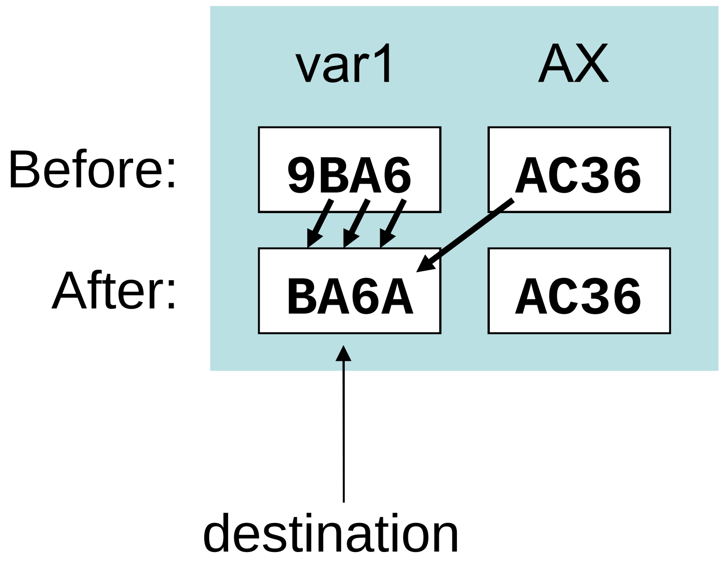
# SHLD Example

Shift variable **var1** 4 bits to the left

Replace the lowest 4 bits of **var1** with the high 4 bits of AX

```
.data
var1 WORD 9BA6h
.code
mov  ax, 0AC36h
shld var1, ax, 4
```

destination source count



|  | var1 | AX |
|---|---|---|
| Before: | 9BA6 | AC36 |
| After: | BA6A | AC36 |

Only the *destination* is modified, not the *source*

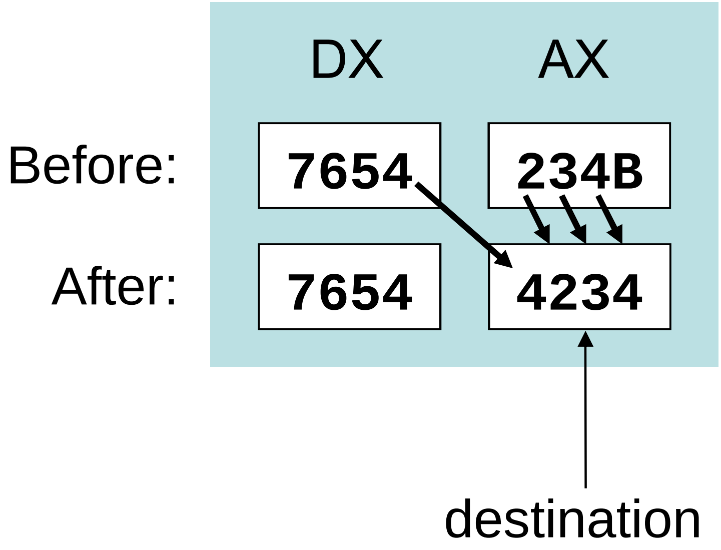
# SHRD Example

Shift AX 4 bits to the right

Replace the highest 4 bits of AX with the low 4 bits of DX

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

destination    source    count



Only the *destination* is modified, not the *source*

# Convert Number to Binary String

Task: Convert Number in EAX to an ASCII Binary String

Receives: EAX = Number
ESI = Address of binary string

Returns: String is filled with binary characters '0' and '1'

```
ConvToBinStr PROC USES ecx esi
    mov  ecx,32
L1: rol  eax,1
    mov  BYTE PTR [esi],'0'
    jnc  L2
    mov  BYTE PTR [esi],'1'
L2: inc  esi
    loop L1
    mov  BYTE PTR [esi], 0
    ret
ConvToBinStr ENDP
```

Rotate left most significant bit of EAX into the Carry flag; If CF = 0, append a '0' character to a string; otherwise, append a '1'; Repeat in a loop 32 times for all bits of EAX.

# Convert Number to Hex String

Task: Convert EAX to a Hexadecimal String pointed by ESI

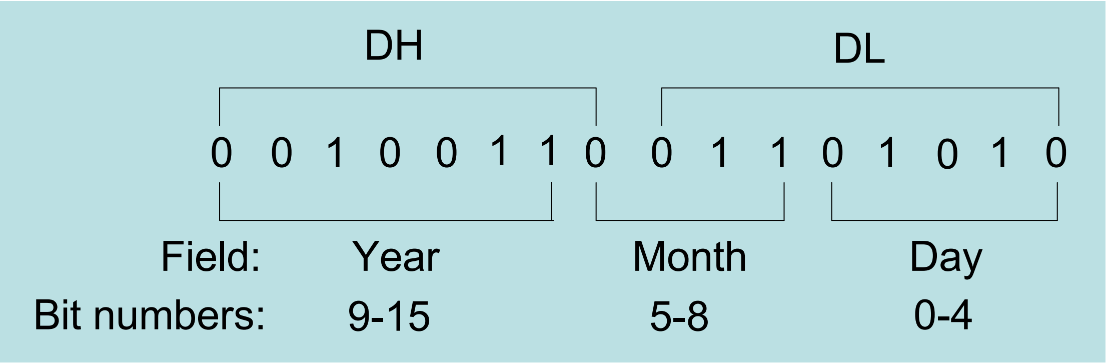
Receives: EAX = Number, ESI= Address of hex string

Returns: String pointed by ESI is filled with hex characters '0' to 'F'

```
ConvToHexStr PROC  USES ebx ecx esi
    mov  ecx, 8                  ; 8 iterations, why?
L1: rol  eax, 4                  ; rotate upper 4 bits
    mov  ebx, eax
    and  ebx, 0Fh                ; keep only lower 4 bits
    mov  bl,  HexChar[ebx]       ; convert to a hex char
    mov  [esi], bl               ; store hex char in string
    inc  esi
    loop L1                      ; loop 8 times
    mov  BYTE PTR [esi], 0       ; append a null byte
    ret
HexChar BYTE "0123456789ABCDEF"
ConvToHexStr ENDP
```

# Isolating a Bit String

- ❖ MS-DOS date packs the year, month, & day into 16 bits
  - ✧ Year is relative to 1980



DH

DL

0 0 1 0 0 1 1 0 0 1 1 0 1 0 1 0

Field: Year Month Day

Bit numbers: 9-15 5-8 0-4

In this example:

Day = 10
Month = 3
Year = 1980 + 19

Date = March 10, 1999

Isolate the Month field:

```
mov ax,dx           ; Assume DX = 16-bit MS-DOS date
shr ax,5            ; shift right 5 bits
and al,00001111b    ; clear bits 4-7
mov month,al        ; save in month variable
```

# Parameter Passing

- Parameter passing in assembly language is different
  - More complicated than that used in a high-level language
- In assembly language
  - Place all required parameters in an accessible storage area
  - Then call the procedure
- Two types of storage areas used
  - Registers: general-purpose registers are used (register method)
  - Memory: stack is used (stack method)
- Two common mechanisms of parameter passing
  - Pass-by-value: parameter **value** is passed
  - Pass-by-reference: **address** of parameter is passed

# Parameter Passing Through Stack

- Parameters can be saved on the stack before a procedure is called.
- The called procedure can easily access the parameters using either the ESP or EBP registers without altering ESP register.
- Example

Suppose you want to implement the following pseudo-code:

```
i = 25;
 j = 4;
Test(i, j, 1);
```

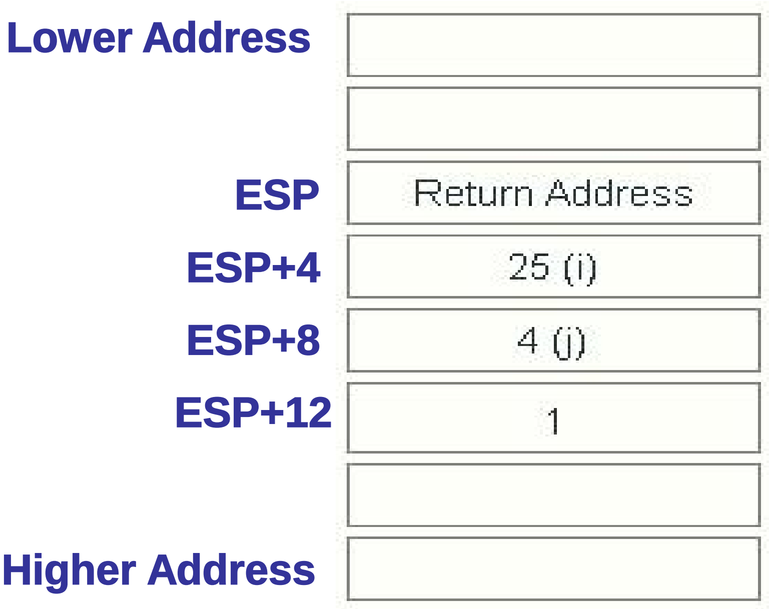
Then, the assembly language code fragment looks like:

```
mov i, 25
mov j, 4
push 1
push j
push i
call Test
```

# Parameter Passing Through Stack

Example: Accessing parameters on the stack

```
Test  PROC
   mov AX, [ESP + 4] ;get i
   add AX, [ESP + 8] ;add j
   sub AX, [ESP + 12] ;subtract parm 3
                              (1) from sum
   ret
Test ENDP
```



Lower Address

ESP Return Address

ESP+4 25 (i)

ESP+8 4 (j)

ESP+12 1

Higher Address

# Freeing Passed Parameters From Stack

❖ Use RET N instruction to free parameters from stack

Example: Accessing parameters on the stack

```
Test  PROC
   mov AX, [ESP + 4] ;get i
   add AX, [ESP + 8] ;add j
   sub AX, [ESP + 12] ;subtract parm. 3
                              (1) from sum
   ret 12
Test ENDP
```