aide C2C.

(valide pour la version 3.26)

Variables.

Les variables 8-bit et 16-bit sont supportées. On peut les définir comme globales ou locales.

Exemple:

```
const char a = 10;

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

main()
{
   char b;
   b = 0;
   fun( b );  //après l'appel, b vaudra 5
}
```

Remarquez qu'en choisissant la seconde option, vous devez être prudent. Un appel de la fonction qui utilise les variables locales peut changer les valeurs de ces dernières dans la fonction appelée. Cette option peut être utile si les variables locales ont utilisés un appel auparavant (par exemple pour un compteur).

Exemple:

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

void fun2( void )
{
   char b;
   b = 1;
   fun1();  //Après l'appel,b devrait être égal à 10
}
```

Si les variables sont déclarées comme constantes, elles seront stockées dans la zone de mémoire programme.Les variables de 8-bit doivent être déclarées en tant que char. Les variables de 16-bit doivent être déclarées en tant que short, int ou long.

Exemple:

```
char  x;   //variable 8bit
short y;  //variable 16bit
long  z;  //variable 16bit, le même que short z;
```

Les Tableaux à une dimension sont supportés. Si un tableau est déclaré comme une constante, seuls les types char sont acceptés.

Variables intégrées supportées. Elles sont stockées dans la mémoire programme.

Absolute addresses

Une variable peut être placée dans une adresse absolue définie par l'utilisateur. Pour ce faire, ajouter un caractère "@" et l'adresse après le nom de la variable.

Exemple pour les PIC::

char a@20; //La variable 'a' sera placé à l'adresse 20
char b@0x20; //La variable 'b' sera placé à l'adresse 32

Pointeurs

Seuls const char * pointeurs sont supportés, comme le montre l'exemple ci-dessous:

*const char *txt = "Un texte";*
output_port_b(txt[1]); //Le 'n' sera envoyé au port b

Variables intégrées

Pour PIC:

INDF	registre indf(0x00)
TMR0	horloge/compteur 8-bit temps réel(0x01)
PCL	compteur programme 8bit partie basse(0x02, 0x82)
STATUS	registre statut(0x03, 0x83)
FSR	pointeur d'adresse indirecte(0x04, 0x84)
PORTA	port A(0x05)
PORTB	port B(0x06)
EEDATA	EEPROM data register(0x08)
EEADR	EEPROM address register(0x09)
PCLATH	compteur programme 5bit partie haute(0x0A, 0x8A)
INTCON	registre intcon(0x0b, 0x8b)
OPTION_REG	registre option(0x81)
TRISA	registre de direction de donnée du port A(0x85)
TRISB	registre de direction de donnée du port B(0x86)
EECON1	registre de contrôle EEPROM(0x88)
EECON2	registre de contrôle EEPROM(0x89)

Tableaux

Les tableaux à une dimension sont supportés. Seuls les types char sont acceptés.

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

Les tableaux à constantes sont supportés. Seuls les types char sont acceptés.Ils doivent être initialisés.

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

Expressions

Les expressions 8 bits peuvent être complexes si besion est.

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

Les expressions 16 bits peuvent contenir jusqu'à 2 opérations.

Exemples:
a16 + b16
a16 & b16++

NB : les complexes créent une variables temporaires.

Utilisez les variables 16 bits en dernier recours. Elles demandent plus de codes que les variables 8 bits.

Les opérations *, / et % sont supportées. L'implémentation peut être définis comme une fonction ou un code en ligne.

Fonctions

Les fonctions peuvent comprendre les types *void*, *char*, *short*, *int* ou *long*.
Ils peuvent comprendre *void*, *const char** ou quelques paramètres *char*, *short*, *int* ou *long*.
Exemple:

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

Si le type de retour n'est pas spécifié, par défaut char est sélectionné.
Exemple:

```
fun( char a ); //est le même que char fun( char a );
```

Si le paramètre n'est pas spécifié, par défaut void est sélectionné.
Exemple:

```
fun(); //est le même que char fun( void );
```

Il y a deux fonctions spéciales pour les interruptions et le point d'entrée.
Les fonctions intégrées peuvent être utilisées pour accéder aux fonctions du micro-controleur.

Pour placer une fonction sur une adresse absolue, ajouter *@ 0x<ADDR>* après la fonction.
Exemple:

```
//Cette fonction sera placée à partir de 0x200
void fun( void ) @ 0x200
{
...
}
```

Fonctions spéciales

Les fonctions spéciales sont *main* et *interrupt*.

main

C'est le point d'entrée et il doit être déclaré comme suit:

```
void main( void )
```

Il est utilisé après une initialisation de variables globales.
Si *main* comprend dans son code quelques préfixes et suffixes
Ce code devra être ajouté au fichier *asm*.
Si *main* n'est pas présent, seuls les tables de variables et les codes devront
être créés dans le fichier *asm*. Ceci fait, le fichier *asm*
généré est inclus dans les autres fichiers *asm*.

interrupt

C'est le point d'entrée pour *interrupt*. Aucunes valeurs
et paramètres ne sont retournés et il doit être déclaré comme suit:

void interrupt(void)

Si aucune fonction n'est déclarée, une option permet d'en déclarer une par défaut.

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

Fonctions intégrées

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);

Rend actif l'interruption ciblée .

Exemple:
enable_interrupt(GIE);

void clear_wdt(void);

Réinitialise le chien de garde.

Exemple:
clear_wdt();

void disable_interrupt(NOMBRE ou MPASM prédéfini);

Désactive l'interruption ciblée.

Exemple:
disable_interrupt(T0IE);

void set_tris_a(expression);

Ecrit l'expression dans le registre trisa.

Exemple:
set_tris_a(0x0);

void set_tris_b(expression);

Ecrit l'expression dans le registre trisb.

Exemple:
set_tris_b(0x0);

void set_tris_c(expression);

Ecrit l'expression dans le registre trisc.

Exemple:
set_tris_c(0x0);

void output_port_a(expression);

Place l'expression dans le Port a.

Exemple:
output_port_a('A');

void output_port_b(expression);

Place l'expression dans le Port b.

Exemple:
output_port_b(0xff);

void output_port_c(expression);

Place l'expression dans le Port c.

Exemple:
output_port_c(0xff);

void output_high_port_a(NOMBRE);

Positionne à l'état haut le bit du porta spécifié.

Exemple:
output_high_port_a(7);

void output_high_port_b(NOMBRE);

Positionne à l'état haut le bit du portb spécifié.

Exemple:
output_high_port_b(6);

void output_high_port_c(NOMBRE);

Positionne à l'état haut le bit du portc spécifié.

Exemple:
output_high_port_c(0);

void output_low_port_a(NOMBRE);

Positionne à l'état bas le bit du porta spécifié.

Exemple:
output_low_port_b(1);

void output_low_port_b(NOMBRE);

Positionne à l'état bas le bitdu portb spécifié.

Exemple:
output_low_port_b(0);

void output_low_port_c(NOMBRE);

Positionne à l'état bas le bit du portc spécifié.

Exemple:
output_low_port_c(04);

char input_port_a(void);

Questionne le porta.

Exemple:
while(input_port_a())

char input_port_b(void);

Questionne le portb.

Exemple:
a = input_port_b();

char input_port_c(void);

Questionne le portc.

Exemple:
x = input_port_c();

char input_pin_port_a(NOMBRE);

Questionne une patte du porta.

Exemple:
if(input_pin_port_a(7))

char input_pin_port_b(NOMBRE);

Questionne une patte du portb.

Exemple:
a = input_pin_port_b(0);

char input_pin_port_c(NOMBRE);

Questionne une patte du portc.

Exemple:
a = input_pin_port_c(5);

void sleep(void);

Met le micro controlleur en mode veille.

Exemple:
sleep();

void nop(void);

Génère une instruction vide.

Exemple:
nop();

void set_bit(VARIABLE, NOMBRE ou MPASM prédéfini);

Met à 1 un bit dans une variable.

Exemple:
set_bit(STATUS, RP0);

Gossaire

void clear_bit(VARIABLE, NOMBRE or MPASM prédéfini);

Met à 0 un bit dans une variable.

Exemple:
clear_bit(a, 1);

void putchar(expression);

Ecrit un caractère dans le port RS232. Les paramètres RS232 peuvent être configurés en utilisant pragmas :
RS232_TXPORT, RS232_TXPIN, RS232_BAUD, TRUE_RS232, CLOCK_FREQ, TURBO_MODE.

Exemple:
putchar('A');

char getchar(void);

Lit un caractère sur le port RS232. Il n'y a aucun retour tant qu'un caractère est lu. Reads a character from RS232 port. Les paramètres RS232 peuvent être configurés en utilisant pragmas :
RS232_RXPORT, RS232_RXPIN, RS232_BAUD, TRUE_RS232, CLOCK_FREQ, TURBO_MODE.

Exemple:
a = getchar();

void delay_s(expression);

Génère un retard en seconde. Les paramètres peuvent être configurés en utilisant les pragmas :
CLOCK_FREQ, TURBO_MODE.

Exemple:
delay_s(15); //délai de15 secondes

void delay_ms(expression);

Génère un retard en milliseconde. Les paramètres peuvent être configurés en utilisant les pragmas :
CLOCK_FREQ, TURBO_MODE.

Exemple:
delay_ms(100); //délai de 100 millisecondes

void delay_us(expression);

Génère un retard en microseconde. Les paramètres peuvent être configurés en utilisant lespragmas :
CLOCK_FREQ, TURBO_MODE.

Exemple:
delay_us(n); //délai de n microsecondes

char char_to_bcd(expression);

Convertit le résultat en un nombre BCD et le retourne. Le résultat est compris entre 0 et 99.

Exemple:
output_port_b(char_to_bcd(10)); //écrit 10(en bcd) dans le port B

char bcd_to_char(expression);

Convertit un résultat comme une valeur BCD en un nombre décimal et le retourne. Le résultat doit être un nombre BCD valide.

Exemple:
a = bcd_to_char(input_port_b()); //lit dur le port B un nombre BCD et le convertit en décimal

C standard

· if, else, while, for, return, break, continue, extern, switch, case, default;
· goto et labels;
· char, short, int, long, void;
· ~, ++, --, +, -, <, <=, >, >=, ==, !=, =, !, &, |, ^, &=, |=, ^=, &&, ||, *, /, %, <<, >>, <<=, >>=;
· tableaux à une dimension;
· pointeurs const char;
· tableaux et variables constants;
· fonctions avec aucun/un/plusieurs paramètres et type de retour void/char;
· assembleur intégré;
· #include
· #define, #undef
· #ifdef, #ifndef, #else, #endif

Assembleur intégré

Le code assembleur peut être inclus dans un code C avec un opérateur *asm*.
Les variables C peuvent être directement utilisées dans le code assembleur. Pour se référrer à celles-ci, on ajoute un 'underscore' devant le nom de la variable ou de la fonction.

Exemple:

```
...
char a;  //a, variable globale
...
void fun1( char a )
{
   ...
}
...
void fun2( void )
{
   char b; //a, variable locale
   ...
   //Le code en-dessous est équivalent à b = 5;
   asm movlw 5
   asm movwf _b_fun2
   ...
   //Le code en-dessous est équivalent à fun1( a );
   asm
   {
      movf _a, W
      movwf param00_fun1
      call _fun1
   }
   ...
}
...
```

Pour produire des lignes en assembleur, commencez par la colonne de gauche (par exemple : les étiquettes) dans le fichier *asm* et ajoutez ‘:’ après le mot de passe ‘asm’.

Exemple:

```
asm: monLabel
asm:
{
 monAutreLabel
}
```

Librairies

Le compilateur peut généré et utilisé des librairies.

La librairie stocke toutes les directives et tous les codes du fichier où elle a été créée.

Preprocessor

Les directives supportées par le préprocesseur sont réunies ci-dessous:

#include "nom_de_fichier" #include <nom_de_fichier>	La directive insère le contenu du fichier spécifié par nom_de_fichier dans le fichier courant.
#define identificateur #define identificateur subs_text	La directive remplace toutes les cases subséquentes de identificateur avec subst_text.
#ifdef identifier	La directive teste l'identificateur où il est actuellement défini.
#ifndef identifier	La directive teste l'identificateur où il est actuellement non défini.
#else	
#endif	
#undef identifier	La directive retire la définition actuelle de l'identificateur.

Les identifieurs prédéfinis:

_C2C_	est toujours défini
_EXT_VERSION_	défini dans la version étendue
_PIC	défini si la cible du PIC est sélectionnée

Pragmas

Les pragmas sont utilisées pour définir quelques valeurs spéciales demandées pour la génération de code, de préférence pour les horloges qui utilisent une fonction delai. Les pragmas supportées sont réunies ci-dessous:

#pragma RS232_RXPORT <num>	Port utilisé pour recevoir les données RS232. La valeur par défaut est PORTA.
#pragma RS232_TXPORT <num>	Port utilisé pour transmettre des données RS232. La valeur par défaut est PORTA.
#pragma RS232_RXPIN <num>	Patte utilisée pour recevoir des données RS232. La valeur par défaut est 1.
#pragma RS232_TXPIN <num>	Patte utilisée pour transmettre des données RS232. La valeur par défaut est 2.
#pragma RS232_BAUD <num>	débit de baud RS232 . La valeur par défaut est 9600bps.
#pragma TRUE_RS232 <num>	Tension pour les niveaux '1' et '0' de la communication RS232. La valeur par défaut est 1(niveau haut pour '1' et niveau bas pour '0')
#pragma CLOCK_FREQ <num>	Fréquence d'horloge du processeur en Hz. La valeur par défaut pour PIC est 4000000

Limitations

Le compilateur possède quelques limitations:
· L'appel de fonctions ne peut être utilisé dans une expression qui contient des opérateurs 16 bits;
· une expression 16 bits peut avoir seulement deux opérateurs;
· un seul type de tableau automatique est supporté: char;
· un seul type de constante est supporté: const char;
· la dimension maximale d'un tableau de constantes doit être inférieure à 256 octets;
· le nombre de paramètres maximum pour une fonction est de 32.