Chapter 3 - Programming in C

Since the heart of an embedded control system is a microcontroller, we need to be able to develop a program of instructions for the microcontroller to use while it controls the system in which it is embedded. When programs are developed on the same type of computer system on which they will be run, as is most commonly done, it is called *native platform development.* An example of native platform development is the use of *Borland's C/C++* to develop a program on an *Intel* PentiumII-based computer such as an *IBM*-compatible PC, and subsequently running the program on the same computer.

However, the type of program development which you will be doing in this course is known as *cross platform development*, where your laptop computer (one platform) is used to develop programs which are targeted to run on the *SiLabs* C8051F020 EVB (another platform). Thus even before such a program can be tested, it must be transmitted (downloaded) from the computer to the EVB.

For this embedded microcontroller, we will be using a programming language called 'C'. C is extremely flexible, and allows programmers to perform many low-level functions which are not easily accessible in languages like FORTRAN or Pascal. Unfortunately, the flexibility of C also makes it easier for the programmer to make mistakes and potentially introduce errors into their program. To avoid this, you should be very careful to organize your program so that it is easy to follow, with many comments so that you and others can find mistakes quickly. The example programs were written with this in mind, so that you get an idea of what well structured programs look like. In order to get you started in C programming, this chapter will explain the basics which you will need to begin. If you would like more examples, they can be found in the *LITEC Tutorials* under *Software: C Programming.*

Brief Overview

As stated above, the C language allows many things which other languages do not, making it very easy to make errors. For this reason, it is suggested that you study this entire unit, including the tutorials if necessary, and become familiar with the specifics before beginning the labs. Although this may seem like a lot of work before the first lab, it will be worth your time and will pay off quickly. If there is a question you have regarding the C language that is not included in the manual, or in the tutorials, you will probably find the answer in a C reference text.

A Simple Program in C

The following program is similar to the first programming example used in most C programming books and illustrates the most basic elements of a C program.

```
#include <stdio.h>  /* include file */

main()      /* begin program here */
{           /* begin program block */

      printf("Hello World\n");

            /* send Hello World
               to the terminal */

}           /* end the program block */
```

The first line instructs the compiler to include the *header file* for the standard input/output functions. This line indicates that some of the functions used in the file (such as printf) are not defined in the file, but instead are in an external library (stdio.h is a standard library header file). This line also illustrates the use of comments in C. Comments begin with the two character sequence "`/*`" and end with the sequence "`*/`". Everything between is ignored and treated as comments by the compiler. Nested comments are not allowed.

The second line in the program is `main()`. Every C program contains a function called `main()` which is the function that executes first. The next line is a curly bracket. Paired curly brackets are used in C to indicate a *block* of code. In the case above, the block belongs to the `main()` statement preceding it.

The `printf` line is the only statement inside the program. In C, programs are broken up into *functions*. The `printf` function sends text to the terminal. In our case, the C8051 will send this text over the serial port to a "computer terminal", where we can view it. (You will use software on your laptop to simulate a terminal.) This line also illustrates that every statement in C is followed by a semicolon. The compiler interprets the semicolon as the end of one statement, and then allows a new statement to begin.

You may also notice that the comment after the `printf` statement continues over more than one line. It is important to remember that *everything* between the comment markers is ignored by the compiler. (see *Specifics of the SDCC Compiler*)

The last line is the closing curly bracket which ends the block belonging to the main function. More examples can be found in the tutorials under the C examples section.

Syntax Specifics

There are many syntax rules in C, but there is neither room nor time here to discuss everything in this manual. Instead, we explain the basics, along with the specifics of the *SDCC C Compiler* which are not in your textbook. Additional information about the C language can be found in the tutorials, and in any C reference text.

Declarations

One thing which was distinctly missing from the first example program was a variable. The type of variables available with the *SDCC C Compiler* for the C8051 microcontroller and their declaration types are listed below in *Table 3.1*:

Table 3.1 - SDCC C Compiler variable types

Type[a]	Size (bytes)	Smallest Value	Largest Value
integer			
(unsigned) char	1	0	255
signed char	1	-128	127
(signed) short	2	-32768	32767
unsigned short	2	0	65535
(signed) int	2	-32768	32767
unsigned int	2	0	65535
(signed) long	4	-2147483648	2147483647
unsigned long	4	0	4294967295
floating point			
float	4	1.2×10^{-38}	1.2×10^{38}
SDCC specific			
bit	1/8 = 1 bit	0	1
sbit	1/8 = 1 bit	0	1

a. The items in parentheses are not required, but are implied by the definition. We recommend that you state these definitions explicitly to avoid errors due to misdefinition.

The format for declaring a variable in a C program is as follows:

```
<type> variablename;
```

For example, the line

```
int i;
```

would declare an integer variable `i`. Although there are a large variety of variable types available, it is important to realize that the larger the size of the data type, the more time will be required by the C8051 to make the calculations. Increased calculation time is also an important consideration when using floating-point variables. It is suggested that in the interest of keeping programs small and efficient, you should not use floating point numbers unless absolutely necessary.

Repetitive Structures

Computers are very useful when repeating a specific task, and almost every program utilizes this capability. The repetitive structures `for`, `while`, and `do..while` are all offered by C.

for Loops

The most common of looping structures is the `for` loop, which looks like this

```
for(initialize_statement; condition; increment){
   ...
}
```

In the example above, the `for` loop will perform the "`initialize_statement`" one time before commencing the loop. The "`condition`" will then be checked to make sure that it is true (non-zero). As long as the "`condition`" is true the statements within the loop block will be performed. After each iteration of the loop, the increment statement is performed. For example:

```
for(i=0; i<10; i++) {
   display(i);
}
```

The statement above will initially set `i` equal to zero, and then call a user-defined function named `display()`10 times. Each time through the loop, the value of `i` is incremented by one. After the tenth time through, `i` is set to 10, and the for loop is ended since `i` is not less than ten.

while Loops

Another frequently used loop structure is the `while` loop, which follows this format

```
while(condition){
   ...
}
```

When a `while` loop is encountered, the condition given in parenthesis is evaluated. If the condition is true (evaluates to non-zero), the statements inside the braces are executed. After the last of these statements is executed, the condition is evaluated again, and the process is repeated. When the condition is false (evaluates to zero), the statements inside the braces are skipped over, and execution continues after the closing brace. As an example, consider the following:

```
i = 0;
while (i<10) {
   i++;
   display(i);
}
```

The above `while` loop will give the same results as the preceding example given with the `for` loop. The variable `i` is first initialized to zero. When the `while` is encountered in the next line, the computer checks to see if `i` is less than 10. Since `i` begins the loop with the value 0 (which is less than ten), the statements inside the braces will be executed. The first line in the loop, `i++,` increments `i` by 1 and is equivalent to `i=i+1`. The second line calls a function named `display` with the current value of `i` passed as a parameter. After `display` is called, the computer returns to the `while` statement and checks the condition `(i < 10)`. Since after the first iteration of the loop the value of `i` is 1, the condition `(i < 10)` evaluates to logical *TRUE* or equivalently "1", the loop will again be executed. The looping will continue until `i` equals 10 when the condition `(i < 10)` will evaluate as being false. The program will then skip over all the statements within the braces of the `while` construct, and proceed to execute the next statement following the closing brace "`}`".

Arrays

It may be necessary to store a list or table of values of the same type that you want to associate in some way. An array can be used to do this. An array is a group of memory locations all of the same name and data type. Each memory location in the array is called an element and is numbered with the first element as number "0". Note: Be aware, though, that arrays can quickly use up the available variable space, and the compiler does not necessarily check this potential problem. The array is declared with the type of data to be stored in the array as follows:

```
<type> arrayname[maxsize];
```

For example, the lines

```
int values[10];
float timer[60]; /* floating point isn't available with our compiler */
```

would declare an array named values that can store up to ten integers (values[0].... values[9]) and an array named timer that can store up to sixty floating point values (timer[0].... timer[59]). The array can be initially filled with values when it is declared, or it can later be filled with data by the program as follows:

```
int c[5]={23, 10, 35, 2, 17};   /* c[0]...c[4] is filled with listed values */
int f[5]={0};                    /* f[0]...f[4] is filled with zeros */
for (i=0;i<=4;i++)
   f[i]=i;                    /* fills the elements of f[0]..f[4] with 0..4 */
```

Arrays can also have multiple dimensions. A simple example is an array with multiple rows and columns. These arrays can also be initialized and filled as follows:

```
int data[3][10]=( {0},{0} );     /* initialized data to have three rows and ten
                                    columns filled with zeros */
data[0][5]=26;        /* puts the value 26 into the element at row 0, column 5 */
data[2][9]=5;        /* puts the value 5 into the element at row 2, column 9 */
```

Operators

In addition to a full complement of keywords and functions, C also includes a full range of operators. Operators usually have two arguments, and the symbol between them performs an operation on the two arguments, replacing them with the new value. You are probably most familiar with the mathematical operators such as + and -, but you may not be familiar with the bitwise and logical operators which are used in C. *Table 3.2 - Table 3.7* list some of the different types of operators available. The operators are also listed in the order of precedence in *Table 3.8*. Just as in algebra where multiplication precedes addition, all C operators obey a precedence which is summarized in table *Table 3.8*.

Mathematical

The symbols used for many of the C mathematical operators are identical to the symbols for standard mathematical operators, e.g., add "+", subtract "-", and divide "/". *Table 3.2* lists the mathematical operators.

Table 3.2 - Mathematical operators

operator	description
*	multiplication
/	division
%	mod (remainder)
+	addition
-	subtraction

Relational, Equality, and Logical

The C language offers a full range of control structures including `if..else`, `while`, `do..while`, and `switch`. Most of these structures should be familiar from previous computing classes, so the concepts are left to a reference text on C. In C, remember that any non-zero value is true, and a value of zero is false. Relational, equality, and logical operators are used for tests in control structures, and are shown in *Table 3.3*. All operators in this list have two arguments (one on each side of the operator).

Table 3.3 - Relational, equality, and logical operators

operator	description
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

operator	description
==	equal to
!=	not equal to
||	logical OR
&&	logical AND

Bitwise

C can perform some low-level operations such as bit-manipulation that are difficult with other programming languages. In fact, some of these bitwise functions are built into the language. *Table 3.4* summarizes the bitwise operations available in C.

Table 3.4 - Bitwise and shift operators

operator	description	example	result
&	bitwise AND	`0x88 & 0x0F`	`0x08`
^	bitwise XOR	`0x0F ^ 0xFF`	`0xF0`
|	bitwise OR	`0xCC | 0x0F`	`0xCF`
<<	left shift	`0x01 << 4`	`0x10`
>>	right shift	`0x80 >> 6`	`0x02`

Table 3.5 - Truth Tables

`X|Y=Q`

X	Y	Q
0	0	0
0	1	1
1	0	1
1	1	1

`X&Y=Q`

X	Y	Q
0	0	0
0	1	0
1	0	0
1	1	1

Unary

Some C operators are meant to operate on one argument, usually the variable immediately following the operator. *Table 3.6* gives a list of those operators, along with some example for reference purposes.

Assignment

Most mathematical and bitwise operators in the C language can be combined with an equal sign to create an assignment operator. For example `a+=3;` is a statement which will add 3 to the current value of `a`. This is a very useful shorthand notation for `a=a+3;`. All of the assignment operators have the same precedence as equals, and are listed in the precedence table.

Miscellaneous

Many of the operators in C are specific to the syntax of the C language, and bear other meanings depending on their operands. *Table 3.7* below is a list of some miscellaneous operators which are specific to the C language. This table has been included only as a reference, and you may wish to refer to a C reference manual for complete descriptions of these operators

Table 3.6 - Unary operators

operator	description	example	equivalent
++	post-increment	`j = i++;`	`j = i;` `i = i + 1;`
++	pre-increment	`j = ++i;`	`i = i + 1;` `j = i;`
--	post-decrement	`j = i--;`	`j = i;` `i = i - 1;`
--	pre-decrement	`j = --i;`	`i = i - 1;` `j = i;`
*	pointer dereference	`*ptr`	value at a memory location whose address is in `ptr`
&	reference (pointer) of	`&i`	the address of `i`
+	unary plus	`+i`	`i`
-	unary minus	`-i`	the negative of `i`
~	ones complement	`~0xFF`	`0x00`
!	logical negation	`!(0)`	`(1)`

.

Table 3.7 - Miscellaneous operators

operator	description
()	function call
[]	array
->	pointer to structure member access
.	structure member access
(*type*)	type cast
sizeof	size of type (in bytes)
?:	if ? then: else
,	combination statement

Precedence and Associativity

All operators have a precedence and an associativity. *Table 3.8* illustrates the precedence of all operators in the language*. Operators on the same row have equal precedence, and precedence decreases as you move down the table.

Table 3.8 - Operator precedence and associativity

operators[a]	associativity
() [] ->.	left to right
! ~ ++ -- * & (type) sizeof	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	right to left
= += -= *= /= %= &= ^= |= <<= >>=	right to left
.	left to right

a. This table is from Kernighan and Ritchie, *The C Programming Language.*

Programming Structure Hints

The C programs you develop should be written in a style that is easy for yourself and others to read and maintain (modify). Since issues pertaining to programming style are the topic of entire text books, the following are a few helpful guidelines and hints which are generally regarded as hallmarks of good programming style.

- Strictly follow the C style convention for the indentation of blocks of code.
- Select identifier names for C variables and functions which implicitly describe their functionality.
- Keep the scope of all variables as *local* within functions unless absolutely necessary to globalize their scope.

* Kernighan and Ritchie, *The C Programming Language*

– As much as possible, encapsulate the functionality of chunks of code into C functions, i.e. adopt a **modular** programming style. It is especially important to avoid having functionally equivalent copies of code dispersed throughout a project comprising one or more files. If the functionality of similar chunks of code can be encapsulated into a single C function, not only will this result in a reduction of the number of lines of code, but more of the code maintenance can be isolated to *solitary* functions.

– Use comments liberally throughout a program. A good rule of thumb is to include a descriptive comment for about every three lines of code.

Specifics of the *SDCC C* Compiler

The Small Device C Compiler is specifically built for programming microcontrollers including the 8051 series microcontroller. SDCC is a free and open source software available for download to anyone.

Library Functions

As seen in the prior example programs, most of the things done in C, with the exception of low-level functions, involve using library functions. All compilers have their own library functions, but they are usually very close to those defined by the ANSI standard. The *SDCC C* compiler is no exception since it includes most of the ANSI functions and contains some extensions specifically for the C8051. A list of useful functions is included in *Appendix A - Programming Information* for your reference.

Comments

In accordance with the ANSI standard, nested comments (comments within comments) are not allowed.

Definition of an Interrupt Handler Function

The SDCC C compiler will generate a function as an interrupt handler if it is defined as such using the `interrupt` keyword in the function definition. The `interrupt` keyword is not portable to most other compilers. If you wish to test compile your code on another compiler, such as *Borland C/C++* or *gcc* on RCS, you will need to remove `interrupt` from your function prototypes and definitions.

Limitations of the demo version

The Silicon Labs IDE that comes with the develop kits natively supports a variety of microcontroller compilers, including SDCC. Since both are available online, all code can be tested for compilation errors just as it would work within the lab.

If you desire to use your knowledge form this course to include an embedded microcontroller in a project in IED or the MDL, you can purchase the microcontroller kit from the Computer Store in the VCC for $45.

Appendix A - Programming Information

C functions

As mentioned many times, the C language does not really support many functions intrinsic to the language, but allows for extensions in the C libraries. The following section outlines the most useful functions that are available with the *SDCC C* compiler. Some of the functions are ANSI C standard functions, and some are specific to the implementation on the C8051. Below is an outline of the structure of this section, along with an example of each section.

Name
: lists the name of the function

Prototype
: shows the prototype of the function which is necessary to know what types are used in the function call and what include file contains the prototype

Description
: describes the use and operation of the function, including options

Portability
: contains the information about the function as applied to other C compilers

Example program
: a short example of how the function is called or used

Related functions
: a list of other functions that are used for the same purpose or that are used with the function being described

abs

Prototype
#include <stdlib.h>
int abs(number);
int number;

Description
abs returns the absolute value of an integer.

Portability
Available on most systems.

Example program
```
/* Absolute value test program
    This program demonstrates the use of the abs function.
*/
/* Include files */
#include <stdlib.h>
main()
{
    int i;
    i = -19;
    printf("\n The absolute value of %d is %d \n\n\n",i,
           abs(i));
}
```

Related functions - fabs

ceil

Prototype
#include <math.h>
double ceil(Number);
double Number;

Description
ceil() is called the ceiling function. It returns the smallest integer greater than Number as a floating point number. For example, ceil(9.3) returns 10.0, and ceil(-6.7) returns -6.0.

Portability
ANSI C compatible

Example program
```
/* Ceiling function test program
    This program demonstrates the use of the ceil function
*/
/* Include files */
#include <math.h>
main()
{
    double i;
    i = -17.569;
    printf("\n The smallest integer greater than %f is \
            %f \n\n",i,ceil(i));
}
```

Related functions -floor

floor

Prototype

#include <math.h>

double floor(Number);
double Number;

Description

floor returns the greatest integer less than Number as a floating point number. For example floor(9.3) returns 9.0, and floor(-6.7) returns -7.0.

Portability

ANSI C compatible

Example program

```
/* Floor function test program
    This program demonstrates the use of the floor function
*/
/* Include files */
#include <math.h>
main()
{
double i;
i = 14.378;
printf("\n The smallest integer less than %f is %f \n\n",i, floor(i));
}
```

Related functions -ceil

getchar

Prototype

int getchar();
int getc(FILE **stream*);

Description

Getchar reads the next character from the terminal port and returns its value as an integer.

Portability

Getchar is available in ANSI C, but its exact behavior is compiler dependent.

Example program

```
/* getchar test program
    This program demonstrates the use of the getchar function.  The function waits for a character
to be sent from the terminal, then transmits a message with the character back.
*/

main()
{
    int c;

    while (1)
    {
       while(!(c = getchar()));
       printf("The character was %c and it's ASCII code \
               is %d \n",c,c);
    }
}
```

Related functions

gets, putchar

gets

Prototype

#include <string.h>

car *gets(String);
car *String;

Description

Gets will read a line of input from the terminal port and will place it in the string pointed to by String.

Portability

Available everywhere.

Example program

```
/* gets test program
    This program illustrates the use of the gets function.  It will receive a line of input from
the terminal, and return it.  It will also print out the first 40 characters in columns to illustrate
simple string handling functions.  The printf function can only output a string of up to 80 char-
acters, so we check for a string that is too long.  If you need to output longer strings use the
puts function instead.   */

#include <string.h>

main()
{
char String[80];
int i, length;

while (1)
    {
    printf("\nPlease enter a string\n");
    gets(String);

    length = strlen(String);

    if (length <= 60)
        {
        printf("\nThe String is \"%s\" \n",String);

        for (i=0;( (i<length) && (i<40) );i++)
            printf("\t%c",String[i]);

        printf("\n");
        }
    else
        printf("\nThat string is too long, try again.\n");
    }
}
```

Related functions

getchar, puts

printf {NOTE: use printf_fast_f to print float values}

Prototype

```
#include <stdio.h>
int printf(controlString [, arg1] . . . );
char *controlString;
```

Description

The formatted input and output functions are reminiscent of the days of hardcopy terminals, which prompted for input from the user in a specific form, and then output information, one line at a time. This is a very simple method of communication in contrast to today's interfaces that usually have full-screen feedback of keyboard and mouse input. This simple interface can be used with the EVB because the HyperTerminal program emulates a terminal to interface with other computers. Printf is used to send information, such as variables, that must be converted to a character string form before sending each character to the terminal. To accomplish this conversion and positioning on the line, printf uses a control string, which instructs the function what types of input are being used and what format the output should be in. The control string flags are listed here along with several examples of the usage of control strings.

When the printf function searches through the control string argument, it looks for flags in the string that indicate that a command follows. There are two characters which are used: a backslash (\) to indicate a control command follows, and a percent sign (%) to indicate the type of the next variable to send to the terminal.

The control commands follow:

\n	start a new line
\r	send a carriage return without a line feed
\t	send a tab
\	control string continues on the following line

The data types and how to print them are included below:

char	%d for a decimal number, %x for a hexadecimal number or %c for a single character display
signed char	%d for a decimal number, %x for a hexadecimal number or %c for a single character display
unsigned char	%u for a decimal number, %x for a hexadecimal number or %c for a single character display
hexadecimal	%x
short int	%d
int	%d
long int	%ld
unsigned short int	%u
unsigned int	%u
unsigned long int	%lu
float, double	%[width].[precision]f

Portability

printf is a standard library function available on all ANSI and original C compilers.

Example program

```
#include <stdio.h>

void test_printf (void)
{
  char a;
  int b;
  long c;
  unsigned char x;
  unsigned int y;
  unsigned long z;
  float f,g;

  a = 1;
  b = 12365;
  c = 0x7FFFFFFF;
  x = 'A';
  y = 54321;
  z = 0x4A6FE00;
  f = 10.0;
  g = 22.95;

  printf("Char %d  int %d  long %ld\n", a,b,c);
  printf("Uchar %u  Uint %u  Ulong %lu\n", x,y,z);
  printf("Xchar %x  xint %x  xlong %lx\n", x,y,z);
  printf("%f != %g\n", f,g);
  printf("%4.2f != %4.2g\n", f,g);
}
```

Related functions

scanf

(scanf is not presently available in the SDCC compiler)

printf_fast_f

Prototype

```
#include <stdio.h>
int printf_fast_f(controlString [, arg1] . . . );
char *controlString;
```

The printf_fast_f function has the same identical functionality as printf, with the exception that printf_fast_f will allow the output of floating point data types.

Example program

```
/* printf_fast_f
    This program illustrates how to print a floating point number
*/

main()
{
float i = 4.0;

printf("The value of i is: %f", i);
}
```

Related functions: printf

putchar, puts

Prototype

void putchar(Character);
int Character;

void puts(String);
char *String;

Description

Putchar sends Character to the terminal port. Character is the ASCII code (integer) for the character to be sent. Puts sends a string to the terminal port.

Portability

ANSI C compatible.

Example program

```
/* putchar & puts test program
    This program illustrates the use of the putchar and puts functions.  The program will use putchar
to output the ASCII characters in the range 33-125 (the printable characters), then print a message
with the puts function.
*/

main()
{
int i;
char String[30];

/* use strcpy() to copy Done message into String */
strcpy(String,"\nDone\n\n\n\n");  puts("\n");

for (i=33;i<=126;i++)
    putchar(i);

puts(String);
}
```

Related functions

gets, getchar

rand

Prototype

#include <stdlib.h>
int rand(void);

Description

Returns a pseudo-random integer between 0 and 32767. The **srand()** function may be used to seed the pseudo-random number generator before calling **rand()**.

Portability

Available on most systems.

Example program

```
/* rand test program
    This program illustrates the generation of 5 random numbers*/

#include<stdlib.h>

main()
{
int num,i;
for (i=0;i<5;i++)
{
    num=rand();
    printf("the random number=%d\n", num);
}
}
```

Related functions -srand

scanf

(NOT PRESENTLY AVAILABLE FOR THE SDCC COMPILER)

Prototype

#include <stdio.h>
int scanf(controlString [, pointer1] . . .);
char *controlString;

Description

Reads formatted data from `stdin` and writes the results to memory at the addresses given by the variable arguments. Each variable argument must be a pointer to a datum of type that corresponds to the format of the data.

Portability

scanf is a standard library function available on all ANSI and original C compilers

Example program

```
#include <stdlib.h>
#include<stdio.h>
void main(void)
{
int a;
printf("Enter an integer:");
scanf("%d", &a);
printf("The value you entered is %d\n", a);
}
```

Related functions - printf

srand

Prototype

#include <stdlib.h>
int srand(int val);

Description

Seed the **rand()** function with **val**.

Portability

Available on most systems.

Example program

```
/* srand test program
    This program demonstrates how to seed the random number generator*/

#include<stdlib.h>

main()
{
int num;

srand(50);
num=rand();


}
```

Related functions

rand

c8051.h header file

```
/*-----------------------------------------------------------------------
   Register Declarations for the Cygnal/SiLabs C8051F02x Processor Range

   Copyright (C) 2004 - Maarten Brock, sourceforge.brock@dse.nl

   This library is free software; you can redistribute it and/or
   modify it under the terms of the GNU Lesser General Public
   License as published by the Free Software Foundation; either
   version 2.1 of the License, or (at your option) any later version.

   This library is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
   Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public
   License along with this library; if not, write to the Free Software
   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
-----------------------------------------------------------------------*/

#ifndef C8051F020_H
#define C8051F020_H


/*  BYTE Registers  */
__sfr __at (0x80) P0          ;  /* PORT 0                                */
__sfr __at (0x81) SP          ;  /* STACK POINTER                         */
__sfr __at (0x82) DPL         ;  /* DATA POINTER - LOW BYTE               */
__sfr __at (0x83) DPH         ;  /* DATA POINTER - HIGH BYTE              */
__sfr __at (0x84) P4          ;  /* PORT 4                                */
__sfr __at (0x85) P5          ;  /* PORT 5                                */
__sfr __at (0x86) P6          ;  /* PORT 6                                */
__sfr __at (0x87) PCON        ;  /* POWER CONTROL                         */
__sfr __at (0x88) TCON        ;  /* TIMER CONTROL                         */
__sfr __at (0x89) TMOD        ;  /* TIMER MODE                            */
__sfr __at (0x8A) TL0         ;  /* TIMER 0 - LOW BYTE                    */
__sfr __at (0x8B) TL1         ;  /* TIMER 1 - LOW BYTE                    */
__sfr __at (0x8C) TH0         ;  /* TIMER 0 - HIGH BYTE                   */
__sfr __at (0x8D) TH1         ;  /* TIMER 1 - HIGH BYTE                   */
__sfr __at (0x8E) CKCON       ;  /* CLOCK CONTROL                         */
__sfr __at (0x8F) PSCTL       ;  /* PROGRAM STORE R/W CONTROL             */
__sfr __at (0x90) P1          ;  /* PORT 1                                */
__sfr __at (0x91) TMR3CN      ;  /* TIMER 3 CONTROL                       */
__sfr __at (0x92) TMR3RLL     ;  /* TIMER 3 RELOAD REGISTER - LOW BYTE    */
__sfr __at (0x93) TMR3RLH     ;  /* TIMER 3 RELOAD REGISTER - HIGH BYTE   */
__sfr __at (0x94) TMR3L       ;  /* TIMER 3 - LOW BYTE                    */
__sfr __at (0x95) TMR3H       ;  /* TIMER 3 - HIGH BYTE                   */
__sfr __at (0x96) P7          ;  /* PORT 7                                */
__sfr __at (0x98) SCON        ;  /* UART0 CONTROL                         */
__sfr __at (0x98) SCON0       ;  /* UART0 CONTROL                         */
__sfr __at (0x99) SBUF        ;  /* UART0 BUFFER                          */
__sfr __at (0x99) SBUF0       ;  /* UART0 BUFFER                          */
__sfr __at (0x9A) SPI0CFG;  /* SERIAL PERIPHERAL INTERFACE 0 CONFIGURATION    */
__sfr __at (0x9B) SPI0DAT;  /* SERIAL PERIPHERAL INTERFACE 0 DATA             */
__sfr __at (0x9C) ADC1   ;  /* ADC 1 DATA                                     */
__sfr __at (0x9D) SPI0CKR;  /* SERIAL PERIPHERAL INTERFACE 0 CLOCK RATE CONTROL*/
__sfr __at (0x9E) CPT0CN      ;  /* COMPARATOR 0 CONTROL                   */
```

```
__sfr __at (0x9F) CPT1CN      ;  /* COMPARATOR 1 CONTROL                        */
__sfr __at (0xA0) P2          ;  /* PORT 2                                      */
__sfr __at (0xA1) EMI0TC      ;  /* External Memory Timing Control              */
__sfr __at (0xA3) EMI0CF      ;  /* EMIF CONFIGURATION                          */
__sfr __at (0xA4) PRT0CF      ;  /* PORT 0 CONFIGURATION                        */
__sfr __at (0xA4) P0MDOUT     ;  /* PORT 0 OUTPUT MODE CONFIGURATION            */
__sfr __at (0xA5) PRT1CF      ;  /* PORT 1 CONFIGURATION                        */
__sfr __at (0xA5) P1MDOUT     ;  /* PORT 1 OUTPUT MODE CONFIGURATION            */
__sfr __at (0xA6) PRT2CF      ;  /* PORT 2 CONFIGURATION                        */
__sfr __at (0xA6) P2MDOUT     ;  /* PORT 2 OUTPUT MODE CONFIGURATION            */
__sfr __at (0xA7) PRT3CF      ;  /* PORT 3 CONFIGURATION                        */
__sfr __at (0xA7) P3MDOUT     ;  /* PORT 3 OUTPUT MODE CONFIGURATION            */
__sfr __at (0xA8) IE          ;  /* INTERRUPT ENABLE                            */
__sfr __at (0xA9) SADDR0      ;  /* UART0 Slave Address                         */
__sfr __at (0xAA) ADC1CN      ;  /* ADC 1 CONTROL                               */
__sfr __at (0xAB) ADC1CF      ;  /* ADC 1 CONFIGURATION                         */
__sfr __at (0xAC) AMX1SL      ;  /* ADC 1 MUX CHANNEL SELECTION                 */
__sfr __at (0xAD) P3IF        ;  /* PORT 3 EXTERNAL INTERRUPT FLAGS             */
__sfr __at (0xAE) SADEN1      ;  /* UART1 Slave Address Enable                  */
__sfr __at (0xAF) EMI0CN      ;  /* EXTERNAL MEMORY INTERFACE CONTROL           */
__sfr __at (0xAF) _XPAGE      ;  /* XDATA/PDATA PAGE                            */
__sfr __at (0xB0) P3          ;  /* PORT 3                                      */
__sfr __at (0xB1) OSCXCN      ;  /* EXTERNAL OSCILLATOR CONTROL                 */
__sfr __at (0xB2) OSCICN      ;  /* INTERNAL OSCILLATOR CONTROL                 */
__sfr __at (0xB5) P74OUT     ;  /* PORT 4 THROUGH 7 OUTPUT MODE CONFIGURATION*/
__sfr __at (0xB6) FLSCL       ;  /* FLASH MEMORY TIMING PRESCALER               */
__sfr __at (0xB7) FLACL       ;  /* FLASH ACESS LIMIT                           */
__sfr __at (0xB8) IP          ;  /* INTERRUPT PRIORITY                          */
__sfr __at (0xB9) SADEN0      ;  /* UART0 Slave Address Enable                  */
__sfr __at (0xBA) AMX0CF      ;  /* ADC 0 MUX CONFIGURATION                     */
__sfr __at (0xBB) AMX0SL      ;  /* ADC 0 MUX CHANNEL SELECTION                 */
__sfr __at (0xBC) ADC0CF      ;  /* ADC 0 CONFIGURATION                         */
__sfr __at (0xBD) P1MDIN      ;  /* PORT 1 Input Mode                           */
__sfr __at (0xBE) ADC0L       ;  /* ADC 0 DATA - LOW BYTE                       */
__sfr __at (0xBF) ADC0H       ;  /* ADC 0 DATA - HIGH BYTE                      */
__sfr __at (0xC0) SMB0CN      ;  /* SMBUS 0 CONTROL                             */
__sfr __at (0xC1) SMB0STA     ;  /* SMBUS 0 STATUS                              */
__sfr __at (0xC2) SMB0DAT     ;  /* SMBUS 0 DATA                                */
__sfr __at (0xC3) SMB0ADR     ;  /* SMBUS 0 SLAVE ADDRESS                       */
__sfr __at (0xC4) ADC0GTL     ;  /* ADC 0 GREATER-THAN REGISTER - LOW BYTE      */
__sfr __at (0xC5) ADC0GTH     ;  /* ADC 0 GREATER-THAN REGISTER - HIGH BYTE     */
__sfr __at (0xC6) ADC0LTL     ;  /* ADC 0 LESS-THAN REGISTER - LOW BYTE         */
__sfr __at (0xC7) ADC0LTH     ;  /* ADC 0 LESS-THAN REGISTER - HIGH BYTE        */
__sfr __at (0xC8) T2CON       ;  /* TIMER 2 CONTROL                             */
__sfr __at (0xC9) T4CON       ;  /* TIMER 4 CONTROL                             */
__sfr __at (0xCA) RCAP2L      ;  /* TIMER 2 CAPTURE REGISTER - LOW BYTE         */
__sfr __at (0xCB) RCAP2H      ;  /* TIMER 2 CAPTURE REGISTER - HIGH BYTE        */
__sfr __at (0xCC) TL2         ;  /* TIMER 2 - LOW BYTE                          */
__sfr __at (0xCD) TH2         ;  /* TIMER 2 - HIGH BYTE                         */
__sfr __at (0xCF) SMB0CR      ;  /* SMBUS 0 CLOCK RATE                          */
__sfr __at (0xD0) PSW         ;  /* PROGRAM STATUS WORD                         */
__sfr __at (0xD1) REF0CN      ;  /* VOLTAGE REFERENCE 0 CONTROL                 */
__sfr __at (0xD2) DAC0L       ;  /* DAC 0 REGISTER - LOW BYTE                   */
__sfr __at (0xD3) DAC0H       ;  /* DAC 0 REGISTER - HIGH BYTE                  */
__sfr __at (0xD4) DAC0CN      ;  /* DAC 0 CONTROL                               */
__sfr __at (0xD5) DAC1L       ;  /* DAC 1 REGISTER - LOW BYTE                   */
__sfr __at (0xD6) DAC1H       ;  /* DAC 1 REGISTER - HIGH BYTE                  */
__sfr __at (0xD7) DAC1CN      ;  /* DAC 1 CONTROL                               */
__sfr __at (0xD8) PCA0CN      ;  /* PCA 0 COUNTER CONTROL                       */
```

```
__sfr __at (0xD9) PCA0MD       ;  /* PCA 0 COUNTER MODE                          */
__sfr __at (0xDA) PCA0CPM0     ;  /* CONTROL REGISTER FOR PCA 0 MODULE 0         */
__sfr __at (0xDB) PCA0CPM1     ;  /* CONTROL REGISTER FOR PCA 0 MODULE 1         */
__sfr __at (0xDC) PCA0CPM2     ;  /* CONTROL REGISTER FOR PCA 0 MODULE 2         */
__sfr __at (0xDD) PCA0CPM3     ;  /* CONTROL REGISTER FOR PCA 0 MODULE 3         */
__sfr __at (0xDE) PCA0CPM4     ;  /* CONTROL REGISTER FOR PCA 0 MODULE 4         */
__sfr __at (0xE0) ACC          ;  /* ACCUMULATOR                                 */
__sfr __at (0xE1) XBR0         ;  /* DIGITAL CROSSBAR CONFIGURATION REGISTER 0*/
__sfr __at (0xE2) XBR1         ;  /* DIGITAL CROSSBAR CONFIGURATION REGISTER 1*/
__sfr __at (0xE3) XBR2         ;  /* DIGITAL CROSSBAR CONFIGURATION REGISTER 2*/
__sfr __at (0xE4) RCAP4L       ;  /* TIMER 4 CAPTURE REGISTER - LOW BYTE         */
__sfr __at (0xE5) RCAP4H       ;  /* TIMER 4 CAPTURE REGISTER - HIGH BYTE        */
__sfr __at (0xE6) EIE1         ;  /* EXTERNAL INTERRUPT ENABLE 1                 */
__sfr __at (0xE7) EIE2         ;  /* EXTERNAL INTERRUPT ENABLE 2                 */
__sfr __at (0xE8) ADC0CN       ;  /* ADC 0 CONTROL                               */
__sfr __at (0xE9) PCA0L        ;  /* PCA 0 TIMER - LOW BYTE                      */
__sfr __at (0xEA) PCA0CPL0     ;/* CAPTURE/COMPARE REGISTER FOR PCA 0 MODULE 0 - LOW
BYTE*/
__sfr __at (0xEB) PCA0CPL1      ;/* CAPTURE/COMPARE REGISTER FOR PCA 0 MODULE 1 - LOW
BYTE*/
__sfr __at (0xEC) PCA0CPL2       ;/* CAPTURE/COMPARE REGISTER FOR PCA 0 MODULE 2 -
LOW BYTE*/
__sfr __at (0xED) PCA0CPL3       ;/* CAPTURE/COMPARE REGISTER FOR PCA 0 MODULE 3 -
LOW BYTE*/
__sfr __at (0xEE) PCA0CPL4       ;/* CAPTURE/COMPARE REGISTER FOR PCA 0 MODULE 4 -
LOW BYTE*/
__sfr __at (0xEF) RSTSRC       ;  /* RESET SOURCE                                */
__sfr __at (0xF0) B            ;  /* B REGISTER                                  */
__sfr __at (0xF1) SCON1        ;  /* UART1 CONTROL                               */
__sfr __at (0xF2) SBUF1        ;  /* UART1 DATA                                  */
__sfr __at (0xF3) SADDR1       ;  /* UART1 Slave Address                         */
__sfr __at (0xF4) TL4          ;  /* TIMER 4 DATA - LOW BYTE                     */
__sfr __at (0xF5) TH4          ;  /* TIMER 4 DATA - HIGH BYTE                    */
__sfr __at (0xF6) EIP1         ;  /* EXTERNAL INTERRUPT PRIORITY REGISTER 1   */
__sfr __at (0xF7) EIP2         ;  /* EXTERNAL INTERRUPT PRIORITY REGISTER 2   */
__sfr __at (0xF8) SPI0CN       ;  /* SERIAL PERIPHERAL INTERFACE 0 CONTROL    */
__sfr __at (0xF9) PCA0H        ;  /* PCA 0 TIMER - HIGH BYTE                     */
__sfr __at (0xFA) PCA0CPH0     ;/* CAPTURE/COMPARE REGISTER FOR PCA 0 MODULE 0 - HIGH
BYTE*/
__sfr __at (0xFB) PCA0CPH1     ;/* CAPTURE/COMPARE REGISTER FOR PCA 0 MODULE 1 - HIGH
BYTE*/
__sfr __at (0xFC) PCA0CPH2     ;/* CAPTURE/COMPARE REGISTER FOR PCA 0 MODULE 2 - HIGH
BYTE*/
__sfr __at (0xFD) PCA0CPH3     ;/* CAPTURE/COMPARE REGISTER FOR PCA 0 MODULE 3 - HIGH
BYTE*/
__sfr __at (0xFE) PCA0CPH4     ;/* CAPTURE/COMPARE REGISTER FOR PCA 0 MODULE 4 - HIGH
BYTE*/
__sfr __at (0xFF) WDTCN        ;  /* WATCHDOG TIMER CONTROL                      */


/*  WORD/DWORD Registers  */

__sfr16 __at (0x8C8A) TMR0     ;  /* TIMER 0 COUNTER                             */
__sfr16 __at (0x8D8B) TMR1     ;  /* TIMER 1 COUNTER                             */
__sfr16 __at (0xCDCC) TMR2     ;  /* TIMER 2 COUNTER                             */
__sfr16 __at (0xCBCA) RCAP2    ;  /* TIMER 2 CAPTURE REGISTER WORD               */
__sfr16 __at (0x9594) TMR3     ;  /* TIMER 3 COUNTER                             */
__sfr16 __at (0x9392) TMR3RL   ;  /* TIMER 3 CAPTURE REGISTER WORD               */
__sfr16 __at (0xF5F4) TMR4     ;  /* TIMER 4 COUNTER                             */
```

```
__sfr16 __at (0xE5E4) RCAP4    ;  /* TIMER 4 CAPTURE REGISTER WORD           */
__sfr16 __at (0xBFBE) ADC0     ;  /* ADC 0 DATA WORD                         */
__sfr16 __at (0xC5C4) ADC0GT   ;  /* ADC 0 GREATER-THAN REGISTER WORD        */
__sfr16 __at (0xC7C6) ADC0LT   ;  /* ADC 0 LESS-THAN REGISTER WORD           */
__sfr16 __at (0xD3D2) DAC0     ;  /* DAC 0 REGISTER WORD                     */
__sfr16 __at (0xD6D5) DAC1     ;  /* DAC 1 REGISTER WORD                     */
__sfr16 __at (0xF9E9) PCA0     ;  /* PCA COUNTER                             */
__sfr16 __at (0xFAEA) PCA0CP0  ;  /* PCA CAPTURE 0 WORD                      */
__sfr16 __at (0xFBEB) PCA0CP1  ;  /* PCA CAPTURE 1 WORD                      */
__sfr16 __at (0xFCEC) PCA0CP2  ;  /* PCA CAPTURE 2 WORD                      */
__sfr16 __at (0xFDED) PCA0CP3  ;  /* PCA CAPTURE 3 WORD                      */
__sfr16 __at (0xFEEE) PCA0CP4  ;  /* PCA CAPTURE 4 WORD                      */


/*  BIT Registers  */

/*  P0  0x80 */
__sbit __at (0x80) P0_0        ;
__sbit __at (0x81) P0_1        ;
__sbit __at (0x82) P0_2        ;
__sbit __at (0x83) P0_3        ;
__sbit __at (0x84) P0_4        ;
__sbit __at (0x85) P0_5        ;
__sbit __at (0x86) P0_6        ;
__sbit __at (0x87) P0_7        ;

/*  TCON  0x88 */
__sbit __at (0x88) IT0         ;  /* EXT. INTERRUPT 0 TYPE                   */
__sbit __at (0x89) IE0         ;  /* EXT. INTERRUPT 0 EDGE FLAG              */
__sbit __at (0x8A) IT1         ;  /* EXT. INTERRUPT 1 TYPE                   */
__sbit __at (0x8B) IE1         ;  /* EXT. INTERRUPT 1 EDGE FLAG              */
__sbit __at (0x8C) TR0         ;  /* TIMER 0 ON/OFF CONTROL                  */
__sbit __at (0x8D) TF0         ;  /* TIMER 0 OVERFLOW FLAG                   */
__sbit __at (0x8E) TR1         ;  /* TIMER 1 ON/OFF CONTROL                  */
__sbit __at (0x8F) TF1         ;  /* TIMER 1 OVERFLOW FLAG                   */

/*  P1  0x90 */
__sbit __at (0x90) P1_0        ;
__sbit __at (0x91) P1_1        ;
__sbit __at (0x92) P1_2        ;
__sbit __at (0x93) P1_3        ;
__sbit __at (0x94) P1_4        ;
__sbit __at (0x95) P1_5        ;
__sbit __at (0x96) P1_6        ;
__sbit __at (0x97) P1_7        ;

/*  SCON  0x98 */
__sbit __at (0x98) RI          ;  /* SCON.0 - RECEIVE INTERRUPT FLAG         */
__sbit __at (0x98) RI0         ;  /* SCON.0 - RECEIVE INTERRUPT FLAG         */
__sbit __at (0x99) TI          ;  /* SCON.1 - TRANSMIT INTERRUPT FLAG        */
__sbit __at (0x99) TI0         ;  /* SCON.1 - TRANSMIT INTERRUPT FLAG        */
__sbit __at (0x9A) RB8         ;  /* SCON.2 - RECEIVE BIT 8                  */
__sbit __at (0x9A) RB80        ;  /* SCON.2 - RECEIVE BIT 8                  */
__sbit __at (0x9B) TB8         ;  /* SCON.3 - TRANSMIT BIT 8                 */
__sbit __at (0x9B) TB80        ;  /* SCON.3 - TRANSMIT BIT 8                 */
__sbit __at (0x9C) REN         ;  /* SCON.4 - RECEIVE ENABLE                 */
__sbit __at (0x9C) REN0        ;  /* SCON.4 - RECEIVE ENABLE                 */
__sbit __at (0x9D) SM2 ;  /* SCON.5 - MULTIPROCESSOR COMMUNICATION ENABLE    */
__sbit __at (0x9D) SM20;  /* SCON.5 - MULTIPROCESSOR COMMUNICATION ENABLE    */
```

```
__sbit __at (0x9D) MCE0;  /* SCON.5 - MULTIPROCESSOR COMMUNICATION ENABLE    */
__sbit __at (0x9E) SM1         ;  /* SCON.6 - SERIAL MODE CONTROL BIT 1      */
__sbit __at (0x9E) SM10        ;  /* SCON.6 - SERIAL MODE CONTROL BIT 1      */
__sbit __at (0x9F) SM0         ;  /* SCON.7 - SERIAL MODE CONTROL BIT 0      */
__sbit __at (0x9F) SM00        ;  /* SCON.7 - SERIAL MODE CONTROL BIT 0      */
__sbit __at (0x9F) S0MODE      ;  /* SCON.7 - SERIAL MODE CONTROL BIT 0      */

/*  P2  0xA0 */
__sbit __at (0xA0) P2_0         ;
__sbit __at (0xA1) P2_1         ;
__sbit __at (0xA2) P2_2         ;
__sbit __at (0xA3) P2_3         ;
__sbit __at (0xA4) P2_4         ;
__sbit __at (0xA5) P2_5         ;
__sbit __at (0xA6) P2_6         ;
__sbit __at (0xA7) P2_7         ;

/*  IE  0xA8 */
__sbit __at (0xA8) EX0         ;  /* EXTERNAL INTERRUPT 0 ENABLE             */
__sbit __at (0xA9) ET0         ;  /* TIMER 0 INTERRUPT ENABLE                */
__sbit __at (0xAA) EX1         ;  /* EXTERNAL INTERRUPT 1 ENABLE             */
__sbit __at (0xAB) ET1         ;  /* TIMER 1 INTERRUPT ENABLE                */
__sbit __at (0xAC) ES0         ;  /* SERIAL PORT 0 INTERRUPT ENABLE          */
__sbit __at (0xAC) ES          ;  /* SERIAL PORT 0 INTERRUPT ENABLE          */
__sbit __at (0xAD) ET2         ;  /* TIMER 2 INTERRUPT ENABLE                */
__sbit __at (0xAF) EA          ;  /* GLOBAL INTERRUPT ENABLE                 */

/*  P3  0xB0 */
__sbit __at (0xB0) P3_0         ;
__sbit __at (0xB1) P3_1         ;
__sbit __at (0xB2) P3_2         ;
__sbit __at (0xB3) P3_3         ;
__sbit __at (0xB4) P3_4         ;
__sbit __at (0xB5) P3_5         ;
__sbit __at (0xB6) P3_6         ;
__sbit __at (0xB7) P3_7         ;

/*  IP  0xB8 */
__sbit __at (0xB8) PX0         ;  /* EXTERNAL INTERRUPT 0 PRIORITY           */
__sbit __at (0xB9) PT0         ;  /* TIMER 0 PRIORITY                        */
__sbit __at (0xBA) PX1         ;  /* EXTERNAL INTERRUPT 1 PRIORITY           */
__sbit __at (0xBB) PT1         ;  /* TIMER 1 PRIORITY                        */
__sbit __at (0xBC) PS0         ;  /* SERIAL PORT PRIORITY                    */
__sbit __at (0xBC) PS          ;  /* SERIAL PORT PRIORITY                    */
__sbit __at (0xBD) PT2         ;  /* TIMER 2 PRIORITY                        */

/*  SMB0CN  0xC0 */
__sbit __at (0xC0) SMBTOE      ;  /* SMBUS 0 TIMEOUT ENABLE                  */
__sbit __at (0xC1) SMBFTE      ;  /* SMBUS 0 FREE TIMER ENABLE               */
__sbit __at (0xC2) AA          ;  /* SMBUS 0 ASSERT/ACKNOWLEDGE FLAG         */
__sbit __at (0xC3) SI          ;  /* SMBUS 0 INTERRUPT PENDING FLAG          */
__sbit __at (0xC4) STO         ;  /* SMBUS 0 STOP FLAG                       */
__sbit __at (0xC5) STA         ;  /* SMBUS 0 START FLAG                      */
__sbit __at (0xC6) ENSMB       ;  /* SMBUS 0 ENABLE                          */
__sbit __at (0xC7) BUSY        ;  /* SMBUS 0 BUSY                            */

/*  T2CON  0xC8 */
__sbit __at (0xC8) CPRL2       ;  /* CAPTURE OR RELOAD SELECT                */
__sbit __at (0xC9) CT2         ;  /* TIMER OR COUNTER SELECT                 */
```

```
__sbit __at (0xCA) TR2        ;  /* TIMER 2 ON/OFF CONTROL                  */
__sbit __at (0xCB) EXEN2      ;  /* TIMER 2 EXTERNAL ENABLE FLAG            */
__sbit __at (0xCC) TCLK       ;  /* TRANSMIT CLOCK FLAG                     */
__sbit __at (0xCD) RCLK       ;  /* RECEIVE CLOCK FLAG                      */
__sbit __at (0xCE) EXF2       ;  /* EXTERNAL FLAG                           */
__sbit __at (0xCF) TF2        ;  /* TIMER 2 OVERFLOW FLAG                   */

/*  PSW  0xD0 */
__sbit __at (0xD0) P          ;  /* ACCUMULATOR PARITY FLAG                 */
__sbit __at (0xD1) F1         ;  /* USER FLAG 1                             */
__sbit __at (0xD2) OV         ;  /* OVERFLOW FLAG                           */
__sbit __at (0xD3) RS0        ;  /* REGISTER BANK SELECT 0                  */
__sbit __at (0xD4) RS1        ;  /* REGISTER BANK SELECT 1                  */
__sbit __at (0xD5) F0         ;  /* USER FLAG 0                             */
__sbit __at (0xD6) AC         ;  /* AUXILIARY CARRY FLAG                    */
__sbit __at (0xD7) CY         ;  /* CARRY FLAG                              */

/*  PCA0CN  0xD8H */
__sbit __at (0xD8) CCF0       ;  /* PCA 0 MODULE 0 INTERRUPT FLAG           */
__sbit __at (0xD9) CCF1       ;  /* PCA 0 MODULE 1 INTERRUPT FLAG           */
__sbit __at (0xDA) CCF2       ;  /* PCA 0 MODULE 2 INTERRUPT FLAG           */
__sbit __at (0xDB) CCF3       ;  /* PCA 0 MODULE 3 INTERRUPT FLAG           */
__sbit __at (0xDC) CCF4       ;  /* PCA 0 MODULE 4 INTERRUPT FLAG           */
__sbit __at (0xDE) CR         ;  /* PCA 0 COUNTER RUN CONTROL BIT           */
__sbit __at (0xDF) CF         ;  /* PCA 0 COUNTER OVERFLOW FLAG             */

/*  ADC0CN  0xE8H */
__sbit __at (0xE8) ADLJST     ;  /* ADC 0 RIGHT JUSTIFY DATA BIT            */
__sbit __at (0xE8) AD0LJST    ;  /* ADC 0 RIGHT JUSTIFY DATA BIT            */
__sbit __at (0xE9) ADWINT     ;  /* ADC 0 WINDOW COMPARE INTERRUPT FLAG     */
__sbit __at (0xE9) AD0WINT    ;  /* ADC 0 WINDOW COMPARE INTERRUPT FLAG     */
__sbit __at (0xEA) ADSTM0     ;  /* ADC 0 START OF CONVERSION MODE BIT 0    */
__sbit __at (0xEA) AD0CM0     ;  /* ADC 0 START OF CONVERSION MODE BIT 0    */
__sbit __at (0xEB) ADSTM1     ;  /* ADC 0 START OF CONVERSION MODE BIT 1    */
__sbit __at (0xEB) AD0CM1     ;  /* ADC 0 START OF CONVERSION MODE BIT 1    */
__sbit __at (0xEC) ADBUSY     ;  /* ADC 0 BUSY FLAG                         */
__sbit __at (0xEC) AD0BUSY    ;  /* ADC 0 BUSY FLAG                         */
__sbit __at (0xED) ADCINT     ;  /* ADC 0 CONVERISION COMPLETE INTERRUPT FLAG*/
__sbit __at (0xED) AD0INT     ;  /* ADC 0 CONVERISION COMPLETE INTERRUPT FLAG*/
__sbit __at (0xEE) ADCTM      ;  /* ADC 0 TRACK MODE                        */
__sbit __at (0xEE) AD0TM      ;  /* ADC 0 TRACK MODE                        */
__sbit __at (0xEF) ADCEN      ;  /* ADC 0 ENABLE                            */
__sbit __at (0xEF) AD0EN      ;  /* ADC 0 ENABLE                            */

/*  SPI0CN  0xF8H */
__sbit __at (0xF8) SPIEN      ;  /* SPI 0 SPI ENABLE                        */
__sbit __at (0xF9) MSTEN      ;  /* SPI 0 MASTER ENABLE                     */
__sbit __at (0xFA) SLVSEL     ;  /* SPI 0 SLAVE SELECT                      */
__sbit __at (0xFB) TXBSY      ;  /* SPI 0 TX BUSY FLAG                      */
__sbit __at (0xFC) RXOVRN     ;  /* SPI 0 RX OVERRUN FLAG                   */
__sbit __at (0xFD) MODF       ;  /* SPI 0 MODE FAULT FLAG                   */
__sbit __at (0xFE) WCOL       ;  /* SPI 0 WRITE COLLISION FLAG              */
__sbit __at (0xFF) SPIF       ;  /* SPI 0 INTERRUPT FLAG  */


/* Predefined SFR Bit Masks */

#define PCON_IDLE         0x01    /* PCON                                  */
#define PCON_STOP         0x02    /* PCON                                  */
```

```
#define PCON_SMOD0        0x80    /* PCON                                */
#define TF3               0x80    /* TMR3CN                              */
#define CPFIF             0x10    /* CPTnCN                              */
#define CPRIF             0x20    /* CPTnCN                              */
#define CPOUT             0x40    /* CPTnCN                              */
#define TR4               0x04    /* T4CON                               */
#define TF4               0x80    /* T4CON                               */
#define ECCF              0x01    /* PCA0CPMn                            */
#define PWM               0x02    /* PCA0CPMn                            */
#define TOG               0x04    /* PCA0CPMn                            */
#define MAT               0x08    /* PCA0CPMn                            */
#define CAPN              0x10    /* PCA0CPMn                            */
#define CAPP              0x20    /* PCA0CPMn                            */
#define ECOM              0x40    /* PCA0CPMn                            */
#define PWM16             0x80    /* PCA0CPMn                            */
#define PORSF             0x02    /* RSTSRC                              */
#define SWRSF             0x10    /* RSTSRC                              */
#define RI1               0x01    /* SCON1                               */
#define TI1               0x02    /* SCON1                               */
#define RB81              0x04    /* SCON1                               */
#define TB81              0x08    /* SCON1                               */
#define REN1              0x10    /* SCON1                               */
#define SM21              0x20    /* SCON1                               */
#define SM11              0x40    /* SCON1                               */
#define SM01              0x80    /* SCON1                               */

#endif
```