CpE 213
Digital Systems Design

Exam 2 Review

Tuesday 11/11/2003

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

Announcements

- Exam 2: Thursday Nov. 13th
  - Topics to be covered and practice exams have been posted.
  - Will cover everything up to, but not including interrupts. Emphasis will be on material not covered in exam 1.
  - Can work in groups. Writeup should be done completely on your own. Mention names of anyone you collaborated with.
  - Exam will be take-home, due at 4:30 pm on Friday 11/14.
  - Drop off at my office or leave with Katie (next door).
  - Email or slide under door at your own risk.
- Download "Writing C Code for the 8051" from Blackboard.

Differences between C and C++

- The main difference between C and C++ is that C isn't object-oriented. Okay - that doesn't actually tell you what you want to know; here are some details:
- structs don't copy in C. That is, if a and b are structs then the line a = b; doesn't work. Nor will structs be fed into functions as arguments. The only way to deal with them sensibly is to use pointers to them, or to write functions to copy their elements explicitly. classes (with private members) don't exist in C.

Differences between C and C++

- There is no operator and function overloading in C. If a function has a name, then that's it---you can't have another version with the same name that does the same thing with different arguments, as you can in C++. The fact that << and >> (left and right shift) do output and input in C++ is a consequence of the ability of the language to overload those operators. Output and input in C are handled by functions called printf and scanf respectively (these also work in C++ if you want, of course).
- In C storage allocation and de-allocation are not handled by new and delete but by a function called malloc. In general, the whole process is a bit more messy in C, but is not too bad once you get the hang of it.

Differences between C and C++

- There are many more detailed differences, of course---see the books on the two languages for information on those. In general, C++ is neater and easier to read than C, and it does not compile to less efficient code *as long as you know what you are doing*. An example inefficiency here would be sending a large struct or class as a function argument by copying , as opposed to by reference using const X& fred (or whatever). The copying will eat stack space and waste time.
- The only real reason for using C in preference to C++ is that the machine on which you wish to compile your program doesn't have a C++ compiler (such as Keil).

Slide by Dr. D. N. Johnston - University of Bath

Control Flow Summary

- if-else: decision making
- else-if: multi-way branch
- switch: another multi-way branch
- while and for: test at top of loop
- do while: test at bottom of loop
- break and continue
- goto and labels (avoid!)

if Statement

- if (*expression*)
  action

Example:

```
char a1 = 'A', a2
  = 'C';
int match = 0;
if (a1 == a2) {
  match++;
}
```

if-else Statement

- if (*expression*)
  action 1
  else
  action 2

Example:

```
char a1 = 'A', a2 = 'C';
int match = 0, gap = 0;
if (a1 == a2) {
   match++;
} else {
   gap++;
}
```

Note: Also see the "switch" statement.

for Statement

for(*expr1; expr2; expr3*)
  action

- *Expr1 – defines initial conditions*
- *Expr2 – tests for continued looping*
- *Expr3 – updates loop*

Example

```
sum = 0;
for(i = 1; i <= 4; i++)
   sum = sum + 1;
```

Iteration 1: sum=0+1=1

Iteration 2: sum=1+2=3

Iteration 3: sum=3+3=6

Iteration 4: sum=6+4=10

while Statement

```
while (expression)
   action
```

Note: skipping do while

Example

```
int x = 0;
while(x != 3) {
   x = x + 1;  2
}
```

Infinite loop!

Iteration 1: x=0+1=1

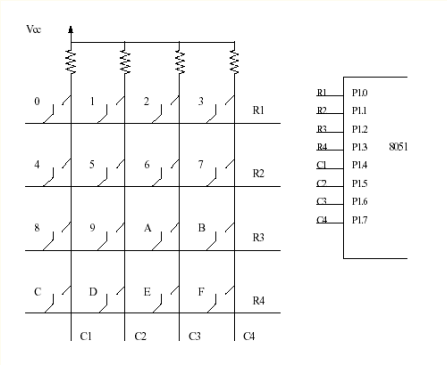
Iteration 2: x=1+1=2

Iteration 3: x=2+1=3

Iteration 4: don't exec

Keypad Interfacing

- 16 Keys arranged as 4x4
- Algorithm:
  - Drive a “0” on a row
  - Read all the columns
  - If any key had been pressed, its column will be “0”, else 1
  - Keep repeating in a loop for each successive row
- Example:
- Switch 4 is pressed
  - R1← 0, C1:C4 = 1111
  - R2← 0, C1:C4 = 0111
- Switch 2 is pressed
  - R1← 0, C1:C4 = 1101



Slide adapted from UT Dallas.

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 = 0x00

y = 1, z = y => P0 = 0000 1000 = 0x08