C Language – Advanced Concepts
Switch Statement

- Syntax

```c
switch (expression) {
    case const_expr1:
        statement1;
        break;
    case const_expr2:
        statement2;
        break;
    :,
    default:
        statementn;
}
```

**Example**

```c
switch (state) {
    case 1:
        nextstate = 2;
        break;
    case 2:
        nextstate = 3;
        break;
    case 3:
        nextstate = 1;
        break;
    default:
        nextstate = 1;
}
```

*Instead of using “switch” you could always use a series of if-then-else statements instead*

*The “break” statement causes control flow to break out of the currently executing statement*
Pointers and Addresses

• A pointer is a variable that holds the address of a variable
• Syntax for pointer declaration:
  
  \[ \text{type\_name} \ *\text{pointer\_name}; \]

• Examples
  
  ```
  int *ax; // declare a variable ax, that is a pointer to an int
  char *cp; // declare a variable cp, that is a pointer to a char
  ```

• Use the dereferencing operator * to access the value pointed by a pointer
  
  ```
  int a, *b;
  ...
  /* get the value pointed to by b */
  a = *b;
  ```

• Pointers provide a way to return multiple data items from a function via function arguments
Pointers and Addresses (continued)

• Use the unary operator & to assign the address of a variable to a pointer
• Example:
  ```
  int x, y;
  int *ip;

  ip = &x;
  y = *ip; // y gets the value of x
  ```

• An array name is actually a pointer to the first element
• When accessing an array you are actually dereferencing a pointer
• Example: if an array is defined as
  ```
  int ax[10];
  ```
• then
  ```
  i = ax[0]; is the same as i = *ax;
  i = ax[1]; is the same as i = *(ax+1);
  i = ax[2]; is the same as i = *(ax+2);
  ```
Example

- Write a function to square every element in an array
  - We pass the array name into the function
  - Remember that the array name is just a pointer to the first element
  - Function can’t modify the pointer, but it can modify what it points to

- Code

```c
void squareall(int a[], int n) {
  int i;
  for (i=0; i<n; i++)
    a[i] = a[i]*a[i];
}

void main(void) {
  int m[] = {1,2,3,4};
  squareall(m, 4);
  /* m now contains 2,4,9,16 */
}
```
Type casting

- Type casting forces a variable of some type into a variable of different type
- The format is
  \[(\text{type}) \; \text{variable}\]

- Example:
  - Assume that \(x_1, x_2\) are of type \text{int} (16 bits)
  - Then the expression \(x_1 \times x_2\) is also of type \text{int}
  - However, the product may be too big to fit into an \text{int}
  - It would however, fit into a \text{long} (32 bits)
  - If we first convert \(x_1, x_2\) into type \text{long}, then the product will be of type \text{long}:
    ```c
    long result;
    int x1, x2;
    ...
    result = ((long) x1) \times ((long) x2);  // no overflow
    ```
Working with fixed addresses

- How to read from a certain memory address?
  - Example: say you want to read the contents of memory at 0x1000

- You can’t simply do
  \[ c = 0x1000; \]

- Instead you have to do
  \[
  \begin{align*}
  \text{char} & \quad \text{*address, n;} \\
  & \\
  \text{address} & \quad \text{=} \quad (\text{char} \quad \text{*}) \; 0x1000; \quad \text{points to location 0x1000} \\
  \text{n} & \quad \text{=} \quad \ast \text{address}; \quad \text{gets contents of memory at that location}
  \end{align*}
  \]

- This is how I/O ports are defined
  \[
  \text{#define PTT } \ast (\text{unsigned char volatile } \ast) \; (0x0240)
  \]

- then you can do
  \[ n = \text{PTT}; \]

“volatile”: Tells compiler that this variable can change its value without specific instructions from the program
Example

- Program continually reads Port T, bit 7

- Whenever that bit is set, it counts the number of “on” switches connected to Port T bits 6..3 and outputs the count to Port T bits 2..0

- If PT7 is not set, it outputs zeros to Port T bits 2..0.
Approach

• Initialize Port T data direction register, then go into an infinite loop

• At each iteration, test PT7

• If PT7 is on, count the 1’s on PT6:PT3 and output count to PT2:PT0; else output zeros to PT2:PT0

• There are a number of ways to do this

• We’ll show a couple of approaches
void main(void) {
    char count;

    EnableInterrupts;

    // Initialize bits 2:0 for output; others for input
    DDRT = 0x07;

    for(;;) {
        if (PTT & 0x80) {   // check bit PT7
            count = 0;

            if (PTT & 0x40) // check PT6
                count++;
            if (PTT & 0x20) // check PT5
                count++;
            if (PTT & 0x10) // check PT4
                count++;
            if (PTT & 0x08) // check PT3
                count++;

            // Write count to bits PT2:PT0. Doesn't affect other
            // bits since they are configured for input.
            PTT = count;
        } else
            PTT = 0;

        __FEED_COP(); /* feeds the dog */
    } /* loop forever */
}
void main(void) {
    char s, count;
    int i;
    EnableInterrupts;
    DDRT = 0x07; // Initialize bits 2:0 for output; others for input
    for(;;) {
        if (PTT & 0x80) {   // check bit PT7
            count = 0;
            s = PTT >> 3;   // shift PT6:PT3 to lower 4 bits
            for (i=0; i<4; i++) {
                count += s & 0x01;  // add next bit to count
                s >>= 1;            // shift right again
            }
            switch (count)  {
                case 0:
                    PTT = 0x0;
                    break;
                case 1:
                    PTT = 0x1;
                    break;
                case 2:
                    PTT = 0x2;
                    break;
                case 3:
                    PTT = 0x3;
                    break;
                case 4:
                    PTT = 0x4;
                    break;
            } // Show the use of a switch statement
        } else
            PTT = 0;
        _FEED_COP(); /* feeds the dog */
    } /* loop forever */
} /* Approach 2 */
Type modifiers

- These can be appended to the declaration of a variable
  - volatile, static, extern, register, const

- Volatile
  - Tells the compiler that the variable can be modified by something else other than the user’s code
  - Example:
    ```c
    volatile unsigned char sw_input;
    ```

- Static
  - Causes a local variable to retain its value when the function exits, and not disappear as what happens normally
  - Example:
    ```c
    static int count;
    ```
Type modifiers (continued)

• Extern
  – Makes a global variable visible in other files
  – Example:
    ```
    extern float ratio;
    ```

• Register
  – A hint to the compiler that the variable will be heavily used and that you want it be kept in a processor register if possible
  – Example:
    ```
    register char i;
    ```

• Const
  – Creates a read-only variable (i.e., a constant)
  – Example:
    ```
    const float pi = 3.14;
    ```
Structures

- A structure is a group of related variables that can be accessed through a common name
- Example: declare a structure type
  ```c
  struct point {
      int x;
      int y;
  };
  ```
- Define an instance
  ```c
  struct point pt;
  ```
- Access the elements of the structure
  ```c
  sq_distance = pt.x * pt.x + pt.y * pt.y;
  ```
Bitfield

• You can specify that a structure is composed of bitfields

• Example

```c
struct mybitfields {
    unsigned short a : 4;  // a is 4 bits wide
    unsigned short b : 5;  // b is 5 bits wide
    unsigned short c : 7;  // c is 7 bits wide
} test;
```

```c
test.a = 2;
test.b = 31;
test.c = 0;
```

• The bits are arranged as follows:

```
00000001 11110010
cccccccc bbbbbaaa
```

This structure has total size of 4+5+7 = 16 bits
Unions

- A union is an object that may hold (at different times) objects of different types and sizes
  - They occupy the same physical memory
- Example: declare a union type
  ```
  union u_tag {
      int x;
      char c[2];
  }
  ```
- Define an instance
  ```
  union u_tag u;
  ```
- Access the elements of the union
  ```
  u.x = 0x1234; // now u.c[0] contains 0x12, u.c[1] contains 0x34
  ```
Example

• Declaration of port T (from include file “mc9s12c32.h”)

```c
typedef union {
    byte Byte;
    struct {
        byte PTT0 :1;  /* Port T Bit 0 */
        byte PTT1 :1;  /* Port T Bit 1 */
        byte PTT2 :1;  /* Port T Bit 2 */
        byte PTT3 :1;  /* Port T Bit 3 */
        byte PTT4 :1;  /* Port T Bit 4 */
        byte PTT5 :1;  /* Port T Bit 5 */
        byte PTT6 :1;  /* Port T Bit 6 */
        byte PTT7 :1;  /* Port T Bit 7 */
    } Bits;
} PTTSTR;

extern volatile PTTSTR _PTT @(REG_BASE + 0x00000240UL);
#define PTT _PTT.Byte
#define PTT_PTT0 _PTT.Bits.PTT0
#define PTT_PTT1 _PTT.Bits.PTT1
#define PTT_PTT2 _PTT.Bits.PTT2
#define PTT_PTT3 _PTT.Bits.PTT3
#define PTT_PTT4 _PTT.Bits.PTT4
#define PTT_PTT5 _PTT.Bits.PTT5
#define PTT_PTT6 _PTT.Bits.PTT6
#define PTT_PTT7 _PTT.Bits.PTT7
```

The “@” is a CodeWarrior-specific extension to C, used for assigning variables to specific addresses.