Embedded C Programming for 8051

EE 337 – Microprocessors Laboratory

- Rajbabu Velmurugan
- Sachin Patkar

Prepared by Mahdev Shirwaikar

Embedded C

- **Why ?**

- For programming microcontrollers used in embedded systems, support for direct access to different registers and reading and setting of bits.
- Language extension for the C programming to cater to the programming of Embedded systems.

- **Difference between Embedded C and C ?**

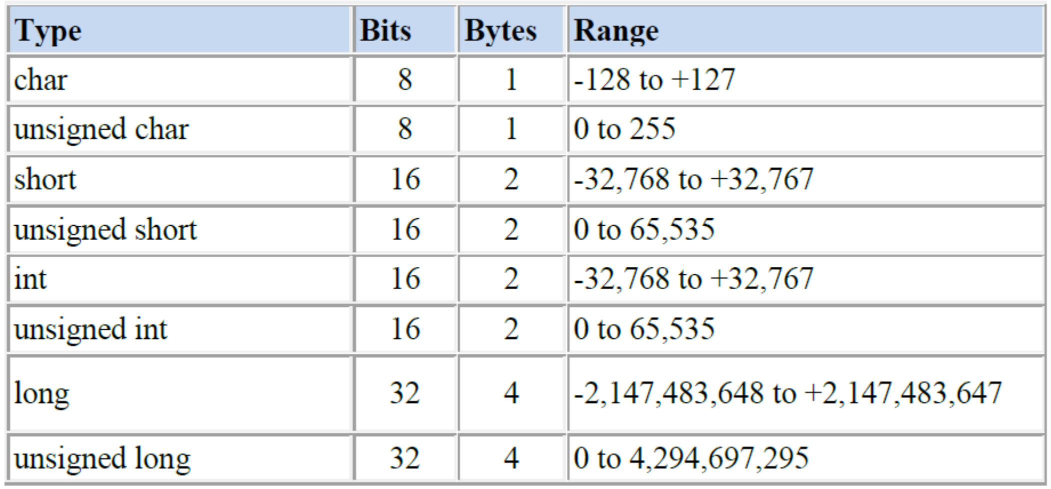
- C : Desktop computers; Embedded C : Microcontroller based applications.
- Embedded C uses limited resources of the processor unlike C which has luxury to use resources like OS, memory of desktop computer.
- C compiler generates OS dependent executables like .exe where as Embedded C requires compilers to create files to be downloaded to the microcontroller/microprocessor where it needs to run.

Keil Compiler

- Most widely used compiler and has an integrated 8051 simulator for debugging.

Variable Types

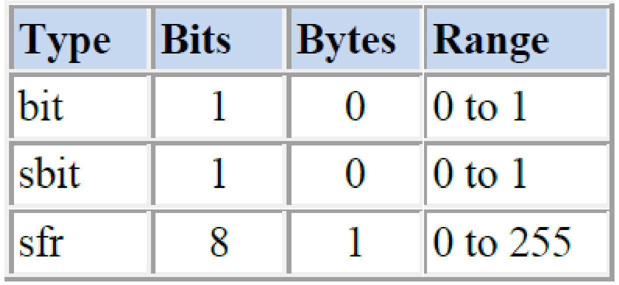
- Standard Types



Type	Bits	Bytes	Range
char	8	1	-128 to +127
unsigned char	8	1	0 to 255
short	16	2	-32,768 to +32,767
unsigned short	16	2	0 to 65,535
int	16	2	-32,768 to +32,767
unsigned int	16	2	0 to 65,535
long	32	4	-2,147,483,648 to +2,147,483,647
unsigned long	32	4	0 to 4,294,697,295

*default is signed unless otherwise specified

- Keil Types

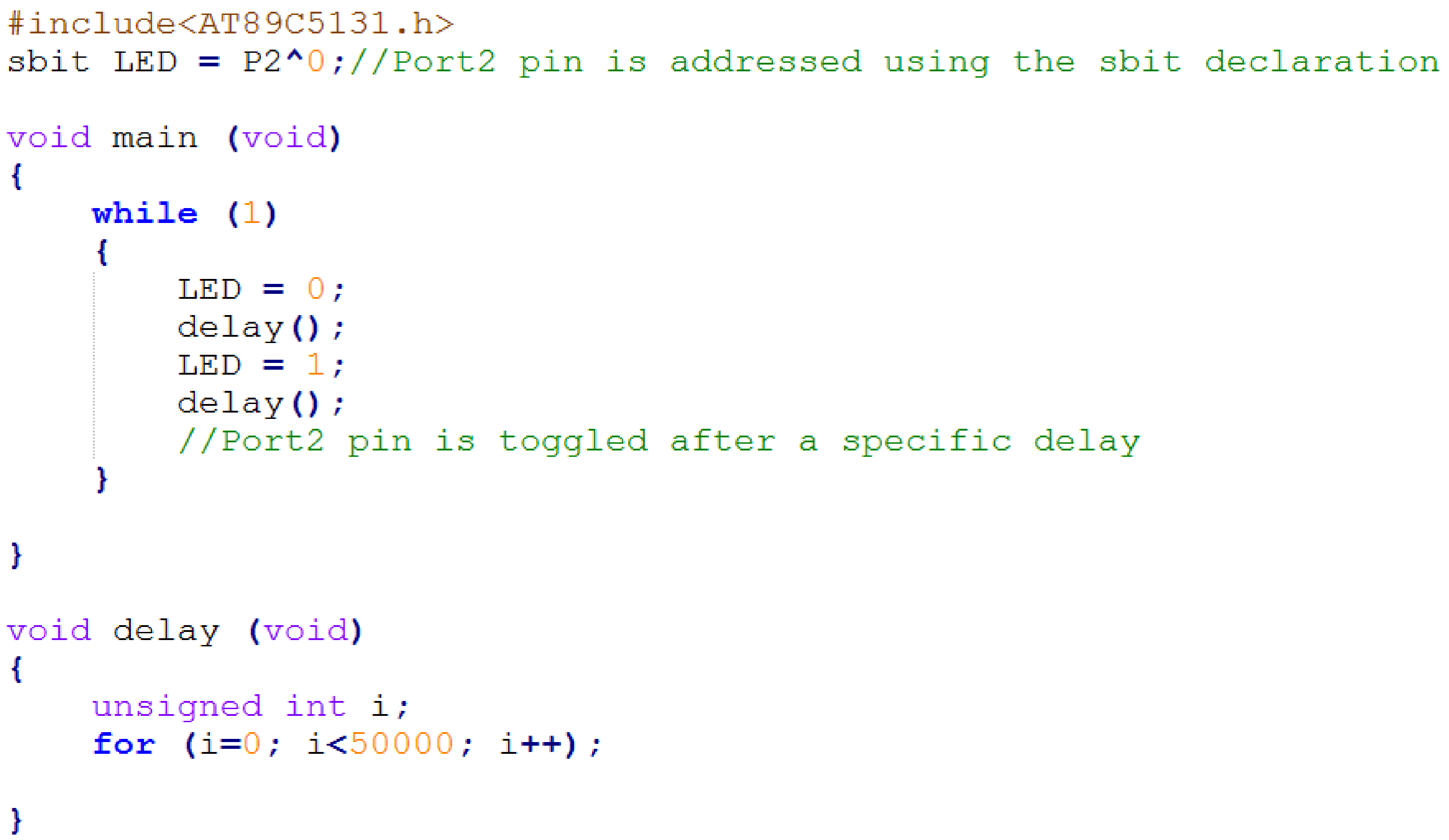


Type	Bits	Bytes	Range
bit	1	0	0 to 1
sbit	1	0	0 to 1
sfr	8	1	0 to 255

Usage Examples : bit

```
//declare two variables, compiler will decide at which addresses
//they will be stored.
bit testbit1 = 0;
bit testbit2 = 1;

//set testbit1 to the value in testbit2
testbit1 = testbit2;

//clear testbit2
testbit2 = 0;
//testbit1 now is 1 and testbit2 is 0.
//Note that the assignment of testbit2 to testbit1 only copies
//the contents of testbit2 to testbit1. It did not change the
//location of testbit1 to that of testbit2.
```

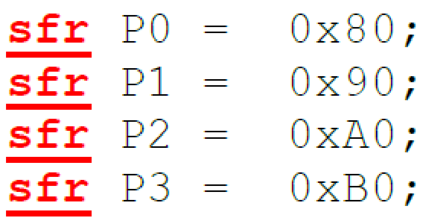
Usage Examples : sbit



```c
#include<AT89C5131.h>
sbit LED = P2^0;//Port2 pin is addressed using the sbit declaration

void main (void)
{
    while (1)
    {
        LED = 0;
        delay();
        LED = 1;
        delay();
        //Port2 pin is toggled after a specific delay
    }

}

void delay (void)
{
    unsigned int i;
    for (i=0; i<50000; i++);

}
```

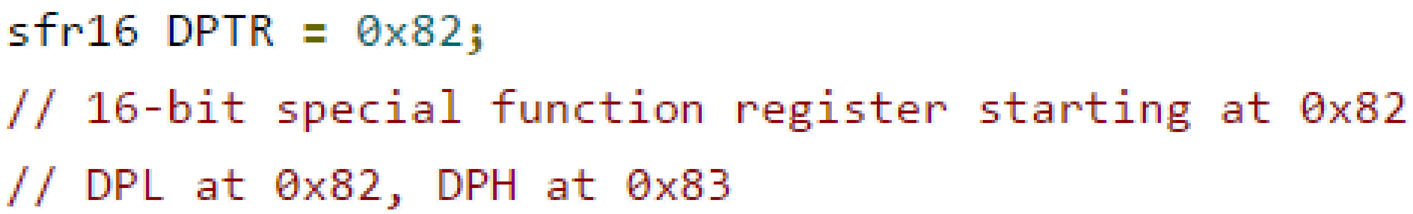
Usage Examples : sfr, sfr16

- **SFR**



```
sfr P0 =  0x80;
sfr P1 =  0x90;
sfr P2 =  0xA0;
sfr P3 =  0xB0;
```

- **SFR16**



```
sfr16 DPTR = 0x82;
// 16-bit special function register starting at 0x82
// DPL at 0x82, DPH at 0x83
```

Operators in C

- Logical operators : AND(&&), OR(||), NOT(!)
- Bitwise operators: AND(&), OR(|),XOR(^), Inverter(~)
- Shift operators: Shift Right(>>), Shift Left(<<)
- Examples

✓ 0x35 & 0x0F = 0x05 /*ANDing*/
✓ 0x04 | 0x68 = 0x6C /*ORing*/
✓ 0x54 ^ 0x78 = 0x2C /*XORing*/
✓ ~0x55 = 0xAA /*Inverting*/
✓ 0x9A >>3 = 0x13 /* Shift right 3 times */
✓ 0x77 >> 4 = 0x07 /* Shift right 4 times */
✓ 0x06 << 4 = 0x60 /*Shift left 4 times */

Functions

- [Return_type] Fucntion_name ([Arguments]) [Memory_model] [reentrant] [interrupt n] [using n]
- Return_type
  - The type of value returned from the function. If return type of a function is not specified, int is assumed by default.
- Function_name
  - Name of function
- Arguments
  - Arguments passed to function

```
unsigned int <function name> (unsigned int var)
{
 ....
 return var;
}
```

Interrupts

```
//this is a function that will be called
//whenever a serial interrupt occurs.
//Note that we need to enable interrupt
//before to make this work
void serial_int (void) interrupt 4
{
    ......
}
```

Interrupt	Vector address	Interrupt number
External 0	0003h	0
Timer 0	000Bh	1
External 1	0013h	2
Timer 1	001Bh	3
Serial	0023h	4

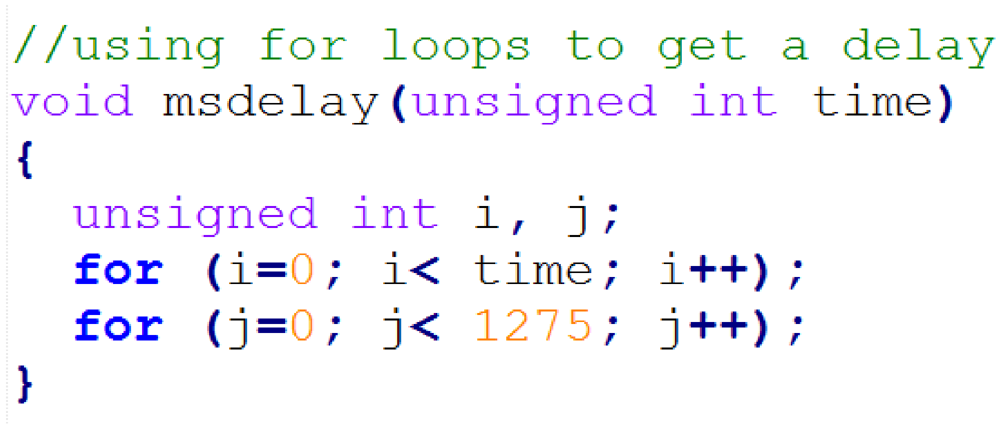
- The Interrupt number is calculated by subtracting 3 from the interrupt vector address and dividing by 8. The five standard interrupts for the 8051 are also shown above.

Time Delay

- Two ways to get time delays in the code

✓ Timers

✓ Simple for loop or nested for loop



```
//using for loops to get a delay
void msdelay(unsigned int time)
{
  unsigned int i, j;
  for (i=0; i< time; i++);
  for (j=0; j< 1275; j++);
}
```

Using

- Keil compiler determines which registers will be used by the interrupt handler function, pushes them out to the stack, executes the handler, and then restores the registers from the stack, before returning to the interrupted code.
- This incurs extra time, especially if a lot of registers will be used. It is preferred that as little time be spent in interrupts as possible.
- To decrease this time, Keil provides an optional extension, using, to the interrupt extension that tells the compiler to change to a new register bank prior to executing the handler, instead of pushing the registers to the stack.

```
//this is a function that will be called whenever a serial interrupt occurs.
//Prior to executing the interrupt, processor will switch to register bank 1
void serial_int (void) interrupt 4 using 1
{
    ......
}
```

Reentrant

- It is possible for a single function to be interrupted by itself.
- A function which can be called in main() as well as interrupt service routine.
- The reentrant extension forces the compiler to maintain a separate data area in memory for each instance of the function.

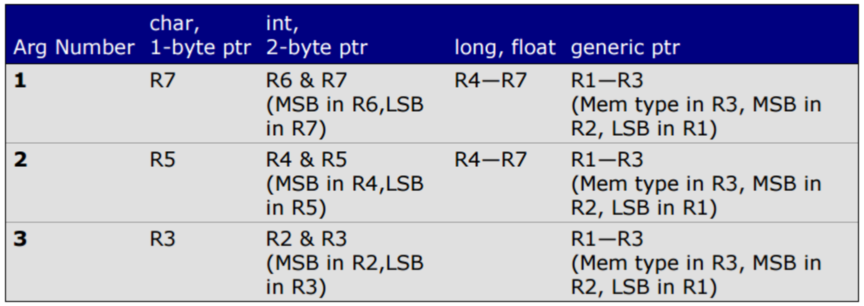
```
//the function may be called both from main program and interrupt handler
int somefunction (int param ) reentrant
{
    ......
    return param;
}
//following interrupt handler also use somefunction
void external0_int (void) interrupt 0
{
    ......
    somefunction(0);
}
//main program also calls somefunction
void main (void)
{
    ......
    while (i==1)
    {
     ......
     somefunction(1);
    }
}
```

Things you might want to know !

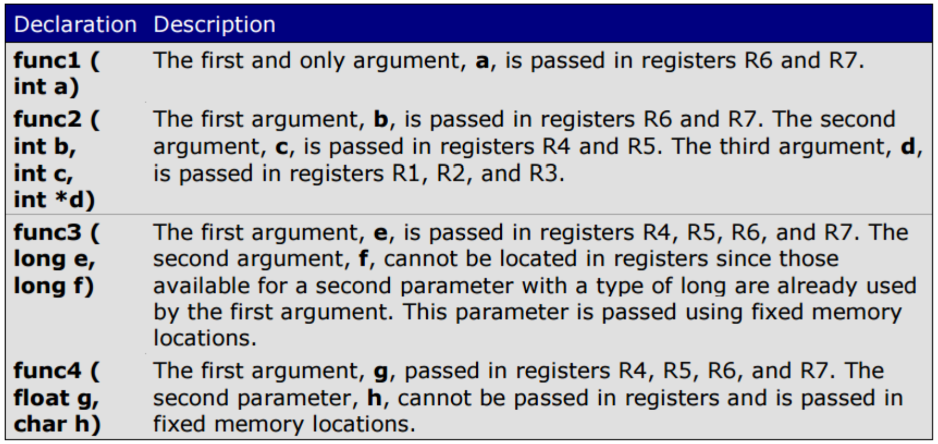
- Can we look at the equivalent assembly code of an embedded C code ?

 - Yes. Using the directive, “#pragma src” at the start of the code, equivalent assembly code can be generated.

- How C functions passes parameters into the registers or fixed memory locations ?



Arg Number	char, 1-byte ptr	int, 2-byte ptr	long, float	generic ptr
1	R7	R6 & R7 (MSB in R6,LSB in R7)	R4—R7	R1—R3 (Mem type in R3, MSB in R2, LSB in R1)
2	R5	R4 & R5 (MSB in R4,LSB in R5)	R4—R7	R1—R3 (Mem type in R3, MSB in R2, LSB in R1)
3	R3	R2 & R3 (MSB in R2,LSB in R3)		R1—R3 (Mem type in R3, MSB in R2, LSB in R1)



Declaration	Description
func1 (int a)	The first and only argument, **a**, is passed in registers R6 and R7.
func2 (int b, int c, int *d)	The first argument, **b**, is passed in registers R6 and R7. The second argument, **c**, is passed in registers R4 and R5. The third argument, **d**, is passed in registers R1, R2, and R3.
func3 (long e, long f)	The first argument, **e**, is passed in registers R4, R5, R6, and R7. The second argument, **f**, cannot be located in registers since those available for a second parameter with a type of long are already used by the first argument. This parameter is passed using fixed memory locations.
func4 (float g, char h)	The first argument, **g**, passed in registers R4, R5, R6, and R7. The second parameter, **h**, cannot be passed in registers and is passed in fixed memory locations.

- Can assembly functions be called from embedded C program and vice-versa ?

 - Yes but we don’t need it for the course. Look at it if you are interested.

Thank you