CpE 213
Digital Systems Design

Lecture 16

Tuesday 10/21/2003

UMR UNIVERSITY OF MISSOURI-ROLLA
The Name. The Degree. The Difference.

Announcements

- Midterm grades have been posted.
- See course objectives under “Course Information” on Blackboard.
- Exam 2: Thursday Nov. 13th
- Review session: Tuesday Nov. 11th, from 7 to 9 pm.
- See me or send me email if you have another exam on that day.

Other announcements

- Winter semester 2004 EIT exam
  - Interested CpE and EE students can see Ms. Martin in EECH 143 for applications.
  - Application deadline: December 1, 2003.
- School of Engineering Honors Program
  - See me THIS WEEK if you are interested.
- IEEE membership drive Oct. 20 through 23
  - Table set up in EE lobby from 9:00 am to 1:00 pm.
  - Questions? Contact Tim Westhoelter, Secretary for IEEE at tswrk5@umr.edu.

C for the 8051

References

- Please download “C for the 8051” from: http://ubermensch.org/Computing/8051/
- Also see Chapters 4 and 5 of the Keil uVision2 “Getting Started” manual.
- See Chapter 3 of “The Final Word on the 8051,” posted on Blackboard.
- See me if you need a handout on C programming in general.

Memory Types

- Memory types (optional)
  - may define the type of memory in which variables are placed.
- Examples:
  - `unsigned char data x;`
  - `char code emsg[ ] = "ERROR";`
  - `int xdata *data ptr; //ptr stored in data`
    `// points to int in xdata`
- Compiler decides if you don’t specify.
  - This is generally the best choice.
  - Function arguments and automatic variables that cannot be located in registers are also stored in the default memory area.

Memory Types

Memory Type	Description
Code	Program memory (64 Kbytes) accessed by opcode MOVC @A+DPTR.
Data	Directly addressable internal data memory; fastest access to variables (128 bytes).
Idata	Indirectly addressable internal data memory; accessed across the full internal address space (256 bytes).
Bdata	Bit-addressable internal data memory; allows bit and byte access (16 bytes).
Xdata	External data memory (64Kbytes); accessed by opcode MOVX @DPTR.
Pdata	Paged (256 bytes) external data memory; accessed by opcode MOVX @Rn.

- Place frequently used variables in internal data memory and less frequently used variables in external data memory.

Location

- Specifying the location of variables is optional. Compiler decides if unspecified.
- See examples in lect16_example.c
- Two ways of specifying:
  - global variables: _at_ addr

```
char data x _at_ 0x2A; //Keil requires 0x
int xdata y _at_ 0x5280;
```

  - direct access within code using built-in commands
    - XBYTE, XWORD for external data memory
    - DBYTE, DWORD for internal data memory
    - CBYTE, CWORD for code memory
    - See example on next slide.

Example

```
//include built-in memory access commands
#include <absacc.h> //REQUIRED!
main(){
  ⋮
  //read from XMEM location 0x5280
  x = XBYTE[0x5280];
  //write x to data memory location 0x42
  DBYTE[0x42] = x;
  ⋮
}
```

New Data Types

type	bits	range	description
bit	1	0-1	bit in bit memory
sbit	1	0-1	SFR bit at specified location
sfr	8	0-255	SFR byte at specified location
sfr16	16	0-64K	16-bit SFR beginning at specified location

- Last 3 types must be global and must specify address.
- Compiler automatically converts between data types when the result implies a different data type.

Example

```
sfr P0 = 0x80;       //sfr P0 located at addr 0x80
sfr seven_seg = 0x90;
sfr16 DPTR = 0x82;  //16-bit sfr named DPTR at 0x82
sbit z = P0^3;       // z is bit sfr located at bit 3 of P0
sbit carry = 0xD7;  // carry is at bit addr 0xD7

main(){
   bit y;                          // a bit variable
   char bdata x = 0xFF;     // a byte located in bit mem
   P0 = 0x00; // clear sfr P0
   y = 1;              // set bit y
   z = y;              // set sbit z equal to bit y
}
```

After executing:

P0 = _____

y = __, z = ____ => P0 = _____

Group Exercise

Memory Models

- Three basic models: small, compact, large.
- SMALL: Total RAM 128 bytes
  - Will support code sizes up to about 4K but a constant check must be kept on stack usage.
  - The number of global variables must be kept to a minimum.
- COMPACT: Total RAM 256 bytes off-chip, or 128 or 256 bytes on-chip
  - Suitable for programs where, for example, the on-chip memory is applied to an operating system.
  - Rarely used on its own, usually with SMALL.
  - Especially useful for programs with a large number of medium speed 8 bit variables, for which the instructionMOVX A, @R0 is very suitable.

Memory Models

- LARGE: Total RAM up to 64Kb, 128 or 256 bytes on-chip
  - Permits slow access to a very large memory space and is perhaps the easiest model to use.
  - Again, not often used on its own but in combination with SMALL.
  - As with COMPACT, register variables are still used, so efficiency remains reasonable.
- Specify model with directive: `#pragma SMALL` or set in project properties in Keil.

Functions

- Specification:

```
return_type name(args) mem_model reentrant
  interrupt n using n {
    //contents of function
}
```

  - everything other than `return_type, name, and args` is optional; do NOT use in prototype
- Examples:

```
char myfun(char x) small {
    //contents of myfun
}
void afun(void) interrupt 0 using 3{
    //contents of afun
 }
```

- Also see lect16_example.c

Function Specification

- interrupt n: function is called in response to interrupt n (n can be 0,1,2,…)
- using n: function will use register bank n exclusively (unless you specify otherwise)
- re-entrant: function may call itself
  - useful for recursion
  - all variables often saved on stack
  - stack may be external
  - default is NOT re-entrant
- memory model can be: small, compact, large

For next lecture

- Download “C for the 8051” from: http://ubermensch.org/Computing/8051/
- HW 5 is due.
- Review lecture notes
- We will discuss the semester project during the lecture.

```
/****************************************
lect16_example.c
D. Beetner

A quick bit of code to show how to do a
few things in C for the 8051.  This program
isn't really useful for anything else
(maybe as a template?).

****************************************/

// include special function register defs
// -- note: commented out for this example (so don't
// -------- redefine some SFRs
// #include <reg51.h>

// include mem-access functions (XBYTE[], etc)
#include <absacc.h>

// Declare some SFRs -- MUST BE GLOBAL!
        sfr P0 = 0x80;              //sfr named P0 located at addr 0x80
        sfr P1 = 0x90;

        sfr16 DPTR = 0x82;  // 16-bit sfr named DPTR at addr 0x82

        sbit z = P0^3;              // z is bit 3 of P0
        sbit carry = 0xD7;      // carry is at bit addr 0xD7

// Declare bit-addressable memory
// (must use sbits to access individual bits, hence global)
        unsigned char bdata bits;       // a byte in bit-mem
        sbit bit4 = bits^4;                     // bit4 is bit 4 of bits

// Declare some variables located _at_ a particular address
// MUST BE GLOBAL to use _at_
        char data myabs _at_ 0x42; // located at data addr 42H
        char xdata seven_seg _at_ 0x5280;       // at xdata addr 5280H

// function prototypes (declarations)
        char myfun(char);
        void afun(void);

// main routine
void main(void){

        //local variables
        bit y;                                  // a bit variable
        unsigned char bdata x = 0x2A;  // a byte located in bit mem
                                                // initialized as 0x2A
        int data aword;                         // in data mem
        char code emsg[] = "ERROR";  // in code mem
        unsigned char xdata xsensor;   // in xdata

        char blah;                      // mem space SPEC'D BY COMPILER!
        int blab;                               // MUCH EASIER ON PROGRAMMER!
        float blob;
```

```
//          sfr P2 = 0xA0;          // NOT ALLOWED

            // Some bit operations
            P0 = 0x00;              // clear sfr P0
            y = 1;                  // set bit y
            z = y;                  // set sbit z equal to bit y

            bits = 0;   // set variable bits (in bit mem) to 0
            bit4 = 1;   // set bit bit4 to 1


            // play with some variables at known locations
            myabs = 0x2A;           // set location 42 (myabs) to 0x2A
            DBYTE[0x42] = 0;

            // call a function
            myabs = myfun(blah);

}


// Define contents of function myfun
char myfun(char x) small{       // runs using a small memory model
            return x--;
}

// Define afun as a function responding to interrupt 0
// and using register bank 3
void afun(void) interrupt 0 using 3{
            static char x;

            x++;
}
```