Microprocessor Programming

Dr. Tin Ni Ni Kyaw

Ph. D (Kumamoto University, Japan)

Associate Professor

Department of Computer Engineering and Information Technology

Yangon Technological University

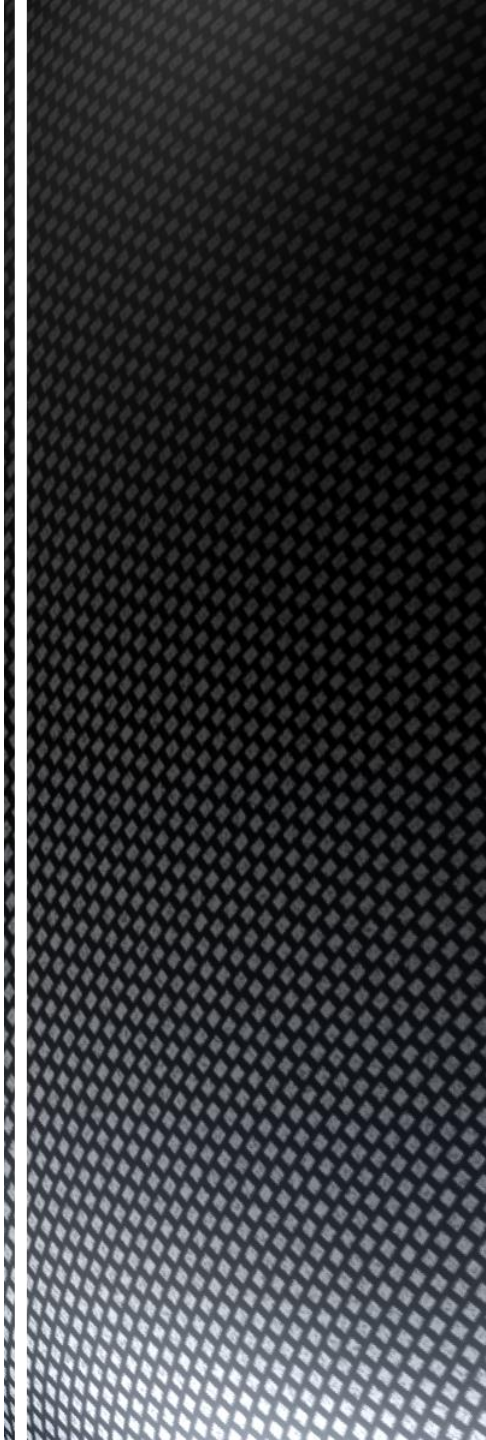
Yangon, Myanmar

Microprocessor Programming

Lecture 7

Logic and Shift Instructions

Contents



Logic Instructions

- In low-level programming, the individual bits always need to be set, tested, or toggled in a register or a memory location.
- In order to do so, the use of the logic operations can be very useful.
- The basic logic instructions include AND, OR, Exclusive-OR, and NOT.
- Another logic instruction is TEST which is a special form of AND instruction.

Logic Instructions (Cont.)

AND

- The AND operation performs logical multiplication as illustrated by the truth table in Figure 1.

A	B	T
0	0	0
0	1	0
1	0	0
1	1	1

Truth table for AND operation Logic Symbol for AND operation

Figure 1. AND operation

Logic Instructions (Cont.)

AND

- Here, two bits, A and B, are ANDed to produce the result T.
- As indicated by the truth table, T is a logic 1 only when both A and B are logic 1s.
- For all other input combinations of A and B, T is a logic 0.
- It is important to remember that 0 AND anything is always 0, and 1 AND 1 is always 1.

Logic Instructions (Cont.)

OR

- The OR operation performs logical addition and is often called the Inclusive-OR function.
- The truth table for the OR function appears in Figure 2.

A	B	T
0	0	0
0	1	1
1	0	1
1	1	1

Truth table for OR operation Logic Symbol for OR operation

Figure 2. OR operation

Logic Instructions (Cont.)

OR

- The OR function generates a logic 1 output if any inputs are 1.
- A 0 appears at the output only when all inputs are 0.
- Here, the inputs A and B OR together to produce the T output.
- It is important to remember that 1 ORed with anything yields a 1.

Logic Instructions (Cont.)

Exclusive-OR

- Figure 3 shows the operation of the Exclusive-OR function.

A	B	T
0	0	0
0	1	1
1	0	1
1	1	0

Truth table for XOR operation Logic Symbol for XOR operation

Figure 3. XOR operation

Logic Instructions (Cont.)

Exclusive-OR

- If the inputs of the Exclusive-OR function are both 0 or both 1, the output is 0.
- If the inputs are different, the output is 1.
- Because of this, the Exclusive-OR is sometimes called a comparator.

Logic Instructions (Cont.)

NOT

- NOT is a logical inversion or the one's complement of a bit.
- The NOT instruction inverts all bits of a byte, word, or doubleword.

Logic Instructions (Cont.)

- The following table shows the format for each logic instructions.
- Similar to the compare and arithmetic instructions, these logic instructions cannot have two operands that are memory locations.

Table 1. Logic instructions

And instructions	Or instructions	Xor instructions	Not instructions
and reg,reg	or reg,reg	xor reg,reg	not reg
and reg,imm	or reg,imm	xor reg,imm	not mem
and reg,mem	or reg,mem	xor reg,mem	
and mem,reg	or mem,reg	xor mem,reg	
and mem,imm	or mem,imm	xor mem,imm	

Logic Instructions (Cont.)

- Now, we will examine the use of AND instruction by comparing the following C code and MASM code segments.

C code	MASM code with high-level directives	MASM code without high-level directives
if(flag & maskit) count++;	mov eax, flag .if eax & maskit inc count .endif	if01: mov eax, flag and eax, maskit jz endif01 then01: inc count endif01: nop

Logic Instructions (Cont.)

- A particular bit can be set in a memory location in the C programming language using the bit-wise OR (|) operator.
- The same can be accomplished in MASM using OR instruction.

C code	MASM code

flag = flag | maskit;	mov eax,flag
	or eax,maskit
	mov flag,eax

Logic Instructions (Cont.)

- A particular bit can be set in a memory location in the C programming language using the bit-wise XOR (^) operator.
- The same can be accomplished in MASM using XOR instruction.

C code	MASM code

flag = flag ^ maskit;	mov eax,flag
	xor eax,maskit
	mov flag,eax

Logic Instructions (Cont.)

- Here is an example to find the one's complement of F0h by using NOT instruction.

```
mov     al,11110000b     ; AL = 11110000b (F0h)
not     al               ; AL = 00001111b (0Fh)
```

Logic Instructions (Cont.)

Test

- The TEST instruction performs the AND operation.
- The difference is that the AND instruction changes the destination operand, whereas the TEST instruction does not.
- A TEST only affects the condition of the flag register, which indicates the result of the test.
- The zero flag (Z) is a logic 1 (indicating a zero result) if the bit under test is a zero, and 0 (indicating a nonzero result) if the bit under test is not zero.

Logic Instructions (Cont.)

Test

- Also, the TEST instruction functions in the same manner as a compare (CMP) instruction.
- The difference is that the TEST instruction normally tests a single bit (or occasionally multiple bits), whereas the CMP instruction tests the entire byte, word, or doubleword.

Logic Instructions (Cont.)

Test

- Usually the TEST instruction is followed by either the JZ (jump if zero) or JNZ (jump if not zero) instruction.
- The destination operand is normally tested against immediate data.
- The value of immediate data is 1 to test the rightmost bit position, 2 to test the next bit, 4 for the next, and so on.

Logic Instructions (Cont.)

Test

- The TEST instruction can check from a bit to several bits at once.
- For example, suppose that we want to know whether bit 0 or bit 3 is set in the AL register.
- We can use the following instruction to find this out:

```
test al,00001001b          ; test bit 0 and bit 3
```

Shift Instructions

- Shifting means to move bits right and left inside an operand.
- In C programming language, it has the ability to shift bits to the left or the right in a memory location by using the << or >> operators respectively.
- For example, if the memory location num contained a 2 and the following instruction was executed, the contents of num would then be a 16:

```
num = num << 3;
```

Shift Instructions (Cont.)

- Similarly, assembly language also has instructions that move bits around inside operands.
- In MASM, there are two ways to shift an operand's bits.
  1. Logical shifting
  2. Arithmetic Shifting

Logical Shifts

- The logical shift fills the newly created bit position with zero.
- As described in Table 2, there exist the two logical shift instructions which shift the contents of a register or a memory location to either the left or the right of a specified number of bits.

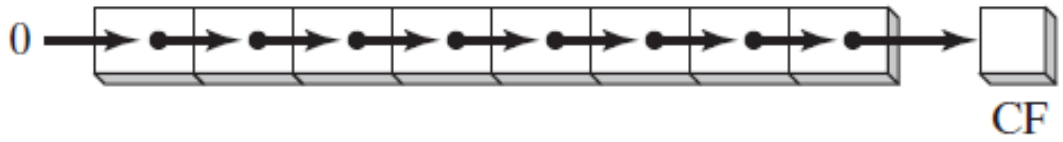
Table 2. Logical Shift Instructions

Instruction	Meaning
shr reg,imm	Shift right
shr mem,imm	Shift right
shl reg,imm	Shift left
shl mem,imm	Shift left

Logical Shifts (Cont.)

SHR Instruction

- The SHR (shift right) instruction performs a logical right shift on the destination operand, replacing the highest bit with a 0.
- The lowest bit is copied into the Carry flag and the bit that was previously in the Carry flag is lost.
- In other words, each bit is moved to the next lowest bit position.
- In the following illustration, a byte is logically shifted one position to the right.



Logical Shifts (Cont.)

SHR Instruction

- Assume that the memory location number contains a 20 as 00010100 in binary.

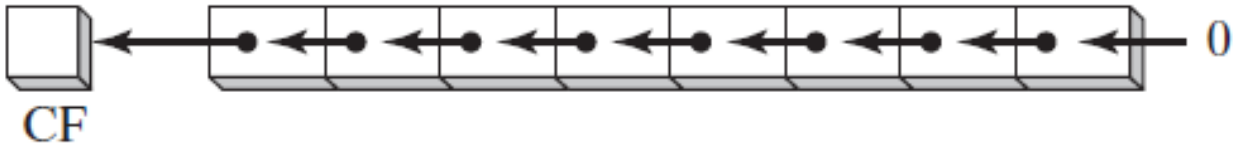
```
mov     al, 00010100b     ; AL = 20
shr     al,1              ; AL = 00001010b or 10
shr     al,1              ; AL = 00000101b or 5
```

- Shifting one bits to right is the equivalent of dividing by 2.

Logical Shifts (Cont.)

SHL Instruction

- The SHL (shift left) instruction performs a logical left shift on the destination operand, filling the lowest bit with 0.
- The highest bit is moved to the Carry flag, and the bit that was in the Carry flag is discarded:



Logical Shifts (Cont.)

SHL Instruction

- Assume that the memory location number contains a 5 as 00000101 in binary.

```
mov     al, 00000101b     ; AL = 5
shl     al,1              ; AL = 00001010b (10)
shl     al,1              ; AL = 00010100b (20)
```

- From the above codes, for every bit position shifted to the left, the number in register is multiplied by a power of 2.

Logical Shifts (Cont.)

Example

- Assume that each bit in the AL register (8 bits) represents a device that is connected to the processor. If a bit is a 1 or a 0, it would indicate whether the device is turned on or off respectively. For this task, also assume that the original data needs to be retained. Given these assumptions, how could one determine how many devices are turned on?

Logical Shifts (Cont.)

- It is probably best to process the bits in the order in which the bit positions are numbered, which is from right to left.

```
mov count,0              ; initialize count to zero
mov ecx,8                ; initialize loop counter to zero
mov temp,al              ; save al in temp
.repeat
mov ah,al                ; mov data in al to ah for testing
and ah,00000001b         ; test bit position zero
.if !ZERO?               ; is the bit set?
inc count                ; yes, count it
.endif
shr al,1                 ; shift al right one bit position
.untilcxz
mov al,temp              ; restore al from temp
```

Logical Shifts (Cont.)

- The above code can be rewritten without the extra mov instruction by using TEST instruction as shown below:

```
mov count,0                ; initialize count to zero
mov ecx,8                  ; initialize loop counter to zero
mov temp,al                ; save al in temp
.repeat
test al,00000001b          ; test bit position zero
.if !ZERO?                 ; is the bit set?
inc count                  ; yes, count it
.endif
shr al,1                   ; shift al right one bit position
.untilcxz
mov al,temp                ; restore al from temp
```

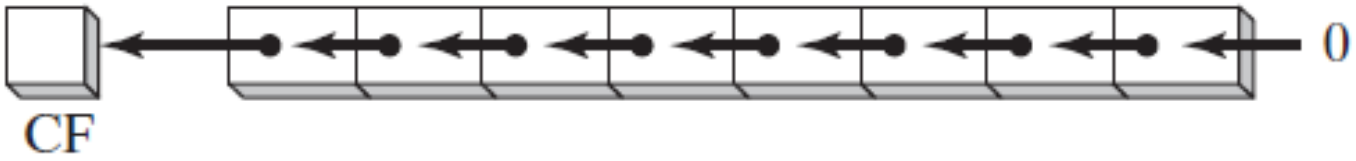
Arithmetic Shifts

- Besides the logical shift, the other shift instruction is the arithmetic shift called SAL and SAR, which stand for shift arithmetic left and shift arithmetic right respectively.
- The arithmetic shifts have the same operand formats as their logical counterparts.
- Unlike the logical shifts, the arithmetic shifts assume that the leftmost bit in a register or memory location is a sign bit.

Arithmetic Shifts (Cont.)

SAL Instruction

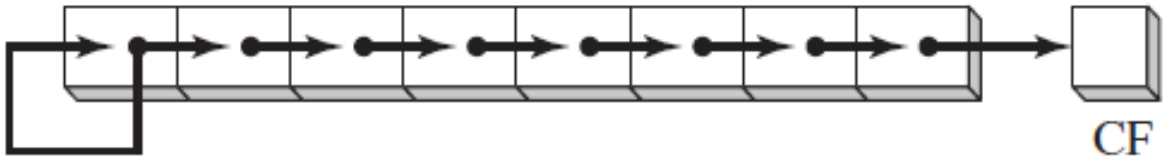
- The SAL (shift arithmetic left) instruction works the same as the SHL instruction.
- For each shift count, SAL shifts each bit in the destination operand to the next highest bit position.
- The lowest bit is assigned 0, the highest bit is moved to the Carry flag, and the bit that was in the Carry flag is discarded.



Arithmetic Shifts (Cont.)

SAR Instruction

- The SAR (shift arithmetic right) instruction performs a right arithmetic shift on its destination operand.
- With the SAR, the leftmost bit is copied to the bit to the right, but instead of bringing in a 0 into the leftmost position, the leftmost position is copied into itself, thus preserving the sign bit.



Arithmetic Shifts (Cont.)

SAR Instruction

- Assume that binary 00010100 is shifted to the right by 2 bit, it becomes 00000101:

```
mov     al, 00010100b     ; AL = 20
sar     al,1              ; AL = 00001010b (10)
sar     al,1              ; AL = 00000101b (5)
```

```
mov     al, 00010100b     ; AL = 20
sar     al,2              ; AL = 00000101b (5)
```

Arithmetic Shifts (Cont.)

- The followings are some codes for shifting negative numbers.

```
mov     al, 11111100      ; AL = -4
shr     al,1              ; AL = 01111110 or 126 (incorrect)
sar     al,1              ; AL = 11111110 or -2 (correct)
```

- This is the reason for the arithmetic SAR instruction, which would cause the sign bit to be copied not only to the right but also back into the leftmost bit position.

Arithmetic Shifts (Cont.)

- As another example, assume that the original number was –8.

```
mov     al, 11111000      ; AL = -8
sar     al,1              ; AL = 11111100 or -4 (correct)
```

- The code demonstrates that it is the equivalent of integer division where the remainder would be in the carry flag.
- The result is that the arithmetic shifts can be used for performing arithmetic.

Arithmetic Shifts (Cont.)

- Consider the following C statement to be implemented using a shift instruction.

```
product = num1 * 8;
```

- Assuming 32-bit words and using the SAL instruction, it could be implemented as follows:

```
mov eax,num1          ; load eax with num1
sal eax,3             ; multiply by 8
mov product,eax       ; store eax in product
```

Arithmetic Shifts (Cont.)

- Again, consider the following C statement to be implemented using a shift instruction.

```
answer = amount / 4;
```

- Assuming 32-bit words and using the SAR instruction, it could be implemented as follows:

```
mov eax,amount
sar eax,2                    ; divide by 4
mov answer,eax
```

Complete Program: Simulating an OCR Machine

- For the following simulation, the memory location used will be called the document status byte (DSB).

Sample Input/Output

```
Enter a hexadecimal number: 1
The hexadecimal number is: 1

SHORT DOCUMENT

Enter a hexadecimal number: 2
The hexadecimal number is: 2

LONG DOCUMENT

Enter a hexadecimal number: 3
The hexadecimal number is: 3

SHORT DOCUMENT
LONG DOCUMENT

Enter a hexadecimal number: ff
The hexadecimal number is: ff

SHORT DOCUMENT
LONG DOCUMENT
CLOSE FEED
MULTIPLE FEED
EXCESSIVE SKEW
DOCUMENT MISFEED
DOCUMENT JAM
UNSPECIFIED ERROR

Enter a hexadecimal number: 100
The hexadecimal number is: 100

Press any key to continue . . .
```

Error messages

Bit	Message	Meaning
0	Short document	The document just read is shorter than anticipated
1	Long document	The document just read is shorter than anticipated
2	Close feed	Current document is too close to the preceding document
3	Multiple feed	Two documents were detected at the same time
4	Excessive skew	The document is skewed (crooked) in the transport
5	Document misfeed	The document fails to feed into the transport
6	Document jam	The document jammed in the transport
7	Unspecified error	An unknown/unspecified error occurred

Complete Program: Simulating an OCR Machine

- The approach taken here is the use of non-nested high-level if directives:

```
                .listall
                .386
                .model flat,c
                .stack 100h
        scanf    PROTO arg2:Ptr Byte, inputlist:VARARG
        printf   PROTO arg1:Ptr Byte, printlist:VARARG
                .data
        msg1fmt  byte "%s",0
        in1fmt   byte "%x",0
        msg2fmt  byte "%s%x",0Ah,0Ah,0
        msg1     byte 0Ah,"Enter a hexadecimal number: ",0
        msg2     byte "The hexadecimal number is: ",0
        msgshort byte "SHORT DOCUMENT",0Ah,0
        msglong  byte "LONG DOCUMENT",0Ah,0
        msgclose byte "CLOSE FEED",0Ah,0
        msgmult  byte "MULTIPLE FEED",0Ah,0
        msgskew  byte "EXCESSIVE SKEW",0Ah,0
        msgfeed  byte "DOCUMENT MISFEED",0Ah,0
        msgjam   byte "DOCUMENT JAM",0Ah,0
        msgerror byte  "UNSPECIFIED ERROR",0Ah,0
        dsb      dword ?
        .code
main    proc

        INVOKE printf, ADDR msg1fmt,ADDR msg1
        INVOKE scanf, ADDR in1fmt,ADDR dsb
        INVOKE printf, ADDR msg2fmt, ADDR msg2, dsb
        .while dsb<=0ffh
        test dsb,00000001b
        .if !zero?                  ; if bit 0 = 1 then
        INVOKE printf, ADDR msg1fmt,ADDR msgshort
        .endif
        test dsb,00000010b
        .if !ZERO?                  ; if bit 1 = 1 then
        INVOKE printf, ADDR msg1fmt,ADDR msglong
        .endif
        test dsb,00000100b
        .if !ZERO?                  ; if bit 2 = 1 then
        INVOKE printf, ADDR msg1fmt,ADDR msgclose
        .endif
        test dsb,00001000b
        .if !ZERO?                  ; if bit 3 = 1 then
        INVOKE printf, ADDR msg1fmt,ADDR msgmult
        .endif
        test dsb,00010000b
        .if !ZERO?                  ; if bit 4 = 1 then
        INVOKE printf, ADDR msg1fmt,ADDR msgskew
        .endif
        test dsb,00100000b
        .if !ZERO?                  ; if bit 5 = 1 then
        INVOKE printf, ADDR msg1fmt,ADDR msgfeed
        .endif
        test dsb,01000000b
        .if !ZERO?                  ; if bit 6 = 1 then
        INVOKE printf, ADDR msg1fmt,ADDR msgjam
        .endif
        test dsb,10000000b
        .if !ZERO?                  ; if bit 7 = 1 then
        INVOKE printf, ADDR msg1fmt,ADDR msgerror
        .endif
        INVOKE printf, ADDR msg1fmt,ADDR msg1
        INVOKE scanf, ADDR in1fmt,ADDR dsb
        INVOKE printf, ADDR msg2fmt,ADDR msg2, dsb
        .endw
        ret
main    endp
        end
```

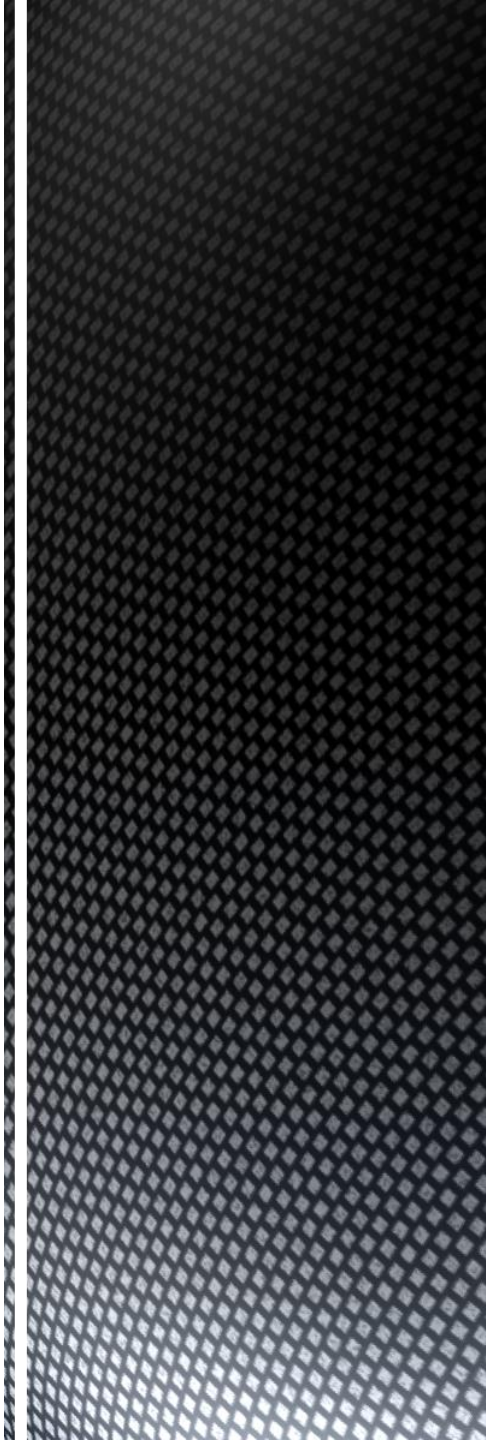
Summary

- To set, test, and toggle bits, use the OR, AND, and XOR instructions respectively.
- If data is needed later, be sure to save the data when using the SHL and SHR instructions.
- The arithmetic shifts assume the leftmost bit as a sign bit and the leftmost position is copied into itself to preserve the sign bit.

Microprocessor Programming

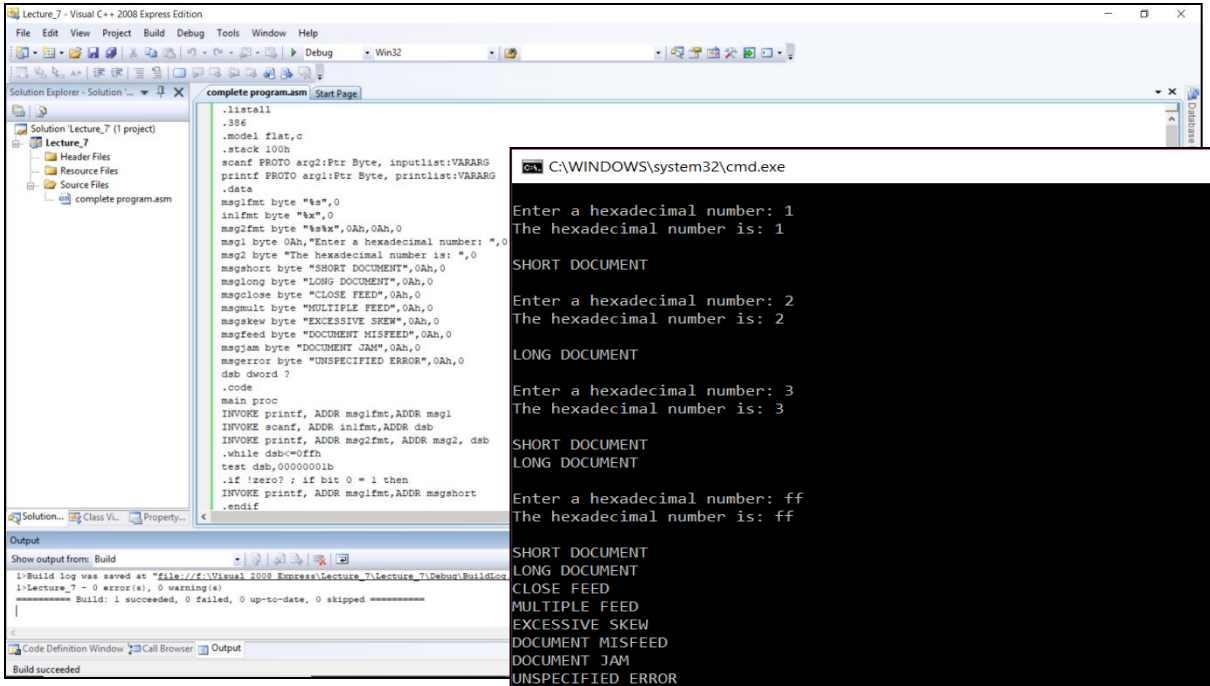
Practical Works

Simulating an OCR Machine



Simulating an OCR Machine

- The implementation of the given MASM code segment and its output are described below:



Microprocessor Programming

Practical Assignments (Instructions)

Assignment 1

- Write a program to simulate a security alarm system according to the following table, where it is possible that any of the first three high-priority items could happen at the same time. Although the last three items can occur at the same time, check and output messages for them only when none of the higher priority first three items have occurred.

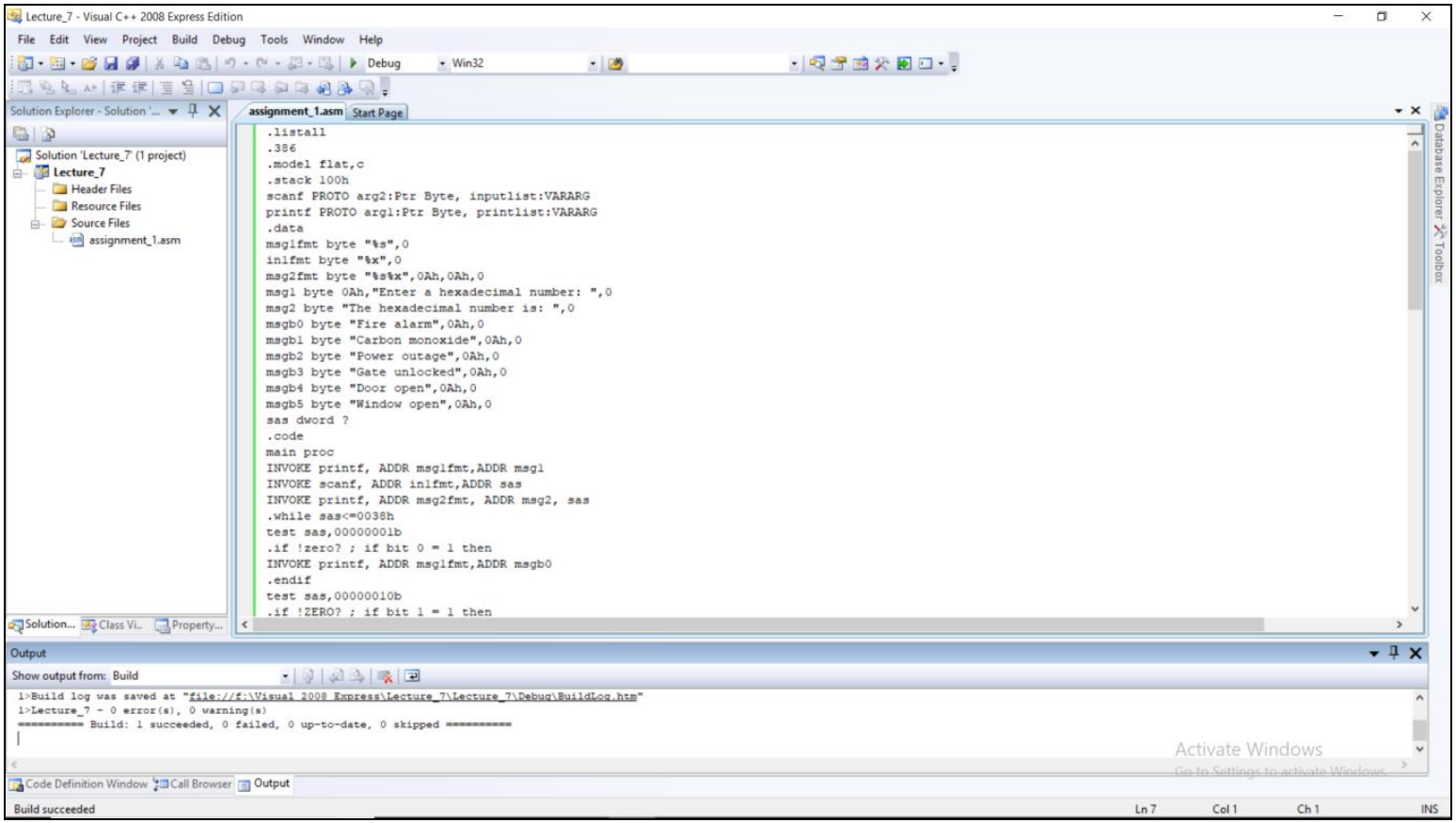
Bit	*Message*
0	Fire alarm
1	Carbon monoxide
2	Power outage
3	Gate unlocked
4	Door open
5	Window open

Microprocessor Programming

Practical Assignments (Report)

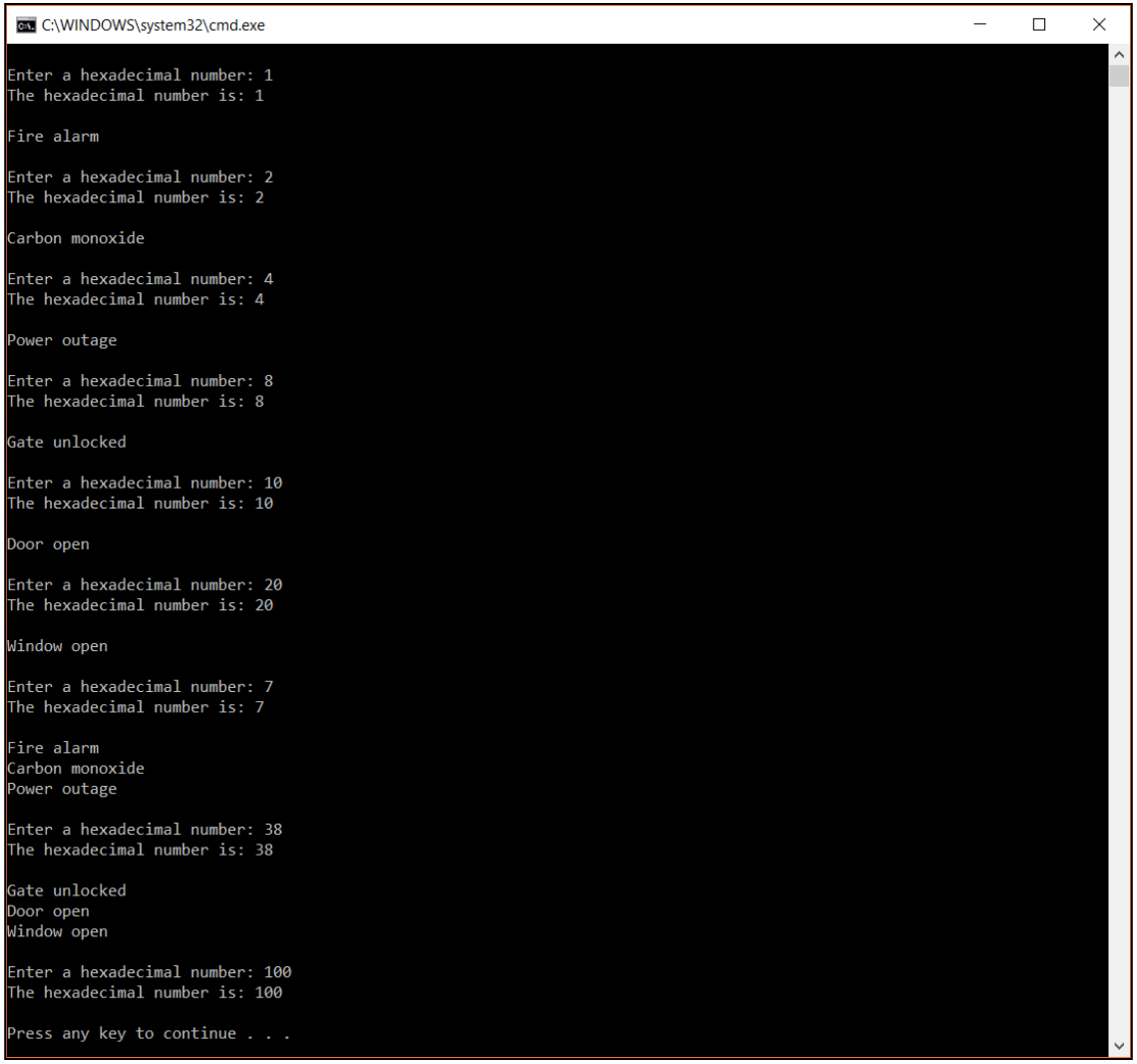
Assignment 1

- A complete assembly program is implemented as follow:



Assignment 1

- The corresponding output is as follow:



Next Lecture

- Rotating
- Stack
- Swapping

Thank You