Lab 9 - Integer Arithmetic

In this Lab we will practice the fundamental binary shift and rotation techniques. Bit manipulation is an intrinsic part of computer graphics, data encryption, and hardware manipulation.

9.1 Shift and Rotate Instructions

Along with bitwise instructions we studied in Lab 6, shift instructions are among the most characteristic of assembly language. *Bit shifting* means to move bits right and left inside an operand. x86 processors provide a particularly rich set of instructions in this area.

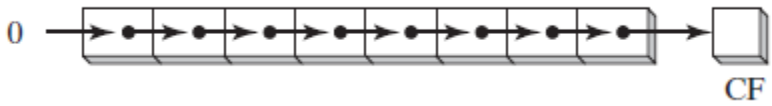
Shift instructions:

SHL	Shift left
SHR	Shift right
SAL	Shift arithmetic left
SAR	Shift arithmetic right
ROL	Rotate left
ROR	Rotate right
RCL	Rotate carry left
RCR	Rotate carry right
SHLD	Double-precision shift left
SHRD	Double-precision shift right

Flags: All shift instructions affect the Overflow and Carry flags.

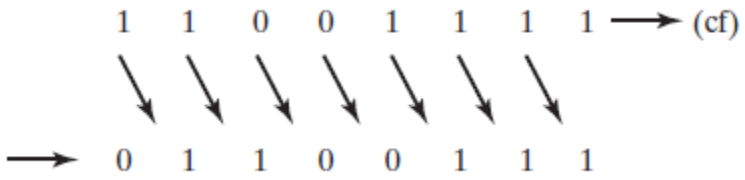
9.1.1 Logical Shifts (SHL, SHR) and Arithmetic Shifts (SAL, SAR)

logical shift fills the newly created bit position with zero.

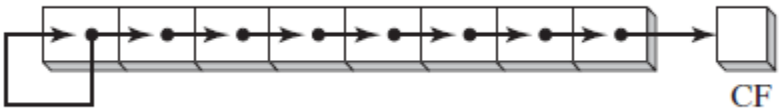


A byte is logically shifted one position to the right. In other words, each bit is moved to the next lowest bit position. Note that bit 7 is assigned 0.

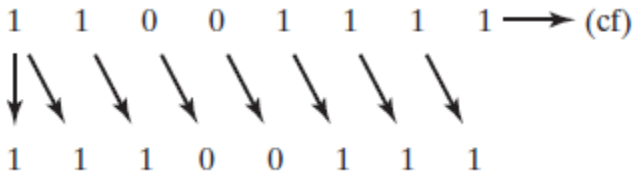
The following illustration shows a single logical right shift on the binary value 11001111, producing 01100111. The lowest bit is shifted into the Carry flag:



In arithmetic shift, the newly created bit position is filled with a copy of the original number's sign bit:



Binary 11001111, for example, has a 1 in the sign bit. When shifted arithmetically 1 bit to the right, it becomes 11100111:



9.1.2 SHL and SHR Instructions

```
SHL destination,count
SHR destination,count
```

The following lists the types of operands permitted by this instruction:

```
SHL reg,imm8
SHL mem,imm8
SHL reg,CL
SHL mem,CL
```

Note: Formats shown above also apply to the SHR, SAL, SAR, ROL, ROR, RCL, and RCR instructions.

Example:

```
Include Irvine32.inc
.data
	testData BYTE 10110001b
	message BYTE "Input binary: 1011 0001",0
	message_SHL1 BYTE "SHL 1 bit: ",0
	message_SHL2 BYTE "SHL 2 bit: ",0
	message_SHR1 BYTE "SHR 1 bit: ",0
	message_SHR2 BYTE "SHR 2 bit: ",0
.code
main proc
	mov EDX, OFFSET message
	call WriteString
	call Crlf

	; Left shift 1 bit
	mov AL,testData		; AL=10110001
	mov CL,1
	SHL AL,CL		; AL=01100010	CF=1
	mov EDX, OFFSET message_SHL1
	call printResult

	; Left shift 2 bits
```

```
        mov AL,testData          ; AL=10110001
        mov CL,2
        SHL AL,CL                ; AL=11000100      CF=0
        mov EDX, OFFSET message_SHL2
        call printResult

        ; Right shift 1 bit
        mov AL,testData          ; AL=10110001
        mov CL,1
        SHR AL,CL                ; AL=01011000      CF=1
        mov EDX, OFFSET message_SHR1
        call printResult

        ; Right shift 2 bits
        mov AL,testData          ; AL=10110001
        mov CL,2
        SHR AL,CL                ; AL=00101100      CF=0
        mov EDX, OFFSET message_SHR2
        call printResult
exit
main endp

printResult proc
        call WriteString
        ;mov EAX,0
        mov EBX,TYPE BYTE
        call WriteBinB
        call Crlf
        ret
printResult endp
end main
```

Example - Bitwise Multiplication using SHL:

Bitwise multiplication is performed when you shift a number's bits in a leftward direction. For example, SHL can perform multiplication by powers of 2. Shifting any operand left by **n** bits multiplies the operand by $\mathbf{2^n}$. So, the formula is:

$$multiplication = operand \times 2^n$$

$$where\ n = number\ of\ bit\ shifts\ towards\ Left$$

For example, shifting the integer 5 left by 1 bit yields the product of $5x2^1 = 10$:

```
mov  dl,5        Before:  0 0 0 0 0 1 0 1  = 5
shl  dl,1        After:   0 0 0 0 1 0 1 0  = 10
```

If binary 00001010 (decimal 10) is shifted left by two bits, the result is the same as multiplying 10 b 2^2:

```
mov AL,10          ; before: 00001010
SHL AL,2           ; after:  00101000
```

Example - Bitwise Division using SHR:

Bitwise division is accomplished when you shift a number's bits in a rightward direction. Shifting an unsigned integer right by **n** bits divides the operand by $\mathbf{2^n}$. So, the formula is:

$$division = operand \div 2^n$$

$$where\ n = number\ of\ bit\ shifts\ towards\ Right$$

In the following statements, we divide 32 by 2^1, producing 16:

```
mov  dl,32
shr  dl,1
```

Before: 0 0 1 0 0 0 0 0 = 32

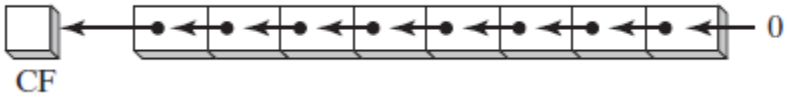
After: 0 0 0 1 0 0 0 0 = 16

In the following example, 64 is divided by 2^3:

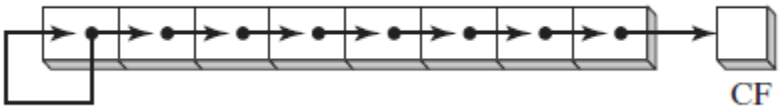
```
mov AL,01000000b          ; AL = 64
SHR AL,3                  ; divide by 8, AL = 00001000b
```

9.1.3 SAL and SAR Instructions

The **SAL** (shift arithmetic left) instruction works exactly 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:



The **SAR** (shift arithmetic right) instruction performs a right arithmetic shift on its destination operand. The difference here is that the SAR instruction duplicates the sign bit in the most significant bit while shifting the bits towards right:



The following example shows how SAR duplicates the sign bit. AL is negative before and after it is shifted to the right:

```
mov AL,0F0h               ; AL = 11110000b (-16)
SAR AL,1                  ; AL = 11111000b (-8), CF = 0
```

Example:

```
Include Irvine32.inc
.data
      testData BYTE 10110001b
      message BYTE "Input binary: 1011 0001",0
      message_SAL1 BYTE "SAL 1 bit: ",0
      message_SAL2 BYTE "SAL 2 bit: ",0
      message_SAR1 BYTE "SAR 1 bit: ",0
      message_SAR2 BYTE "SAR 2 bit: ",0
.code
```

```
main proc
	mov EDX, OFFSET message
	call WriteString
	call Crlf

	; Left arithmetic shift 1 bit
	mov AL,testData		; AL=10110001
	mov CL,1
	SAL AL,CL		; AL=01100010	CF=1
	mov EDX, OFFSET message_SAL1
	call printResult

	; Left arithmetic shift 2 bits
	mov AL,testData		; AL=10110001
	mov CL,2
	SAL AL,CL		; AL=11000100	CF=0
	mov EDX, OFFSET message_SAL2
	call printResult

	; Right arithmetic shift 1 bit
	mov AL,testData		; AL=10110001
	mov CL,1
	SAR AL,CL		; AL=11011000	CF=1
	mov EDX, OFFSET message_SAR1
	call printResult

	; Right arithmetic shift 2 bits
	mov AL,testData		; AL=10110001
	mov CL,2
	SAR AL,CL		; AL=11101100	CF=0
	mov EDX, OFFSET message_SAR2
	call printResult
exit
main endp

printResult proc
	call WriteString
	;mov EAX,0
	mov EBX,TYPE BYTE
	call WriteBinB
	call Crlf
	ret
printResult endp
end main
```

Example - Signed Division using SAR:

You can divide a signed operand by a power of 2, using the SAR instruction. In the following example, –128 is divided by 2^3. The quotient is –16:

```
mov AL,-128		; AL = 10000000b (-128)
SAR AL,3		; AL = 11110000b (-16)
```

Example – Sign-Extend AX into EAX using SAR:

Suppose **AX** contains a signed integer and you want to extend its sign into EAX. First shift EAX 16 bits to the left, then shift it arithmetically 16 bits to the right:

```
mov AX,-128         ; EAX = ????FF80h
SHL EAX,16          ; EAX = FF800000h
SAR EAX,16          ; EAX = FFFFFF80h
```

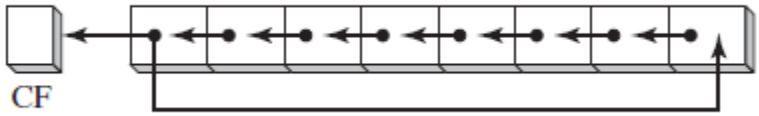
This code works same as we studied earlier using **MOVSX** instruction:

```
mov AX,-128         ; EAX = ????FF80h
movsx EAX,AX        ; EAX = FFFFFF80h
```

9.1.4 ROL Instruction

Bitwise rotation occurs when you move the bits in a circular fashion. In some versions, the bit leaving one end of the number is immediately copied into the other end. Another type of rotation uses the Carry flag as an intermediate point for shifted bits.

The **ROL** (rotate left) instruction shifts each bit to the left. The highest bit is copied into the Carry flag and the lowest bit position. The instruction format is the same as for **SHL**:



Note: Bit rotation does not lose bits. A bit rotated off one end of a number appears again at the other end.

```
Include Irvine32.inc
.code
main proc
      MOV AL,40h          ; AL = 01000000b
      ROL AL,1            ; AL = 10000000b, CF = 0
      ROL AL,1            ; AL = 00000001b, CF = 1
      ROL AL,1            ; AL = 00000010b, CF = 0
exit
main endp
end main
```

When using a rotation count greater than 1, the Carry flag contains the last bit rotated out of the MSB position:

```
MOV AL,00100000b
ROL AL,3                  ; AL = 00000001b, CF = 1
```

Example - Exchanging Groups of Bits:

We can use ROL to exchange the upper (bits 4–7) and lower (bits 0–3) halves of a byte. For example, 26h rotated four bits in either direction becomes 62h:

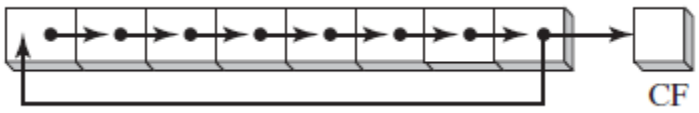
```
Include Irvine32.inc
.code
main proc
      MOV AL,26h
      ROL AL,4            ; AL = 62h
exit
main endp
end main
```

When rotating a multibyte integer by **four** bits, the effect is to rotate each hexadecimal digit one position to the right or left. Here, for example, we repeatedly rotate 6A4Bh left four bits, eventually ending up with the original value:

```
MOV AX,6A4Bh
ROL AX,4            ; AX = A4B6h
ROL AX,4            ; AX = 4B6Ah
ROL AX,4            ; AX = B6A4h
ROL AX,4            ; AX = 6A4Bh
```

9.1.5 ROR Instruction

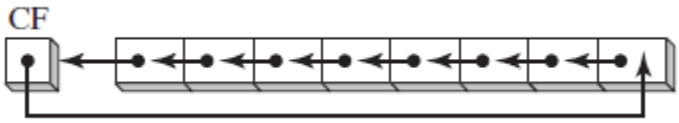
The **ROR** (rotate right) instruction shifts each bit to the right and copies the lowest bit into the Carry flag and the highest bit position. The instruction format is the same as for **SHL**:




```
Include Irvine32.inc
.code
main proc
      MOV AL,01h          ; AL = 00000001b
      ROR AL,1            ; AL = 10000000b, CF = 1
      ROR AL,1            ; AL = 01000000b, CF = 0
exit
main endp
end main
```

9.1.6 RCL (Rotate Cary Left) Instruction

The **RCL** (rotate carry left) instruction shifts each bit to the left, first copies the Carry flag to the LSB, then copies the MSB into the Carry flag:



If we imagine the Carry flag as an extra bit added to the high end of the operand, RCL looks like a rotate left operation.

```
Include Irvine32.inc
.code
main proc
      clc                 ; CLC clears the Carry flag (CF = 0)
      MOV BL,88h          ; CF,BL = 0 10001000b
      RCL BL,1            ; CF,BL = 1 00010000b
      RCL BL,1            ; CF,BL = 0 00100001b
exit
main endp
end main
```

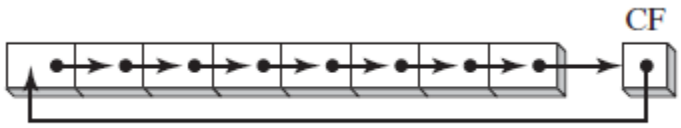
Example - Recover a Bit from the Carry Flag:

RCL can recover a bit that was previously shifted into the Carry flag (e.g., due to binary addition).

```
Include Irvine32.inc
.data
      testval BYTE 01101010b
.code
main proc
      SHR testval,1            ; shift LSB into Carry flag
      JC quit                  ; exit if Carry flag set
      RCL testval,1            ; else restore the number
quit:
      exit
main endp
end main
```

9.1.7 RCR (Rotate Cary Right) Instruction

The **RCR** (rotate carry right) instruction shifts each bit to the right, first copies the Carry flag into the MSB, then copies the LSB into the Carry flag:



As in the case of RCL, it helps to visualize the integer in this figure as a 9-bit value, with the Carry flag to the right of the LSB.

```
Include Irvine32.inc
.code
main proc
      stc                ; CF = 1
      MOV AH,10h         ; AH, CF = 00010000 1
      RCR AH,1           ; AH, CF = 10001000 0
exit
main endp
end main
```

9.1.8 Signed Overflow

The Overflow flag is set if the act of shifting or rotating a signed integer by one bit position generates a value outside the signed integer range of the destination operand. To put it another way, the number's sign is reversed.

In the following example, a positive integer (+127) stored in an 8-bit register becomes negative (-2) when rotated left:

```
MOV AL,+127        ; AL = 01111111b
ROL AL,1           ; OF = 1, AL = 11111110b
```

Similarly, when -128 is shifted one position to the right, the Overflow flag is set. The result in AL (+64) has the opposite sign:

```
MOV AL,-128          ; AL = 10000000b
SHR AL,1             ; OF = 1, AL = 01000000b
```

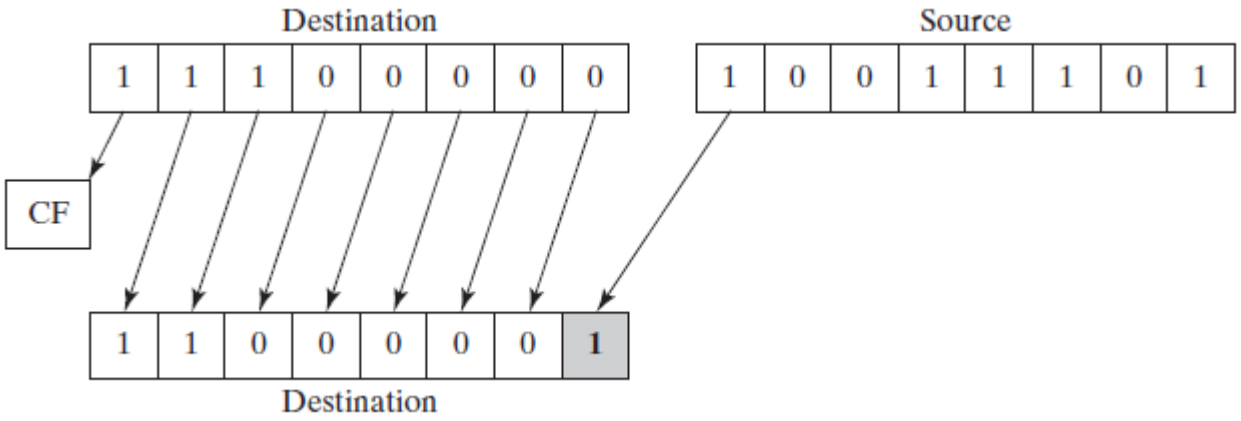
Note: The value of the Overflow flag is undefined when the shift or rotation count is greater than 1.

9.1.9 SHLD and SHRD Instructions

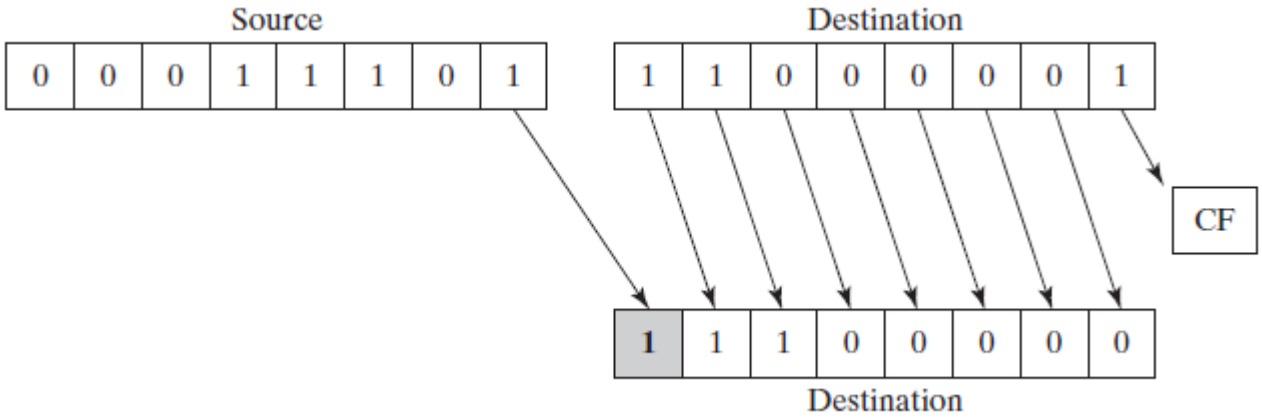
The SHLD (shift left double) instruction shifts a destination operand a given number of bits to the left.

The bit positions opened up by the shift are filled by the most significant bits of the source operand. The source operand is not affected, but the Sign, Zero, Auxiliary, Parity, and Carry flags are affected:

```
SHLD dest, source, count
```



The SHRD (shift right double) instruction shifts a destination operand a given number of bits to the right. The bit positions opened up by the shift are filled by the least significant bits of the source operand:



The following instruction formats apply to both SHLD and SHRD:

```
SHLD reg16,reg16,CL/imm8
SHLD mem16,reg16,CL/imm8
SHLD reg32,reg32,CL/imm8
SHLD mem32,reg32,CL/imm8
```

Example 1:

```
Include Irvine32.inc
.data
	testVal WORD 9BA6h
.code
main proc
	MOV AX,0AC36h
	SHLD testVal,AX,4		; wval = BA6Ah
exit
main endp
end main
```

Example 2:

```
Include Irvine32.inc
.code
main proc
	MOV AX,234Bh
	MOV DX,7654h
	SHRD AX,DX,4
exit
main endp
end main
```

Example 3 - Shifting an array of doublewords to the right by 4 bits:

```
Include Irvine32.inc
.data
	array DWORD 648B2165h,8C943A29h,6DFA4B86h,91F76C04h,8BAF9857h
.code
main proc
	mov BL,4				; shift count
	mov ESI,OFFSET array			; offset of the array
	mov ECX,(LENGTHOF array) - 1		; number of array elements
L1:
	push ECX				; save loop counter
	mov EAX,[ESI + TYPE DWORD]
	mov CL,BL				; shift count
	SHRD [ESI],EAX,CL			; shift EAX into high bits of [ESI]
	add ESI,TYPE DWORD			; point to next doubleword pair
	pop ECX				; restore loop counter
	LOOP L1
	SHR DWORD PTR [ESI],4			; shift the last doubleword

exit
main endp
end main
```

- SHLD and SHRD can be used to manipulate bit-mapped images, when groups of bits must be shifted left and right to reposition images on the screen.
- Another potential application is data encryption, in which the encryption algorithm involves the shifting of bits.
- These two instructions can be used when performing fast multiplication and division with very long integers.

9.2 Shift and Rotate Applications

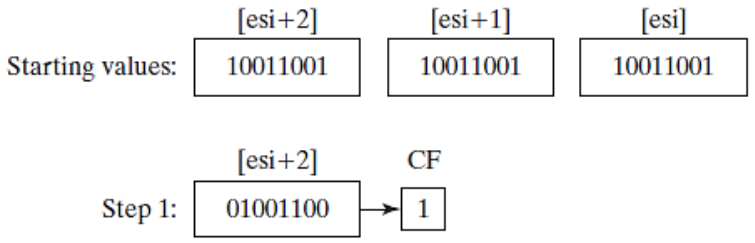
When a program needs to move bits from one part of an integer to another, assembly language is a great tool for the job. Sometimes, we move a subset of a number's bits to position 0 to make it easier to isolate the value of the bits. In this section, we show a few common bit shift and rotate applications that are easy to implement. More applications will be found in the chapter 7 exercises.

9.2.1 Shifting Multiple Doublewords

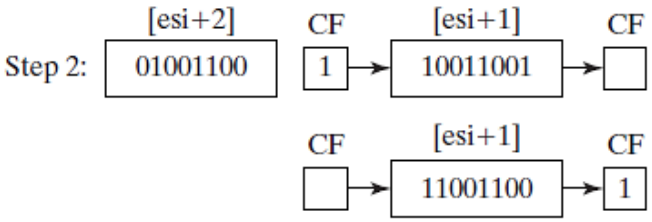
We can shift an integer that has been divided into an array of bytes, words, or doublewords.

The following steps show how to shift an array of bytes 1 bit to the right:

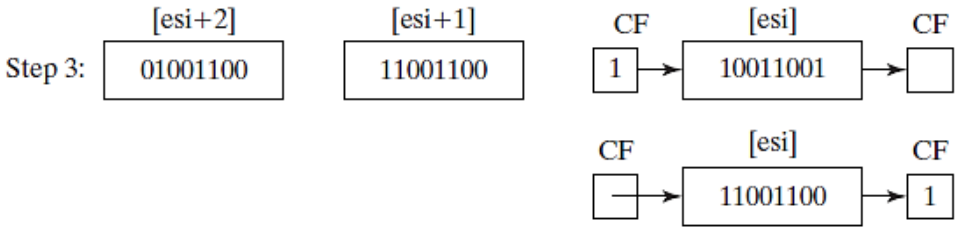
Step 1: Shift the highest byte at [ESI+2] to the right, automatically copying its lowest bit into the Carry flag.



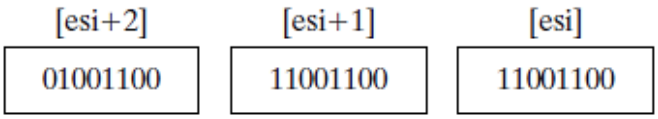
Step 2: Rotate the value at [ESI+1] to the right, filling the highest bit with the value of the Carry flag, and shifting the lowest bit into the Carry flag:



Step 3: Rotate the value at [ESI] to the right, filling the highest bit with the value of the Carry flag, and shifting the lowest bit into the Carry flag:



After Step 3 is complete, all bits have been shifted 1 position to the right:



```
Include Irvine32.inc
.data
	ArraySize = 3
	array BYTE ArraySize DUP(99h)	; 1001 pattern in each nybble
.code
main PROC
	call DisplayArray		; display the array

	mov esi,0
	shr array[esi+2],1		; high byte
	rcr array[esi+1],1		; middle byte, include Carry flag
	rcr array[esi],1		; low byte, include Carry flag

	call DisplayArray		; display the array
	exit
main ENDP

;-----------------------------------------------------
DisplayArray PROC
; Display the bytes from highest to lowest
;-----------------------------------------------------
	pushad

	mov ecx,ArraySize
	mov esi,ArraySize-1
L1:
	mov  al,array[esi]
	mov  ebx,1			; size = byte
	call WriteBinB			; display binary bits
	mov  al,' '
	call WriteChar
	sub  esi,1
	Loop L1

	call Crlf
	popad
	ret
DisplayArray ENDP

END main
```

9.2.2 Binary Multiplication

We can get performance advantage of Assembly language in integer multiplication by using bit shifting rather than the MUL instruction. The **SHL** instruction performs unsigned multiplication when the multiplier is a power of 2. Shifting an unsigned integer n bits to the left multiplies it by 2^n.

Any other multiplier can be expressed as the sum of powers of 2. For example, to multiply unsigned EAX by 36, we can write 36 as $2^5 + 2^2$ and use the distributive property of multiplication:

```
EAX * 36     = EAX * (32 + 4)
             = EAX * (2^5 + 2^2)
             = (EAX * 2^5) + (EAX * 2^2)
```

Code will be:

```
;mov EAX,123        ; multiplicaton using mul instruction
;mul 36

; multiplication using shift and rotate method (more efficient)
mov EAX,123
mov EBX,EAX
shl EAX,5           ; multiply by 2^5 (32)
shl EBX,2           ; multiply by 2^2 (4)
add EAX,EBX         ; add the products (32 + 4)
```

9.2.3 Displaying Binary Bits

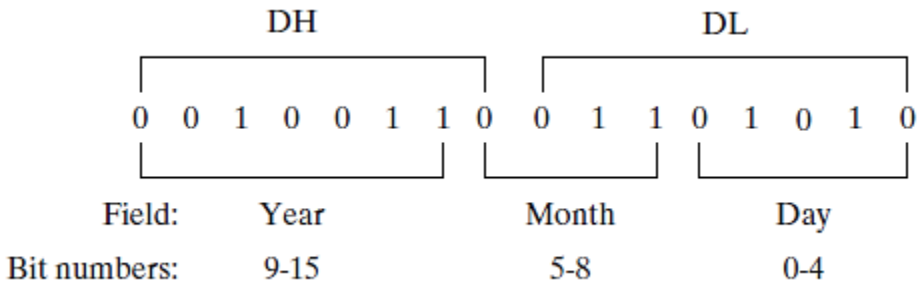
A common programming task is converting a binary integer to an ASCII binary string, allowing the latter to be displayed. The **SHL** instruction is useful in this regard because it copies the highest bit of an operand into the Carry flag each time the operand is shifted left. The following *BinToAsc* procedure is a simple implementation:

```
Include Irvine32.inc
.data
	binVal DWORD 1234ABCDh		; sample binary value
	buffer BYTE 32 dup(0),0
.code
main PROC
	mov	eax,binVal		; EAX = binary integer
	mov	esi,OFFSET buffer	; point to the buffer
	call	BinToAsc		; do the conversion

	mov	edx,OFFSET buffer	; display the buffer
	call WriteString		; output: 00010010001101001010101111001101

	call Crlf
	exit
main ENDP

;-----------------------------------------------------
; BinToAsc PROC
;
; Converts 32-bit binary integer to ASCII binary.
; Receives: EAX = binary integer, ESI points to buffer
; Returns: buffer filled with ASCII binary digits
;-----------------------------------------------------

BinToAsc PROC
	push	ecx
	push	esi

	mov	ecx,32			; number of bits in EAX

L1:	shl	eax,1			; shift high bit into Carry flag
	mov	BYTE PTR [esi],'0'	; choose 0 as default digit
	jnc	L2			; if no Carry, jump to L2
	mov	BYTE PTR [esi],'1'	; else move 1 to buffer

L2:	inc	esi			; next buffer position
	loop	L1			; shift another bit to left
```

```
        pop    esi
        pop    ecx
        ret
BinToAsc ENDP

END main
```

9.2.4 Extracting File Date Fields

When storage space is at a premium, system-level software often packs multiple data fields into a single integer. To uncover this data, applications often need to extract sequences of bits called bit strings. For example, in real-address mode, MS-DOS function 57h returns the date stamp of a file in DX. (The date stamp shows the date on which the file was last modified.) Bits 0 through 4 represent a day number between 1 and 31, bits 5 through 8 are the month number, and bits 9 through 15 hold the year number. If a file was last modified on March 10, 1999, the file's date stamp would appear as follows in the DX register (the year number is relative to 1980):



To extract a single bit string, shift its bits into the lowest part of a register and clear the irrelevant bit positions. The following code example extracts the day number field of a date stamp integer by making a copy of DL and masking off bits not belonging to the field:

```
mov al,dl          ; make a copy of DL
and al,00011111b   ; clear bits 5-7
mov day,al         ; save in day
```

To extract the month number field, we shift bits 5 through 8 into the low part of AL before masking off all other bits. AL is then copied into a variable:

```
mov ax,dx          ; make a copy of DX
shr ax,5           ; shift right 5 bits
and al,00001111b   ; clear bits 4-7
mov month,al       ; save in month
```

The year number (bits 9 through 15) field is completely within the DH register. We copy it to AL and shift right by 1 bit:

```
mov al,dh          ; make a copy of DH
shr al,1           ; shift right one position
mov ah,0           ; clear AH to zeros
add ax,1980        ; year is relative to 1980
mov year,ax        ; save in year
```