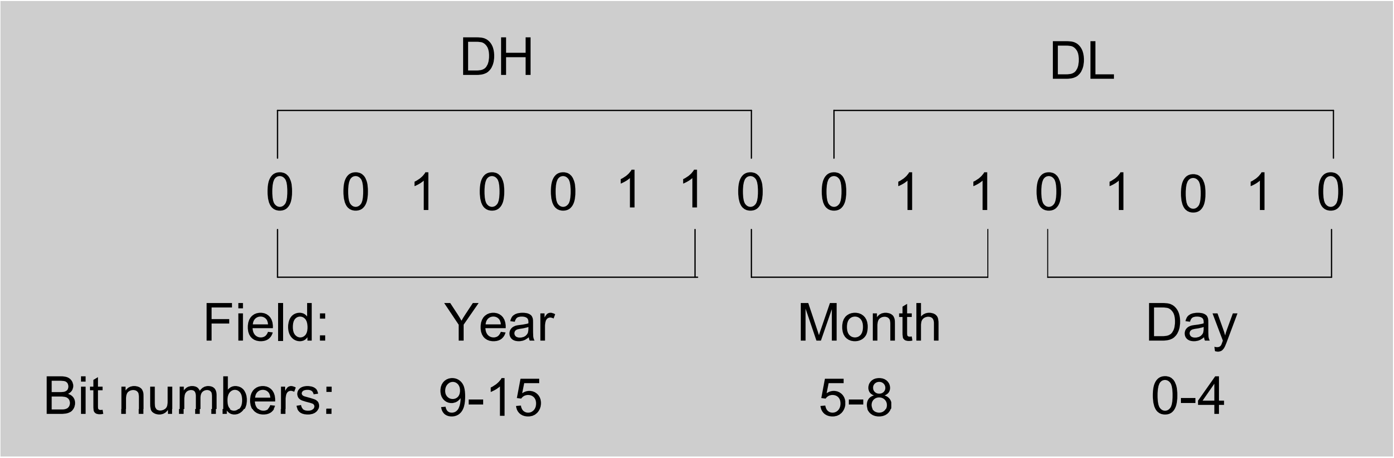
Isolating a bit string

- The MS-DOS file date field packs the year (relative to 1980), month, and day into 16 bits:



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

Isolating a bit string

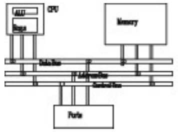
```
mov al,dl          ; make a copy of DL
and al,00011111b ; clear bits 5-7
mov day,al         ; save in day variable
```

```
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 variable
```

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

Multiplication and division

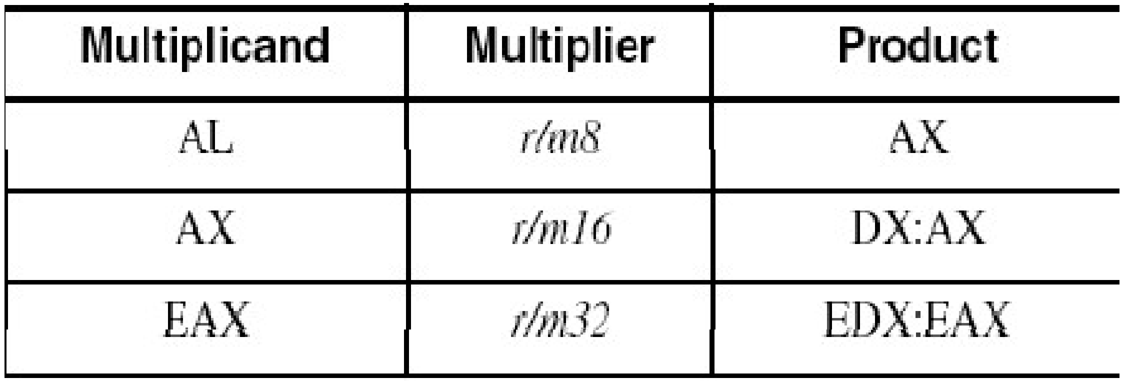
MUL instruction



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

```
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

MUL examples

100h * 2000h, using 16-bit operands:

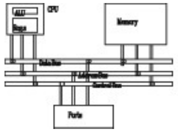
```
.data
val1 WORD 2000h
val2 WORD 100h
.code
mov ax,val1
mul val2  ; DX:AX=00200000h, CF=1
```

The Carry flag indicates whether or not the upper half of the product contains significant digits.

12345h * 1000h, using 32-bit operands:

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

IMUL instruction



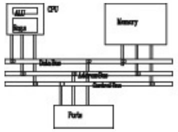
- IMUL (signed integer multiply) multiplies an 8-, 16-, or 32-bit signed operand by either AL, AX, or EAX (there are one/two/three operand format)
- Preserves the sign of the product by sign-extending it into the upper half of the destination register

Example: multiply 48 * 4, using 8-bit operands:

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

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

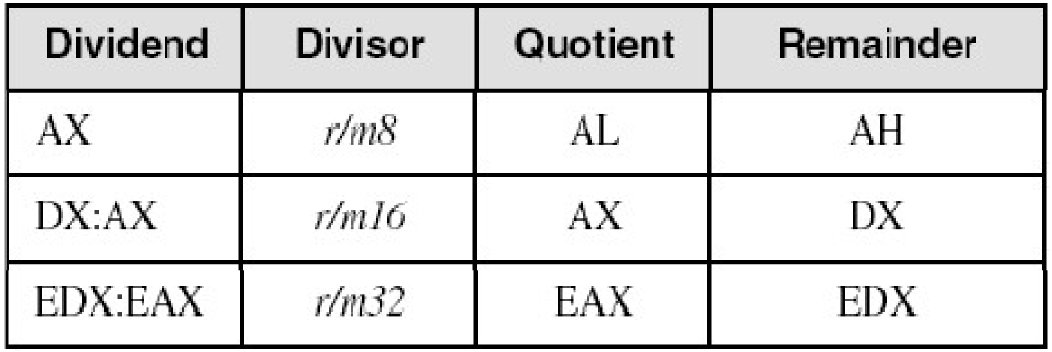
DIV instruction



- 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
```

Default Operands:



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

DIV examples

Divide 8003h by 100h, using 16-bit operands:

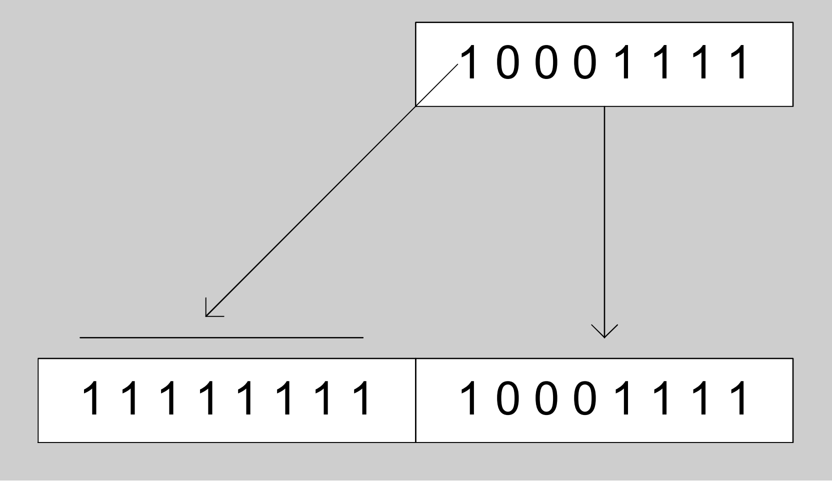
```
mov dx,0                  ; clear dividend, high
mov ax,8003h              ; dividend, low
mov cx,100h               ; divisor
div cx                    ; AX = 0080h, DX = 3
```

Same division, using 32-bit operands:

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

Signed integer division

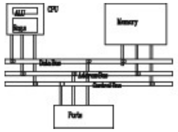
- Signed integers must be sign-extended before division takes place
  - fill high byte/word/doubleword with a copy of the low byte/word/doubleword's sign bit
- For example, the high byte contains a copy of the sign bit from the low byte:



1 0 0 0 1 1 1 1

1 1 1 1 1 1 1 1 | 1 0 0 0 1 1 1 1

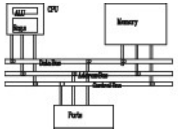
CBW, CWD, CDQ instructions



- The CBW, CWD, and CDQ instructions provide important sign-extension operations:
  - CBW (convert byte to word) extends AL into AH
  - CWD (convert word to doubleword) extends AX into DX
  - CDQ (convert doubleword to quadword) extends EAX into EDX
- For example:

```
mov eax,0FFFFFF9Bh      ; -101 (32 bits)
cdq          ; EDX:EAX = FFFFFFFFFFFFFF9Bh
             ; -101 (64 bits)
```

IDIV instruction

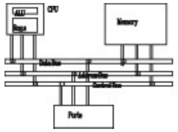


- IDIV (signed divide) performs signed integer division
- Uses same operands as DIV

Example: 8-bit division of –48 by 5

```
mov al,-48
cbw              ; extend AL into AH
mov bl,5
idiv bl          ; AL = -9,  AH = -3
```

IDIV examples



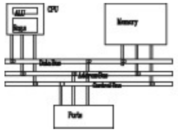
Example: 16-bit division of –48 by 5

```
mov  ax,-48
cwd                  ; extend AX into DX
mov  bx,5
idiv bx         ; AX = -9,  DX = -3
```

Example: 32-bit division of –48 by 5

```
mov  eax,-48
cdq             ; extend EAX into EDX
mov  ebx,5
idiv ebx        ; EAX = -9,  EDX = -3
```

Divide overflow



- *Divide overflow* happens when the quotient is too large to fit into the destination.

```
mov ax, 1000h
mov bl, 10h
div bl
```

It causes a CPU interrupt and halts the program. (divided by zero cause similar results)