C short tuor

*Abstract
  Now I take you a short tuor.
  In this short tuor, you will get the outline of hardware control with C.

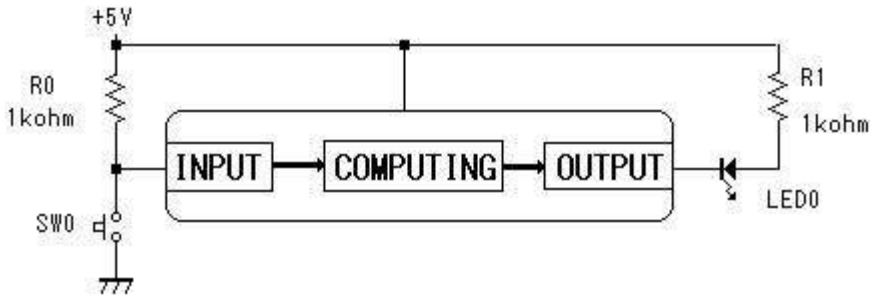
  After this tuor, you will find the difference of ANSI C text explanations.

  There are many C language books but I had not found the adequet book which explains about hardware control with C.

  Shall we go !

* Simple program
   <Specifications>
     Push switch (SW0) , then turn on a LED (LED0).

     During pushing, LED enlights.



  C source code

```
/**************************************
    File Name => Simple program

    Hardware Property
        Port 7 : Input
        Port B : Output

**************************************/
#include <j3048f.h>

/**************************************
    Global labels
**************************************/
#define FALSE  0
#define TRUE   FALSE+1

/**************************************
    Global variables
**************************************/

/**************************************
    Function Proto type
**************************************/
void          initialize_io(void);
unsigned char get_sensor(void);
void          put_data(unsigned char x);

/**************************************
    Main (Entry Point)
**************************************/
void main(void)
{
```

```
    unsigned char now ;

    /*******************************
        initialize port
    *******************************/
    initialize_io();

    /*******************************
        endless loop
    *******************************/
    while ( TRUE ) {
        /* get external information */
        now = get_sensor();
        /* modify or generate information */

        /* put some information to external world */
        put_data(now);
    }
}

/*************************************
*          Port Initialization       *
*************************************/
void initialize_io(void)
{
    /* set port direction */
    PB.DDR = 0xff ;
    /* set port value */
    put_data(0xf6) ;
}

/*************************
* Get a value from Port 7 *
*************************/
unsigned char get_sensor(void)
{
    return P7.DR.BYTE ;
}

/************************
* Put a value to Port B *
************************/
void put_data(unsigned char x)
{
    PB.DR.BYTE = x ;
}
```

(1) #include <j3048f.h>
    This statement signals C that we are going to use the j3048 package.
    The statement is a type of data declaration.
    Later we use the function puts from this package.

(2) #define FALSE 0
    #define TRUE   FALSE+1

    If you define some labels, you can get this C source code easy.
    Magic number is harmful to get C source code.
    When you use some constant numbers, I recommend you define labels.

(3) Our single function is named main .
    The name main is special, because it is the first function called.
    Other functions are called directly or indirectly from main function.

(4) A comment box is enclosed /* and */.
    It's not always necessary for program, but it's useful to get
    when a programmer and other person read the source.

(5) now = get_sensor();
    This line is executable statement instructing C
    to get external information.
    To end a statement C uses a semicolon (;).

(6) put_data(now);
    This line is executable statement instructing C
    to put a value to external world.

(7) C source code is written with small letters.
    small letters   -> statements , variables and pre processor
    capital letters -> labels , constant values and predefined value

* Background about simple program

Data types

C has several native data types.

Data types	Size	Range
char	1 byte	-128 <-> 127
short	2 bytes	-32768 <-> 32767
int	most natural size in MPU (depends on C compiler)	
long	more long than int bit size(depends on C compiler)	
float	depends on C compiler	
double	depends on C compiler	

C allows programmers to make user define data types.

If you want to define data types, use 'struct' or 'union'.

Variable Declarations

C allows us to store values in variables.

Each variable is identified by a variable name.
In addition, each variable has a variable type.

The type tells C how the variable is going to be used
and what kind of value (real, integer, character etc.) it can be hold.

Rules for variables (illegal case)
- x Begin with a number (example => 1st)
- x Contain a "$" (example => work$well)
- x Contain a space (example => the basic)
- x Reserved word (example => while , int , for)

Before you can use a variable in C,
it must be defined in a declaration statement.

The general form of a variable declaration is:

```
data_type  variable_name ;  /* comment */
```

(For example)

```
int avr;    /* average */
```

We can also initialize variables like this :

```
int avr = 2 ; /* average */
```

But many C for embedded system does not allow you the aboves.
Because of reduce program memory and easy to compile.

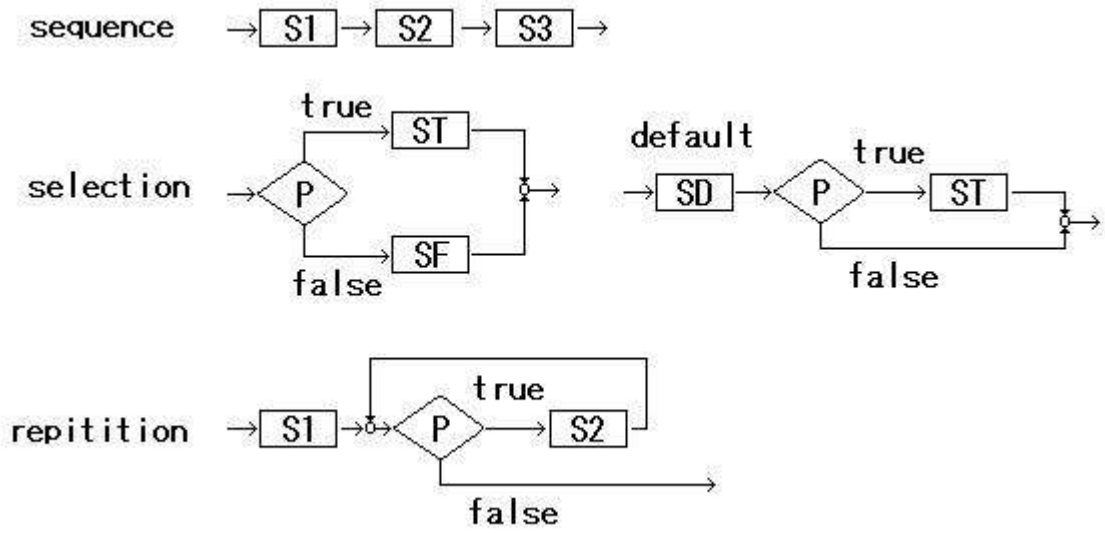
Control Statements

Since C has several control statements,
you will write structured program easily.

And the concept of C language is to write proper program.

if statement (branch)
switch statement (multi branch)
while statement (repetition)
for statement (repetition)

Proper program has one entry and one exit.
Proper program has only three control structure.



if statement

The if statement allows us to put some decision-making into our program.
The general form of the if statement is :

```
if (condition)
    statement;
```

If the condition is true , the statement will be executed.
If the condition is false, the statement will be skipped.

And multiple statements may be grouped by putting them
inside curly braces({}).

```
if (condition) {
    statement1;
    statement2;
}
```

To express complexed condition, we must use relational operators.

Relational Operator	Meaning
<=	Less than or equal to
<	Less than
>	Greater than
>=	Greater than or equal to
==	Equal
!=	Not equal

An alternate form of the if statement is :

```
if (condition)
    statement1;
else
    statement2;
```

If the condition is true, statement1 is executed.
If it is false, statement2 is executed.

When you want more than two conditions, we can use these :

Operator	Meaning
A && B	A and B
A || B	A or B

switch statement

The switch statement is similar to a chain of if/else statements.
The general form of a switch statement is :

```
switch (expression) {
    case constant1:
        statement1;
        . . . .
        break;
    case constant2:
        statement2;
        . . . .
        break;
    /* write more cases */
    default:
        statementN;
        . . . .
        break;
}
```

The switch statement evaluates the value of an expression and
branches to one of the case labels.
Duplicate labels are not allowed, so only one case will be selected.
The expression must evaluate an integer, character, or enumeration.

The case labels can be in any order and must be constants.
The default label can be put anywhere in the switch .

No two case labels can have the same value.

When C sees a switch statement, it evaluates the expression
and the looks for a matching case label.
If none is found, the default label is executed.
If no default is found, the statement does nothing.

while statement

The while statement is one of the loop statements that
allow the program to repeat a section of code any number of times
or until some condition occurs.
The while statement is used when the program needs to perform
repetitive tasks.

The general form of a <U>while</U> statement is :

```
while (condition)
    statement;
```

The program will repeatedly execute the statement inside
the while until the condition becomes false.
(If the condition is initially false, the statement will not be executed.)

for statement
  The for statement allow the programmer to execute a block of code
  for a specified number of times.

  The general form of the for statement is :
    for (expression1 ; expression2 ; expression3)
        body-statement;

  This statement is equivalent to :
    expression1 ;
    while (expression2) {
        body-statement;
        expression3 ;
    }

* Operation
  1 Arithmetic Operation
    Variables are given a value through the use of assignment statements.

    For example:
      answer = (1 + 2) * 4;

    The five simple operators are these:

    Simple Operators

Operator	Meaning
*	Multiply
/	Divide
+	Add
-	Subtract
%	Modulus (return the remainder after division)

    C has several special expressions for arithmetic operation like this:

Normal expression	Special expression
a = a + b	a += b
a = a - b	a -= b
a = a * b	a *= b
a = a / b	a /= b
a = a % b	a %= b
a = a + 1	++a
a = a + 1	a++
a = a - 1	--a
a = a - 1	a--

    Of cource both is accepted, but C programmer usually use the right ones.
    Because of the concept of C reduces key type count,
    C programmer use the right ones.

    When I met a fresh person, I made him writing a simple C program and
    checked which use normal or special expression.
    With expression I have gotton that he or she has C programming.

  2 Logical Operation
    C is called as high level assembly language.
    The reason why we can write logical and shift operation easily
    than assembly language.

    In FA (Factory Automation) programming we must have many
    logical operations. Like bit set or bit clear.

    Please remember H8 assembly instructions : BCLR BTST BSET BNOT
    if you had mastered assembly language.

    The four simple operators are these :
    Simple Operators

Operator	Meaning
&	AND
|	OR (inclusive or)
~	EOR (exclusive or)
!	NOT

3 Shift Operation

In order to write sophistcated program you should use shift operator.

The two simple operators are these :

Simple Operators

Operator	Meaning
<<	Left shift
>>	Right shift

4 Special Expression

C has several special expressions for arithmetic operation like this :

Normal expression	Special expression
a = a & b	a &= b
a = a | b	a |= b
a = a ^ b	a ^= b
a = a << b	a <<= b
a = a >> b	a >>= b

Of cource both is accepted, but C programmer usually use the right ones.
Remember the C concept!

*Functions
Functions break large task into small tasks.

Functions allow programmers to group commonly used code into
a compact unit that can be used repeatedly.

Already have encountered one function, main .

It is a special function called at the beginning of the program.

All other functions are directly or indirectly called from main.

Suppose you want to write a program to get a bit from input port.
You could write out the formula three times, or you could create
a function to do the work.
Each function begin with a comment block containing the following:
  Name   : Name of the function
  Description : Description of what the function does
  Parameters : Description of each of the parameters to the function
  Returns   : Description of the return value of the function

One function to get a bit from input port begins with :

```
/*****************************************/
/* get_a_bit - get a bit from input port */
/*                                       */
/* Parameters                            */
/*     bit  - bit number                 */
/*     port - port number                */
/*                                       */
/* Returns                               */
/*     1 or 0                            */
/*****************************************/
```

The function proper begin with the line :
  int get_a_bit(int bit,int port)

int is the function type.

The two parameters are bit and port.
They are of type int also.

C uses a form of parameter passing called "Call by value".
When our procedure get_a_bit is called , with code such as :
  get_a_bit(4,7);

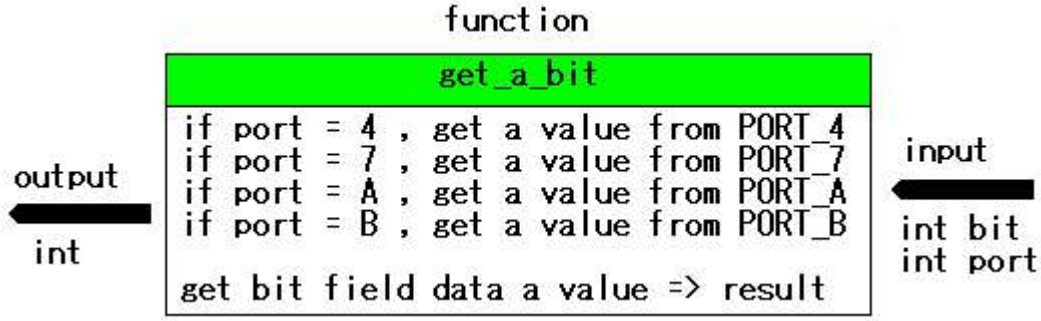
C copies the value of the parameters (in this case 4 and 7) into
the function's parameters (bit and port) and then starts executing
the function's code.

The function get_a_bit area with the statement :

```
  switch ( port ) {
      case 0x04 : result = *p4dr ; break ;
      case 0x07 : result = *p7dr ; break ;
      case 0x0a : result = *padr ; break ;
      case 0x0b : result = *pbdr ; break ;
      default   : result = 0     ; break ;
  }
  result = (result >> bit) & 1 ;
```

What's left is to give the result to the caller.
This step is done with the return statement :

```
   return result ;
```



If you want to write this get_a_bit function after main function,
you must add one line before main :

```
  int get_a_bit(int bit,int port);
```

and write this get_a_bit function after main.

This declaration , which is called a function prototype,
has to agree with the definition and uses of get_a_bit.

Look the get_a_bit function definition :

```
/*****************************************/
/* get_a_bit - get a bit from input port   */
/*                                         */
/* Parameters                              */
/*     bit  - bit number                   */
/*     port - port number                  */
/*                                         */
/* Returns                                 */
/*     1 or 0                              */
/*****************************************/
int get_a_bit(int bit,int port)
{
    int result ;

    switch ( port ) {
        case 0x04 : result = *p4dr ; break ;
        case 0x07 : result = *p7dr ; break ;
        case 0x0a : result = *padr ; break ;
        case 0x0b : result = *pbdr ; break ;
        default   : result = 0     ; break ;
    }
    result = (result >> bit) & 1 ;
}
```

* Array , Pointer , Structure and Union
(Concepts)
Arrays allow us to do something similar with variables.

An array is a set of consecutive memory locations used to store data.
Each item in the array is called an element.
The number of elements in an array is called the dimention of the array.

A typical array declaration is :

```
/* List of data to be sorted and averaged */
int a_list[3];
```

The above example declares a_list to be an array of three elements.

a_list[0],a_list[1], and a_list[2] are individual variables.

To get an element of an array, you use a number called
the index-the number inside the square brackets [ ].

C is a assembly-like language that would like to start counting at 0.
So, the above three elements are numbered 0 to 2.

In hardware application I have defined ring buffer with array.
<Example>

```
#define SIZE 256

int  rd_ptr     ; /* read pointer  */
int  wr_ptr     ; /* write pointer */
char rbuf[SIZE] ; /* ring buffer   */
```

(store data to ring buffer)

```
rbuf[wr_ptr] = x ;
wr_ptr++ ;
wr_ptr %= SIZE ;
```

(get data from ring buffer)

```
x = rbuf[rd_ptr] ;
rd_ptr++ ;
rd_ptr %= SIZE ;
```

(Multidimensional Arrays)
Arrays can be have more than two dimension.
The declaration for a two-dimensional array is :

```
type variable[size1][size2];
```

For example :

```
int matrix[2][4] ;  /* a typical matrix */
```

Notice that C does not follow the notation used in other languages of
matrix[10,12].

To access an element of the matrix , use the following notation :

```
Matrix[1][2] = 10;
```

C allows the programmer to use as many dimensions as needed
(limited only by the amount of memory available).

Additional dimensions can be tacked on :

```
int four_dimensions[10][12][9][5];
```

I didn't apply multidimensional arrays to control hardwares.

Though multidimensional arrays will need many memory area,
single chip micro computer has not sufficient memory capacity.

Instead of multidimensional arrays I have apply some single arrays
to control hardwares.

(strings)
Strings are sequence of characters.

C does not have a built-in string type ;
instead , strings are created out of character arrays.

In fact, strings are just character arrays with a few restrictions.
One of these restrictions is that special character '¥0' (NULL) is used
to indicate the end of a string.

For example :

```
char name[4]

void main(void)
{
    name[0] = 'S';
    name[1] = 'E';
    name[2] = 'C';
    name[3] = '¥0';
}
```

This code consists of a character array of four elements.
Note that we had to allocate one character for the end-of-string delimiter.

String constants consist of text enclosed in double quotes (" ") .
You may have noticed that the first parameter to printf is a string constant.

C does not allow one array to be assigned to another
so we can't write an assignment of the form :

```
name = "Sam";    /* Illegal */
```

Instead we must use the standard library function strcpy to copy the string
constant into the variables.

For example :

```
strcpy(name,"Sam");    /* Legal */
```

String Functions

Function	Description
strcpy(str1, str2)	Copy str2 into str1
strcat(str1, str2)	Concatenate str2 onto the end of str1
length = strlen(str)	Get the length of a <I>str</I>
strcmp(str1, str2)	if str1 equals str2 then 0 , otherwise nonzero

(pointer)
Pointers are often used in C programming and it's one of the most important concept in C.

But, it's a little difficult to get without the assembly language backborn.
So I'll tell you easier as possible as I can.

Pointer Variables
Normal variables store data like integer, float, character and all,
but pointer variables store address.

In Figure11-1 , variable x is stored at address 100 and variable y
is stored at address 106.

These addresses are automatically assigned by the C compiler
to each variable.

They are declared like this :
```
int x;
int y;
```

And ptr is a pointer and declared like this :
```
int *ptr;
```

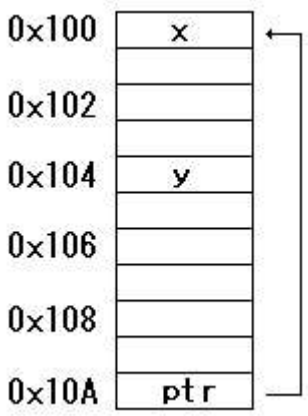


Figure11-1

It means ptr is a pointer to int type.
A variable with pre asterisk is pointer in C.

If you want to declare a variable, don't start from '*' except a pointer.
ptr is a pointer, *ptr is a context of the pointer.

Please remember assembly language specification.
Since CPU's memory has addresses and data, general purpose registers
sometimes are pointers sometimes are holder of data.

(Pointer Operation)
In Figure11-1, ptr point variable x, it means ptr stores an address of
variable x.

And if x is 1, they can be written like this :
```
x   = 1  ;
ptr = &x ;  /* &x = address of x */
```

This & is called an address operator.

And if you write like this:

```
  *ptr = 2 ;  /* *p indicate x */
```

*ptr means the context of x, so x will be 2.

This * is called a dereference operator.
So in this case each value of these variables are :

Variable	Value
x	1
&x	100
*x	
ptr	100
&ptr	112
*ptr	1

Why use 'Structure' , 'Union' and 'Bit-fields' ?
The C program on UNIX, MS-DOS and WindowsXX may not need 'Structure', 'Union' and 'Bit-fields'.

But the C program on single chip computer will need these notation.
Usual single chip computer has bitwise operation like
    'BTST' (bit test)
    'BCLR' (bit clear)
    'BSET' (bit set)
etc.

We can use these bitwise operation easily with 'Structure', 'Union' and 'Bit-fields'.

Example

```
             C statement              Assembly instruction
set 0 bit => variable |= 0x01 ; => MOV.B @VARIABLE,R0
                                    OR.B  #H'01,R0
                                    MOV.B R0,@VARIABLE

set 0 bit => P4.DR.BIT.B0 = 1 ; => BSET #0,@P4DR
```

Compare both C statements.

The upper C statement is to be converted 3 assembly instruction.
But the lower one is to be converted only 1 assembly instruction.

In order to make rapid and small program in C , you should use 'Structure', 'Union' and 'Bit-fields'.

*Structure

A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling.

Structures help to organize complicated data , particularly in large programs, because they permit a group of related variables to be treated as a unit instead of as separate entities.

The general form of a structure definition is :

```
  struct [tag] { member-list } [declarators];
  struct tag declarators;
```

*Union

A union is a variable that may hold (ad different times) objects of different type and sizes, with the compiler keeping track of size and alignment requirements.

The general form of a <U>union</U> definition is :

```
union [tag] { member-list } [declarators];
union tag declarators;
```

We can operate a variable with several diffrent manners.

```
struct st_p4 {                          /* struct P4    */
    unsigned char      DDR;             /* P4DDR        */
    char               wk1;             /*              */
    union {                             /* P4DR         */
        unsigned char BYTE;             /*  Byte Access */
        struct {                        /*  Bit  Access */
            unsigned char B7:1;         /*    Bit 7      */
            unsigned char B6:1;         /*    Bit 6      */
            unsigned char B5:1;         /*    Bit 5      */
            unsigned char B4:1;         /*    Bit 4      */
            unsigned char B3:1;         /*    Bit 3      */
            unsigned char B2:1;         /*    Bit 2      */
            unsigned char B1:1;         /*    Bit 1      */
            unsigned char B0:1;         /*    Bit 0      */
        } BIT;                          /*               */
    } DR;                               /*               */
    char               wk2[18];         /*               */
    union {                             /* P4PCR        */
        unsigned char BYTE;             /*  Byte Access */
        struct {                        /*  Bit  Access */
            unsigned char B7:1;         /*    Bit 7      */
            unsigned char B6:1;         /*    Bit 6      */
            unsigned char B5:1;         /*    Bit 5      */
            unsigned char B4:1;         /*    Bit 4      */
            unsigned char B3:1;         /*    Bit 3      */
            unsigned char B2:1;         /*    Bit 2      */
            unsigned char B1:1;         /*    Bit 1      */
            unsigned char B0:1;         /*    Bit 0      */
        } BIT;                          /*               */
    } PCR;                              /*               */
};

#define P4 (*(volatile struct st_p4 *)0xFFFFC5) /* P4 Address*/
```

In the above case P4 has three different addresses.

DDR (1 byte)
  P4.DDR => 0xFFFFC5

DR (1 byte)
  P4.DDR   => 0xFFFFC5
  P4.wk1   => 0xFFFFC6 (dummy)
  P4.DR.BYTE   => 0xFFFFC7

  P4.DR.BIT.B7 => 0xFFFFC7 bit 7(MSB)
  P4.DR.BIT.B6 => 0xFFFFC7 bit 6
  P4.DR.BIT.B5 => 0xFFFFC7 bit 5
  P4.DR.BIT.B4 => 0xFFFFC7 bit 4
  P4.DR.BIT.B3 => 0xFFFFC7 bit 3
  P4.DR.BIT.B2 => 0xFFFFC7 bit 2
  P4.DR.BIT.B1 => 0xFFFFC7 bit 1
  P4.DR.BIT.B0 => 0xFFFFC7 bit 0(LSB)

  MSB : Most Significant Bit
  LSB : Least Significant Bit

```
 PCR (1 byte)
   P4.DDR         => 0xFFFFC5
   P4.DR.BYTE     => 0xFFFFC7
   P4.wk2[0]      => 0xFFFFC8 (dummy)
     :              :
   P4.wk2[17]     => 0xFFFFD9 (dummy)

   P4.PCR.BYTE    => 0xFFFFDA
   P4.PCR.BIT.B7 => 0xFFFFDA bit 7(MSB)
   P4.PCR.BIT.B6 => 0xFFFFDA bit 6
   P4.PCR.BIT.B5 => 0xFFFFDA bit 5
   P4.PCR.BIT.B4 => 0xFFFFDA bit 4
   P4.PCR.BIT.B3 => 0xFFFFDA bit 3
   P4.PCR.BIT.B2 => 0xFFFFDA bit 2
   P4.PCR.BIT.B1 => 0xFFFFDA bit 1
   P4.PCR.BIT.B0 => 0xFFFFDA bit 0(LSB)
```

Usage of P4

```
/* P4DDR byte access */
void initialize_io(void)
{
    /* Port 4 : out */
    P4.DDR = 0xff ;
}

/* P4DR byte access */
void put_a_byte(int bytex)
{
    P4.DR.BYTE = (unsigned char)bytex ;
}
```

*Bit-fields

A bit-field, or field for short, is a set of adjacent bits within a single implementation-defined storagee unit that we will call a "word."

The general form of a bit-field definition is :

```
 struct {
   total bit size  BitName#1 : BitSize ;
    :
   total bit size  BitName#N : BitSize ;
 } [declarators];
```

You know that H8 port 7 is only input.
If you want to access port 7 with bit access,
define 'structure' BIT like next declaration in header file.

```
 struct st_p7 {                          /* struct P7    */
     union {                             /* P7DR         */
         unsigned char BYTE;             /*  Byte Access */
         struct {                        /*  Bit  Access */
             unsigned char B7:1;         /*    Bit 7      */
             unsigned char B6:1;         /*    Bit 6      */
             unsigned char B5:1;         /*    Bit 5      */
             unsigned char B4:1;         /*    Bit 4      */
             unsigned char B3:1;         /*    Bit 3      */
             unsigned char B2:1;         /*    Bit 2      */
             unsigned char B1:1;         /*    Bit 1      */
             unsigned char B0:1;         /*    Bit 0      */
         } BIT;                          /*               */
     } DR;                               /*               */
 };

 #define P7 (*(volatile struct st_p7 *)0xFFFFCE) /* P7 Address*/
```

Example of structure , union and bit-fields
  See the following definitions.

```
 struct st_itu0 {                        /* struct ITU0  */
     union {                             /* TCR          */
         unsigned char BYTE;             /*  Byte Access */
         struct {                        /*  Bit  Access */
             unsigned char wk  :1;       /*              */
             unsigned char CCLR:2;       /*    CCLR      */
             unsigned char CKEG:2;       /*    CKEG      */
             unsigned char TPSC:3;       /*    TPSC      */
         } BIT;                          /*              */
     } TCR;                              /*              */

     union {                             /* TIOR         */
         unsigned char BYTE;             /*  Byte Access */
         struct {                        /*  Bit  Access */
             unsigned char wk :1;        /*              */
             unsigned char IOB:3;        /*    IOB       */
             unsigned char    :1;        /*              */
             unsigned char IOA:3;        /*    IOA       */
         } BIT;                          /*              */
     } TIOR;                             /*              */

     union {                             /* TIER         */
         unsigned char BYTE;             /*  Byte Access */
         struct {                        /*  Bit  Access */
             unsigned char wk   :5;      /*              */
             unsigned char OVIE :1;      /*    OVIE      */
             unsigned char IMIEB:1;      /*    IMIEB     */
             unsigned char IMIEA:1;      /*    IMIEA     */
         } BIT;                          /*              */
     } TIER;                             /*              */

     union {                             /* TSR          */
         unsigned char BYTE;             /*  Byte Access */
         struct {                        /*  Bit  Access */
             unsigned char wk  :5;       /*              */
             unsigned char OVF :1;       /*    OVF       */
             unsigned char IMFB:1;       /*    IMFB      */
             unsigned char IMFA:1;       /*    IMFA      */
         } BIT;                          /*              */
     } TSR;                              /*              */

     unsigned int     TCNT;              /* TCNT         */
     unsigned int     GRA;               /* GRA          */
     unsigned int     GRB;               /* GRB          */
 };
```

These definitions are in header file '3048f.h'.

* State machine
What is 'State machine' ?

'State machine' is one of mechanism that realizes sequencer.

We can find 'State machine' in embedded micro computer programs.

'State machine' compares current state and now event.

And decides to change or keep 'state'.

'State machine' makes us to write the program of hardware control with ease.

Example => audio tape recoder
Audio tape recoder has several buttons .

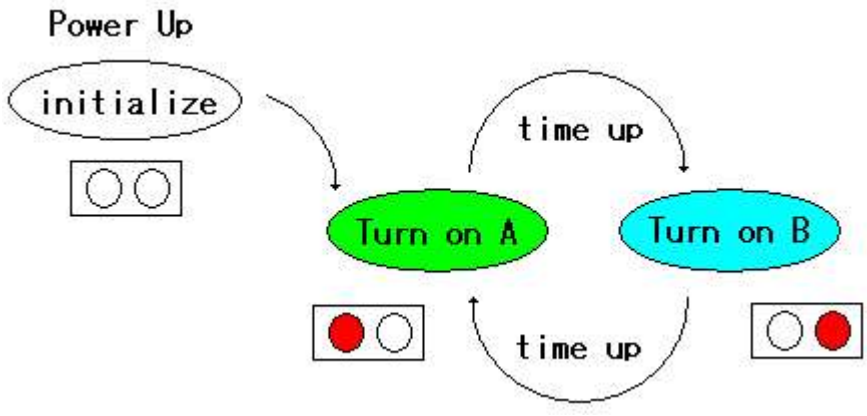
REC FORWORD PLAY REWIND PAUSE EJECT

* Push PAUSE button
If state is PLAY, change state PAUSE and stop.
If state is not PLAY, keep now state.

* Push EJECT button
If state is HOLD TAPE, change state NOT HOLD and eject tape.
If state is NOT HOLD, keep now state.

-> NOW	EVENT	PERFORM	NEXT
PLAY	Push PAUSE	stop	PAUSE
PAUSE	Push PAUSE	play	PLAY
STOP	Push PAUSE	nothing	STOP
NOT HOLD	Push PAUSE	nothing	NOT HOLD
HOLD TAPE	Push PAUSE	nothing	HOLD TAPE
PLAY	Push EJECT	stop & eject	NOT HOLD
HOLD TAPE	Push EJECT	eject	NOT HOLD
NOT HOLD	Push EJECT	nothing	NOT HOLD

About 'State machine' of railroad warning machine



This case has three state :
  1 Initialize
  2 Turn on A
  3 Turn on B

```
 /* initialize */
 initialize();
 /* endless loop */
 while ( TRUE ) {
     /* Turn on A */
     /* Turn on B */
 }
```

Assign some numbers to states
  1 Turn on A -> state = 0
  2 Time out A -> state = 1
  3 Turn on B -> state = 2
  4 Time out B -> state = 3

```
 switch ( state ) {
     case  0 : /* Turn on A */
               break ;
     case  1 : /* Time out A */
               break ;
     case  2 : /* Turn on B */
               break ;
     case  3 : /* Time out B */
               break ;
     default :
               break ;
 }
```

Add some statements.

```
 while ( TRUE ) {
     switch ( state ) {
         case  0 : /* Turn on A */
                   lamp(TURN_ON_A);
                   set_wait(MS100);
                   state = 1 ;
                   break ;
         case  1 : /* Time out A */
                   dec_wait();
                   if ( get_wait() == 0 ) {
                       state = 2 ;
                   }
                   break ;
         case  2 : /* Turn on B */
                   lamp(TURN_ON_B);
                   set_wait(MS100);
                   state = 3 ;
                   break ;
         case  3 : /* Time out B */
                   dec_wait();
                   if ( get_wait() == 0 ) {
                       state = 0 ;
                   }
                   break ;
         default :
                   state = 0 ;
                   break ;
     }
 }
```

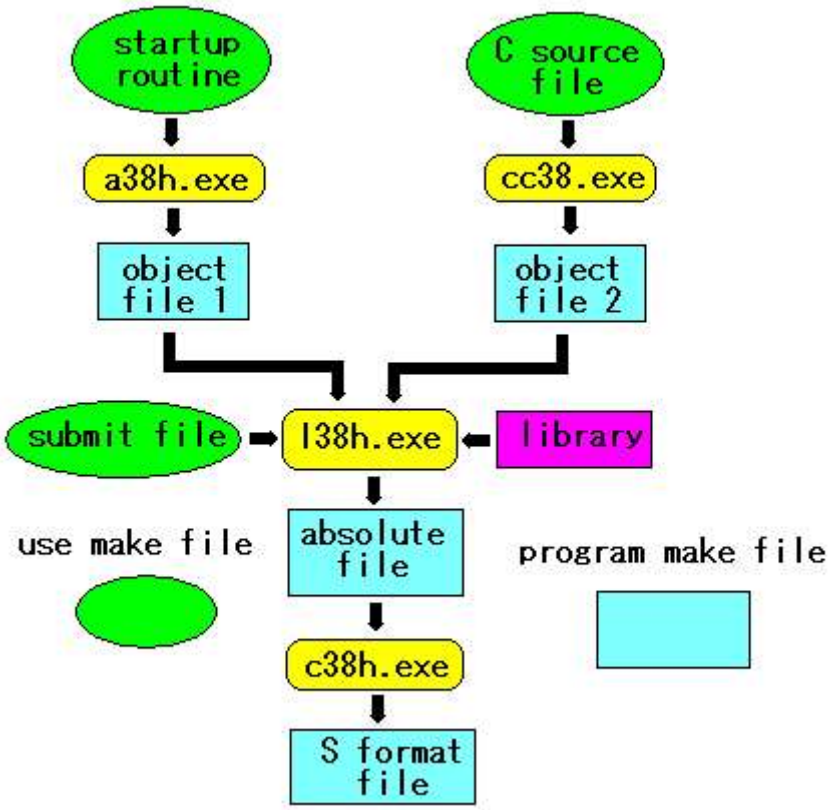
* Compile , Link and Transfer program
  You made some programs with H8 assembly language.
  If you have not the knowledges about H8 assembly language,
  you can not write 'Startup Routine' for C program.

  To compile and link C program for H8/3048, you must write the following files.
    1 Startup Routine file (file extention -> scr)
    2 Source file (file extention -> c)
    3 Submit file (file extention -> sub)

  Example 'Submit file'

```
  INPUT  start,test   -> Input file : 'start.obj' and 'test.obj'
  OUTPUT test         -> Output file : 'test.abs'
  LIB    c38hab       -> Run time library : c38hab.lib
  PRINT  test         -> Print map file : test.map
  START  P(100)       -> Program entry point address $100
  EXIT
```

  *How to compile and link and flow



In DOS prompt type the following :

```
c:¥H8C>cl start test{enter}
```

Content of batch file 'cl.bat'

```
A38H  %1.src
CC38H -include=c:¥h8c -list %2.C
L38H -SUBCOMMAND=%2.SUB
C38H %2.ABS
```

A38H %1.src => Assemble %1.src (Write startup routine)

CC38H -include=c:¥h8c -list %2.C => Compile %2.c
-include=c:¥h8c =>Indicate the store folder of include files
-list =>Show the process of compile

L38H -SUBCOMMAND=%2.SUB =>Link some object files
-SUBCOMMAND=%2.SUB =>Indicate the name of submit command file

C38H %2.ABS => Convert %2.ABS to %2.MOT
%2.MOT => Motolora S format file for flash ROM

If you make the object file from C source file, use batch file 'cc.bat'.

Content of batch file 'cc.bat'

```
cc38h -include=e:¥h8c %1.c
```

Usage :

```
c:¥H8C>cc test{enter}
```

What is 'Startup Routine' ?

In order to run C application program on single chip computer, we must make 'Startup Routine'.

In general 'Startup Routine' is written by assembly language.

'Startup Routine' will define the following articles.

1. Memory assignment like ROM or RAM
2. Program code
3. Vector table
4. Heap area
5. Stack area

Example 'Startup Routine'

```
;************************************************
; Startup routine for H8/3048F
; File Name -> Startup.src
; Writer    -> Kensuke Ooyu
; Date      -> Jan.16 2002
;************************************************

;************************************************
    .cpu 300ha:20           ; H8/300H advanced
;************************************************

;************************************************
; Declare Global labels
;************************************************
    .import _main           ; FUNCTION main
    .import _intimia1       ; FUNCTION intimia1
    .import _err_in         ; FUNCTION err_in
    .import _in_chr         ; FUNCTION in_chr

;************************************************
; Declare Global variables
```

```
;***********************************************
    .export _cnt            ; VARIABLE cnt
    .export _mode           ; VARIABLE mode
    .export _rdata          ; VARIABLE receive  data
    .export _tdata          ; VARIABLE transmit data

;***********************************************
    .SECTION VECT,DATA,LOCATE=H'000000

    .DATA.L H'00100         ; RESET VECTOR

    .ORG    H'0070          ;
    .DATA.L _intimia1       ; ITU1 GRA INTERRUPT
    .ORG    H'00D0          ;
    .DATA.L _err_in         ; SCI0 Error
    .DATA.L _in_chr         ; SCI0 RxD

;***********************************************

;***********************************************
; GOTO MAIN FUNCTION
;***********************************************
    .ORG    H'00100
INIT:
    MOV.L   #H'FEF10,ER7    ; SET STACK POINTER
    LDC     #0,CCR          ; CLEAR INTERRUPT MASK
                            ; NOT USE UI BIT
    JMP     @_main          ; JUMP TO C MAIN FUNCTION

;***********************************************
    .SECTION D,DATA,LOCATE=H'FEF10
;***********************************************
    _cnt:   .res.w  1       ; int   cnt
    _mode:  .res.w  1       ; int   mode
    _rdata: .res.b  1       ; char  receive  data
    _tdata: .rew.b  1       ; char  transmit data

    .end
```

Other example 'Startup Routine'

```
;***********************************************
          .CPU    300HA
;***********************************************

RESET:      .EQU    H'0000          ; Reset Vector Address
STACK_TOP: .EQU    H'FFF10         ; Stack Entry (mode 7)

;***********************************************
; Declare Global labels
;***********************************************
    .IMPORT _main

;***********************************************
    .SECTION VECTOR,COMMON,LOCATE=H'0000
    .ORG    RESET
    .DATA.L START

;***********************************************
    .SECTION P,CODE,ALIGN=2
START:
    MOV.L   #STACK_TOP,ER7  ; Set stack pointer
    JMP     @_main          ; Branch main function

    .end
```