Assembly Programming

Chapter 7: Integer Arithmetic

CSE3030

Prof. Youngjae Kim
Distributed Computing and Operating Systems Laboratory (DISCOS)
https://discos.sogang.ac.kr

Integer Arithmetic

- Shift and Rotate Instructions
- Shift and Rotate Applications
- Multiplication and Division Instructions
- Extended Addition and Subtraction
- ASCII and Packed Decimal Arithmetic

Shift and Rotate Instructions

- Logical vs Arithmetic Shifts
- SHL and SHR Instruction
- SAL and SAR Instructions
- ROL and ROR Instruction
- RCL and RCR Instructions
- Operation types for shift and rotate instructions:

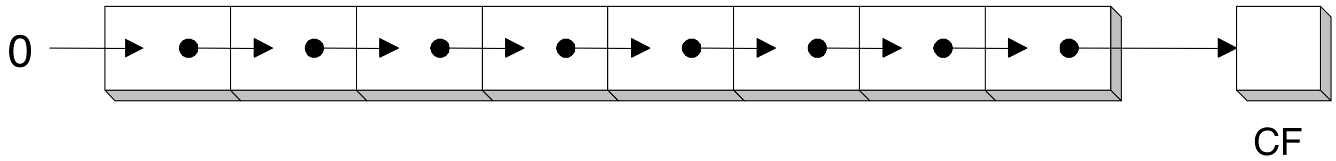
```
SHL reg,imm8
SHL mem,imm8
SHL reg,CL
SHL mem,CL
```

(Same for all shift and rotate instructions)

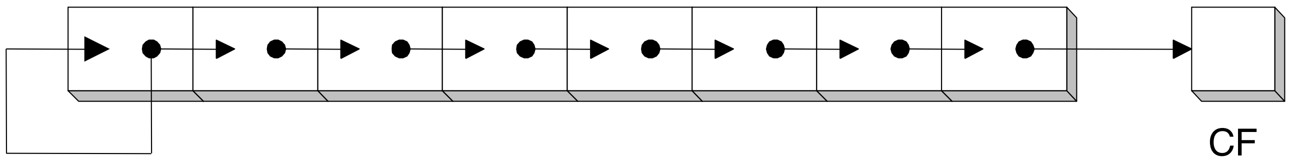
- SHLD/SHRD Instructions

Logical vs Arithmetic Shifts

- A logical shift fills the newly created bit position with zero:

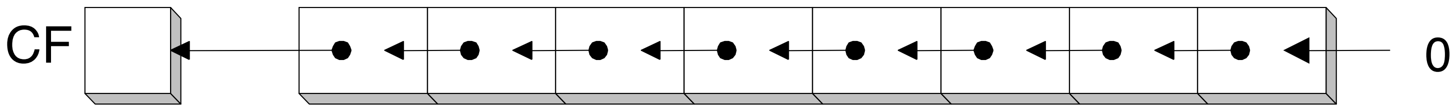


- An arithmetic shift fills the newly created bit position with a copy of the number's sign bit:



SHL(Shift Left) Instruction

- Performs a logical left shift on the destination operand, filling the lowest bit with 0.

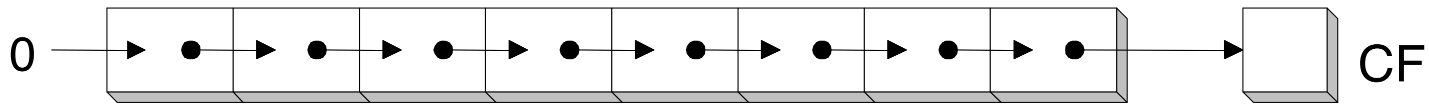


- Shifting left n bits multiplies the operand by 2^n

```
mov dl,5
shl dl,2        ; DL = 20
```

SHR(Shift Right) Instruction

- Performs a logical right shift on the destination operand. The highest bit position is filled with a zero.

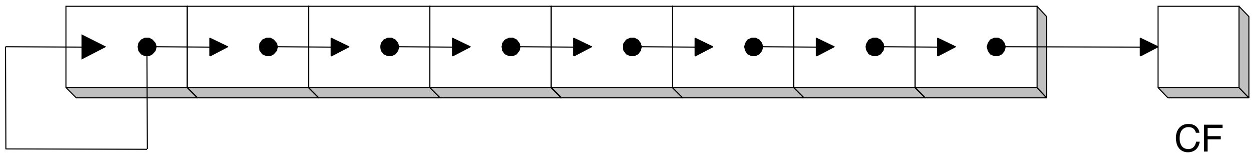


- Shifting right n bits divides the operand by 2^n

```
mov dl,80
shr dl,1      ; DL = 40
shr dl,2      ; DL = 10
```

SAL/SAR (Shift Arithmetic Left/Right) Instructions

- SAL is identical to SHL.
- SAR performs a right arithmetic shift on the destination operand.



- An arithmetic shift preserves

```
mov dl,-80
sar dl,1        ; DL = -40
sar dl,2        ; DL = -10
```

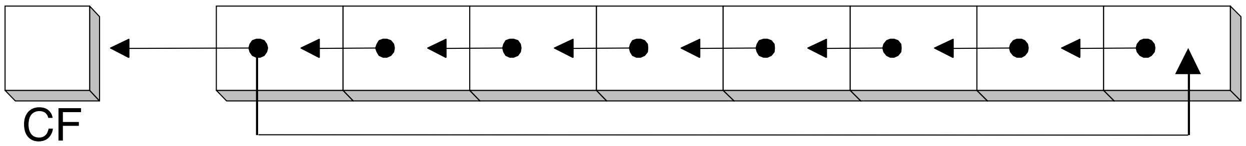
- Indicate the hexadecimal value of AL after each shift:

```
mov al,6Bh
shr al,1            a.  35h
shl al,3            b.  A8h
mov al,8Ch
sar al,1            c.  C6h
sar al,3            d.  F8h
```

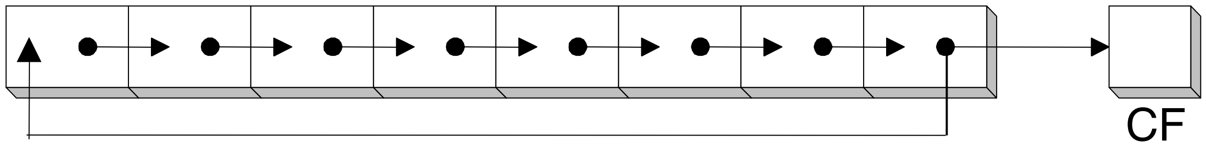
al: 01101011
shr al, 1 ; al: 00110101 (35h) CF: 1
shl al, 3 ; CF: 1 al: 10101000 (A8h)
mov al, 8Ch ; al: 10001100
sar al, 1; 11000110 (C6h)
sar al, 3: 11111000 (F8h)

ROL(Rotate Left) Instruction

- ROL
  - Shifts each bit to the left. The highest bit is copied into both the Carry flag and into the lowest bit. No bits are lost.



- ROR Instruction



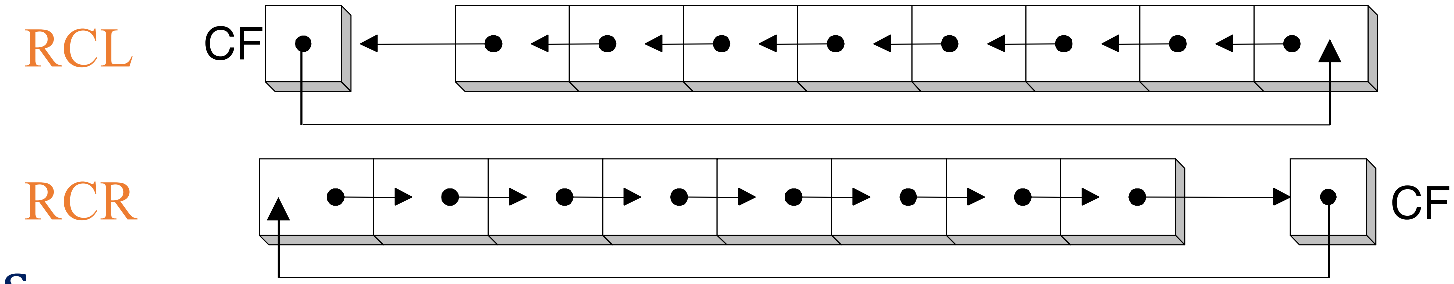
- Examples

```
mov al,11110000b
rol al,1                            ; AL = 11100001b
ror al,1                            ; AL = 11110000b
mov dl,3Fh
rol dl,4                            ; DL = F3h
ror dl,4                            ; DL = 3Fh

mov al,6Bh
ror al,1                          ; a. B5h
rol al,3                          ; b. ADh
```

RCL/RCR(Rotate Carry Left/Right) Instruction

- RCL/RCR
  - Shifts each bit to the left/right. Copies the Carry flag to the LSB/MSB. Copies the MSB/LSB to the Carry flag.



- Examples

```
clc                 ; CF = 0
mov bl,88h          ; CF,BL = 0 10001000b
rcl bl,1            ; CF,BL = 1 00010000b
rcl bl,1            ; CF,BL = 0 00100001b
stc                 ; CF = 1
mov ah,10h          ; CF,AH = 1 00010000b
rcr ah,1            ; CF,AH = 0 10001000b
stc
mov al,6Bh
rcr al,1            ; a. B5h
rcl al,3            ; b. AEh
```

SHLD/SHRD Instruction

- Shifts a destination operand a given number of bits to the left/right.
- The bit positions opened up by the shift are filled by the MSBs/LSBs of the source operand
- The source operand is not affected
- Syntax: **SHLD** ***destination, source, count***

  SHRD ***destination, source, count***
- Operand types:

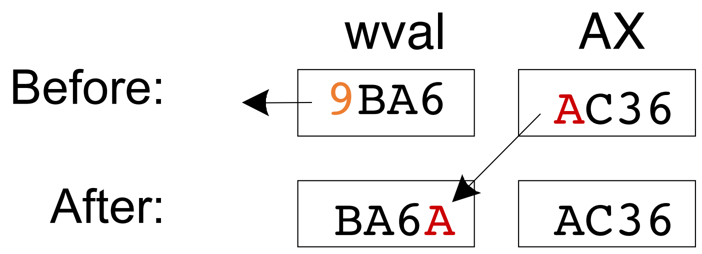
```
SHLD reg16/32, reg16/32, imm8/CL
SHLD mem16/32, reg16/32, imm8/CL
```

Same operand types for SHRD

Exmaples

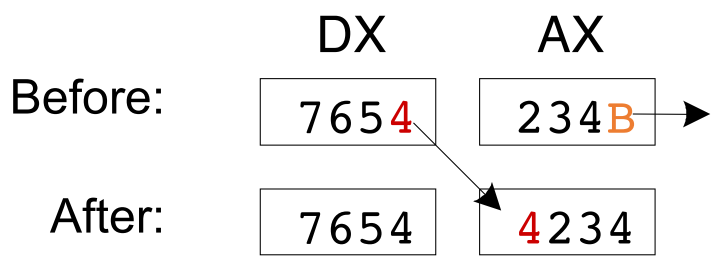
SHLD

```
.data
wval WORD 9BA6h
.code
mov  ax,0AC36h
shld wval,ax,4
```



SHRD

```
mov  ax,234Bh
mov  dx,7654h
shrd ax,dx,4
```



Example:

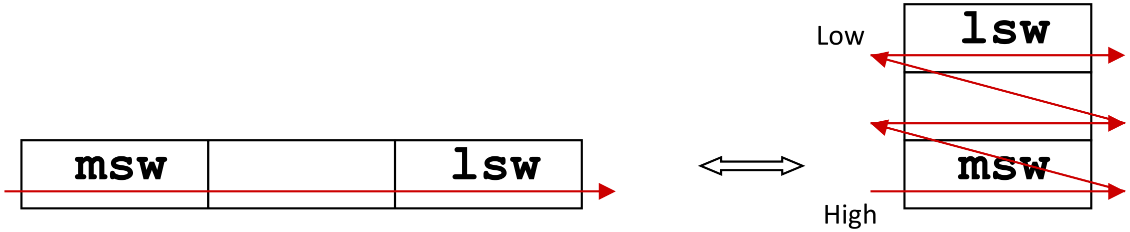
```
mov  ax,7C36h
mov  dx,9FA6h
shld dx,ax,4            ; DX =  FA67h
shrd dx,ax,8            ; DX =  36FAh
```

Shift and Rotate Applications

- Shifting Multiple Doublewords
- Isolating a Bit String
- Binary Multiplication
- Displaying Binary Bits

Shifting Multiple Doublewords

- We can shift an extended-precision integer that has been divided in an array of bytes, words, or double words.

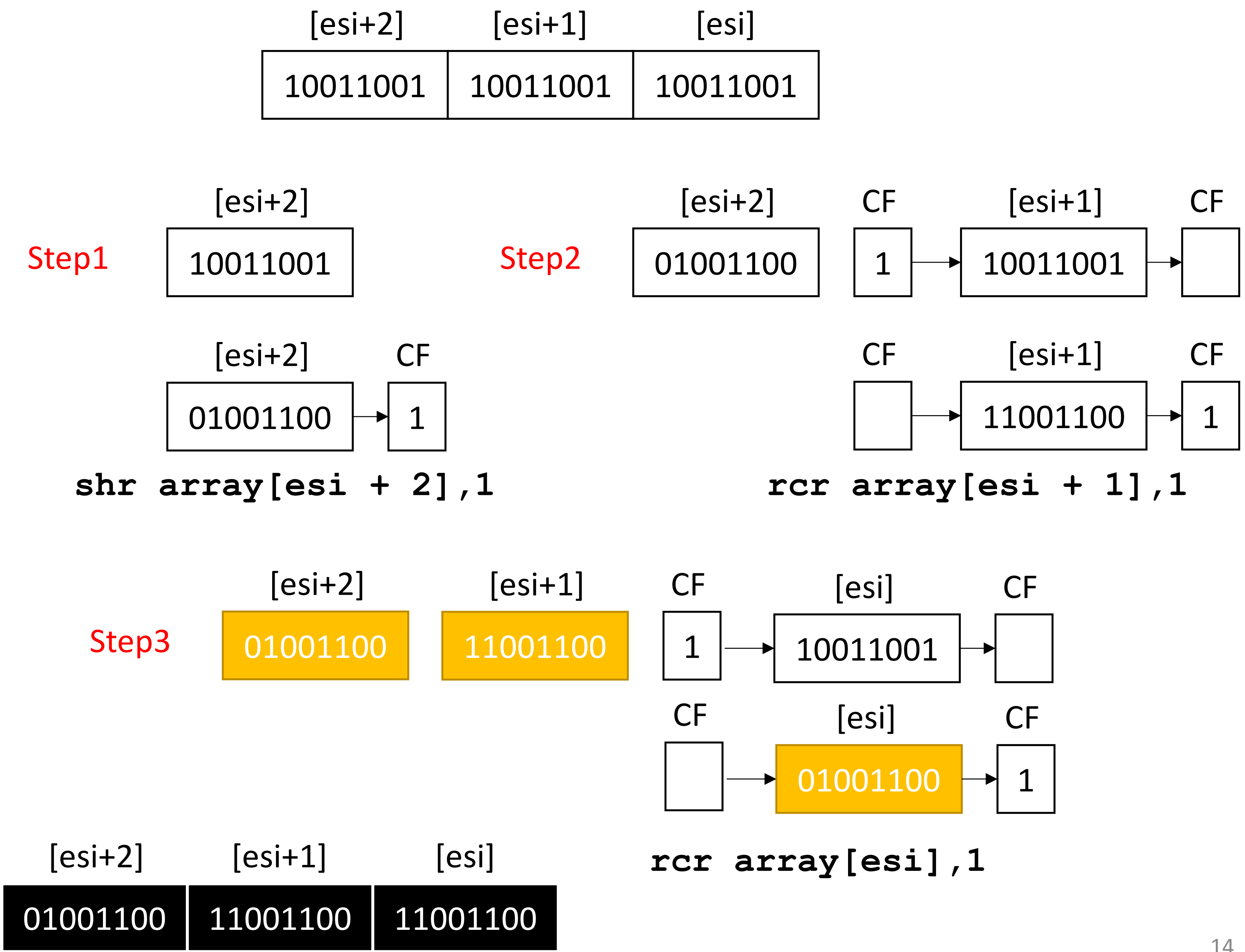


- Example for shifting multiple bytes

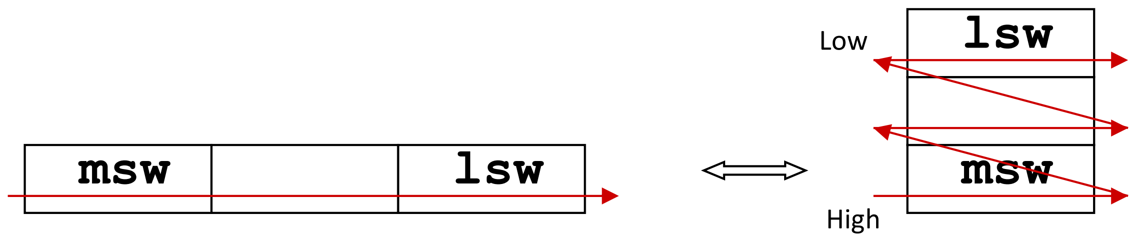
10011001

```
.data
ArraySize = 3
array BYTE ArraySize DUP(99h) ; 1001 pattern in each nybble.
.code
mov esi,0
shr array[esi + 2],1   ; high byte
rcr array[esi + 1],1   ; middle byte, include Carry
rcr array[esi],1       ; low byte, include Carry
```

four-bit aggregation

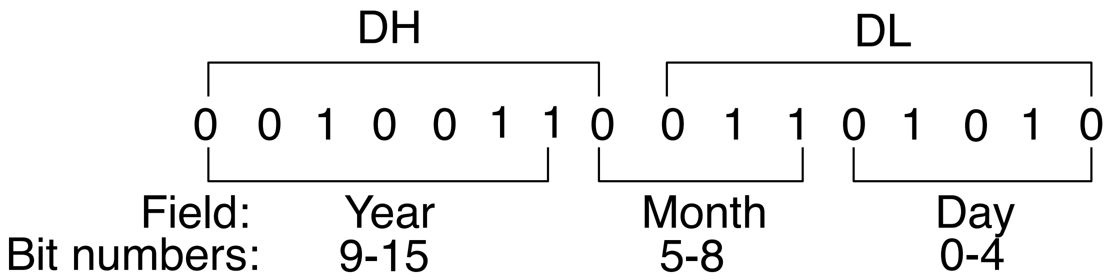

[esi+2] [esi+1] [esi]
10011001 10011001 10011001
Step1
[esi+2]
10011001
[esi+2] CF
01001100 1
shr array[esi + 2],1
Step2
[esi+2] CF [esi+1] CF
01001100 1 10011001
CF [esi+1] CF
11001100 1
rcr array[esi + 1],1
Step3
[esi+2] [esi+1] CF [esi] CF
01001100 11001100 1 10011001
CF [esi] CF
01001100 1
rcr array[esi],1
[esi+2] [esi+1] [esi]
01001100 11001100 11001100

- Shifting Multiple Doublewords




```
.data
ArraySize = 3
array DWORD ArraySize DUP(99999999h) ; 1001 1001...
.code
mov esi,0
shr array[esi + 8],1   ; high dword
rcr array[esi + 4],1   ; middle dword, include Carry
rcr array[esi],1       ; low dword, include Carry
```

- Isolating a Bit String: Data field of MS-DOS files



Isolating the month field

```
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 variable
```

- Binary Multiplication
  - Factor the multiplier into powers of 2 and shift.

```
EAX * 36
= EAX * (32 + 4)
= (EAX * 32)+(EAX * 4)
```

```
mov eax,123
mov ebx,eax
shl eax,5            ; mult by 2^5
shl ebx,2            ; mult by 2^2
add eax,ebx
```

  - Multiply AX by 26, using shifting and addition instructions. *Hint:* 26 = 16 + 8 + 2.

```
mov ax,26                ; test value
mov dx,ax
shl dx,4                 ; AX * 16
push dx                  ; save for later
mov dx,ax
shl dx,3                 ; AX * 8
shl ax,1                 ; AX * 2
add ax,dx                ; AX * 10
pop dx                   ; recall AX * 16
add ax,dx                ; AX * 26
```

- Displaying Binary Bits
  - Shift MSB into the Carry flag; If CF = 1, append a "1" character to a string; otherwise, append a "0" character. Repeat in a loop, 32 times.

; 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

For example:
32-bit binary integer
0101 0101 0101 0101 0101 0101 0101 0101
-> display ASCII binary string
0101 0101 0101 0101 0101 0101 0101 0101

```
.data
buffer BYTE 32 DUP(0),0
.code
    mov ecx,32
    mov esi,OFFSET buffer
L1: shl eax,1
    mov BYTE PTR [esi],'0'
    jnc L2                        ; Jump if not carry
    mov BYTE PTR [esi],'1'
L2: inc esi
    loop L1
```

Multiplication and Division Instructions

- MUL Instruction
- IMUL Instruction
- DIV Instruction
- Signed Integer Division
- CBW, CWD, CDQ Instructions
- IDIV Instruction
- Implementing Arithmetic Expressions

- MUL Instruction
  - The MUL (unsigned multiply) instruction multiplies an 8-, 16-, or 32-bit operand by either AL, AX, or EAX.
  - The instruction formats:

```
MUL r/m8
MUL r/m16
MUL r/m32
```

  - Implied Operands

Multiplicand	Multiplier	Product
AL	*r/m8*	AX
AX	*r/m16*	DX:AX
EAX	*r/m32*	EDX:EAX

- IMUL Instruction
  - IMUL (signed integer multiply) multiplies an 8-, 16-, or 32-bit signed operand by either AL, AX, or EAX.
  - Preserves the sign of the product by sign-extending it into the upper half of the destination register.

- MUL and IMUL Examples
  - 100h × 2000h, using 16-bit operands:

```
.data
val1 WORD 100h
.code
mov ax,2000h
mul val1       ; DX:AX = 00200000h, CF=1
```

Carry flag is set because the upper part of the product (DX) is not equal to zero.

```
mov eax,12345h
mov ebx,1000h
mul ebx         ; EDX:EAX = 0000000012345000h, CF=0
```

```
mov  al,48
mov  bl,4
imul bl               ; AX = 00C0h, OF=1
```

OF=1 because AH is not a sign extension of AL.

```
mov eax,4823424
mov ebx,-423
imul ebx ; EDX:EAX = FFFFFFFF86635D80h, OF=0
```

OF=0 because EDX is a sign extension of EAX.

- Exercises

```
mov ax,1234h
mov bx,100h
mul bx      ; DX = 0012h , AX = 3400h , CF = 1
```

```
mov eax,00128765h
mov ecx,10000h
mul ecx     ; EDX= 00000012h EAX=  87650000h CF=  1
```

```
mov ax,8760h
mov bx,100h
imul bx     ; DX=  FF87h      AX=   6000h     OF=   1
```

- DIV Instruction
  - The DIV (unsigned divide) instruction performs 8-bit, 16-bit, and 32-bit division on unsigned integers
  - A single operand is supplied (register or memory operand), which is assumed to be the divisor
  - Instruction formats:

DIV *r/m8*
DIV *r/m16*
DIV *r/m32*

Dividend	Divisor	Quotient	Remainder
AX	*r/m8*	AL	AH
DX:AX	*r/m16*	AX	DX
EDX:EAX	*r/m32*	EAX	EDX

  - Examples:

```
mov dx,0      ; clear dividend, high
mov ax,8003h ; dividend, low
mov cx,100h  ; divisor
div cx    ; AX = 0080h(quotient), DX = 3(remainder)
```

```
mov edx,0                   ; clear dividend, high
mov eax,8003h               ; dividend, low
mov ecx,100h                ; divisor
div ecx                     ; EAX = 00000080h, DX = 3
```

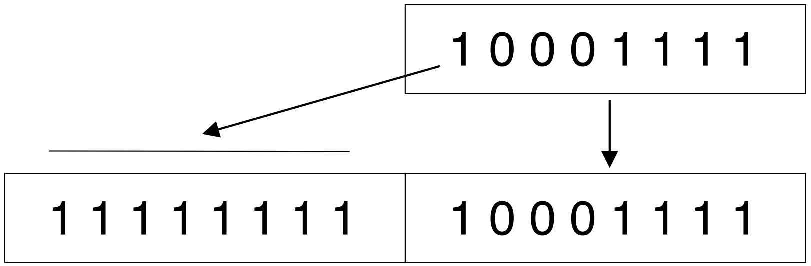
- Exercises

```
mov dx,0087h
mov ax,6000h
mov bx,100h
div bx          ; DX =  0000h     AX = 8760h
```

```
mov dx,0087h
mov ax,6002h
mov bx,10h
div bx     ;  Divide Overflow(Program will die)
```

DX:AX 0087:6002 ->
if it is divided by bx (10h) then,
the quotient is 87600 (5 bytes over 4 bytes), which does not fit in DX (4 bytes).
So, overflow occurs.

- Signed Integer Division
  - Signed integers must be sign-extended before division takes place.
    - Fill high byte/word/doubleword with a copy of the low byte/word/doubleword's sign bit.
  - An Illustration:



- CBW, CWD, CDQ Instructions
  - CBW (convert byte to word) extends AL into AH
  - CWD (convert word to doubleword) extends AX into DX
  - CDQ (convert dword to qword) extends EAX into EDX

```
mov eax,0FFFFFF9Bh        ; (-101)
cdq               ; EDX:EAX = FFFFFFFFFFFFFF9Bh
```

- IDIV Instruction
  - IDIV (signed divide) performs signed integer division.
  - The same syntax and operands as in DIV instruction.
  - Examples:

```
mov  al,-48   ; 8-bit division of -48 by 5
cbw  ; extend AL into AH
mov  bl,5
idiv bl        ; AL = -9,  AH = -3
```

```
mov  ax,-48    ; 16-bit division of -48 by 5
cwd            ; extend AX into DX
mov  bx,5
idiv bx        ; AX = -9,  DX = -3
```

```
mov  eax,-48   ; 32-bit division of -48 by 5
cdq            ; extend EAX into EDX
mov  ebx,5
idiv ebx       ; EAX = -9,  EDX = -3
```

- Examples (Cont')

```
mov   ax, FDFFh        ; -513
cwd
mov   bx,100h
idiv  bx          ; AX = FFFEh (-2)        DX = FFFFh (-1)
```

100h -> 256

-513/512 = 256*(-2)-1

AX (Quotient): -2 = FFFEh

DX (Remainder): -1 = FFFFh

```
mov   ax, 1000h
mov   bl, 10h
div   bl          ; AL =                   AH =
```

Divide overflow!

AL can't hold 100h

- Unsigned Arithmetic Expressions
  - Reasons to learn how to implement integer expressions:
    - Learn how do compilers do it
    - Test your understanding of MUL, IMUL, DIV, IDIV
    - Check for overflow (Carry and Overflow flags)
  - Example: `var4 = (var1 + var2) * var3`

```
; Assume unsigned operands
mov  eax,var1
add  eax,var2         ; EAX = var1 + var2
mul  var3             ; EAX = EAX * var3
jc   TooBig           ; check for carry
mov  var4,eax         ; save product
```

- Signed Arithmetic Expressions(by Examples)

```
mov   eax,var1   ; eax = (-var1 * var2) + var3
neg   eax
imul  var2
jo    TooBig             ; check for overflow
add   eax,var3
jo    TooBig             ; check for overflow
```

```
;   var4 = (var1 * 5) / (var2 – 3)
mov   eax,var1                 ; left side
mov   ebx,5
imul  ebx                      ; EDX:EAX = product
mov   ebx,var2                 ; right side
sub   ebx,3
idiv  ebx                      ; EAX = quotient
mov   var4,eax
```

Starting from the left is better

```
; var4 = (var1 * -5) / (-var2 % var3);
mov   eax,var2                 ; begin right side
neg   eax
cdq                            ; sign-extend dividend
idiv  var3                     ; EDX = remainder
mov   ebx,edx                  ; EBX = right side
mov   eax,-5                   ; begin left side
imul  var1                     ; EDX:EAX = left side
idiv  ebx                      ; final division
mov   var4,eax                 ; quotient
```

Starting from the right is better

- Exercises
  - **eax = (ebx * 20)/ecx** (use 32bit integers)

```
mov eax,20
imul ebx
idiv ecx
```

  - **eax = (ecx * edx)/eax** (use 32bit integers and save and restore ECX and EDX)

```
push  edx
push  eax                    ; EAX needed later
mov   eax,ecx
imul  edx                    ; left side: EDX:EAX
pop   ebx                    ; saved value of EAX
idiv  ebx                    ; EAX = quotient
pop   edx                    ; restore EDX, ECX
```

  - **var3 = (var1 * -var2)/(var3 – ebx)** (Do not modify any variables other than var3)

```
mov  eax,var1
mov  edx,var2
neg  edx
imul edx                 ; left side: EDX:EAX
mov  ecx,var3
sub  ecx,ebx
idiv ecx                 ; EAX = quotient
mov  var3,eax
```

- Extended Precision Arithmetic
  - Extended Precision Addition and Subtraction
  - ADC and SBB Instructions

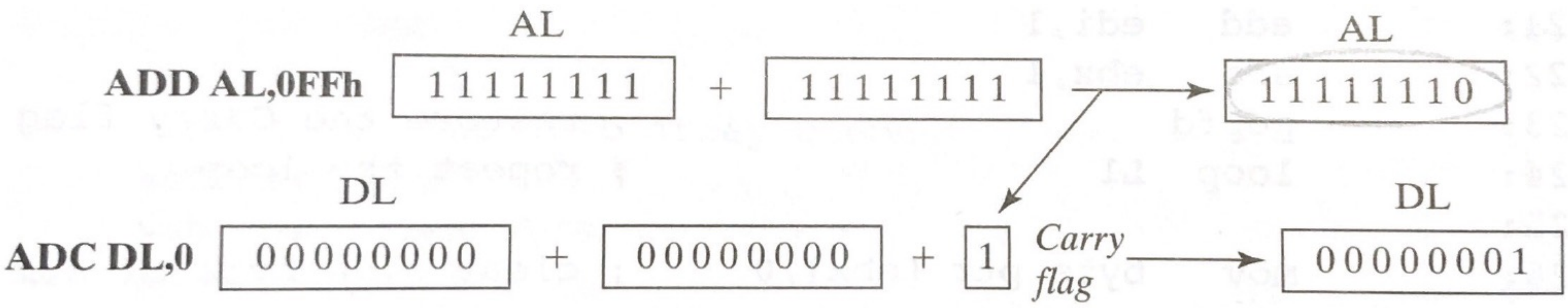
- Extended Precision Addition
  - Adding two operands that are longer than the computer's word size (32 bits).
    - Virtually no limit to the size of the operands
  - The arithmetic must be performed in steps
    - The Carry value from each step is passed on to the next step.

Extended Addition and Substraction

- ADC Instruction
  - ADC(add with carry) adds both a source operand and the contents of the Carry flag to a destination operand.
  - The same syntax as ADD, SUB, etc.
  - Example: Add two 32-bit integers (FFFFFFFFh + FFFFFFFFh), producing a 64-bit sum in EDX:EAX:

```
mov edx,0
mov eax,0FFFFFFFFh
add eax,0FFFFFFFFh
adc edx,0   ; EDX:EAX = 00000001FFFFFFFEh
```

```
mov dl,0
mov al,0FFh
add al,0FFh          ; AL = FFh
adc dl,0             ; DL:AL = 01FFh
```



- SBB Instruction
  - The SBB (subtract with borrow) instruction subtracts both a source operand and the value of the Carry flag from a destination operand.
  - The same syntax as the ADC instruction

- Extended Addition Example
  - Add 1 to EDX:EAX= 00000000FFFFFFFFh
    - Add the lower 32 bits first, setting the Carry flag.
    - Add the upper 32 bits, and include the Carry flag.

```
mov edx,0               ; set upper half
mov eax,0FFFFFFFFh      ; set lower half
add eax,1               ; add lower half
adc edx,0               ; add upper half
           ; EDX:EAX = 00000001 00000000
```

- Extended Subtraction Example
  - Subtract 1 from EDX:EAX= 0000000100000000h
    - Subtract the lower 32 bits first, setting the Carry flag.
    - Subtract the upper 32 bits, and include the Carry flag.

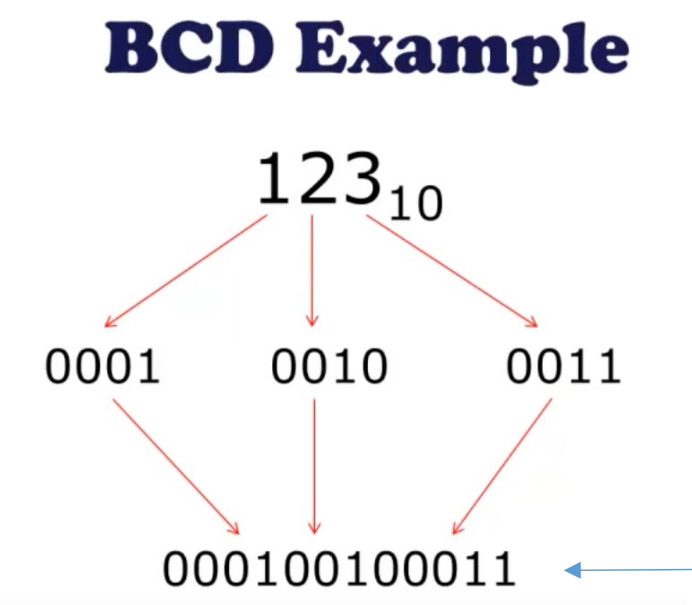
```
mov edx,1               ; set upper half
mov eax,0               ; set lower half
sub eax,1               ; subtract lower half
sbb edx,0               ; subtract upper half
           ; EDX:EAX = 00000000 FFFFFFFF
```

ASCII and Unpacked Decimal Arithmetic

- Binary Coded Decimal
- ASCII Decimal
- AAA Instruction
- AAS Instruction
- AAM Instruction
- AAD Instruction

- Binary-Coded Decimal
  - Binary-coded decimal (BCD) integers use 4 binary bits to represent each decimal digit
  - A number using unpacked BCD representation stores a decimal digit in the lower four bits of each byte
    - For example, 5,678 is stored as the following sequence of hexadecimal bytes:

05	06	07	08



Binary Coded Decimal

- # ASCII Decimal
  - A number using ASCII Decimal representation stores a single ASCII digit in each byte
    - For example, 5,678 is stored as the following sequence of hexadecimal bytes:

35	36	37	38

48	30	00110000	**0**	zero
49	31	00110001	**1**	one
50	32	00110010	**2**	two
51	33	00110011	**3**	three
52	34	00110100	**4**	four
53	35	00110101	**5**	five
54	36	00110110	**6**	six
55	37	00110111	**7**	seven
56	38	00111000	**8**	eight
57	39	00111001	**9**	nine
58	3A	00111010	**:**	colon
59	3B	00111011	**;**	semicolon
60	3C	00111100	**<**	less than
61	3D	00111101	**=**	equality sign
62	3E	00111110	**>**	greater than
63	3F	00111111	**?**	question mark
64	40	01000000	**@**	at sign
65	41	01000001	**A**	
66	42	01000010	**B**	

Some exert from ASCII table

AAA Instruction

- The aaa instruction is used to adjust the content of the AL register after that register has been used to perform the addition of two unpacked BCDs.
- The CPU uses the following logic:

```
If (al AND 0Fh > 9) or (the Auxilliary Flag is set))
    al = al + 6
    ah = ah + 1
    CF set
    AF set
ENDIF
Al = al AND 0Fh
```

AAA Instruction (Example1)

- Example: Add ‘2' and '2‘

```
mov al,'7'              ; al = 37h
add al,'2'              ; al = 69h
aaa                     ; ah = 9 (ah is unchanged.)
```

Why?

```
mov al, '7'        ; al = 37h
add al, '2'        ; al = 37h + 32h = 69h
aaa                ; al AND 0Fh = 69h AND 0Fh = 09h
                        (ah is unchanged. CF clear)
```

AAA Instruction (Example2)

- Example: Add '8' and '2'

```
mov ah,0
mov al,'8'           ; AX = 0038h
add al,'2'           ; AX = 006Ah
aaa                  ; AX = 0100h (adjust result)
or  ax,3030h         ; AX = 3130h = '10'
```

Why?

```
mov ah, 0        ; ah = 00h
mov  al, '8'     ; al = 38h
add al, '2'      ; al = 38h + 32h = 6Ah
aaa              ; al AND 0Fh -> 6Ah AND 0Fh = 0Ah
                 ; 0Ah > 9 then CF set
                           al = al+6 = 10h
                           ah = ah+1=01h
                           ax = 0101h (adjust result)
```

AAS Instruction

- The aas instruction is used to adjust the content of AL register after that register has been used to perform the subtraction of two unpacked BCDs.
- The CPU uses the following logic:

```
If (al AND 0Fh > 9) or (the Auxilliary Flag is set))
    al = al - 6
    ah = ah - 1
    CF set
ENDIF
Al = al AND 0Fh
```

AAA Instruction (Example1)

- Example: Subtract ‘7' and ‘9‘

```
mov al,'7'          ; al = 37h
sub al,'9'          ; al = FEh
aas                 ; ah = 9 (ah is unchanged.)
```

Why?

```
mov al, '7'     ; al = 37h
sub al, '9'     ; al = 37h – 39h  = FEh
aas             ; al AND 0Fh = FEh AND 0Fh = 0Eh
                ; 0Eh > 9
                        al = al – 6 = 0FEh – 6 = 0F8h
                        al = 0F8h AND 0Fh = 08h
                        ah = ah – 1, CF set
```

AAM Instruction

- The aam instruction is used to adjust the content of the AL and AH registers after the AL register has been used to perform the multiplication of two unpacked BCD bytes.
- The CPU uses the following simple logic:

```
al = al mod 10
ah = al/10
```

- Example

```
mov bl,05h    ; first operand
mov al,06h    ; second operand
mul bl        ; AX = 001Eh
aam           ; AX = 0300h
```

al = al mod 10 = 1Eh mod 10 = 00h
ah = al / 10 = 03h
AX = 0300h

AAD Instruction

- The aad instruction is used to adjust the content of the AX register before the register is used to perform the division of two unpacked BCDs by another unpacked BCD digit.
- The CPU uses the following logic:

```
al = ah*10 + al
ah = 0
```

- Example

```
.data
quotient  BYTE ?
remainder BYTE ?
.code
mov ax,0307h                  ; dividend
aad                           ; AX = 0025h
mov bl,5                      ; divisor
div bl                        ; AX = 0207h
mov quotient,al
mov remainder,ah
```