CpE 213
Digital Systems Design

Lecture 15

Thursday 10/16/2003

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

Announcements

- Exams can be picked up during office hours.
- Send me one email per group with names of all group members. Indicate which members are taking CpE 214.
- Assignment #5 has been posted. Work on it as soon as possible after this lecture.

Exam 2

- Date: Thursday Nov. 13th
- Review session: Tuesday Nov. 11th, from 7 to 9 pm
- Any objections to the new date of the exam or the time or date of the review session should be sent to me by 5 pm on Monday Oct. 20.

Thought for the Day

from comp.arch.embedded newsgroup

"Always code as if the guy that ends up maintaining and/or testing your code is a violent psychopath who knows exactly where you live."

Assembly Language Programming for the 8051
ISM Chapter 7
Sections 7.1 – 7.5

ASM Directives

- An ASM directive (or pseudo-opcode) has the same format as assembler instruction (opcode), but it is NOT assembled into machine code.
- Directives function as commands to the assembler to define properties of the code. These properties include items such as:
  - Addresses to start code or data segments
  - Allocation of memory space
  - Constant or symbol definitions
- Beware! Assembler directives vary from one software (assembler) vendor to another.
- Keil directives have been posted on Blackboard.

More Assembler Directives

- EXTERN: declare variables from other modules (ASM files)
- PUBLIC: declare variable to be used in other module
- USING: tell compiler current register bank

  Ex:
  ```
  USING 1
  MOV A, AR0
  MOV A, R0
  ```
  Will still be switching banks with PSW.
- EQU: create “assembler” constant

  Ex:
  ```
  answer EQU #42H
  MOV A, answer
  ```
- END: last statement in source file

Immediate Data

Example:

- MOV A, #2AH ; hex
- MOV A, #42D ; decimal
- MOV A, #0010 1010B ; binary
- MOV A, #42 ; decimal is default
- MOV A, #41+1

SFRs

- Example:
  - SETB D7H
  - SETB
- Example:
  - SETB D0H
  - SETB
- SFRs can generally be referenced by name.
- Reminder: SFRs are generally accessible only by direct addressing. See 8051 Data Sheet.

General Program Layout

- Data Segments
  - declare segments
  - declare variables
- Code segments
  - declare segments
  - declare constants

C Programming Review

Some slides adapted from C reviews by Dr. Warter-Perez and Mr. Anubhav Gupta

Reference Links for C

- An online C Primer:
  http://occs.cs.oberlin.edu/faculty/jdonalds/341/CPrimer.html
- Another C Primer:
  - http://www.vectorsite.net/tscpp.html
- A more detailed reference:
  - http://www.cs.cf.ac.uk/Dave/C/CE.html
- A number of books on the topic are listed in your course syllabus.

C Basics - Variables

- Variables have a data type and name or identifier
- Identifiers
  - Have the following restrictions:
    - Must start with a letter or underscore (_)
    - Must consist of only letters, numbers or underscore
    - Must not be a keyword
  - Have the following conventions:
    - All uppercase letters are used for constants
    - Variable names are meaningful – thus, often multi-word
      - Convention 1: alignment_sequence
      - Convention 2: AlignmentSequence

C Basics – Data Types (1)

- 3 basic data types: integer, float, char
  - Integer (int) – represent whole numbers
    - long (32-bits same as default), short (16-bits)
      - System dependent
    - signed (positive and negative, default), unsigned (positive)
  - Ex 1: define an integer variable y
  - Ex 2: define an unsigned short integer variable month initialized to 4 (April)
    -

C Basics – Data Types (2)

- Floating point – represent real numbers
  - IEEE Standards
    - Single-precision (float, 32-bits)
    - Double-precision (double, 64-bits)
  - Ex 1: define a single-precision floating-point variable named error_rate and initialize to 3.5
    -
  - Ex 2: define a double-precision floating-point variable named score and initialize it to .004 using scientific notation
    -

C Basics – Data Types (3)

- Character – represent text
  - ASCII – American Standard Code for Information Interchange
  - Represents characters, numbers, punctuation, spacing and special non-printable control characters
  - Example ASCII codes: 'A' = 65, 'B' = 66, … 'a' = 97, 'b' = 98, '\n' = 10
  - Ex 1: define a character named AminoAcid and initialize it to 'C'
    -
    -

Summary of Data Types

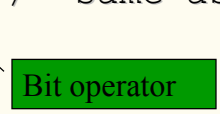
data type	size (bytes)	values (range)
char	1	-128 to 127
short	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647
float	4	3.4E+/-38 (7 digits)
double	8	1.7E+/-308 (15 digits long)

Warning: Using the float and double data types is very difficult on the 8051.

C symbol definition

- Normally put into *include* files (*.h)
- Some examples (see reg51.h):

```
sfr P0   = 0x80;
sfr P1   = 0x90;
sfr P2   = 0xA0;
sfr PSW  = 0xD0;
sbit CY   = 0xD7;
sbit P0_1 = P0^1; /* same as 81h */
```



Arithmetic Operators

Operator		Example
		`int x, y=5, z=3;`
+	add	`x = y + z;` x = 8
-	subtract	`x = y – z;` x = 2
*	multiply	`x = y * z;` x = 15
/	divide	`x = y / z;` x = 1
%	modulus	`x = y % z;` x = 2

Auto Increment and Decrement

- Pre-increment/decrement
  - `y = ++ x;` *equivalent to*
  - `y = --x;` *equivalent to*
- Post-increment/decrement
  - `y = x++;` *equivalent to*
  - `y = x--;` *equivalent to*

```
                    x = 3
x = x+1;    x = 4
y = x;      y = 4
x = x-1;    x = 2
y = x;      y = 2

y = x;      y = 3
x = x+1;    x = 4
y = x;      y = 3
x = x-1;    x = 2
```

Relational and Logical Operators

- Relational operators
  - == equal
  - != not equal
  - > greater than
  - >= greater than or equal
  - < less than
  - <= less than or equal
- Logical operators
  - && and
  - || or
  - ! not

Relational Operators

- Assume x is 1, y is 4, z = 14

Expression	*Value*	*Interpretation*
`x < y + z`	1	True
`y == 2 * x + 3`	0	False
`z <= x + y`	0	False
`z > x`	1	True
`x != y`	1	True

Logical Operators

- Assume x is 1, y is 4, z = 14

Expression	*Value*	*Interpretation*
`x<=1 && y==3`	0	False
`x<= 1 || y==3`	1	True
`!(x > 1)`	1	True
`!x > 1`	0	False
`!(x<=1 || y==3)`	0	False

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

1-D Arrays

- char amino_acid;
  - Defines one amino_acid as a character

1 cell 

- char sequence[5];
  - Defines a sequence of 5 elements of type character (where each element may represent an amino acid)

5 cells with indices

0	1	2	3	4

Initializing Arrays

- char seq [5] = “ACTG”;

5 cells with values

'A'	'C'	'T'	'G'	'\0'

seq[0] = 'A'

seq[1] = 'C'

...

- float hydro[6] = {-0.2, 0, -0.67, -3.5, 2.8};

5 cells with values

-.2	0	-.67	-3.5	2.8	0

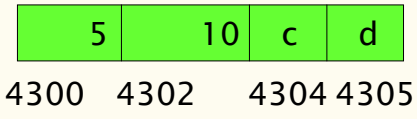
hydro['A' - 'A'] = -.2 hydro['C' - 'A'] = -.67 hydro[5] = 0

- No initialization – each cell has “garbage” – unknown value

Pointers

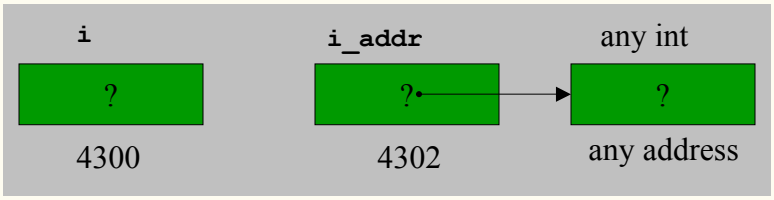
- Pointers are variables that represent an address in memory.
- That location a pointer addresses contains another variable.

```
int i = 5, j = 10;
char c = 'c', d = 'd';
```

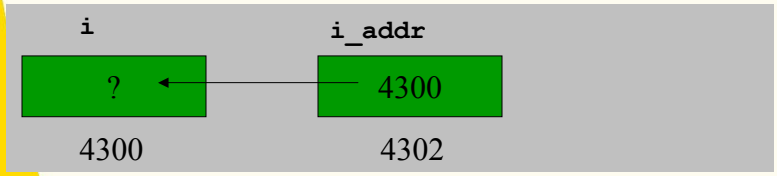


Using Pointers (1)

```
int i;          /* data variable */
int *i_addr;    /* pointer variable */
```

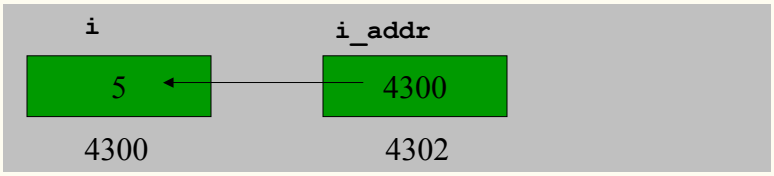



```
i_addr = &i;     /* & = address operator */
```

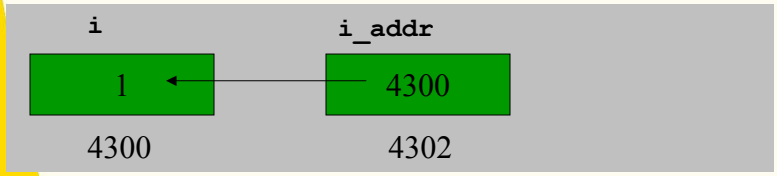


Pointers Made Easy (2)

```
*i_addr = 5;        /* indirection operator */
```




```
int k = *i_addr;/* indirection: k is now 5 */
i = 1;          /* but k is still 5 */
```



Subprograms

- Functions
  - `x= f(y)`
  - `f(x,y)`
  - Similar to procedures and subroutines in Fortran or Pascal
- Implemented with LCALL and RET
- Functions are useful for:
  - making code easier to read (structure)
  - reusing code

Program Structure

- Makes code easier to read:

```
init();
while(1) {
  doit();
  if (error) fixup();
  }
```

- Reuse blocks of code:

```
move(a,b); move(c,d)
```

Example function

PULSEP1.C

```
#include <reg51.h>
sbit P1_1= P1^1;
void pulseP1B1(void){
 P1_1= 1; P1_1= 0; }
void main(void){
 pulseP1B1();
 pulseP1B1();
}
```

```
 ; FUNCTION pulseP1B1 (BEGIN)
0000 D291           SETB    P1_1
0002 C291           CLR     P1_1
0004 22             RET
; FUNCTION main (BEGIN)
0000 120000   R     LCALL   pulseP1B1
0003 120000   R     LCALL   pulseP1B1
```

Function Parameters

- Function arguments are passed “by value”.
- What is “pass by value”?
  -
- What does this imply?
  -

Example 1: swap_1

```
void swap_1(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
```

Q: Let x=3, y=4, after swap_1(x,y); x =? y=?

Example 2

- pass by value
  - `f(x){ x=2 }; //function definition`
  - `f(5);  //does this set 5=2? (no!)`
  - example:

```
; FUNCTION _f (BEGIN)
0000 7F02           MOV     R7,#02H
0002 22             RET
; FUNCTION main (BEGIN)
0000 7F05           MOV     R7,#05H
0002 120000  R      LCALL   _f
```

Output parameters

- So, how do we return something to the caller?
- Pointer parameters

```
void f(char *p){*p= 2}
main() { f(&x); } //sets x=2
```

- Non-void return value:

```
char f(void) {return 2; }
main() { x=f() } //sets x=2;
```

Example 1: swap_2

```
void swap_2(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
```

Q: Let x=3, y=4, after swap_2(&x,&y); x =? y=?

Example 2

```
void f(char *p){*p= 2}
main() { f(&x); }      //sets x=2
```

```
 ; FUNCTION _f (BEGIN)
0000 A807           MOV     R0,AR7
0002 7602           MOV     @R0,#02H
0004 22             RET
; FUNCTION main (BEGIN)
0000 7F00     R     MOV     R7,#LOW x
0002 120000   R     LCALL   _f
```

Non-void return value

```
char f(void) {return 2; }
main() { x=f() } //sets x=2;
```

```
 ; FUNCTION f (BEGIN)
0000 7F02          MOV     R7,#02H
0002 22            RET
; FUNCTION main (BEGIN)
0000 120000  R     LCALL   f
0003 8F00    R     MOV     x,R7
```

Group Exercise (Take-Home)

1. Make a list of all of the C topics that are fuzzy for you.
2. Look at the code in example.c (posted on Blackboard).
3. Find:
   - one critical error
   - two instances of poor programming
4. Bring your answers to class on Tuesday.

Group Exercise

Modification

Shift Operator

```
x=0xC2;    // x = 1100 0010
           //shift x left by one bit
x = x<<1; // x = 1000 0100
```

How would we code `x = x<<3` in ASM?

Functions and Variable Scope

- Functions:
  - prototype at top
  - pass/return nothing – use type void
  - generally define after main
- Variable Scope
  - may only declare vars at top of program/function
  - globals known everywhere (BAD!)
  - locals known only within their own function
  - static vars keep their value between calls
  - other variables are initialized at each function call

Functions and Variable Scope

```
int blah (int x);
int z;

void main(void){
   int x,y;

   y = 0x5280;
   x = 1;
   z = blah(x);
   z = y;
   x ++;
   z = blah(x);
}

int blah (int v){
   int y = 0;
   static int x=0;

   y ++;
   x ++;

   return v*42;
}
```

Modification

- What if we wanted to return two variables?
- Use pointers