C2C help index.

(valid from the compiler version 3.27)

Variables.

8-bit and 16-bit variables are supported. They can be defined as global or local ones. Local variables can have unique addresses or will share the same address space.
Example:

```
const char a = 10;

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

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

Note by choosing the second option be very careful. A call of the function which uses local variables may change the values of the local variables in the calling function. This option may be useful if local variables are used before any calls (for example as counters).
Example:

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

void fun2( void )
{
```

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

Variable can be declared as const. It will be stored
in the program area.

The 8 bit long variables are should be declared as char. The 16 bit long variables may be declared as short, int or long.
Example:

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

One-dimensional arrays are supported. If an array is declared as const it can have only char type.

Built-in variables are supported. They are placed into program space.

Absolute addresses

A variable can be placed on a user defined absolute address. To do this add a character '@' and the address after the variable name. For the SX target the address is calculated as

*<address> + <bank_number> * 16*

Example for the PIC target:

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

Example for the SX target:

```
char  a@32;   //Variable 'a' will be placed on address 16
          //bank 1 ( 32 = 16 + 1 * 16 )
char  b@0x31; //Variable 'b' will be placed on address 17
          //bank 2 ( 0x31 = 49 = 17 + 2 * 16 )
```

Index

Pointers

Only const char * pointers are supported now as shown in the example below:

```
const char *txt = "This is a text";
```

output_port_b(txt[1]); //the 'h' will be put into port b

Index

Built-in variables

For PIC target:

INDF	indf register(0x00)
TMR0	8-bit real-time clock/counter(0x01)
PCL	low order 8 bits program counter(0x02, 0x82)
STATUS	status register(0x03, 0x83)
FSR	indirect data memory address pointer(0x04, 0x84)
PORTA	port A(0x05)
PORTB	port B(0x06)
EEDATA	EEPROM data register(0x08)
EEADR	EEPROM address register(0x09)
PCLATH	high order 5 bits program counter(0x0A, 0x8A)
INTCON	intcon register(0x0b, 0x8b)
OPTION_REG	option register(0x81)
TRISA	port A data direction register(0x85)
TRISB	port B data direction register(0x86)
EECON1	EEPROM control register(0x88)
EECON2	EEPROM control register(0x89)

For SX target:

RTCC	real-time clock/counter
PC	program counter
STATUS	status register
FSR	file select register
RA	port A
RB	port B
RC	port C
INDF	indf register

Index

Arrays

one-dimensional arrays are supported. They can have only char type.

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

Const arrays are supported. They can have only char type.They have to be initialised.

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

Index

Expressions

8 bit expressions can be as complex as needed.

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

16 bit expressions usually can have only 2 operands.

Examples:
a16 + b16
a16 & b16++

Note that a complex expression will produce temporary variables.

Use 16 bit variables only if you really need them. They need much mode code than 8 bit variables.

The operations *, / and % are supported . The implementation
can be defined as a function or inline code.

Index

Functions

Functions can have *void*, *char*, *short*, *int* or *long* return types.
They can have *void*, *const char** or several *char*, *short*, *int* or *long* parameters.
Example:

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

If the return type is not specified it is char.
Example:

fun(char a); //is the same as char fun(char a);

If the parameter is not specified it is void.
Example:

fun(); //is the same as char fun(void);

There are two special functions for entry point and interrupt.
Built-in functions can be used to access micro controller features.

To place a function on an absolute address place *@ 0x<ADDR>* after it.
Example:

```
//This function will be placed started from 0x200
void fun( void ) @ 0x200
{
…
}
```

Index

Special functions

The special functions are main and interrupt.

main

It is the entry point and should be declared as:

void main(void)

It is called after global variables initialisation.
If main presents in the code some prefix and postfix
code will be added to asm file.
If main is not present only variable table and code will
be produced for asm file. This is done in case the generated
asm file is included into the other asm file.

The main code has separate from the interrupt code temporary variables.

interrupt

It is the entry point for interrupt. It has no
return value and no parameters and must be declared as:

void interrupt(void)

If no interrupt function is declared there is an option to
add a default one or do not add any.

The interrupt code has separate from the main code temporary variables.

Index

Built-in functions

void enable_interrupt(NUMBER or MPASM predefined constant);

Enables the relevant interrupt .

Example:
enable_interrupt(GIE);

Index

void clear_wdt(void);

Resets the watchdog timer.

Example:
clear_wdt();

Index

void disable_interrupt(NUMBER or MPASM predefined constant);

Disables the relevant interrupt.

Example:
disable_interrupt(T0IE);

Index

void set_mode(expression);

Sets the mode register by the expression result (only for the SX target).

Example:
set_mode(10);

Index

void set_option(expression);

Sets the option register by the expression result (only for the SX target).

Example:
set_option(0);

Index

void set_tris_a(expression);

Sets the trisa register by the expression result.

Example:

set_tris_a(a>=10);

Index

void set_tris_b(expression);

Sets the trisb register by the expression result.

Example:
set_tris_b(a+b+c);

Index

void set_tris_c(expression);

Sets the trisc register by the expression result.

Example:
set_tris_c('A');

Index

void output_port_a(expression);

Puts the expression result into porta.

Example:
output_port_a('A');

Index

void output_port_b(expression);

Puts the expression result into portb.

Example:
output_port_b(0xff);

Index

void output_port_c(expression);

Puts the expression result into portc.

Example:
output_port_c(0xff);

Index

void output_high_port_a(NUMBER);

Sets a pin of porta.

Example:
output_high_port_b(7);

Index

void output_high_port_b(NUMBER);

Sets a pin of portb.

Example:
output_high_port_b(6);

Index

void output_high_port_c(NUMBER);

Sets a pin of portc.

Example:
output_high_port_c(0);

Index

void output_low_port_a(NUMBER);

Clears a pin of porta.

Example:
output_low_port_b(1);

Index

void output_low_port_b(NUMBER);

Clears a pin of portb.

Example:
output_low_port_b(0);

Index

void output_low_port_c(NUMBER);

Clears a pin of portc.

Example:
output_low_port_c(04);

Index

char input_port_a(void);

Inputs from porta.

Example:
while(input_port_a())

Index

char input_port_b(void);

Inputs from portb.

Example:
a = input_port_b();

Index

char input_port_c(void);

Inputs from portc.

Example:
x = input_port_c();

Index

char input_pin_port_a(NUMBER);

Inputs from pin of porta.

Example:
if(input_pin_port_a(7))

Index

char input_pin_port_b(NUMBER);

Inputs from pin of portb.

Example:
a = input_pin_port_b(0);

Index

char input_pin_port_c(NUMBER);

Inputs from pin of portb.

Example:
a = input_pin_port_c(5);

Index

void sleep(void);

Puts the micro controller into sleep mode.

Example:
sleep();

Index

void nop(void);

Generates an empty instruction.

Example:
nop();

Index

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

Sets a bit in variable.

Example:
set_bit(STATUS, RP0);

Index

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

Clears a bit in variable.

Example:
clear_bit(a, 1);

Index

void putchar(expression);

Writes a character into RS232 port. The RS232 parameters could be setup using pragmas : RS232_TXPORT, RS232_TXPIN, RS232_BAUD, TRUE_RS232, CLOCK_FREQ, TURBO_MODE.

Example:
putchar('A');

Index

char getchar(void);

Reads a character from RS232 port. Doesn't return till a character is read. The RS232 parameters could be setup using pragmas :
RS232_RXPORT, RS232_RXPIN, RS232_BAUD, TRUE_RS232, CLOCK_FREQ, TURBO_MODE.

Example:
a = getchar();

Index

void delay_s(expression);

Delays for the given number of seconds. The timing parameters could be setup using pragmas : CLOCK_FREQ, TURBO_MODE.

Example:
delay_s(15); //delays for 15 seconds

Index

void delay_ms(expression);

Delays for the given number of milliseconds. The timing parameters could be setup using pragmas : CLOCK_FREQ, TURBO_MODE.

Example:
delay_ms(100); //delays for 100 milliseconds

Index

void delay_us(expression);

Delays for the given number of microseconds. The timing parameters could be setup using pragmas : CLOCK_FREQ, TURBO_MODE.

Example:
delay_us(d); //delays for d microseconds

Index

char char_to_bcd(expression);

Converts the expression result into a BCD number and returns it. The expression result must be between 0 and 99.

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

Index

char bcd_to_char(expression);

Converts the expression result as a BCD value into a decimal number and returns it. The expression result must a valid BCD number.

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

Index

Standard C

- 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

Index

Built-in assembler

Assembler code can be included into c-code with asm operator.
C- variables or call c-functions calls can be referred to. To refer to them put underscore in front of variable/function name. In case a variable is local add underscore and function name after its

name.

Example:

```
...
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
   }
   ...
}
...
```

To produce assembler lines started from the left column (for example to produce labels) in the output .asm file add ‘:’ after the ‘asm’ keyword.

Example:

```
asm: myLabel
asm:
{
 myOtherLabel
}
```

Page instructions

Programming for the SX target there is no need to use 'page' instructions into the built-in assembler. These instructions will be inserted by the compiler.

Bank instructions

If a c-variable is used in the built-in assembler for the SX target the 'bank' instruction should be inserted in the beginning of such code. The compiler will remove these instructions if they are unnecessary.

Example:
char b;
...
//The code below is equal to b = 5;
asm mov w, #5
asm bank _b
asm mov _b, w

Index

Libraries

The compiler can generate and consume libraries.

The library stores all conditional directives and all code from the files the library was built from.

Index

Preprocessor

The supported preprocessor directives are listed below:

#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

Index

Pragmas

The pragmas are used to define some special values needed for code generation like for example clock frequency which is used in delay function generation.The supported pragmas are listed below:

#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.

Index

Data Types

The compiler uses the following data types:

Data Type	**Syntax**	**Example**
Decimal	XXXXX	1563
Octal	0XXXX	0234
Hexadecimal	0xXXXX	0xFFA8
Binary	XXXXXb	10000001b
ASCII	'X'	'S'

Index

Limitations

The compiler has the following limitations:
- function calls usually can not be used in expressions which contain 16bit operands;
- a 16bit expression usually can have only 2 operands;
- only one type of auto arrays is supported: char;
- only one constant type is supported: const char;
- the maximum const array size is less than 256 bytes;
- the maximum function parameter number is 32.

These limitations are ranged. The ones in the beginning probably will be solved in the coming compiler

releases.

Index