun2( void );
void printf( const char* );
```

Hay 2 funciones especiales, la **main ()** y para **interrupciones**. Las funciones internas pueden ser usadas para acceder a las características del micro.

Para poner una función en una dirección absoluta concreta usar **@ 0x<ADDR** después de la función.

Ejemplo:

```
//Esta funcion se situará en la direccion 0x200
void fun( void ) @ 0x200
{
...
}
```

Funciones especiales

main

Es la función principal del programa en C y debe declararse como en C++:

```
void main (void )
```

interrupt

No retorna valores ni acepta parámetros y se declara como:

```
void interrupt( void )
```

El código de la RSI estará separado del código del programa principal.

Funciones Internas

clear_wdt
enable_interrupt
disable_interrupt
set_mode
set_option
set_tris_a
set_tris_b
set_tris_c
output_port_a
output_port_b
output_port_c
output_high_port_a
output_high_port_b
output_high_port_c
output_low_port_a
output_low_port_b
output_low_port_c
input_port_a
input_port_b
input_port_c
input_pin_port_a
input_pin_port_b
input_pin_port_c
sleep
nop
set_bit
clear_bit
putchar
getchar
delay_s
delay_ms
delay_us
char_to_bcd
bcd_to_char

void enable_interrupt(NUMBER or MPASM predefined constant);

Habilita la interrupción indicada.

Ejemplo:
`enable_interrupt(GIE);`

void clear_wdt(void);

Resetea el perro guardián.

Ejemplo:
`clear_wdt();`

void disable_interrupt(NUMBER or MPASM predefined constant);

Deshabilita la interrupción indicada.

Ejemplo:
`disable_interrupt(TOIE);`

void set_mode(expression);

Inicializa el registro de modo. (sólo en el tipo sx).

Ejemplo:
`set_mode(10);`

void set_option(expression);

Inicializa el registro de opciones. (sólo en el tipo sx).

Ejemplo:
`set_option(0);`

```
void set_tris_a( expression );  
void set_tris_b( expression );  
void set_tris_c( expression );
```

Inicializa el registro TRIS-A,B,C.

Ejemplo:
`set_tris_a(a+b+c);`

```
void output_port_a( expression );  
void output_port_b( expression );  
void output_port_c( expression );
```

Escribe el valor de la expresión en el puerto A,B,C.

Ejemplo:
`output_port_a('A');`

```
void output_high_port_a( NUMBER );  
void output_high_port_b( NUMBER );  
void output_high_port_c( NUMBER );
```

Pone a 1 el pin del número indicado del puerto A,B,C.

Ejemplo:
`output_high_port_a(7);`

```
void output_low_port_a( NUMBER );  
void output_low_port_a( NUMBER );  
void output_low_port_a( NUMBER );
```

Pone a 0 el pin indicado del puerto A,B,C.

Ejemplo:
output_low_port_b(1);

```
char input_port_a( void );  
char input_port_a( void );  
char input_port_a( void );
```

Lee el puerto A,B,CInputs from porta.

Ejemplo:
while(input_port_a())

```
char input_pin_port_a( NUMBER );  
char input_pin_port_a( NUMBER );  
char input_pin_port_a( NUMBER );
```

Lee el pin indicado del puerto A,B,C.

Ejemplo:
if(input_pin_port_a(7))

void sleep(void);

Pone al micro en modo sleep.

Ejemplo:
`sleep();`

void nop(void);

Genera una operación vacía NOP.

Ejemplo:
`nop();`

void set_bit(VARIABLE, NUMBER or MPASM predefined constant);

Pone a 1 el bit indicado en una variable.

Ejemplo:
`set_bit(STATUS, RP0);`

void clear_bit(VARIABLE, NUMBER or MPASM predefined constant);

Pone a 0 el bit indicado en una variable.

Ejemplo:
`clear_bit(a, 1);`

void putchar(expression);

Escribe un carácter en el puerto RS232 . La configuración del RS232 puede ser realizada utilizando las [pragmas](#) :

RS232_TXPORT, RS232_TXPIN, RS232_BAUD, TRUE_RS232, CLOCK_FREQ,
TURBO_MODE.

Ejemplo:
`putchar('A');`

char getchar(void);

Lee un carácter desde el puerto RS232 port. No se retorna hasta que el carácter haya sido leído. La configuración del RS232 puede ser realizada utilizando las [pragmas](#) : RS232_RXPORT, RS232_RXPIN, RS232_BAUD, TRUE_RS232, CLOCK_FREQ, TURBO_MODE.

Ejemplo:
`a = getchar();`

void delay_s(expression);

Retardo de n segundos. La temporización puede ser configurada con los [pragmas](#) : CLOCK_FREQ, TURBO_MODE.

Ejemplo:
`delay_s(15); //delays for 15 seconds`

void delay_ms(expression);

Retardo de n milisegundos. La temporización puede ser configurada con los [pragmas](#) : CLOCK_FREQ, TURBO_MODE.

Ejemplo:
`delay_ms(100); //delays for 100 milliseconds`

void delay_us(expression);

Retardo en microsegundos. La temporización puede ser configurada con los [pragmas](#) : CLOCK_FREQ, TURBO_MODE.

Ejemplo:
`delay_us(d); //delays for d microseconds`

char char_to_bcd(expression);

Convierte un valor entre 0 y 99 a BCD.

Ejemplo:

```
output_port_b( char_to_bcd( 10 ) ); //write 10(bcd) into port B
```

char bcd_to_char(expression);

Convierte un valor BCD a decimal.

Ejemplo:

```
a = bcd_to_char( input_port_b() ); //reads port B as a BCD number and converts it into decimal
```

C estandard

- if, else, while, for, return, break, continue, extern, switch, case, default;
- goto and labels;
- char, short, int, long, void;
- ~, ++, --, +, -, <, <=, =, ==, !=, =, !, &, |, ^, &=, |=, ^=, &&, ||, *, /, %, <<, , <<=, =;
- one-dimensional arrays;
- const char pointers;
- const variables and arrays;
- functions with no/one/many parameters and void/char return type;
- built-in assembler;
- #include
- #define, #undef
- #ifdef, #ifndef, #else, #endif

Código ensamblador insertado

Ejemplo:

```
...
char a; //a global variable
...
void fun1( char a )
{
    ...
}
...
void fun2( void )
{
    char b; //a local variable
    ...
    //The code below is equal to b = 5;
    asm movlw 5
    asm movwf _b_fun2
    ...
    //The code below is equal to fun1( a );
    asm
    {
        movf _a, W
        movwf param00_fun1
        call _fun1
    }
    ...
}
...
```

Preprocesador

directivas soportadas:

#include "file_name" #include <file_name	The directive inserts the contents of the file specified by file_name into the current file.
#define identifier #define identifier subs_text	The directive replaces all subsequent cases of identifier with the subst_text.
#ifdef identifier	The directive tests whether identifier is currently defined.
#ifndef identifier	The directive tests whether identifier is currently undefined.
#else	
#endif	
#undef identifier	The directive removes the current definition of identifier.

The predefined identifiers:

C2C	is always defined
_EXT_VERSION_	defined in extended version
_PIC	defined if PIC target is selected
_SX	_SX defined if SX target is selected

Pragmas

Se utilizan para definir algunos valores especiales que se necesitan en la generación de código, como por ejemplo la generación de frecuencias de reloj:

#pragma RS232_RXPORT <num>	Port used to receive RS232 data. Default value is PORTA.
#pragma RS232_TXPORT <num>	Port used to transmit RS232 data. Default value is PORTA.
#pragma RS232_RXPIN <num>	Pin used to receive RS232 data. Default value is 1.
#pragma RS232_TXPIN <num>	Pin used to transmit RS232 data. Default value is 2.
#pragma RS232_BAUD <num>	RS232 baudrate. Default value is 9600bps.
#pragma TRUE_RS232 <num>	Voltage level for '1' and '0' is SR232 communication. Default value is 1 (high level for '1' and low level for '0')
#pragma CLOCK_FREQ <num>	Processor clock frequency in Hz. Default value for PIC is 4000000 and for SX 50000000.
#pragma TURBO_MODE <num>	Processor mode. Used only for SX (ignored for PIC). Default value is 1.

Limitaciones del compilador

- Las funciones no trabajan con operandos de 16 bits.
- Una expresión de 16 bits solo puede tener 2 operandos.
- Sólo auto arrays tipo char;
- Sólo constantes tipo const char;
- Un array tipo const array es como máximo de 256 bytes;
- El máximo numero de parámetros en una función es 32.