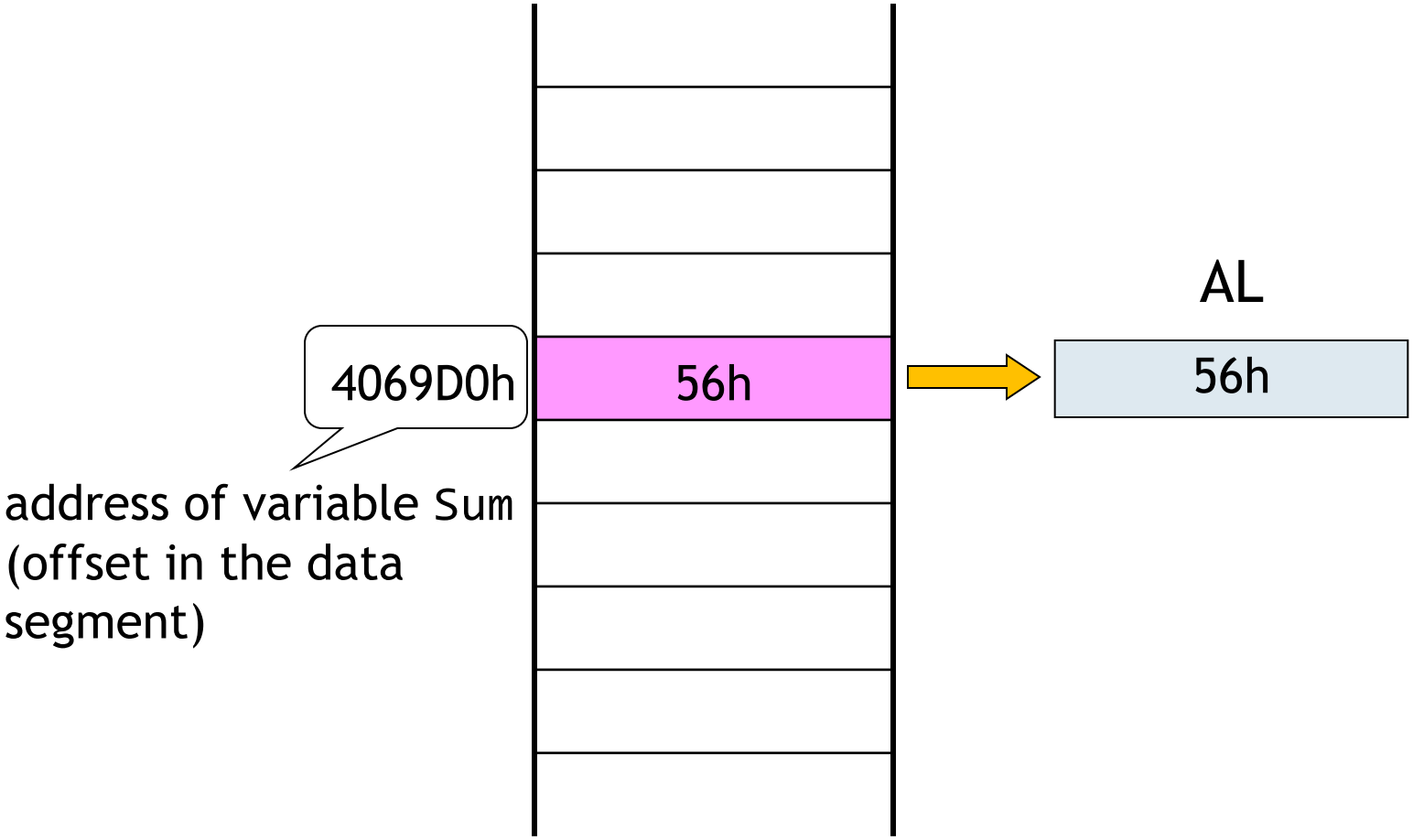
Instructions moving data

- do not affect flags.

mov register/mem, register/mem/number (move data)

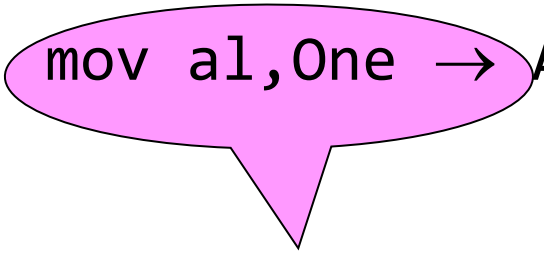
The difference between the value and the address of a variable

```
mov al,Sum; value 56h → al
mov ebx,offset Sum; address 4069D0h → ebx
```



When moving data to and from memory, the EAX register (or its part) is usually used, because:

```
One DB 1
mov bl,One → 8A 1D D0 69 40 00
mov al,One → A0 D0 69 40 00
```



better!

Hyde, R.: The Art of Assembly Language Programming, cap. 5.4 Encoding 80x86 Instructions

Operator offset cannot be used to get the address of a memory location pointed to by an indirect addressing mode (using a base or index register).

- Store the address of variable Table increased by EDI to EBX.

```
mov ebx,offset Table[edi]
```

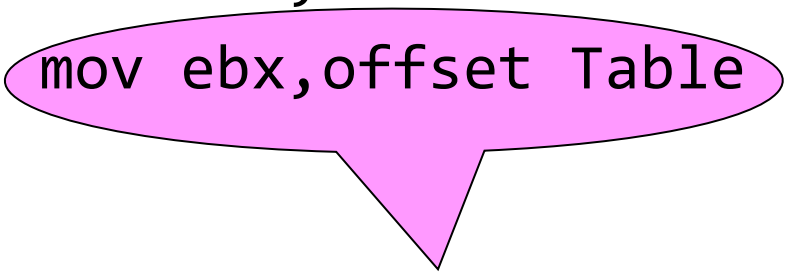
Does not work! (“Invalid operand for offset.”)

lea register,memory (load effective address)

- loads the offset of a memory location to a general-purpose register.

`lea ebx,Table` is the same as `mov ebx,offset Table`

```
lea ebx,Table          → 8D 1D 10 60 40 00
mov ebx,offset Table   → BB 10 60 40 00
```



better!

```
lea ebx,Table[edi]; store the address of variable
Table increased by EDI to EBX
```

movzx register, register/memory (move with zero-extend)

- copies the right operand into the lower half of the left operand and sets the upper half to 0

```
movzx bx,dl   is the same as   mov bl,dl
                               mov bh,0
```

movsx register, register/memory (move with sign-extend)

- copies the right operand into the lower half of the left operand and duplicates the highest order bit of the right operand throughout the upper half of the left operand.

```
Number DB -5

movsx ax,Number; ax = 1111 1111 1111 1011
```

is the same as

```
mov al,Number
mov ah,0FFh
```

xchg register/memory, register/memory (exchange)

- exchanges the contents of the operands

Arithmetic instructions

- affect flags ZF, CF, OF, SF, AC, PF

add register/memory, register/memory/number (add)

- adds the operands, stores the result to the left operand

sub register/memory, register/memory/number (subtract)

- subtracts the right operand from the left one, stores the result to the left operand

adc register/mem,register/mem/number (add with carry)

- adds the operands and CF, stores the result to the left operand
- useful when adding operands longer than a general-purpose register

```
X DQ 0FEDCBA98h    ; 98 BA DC FE 00 00 00 00
Y DQ 10000000h     ; 00 00 00 10 00 00 00 00
Z DQ ?; Z = X + Y
```


```
mov eax,dword ptr X;  eax = FEDCBA98
add eax,dword ptr Y;  eax = 0EDCBA98, CF = 1
mov dword ptr Z,eax
mov eax,dword ptr X+4; eax = 0
adc eax,dword ptr Y+4; eax = 1
mov dword ptr Z+4,eax
```

sbb register/memory,register/memory/number (subtract with borrow)

- subtracts the right operand and CF from the left operand, stores the result to the left operand
- useful when subtracting operands longer than a general-purpose register

inc register/memory (increment)

- increments the operand by 1
- does not affect CF!

```
add edi,1 → 83 C7 01
inc edi → 47
```

better!

dec register/memory (decrement)

- decrements the operand by 1
- does not affect CF!

cmp register/memory, register/memory/number (compare)

- compares the operands: subtracts the right operand from the left one, sets flags according to the result (does not store the result itself)

mul register/memory (unsigned multiply)

Operand type	Multiplied by	Result
byte	AL	AX
word	AX	DX:AX
dword	EAX	EDX:EAX

- If the upper half of the result is 0, then OF and CF are set to 0.
- Otherwise OF and CF are set to 1.

imul register/memory (signed multiply)

- multiplication of signed numbers
- implicit operands as mul

imul register, register/memory/number

- left operand = left * right
- Both operands must be of the same type (word or dword).

```
imul dx,word ptr [edi]
```

imul register, register/memory, number

- left operand = intermediate * right
- The leftmost and intermediate operands must be of the same type (word or dword).

If the upper half of the result is a signed extension of the lower half, then OF and CF are set to 0, otherwise to 1.

div register/memory (unsigned divide)

- integer division of unsigned numbers

Dividend	Type of the operand (divisor)	Quotient	Remainder
AX	byte	AL	AH
DX:AX	word	AX	DX
EDX:EAX	dword	EAX	EDX

- Divide decimal number 4001 in the AX register by ten.

```
mov dx,0
mov bx,10
div bx; ax = 400, dx = 1
```

If the divisor is zero or the quotient does not fit into the implicit register, an internal interrupt is generated.

idiv register/memory (signed divide)

- integer division of signed numbers
- implicit operands as div

Sign extension

If you want to divide signed operands of the same type, you must sign extend the dividend.

cbw (convert byte to word)

- Converts the byte in AL to a word in AX via sign extension (bit 7 of the AL register is copied into all bits of the AH register).

➢ Divide decimal number -15 in BH by 7 in BL.

```
mov al,bh;     al = 0F1h = -15
cbw;           ax = 0FFF1h = -15
idiv bl;       al = 0FEh = -2, ah = 0FFh = -1
```

cwd (convert word to doubleword)

- Converts the word in AX to a doubleword in DX:AX.

cwde (convert word to doubleword extended)

- Converts the word in AX to a doubleword in EAX.

cdq (convert doubleword to quadword)

- Converts the doubleword in EAX to the quadword in EDX:EAX.

Instructions cbw, ... do not affect flags.

neg register/memory (two's complement negation)

- switches the sign of the operand (replaces the operand by its two's complement)

```
mov al,-15 ; al = -15 = 11110001b
neg al     ; al = 15 = 00001111b
mov bl,al  ; bl = 15
neg bl     ; bl = -15
```

Logical instructions

- perform logical operations on the corresponding bits of the operands. The result is stored to the left operand.
- set flags:
  - CF and OF to 0
  - ZF, SF, PF according to the result of the operation
  - AC remains undefined

and register/memory, register/memory/number

- is useful when a single bit (several bits) of the left operand are to be set to 0. The right operand is a mask having 0 in the given bits and 1 in the other bits.

➢ Set the 3rd bit of BL to zero.

```
and bl,11110111b
```

➢ Convert the ASCII code of a digit to the value.

‘0’ = 00110000b, ‘1’ = 00110001b, ...

```
mov al,’9’
and al,00001111b; ’9’ → 9
```

or register/memory, register/memory/number

- is useful when a single bit (several bits) of the left operand are to be set to 1. The right operand is a mask having 1 in the given bits and 0 in the other bits.

➢ Set the 3rd bit of BL to 1.

```
or bl,00001000b
```

➢ Convert a number to the ASCII code of the corresponding digit:

```
mov al,9
or al,110000b; 9 → '9'
```

xor register/memory, register/memory/number

- is useful when a single bit (several bits) of the left operand are to be inverted. 1 in the mask inverts the corresponding bit, 0 leaves the bit unchanged.

➢ Invert the upper 4 bits of the BL register.

```
mov bl,01010101b
xor bl,11110000b ; bl = 10100101b
```

➢ Set the ECX register to zero:

```
mov ecx,0   → B9 00 00 00 00
xor ecx,ecx → 33 C9
```

better!

test register/memory, register/memory/number (logical compare)

- AND without saving the result
- is useful when we want to branch the execution according to the value of a chosen bit of the left operand.

➢ Branch according to the 3rd bit of the BL register.

```
test bl,00001000b
jz Zero; 3rd bit is 0
One: ...
```

not register/memory (one's complement negation)

- inverts all bits of the operand
- does not change flags

Shift and rotate instructions

The left operand is a register or memory location (byte, word or doubleword), whose bits are shifted or rotated to the left or right.

The right operand is a constant or register CL. The right operand determines the number of positions to shift.

Shift instructions

- set flags:
  - CF, OF, ZF, SF, PF according to the result of the operation
  - AC - undefined

shl register/memory, number/CL (shift logical left)

sal register/memory, number/CL (shift arithmetic left)

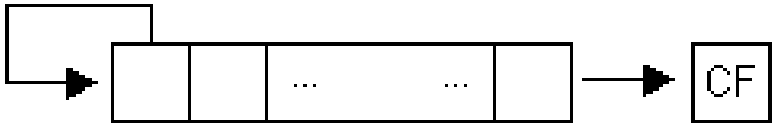
- useful for fast doubling (or multiplying by 4, 8 or 16) unsigned numbers.

```
shl dx,4; dx = 16*dx
```

shr register/memory, number/CL (shift logical right)

- halving (or dividing by 4, 8 or 16) unsigned numbers

sar register/memory, number/CL (shift arithmetic right)



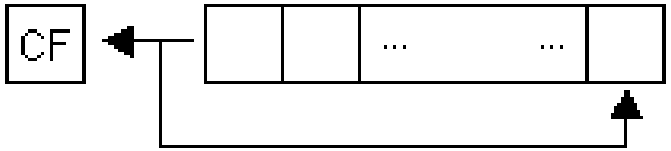
- halving (or dividing by 4, 8 or 16) signed numbers

```
mov bl,-4 ; bl = -4 = 11111100b
sar bl,1  ; bl = -2 = 11111110b
```

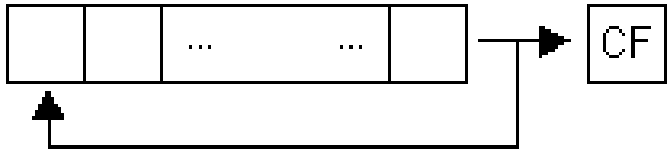
Rotate instructions

- set CF and OF

rol register/memory, number/CL (rotate left)



ror register/memory, number/CL (rotate right)



rcl register/memory, number/CL (rotate left through carry)

- useful when doubling operands longer than 32 bits. Because one operation can shift at most 32 bits, shifts of longer operands must be done by parts.

➢ Double variable `BigNumber` of type qword.

```
shl dword ptr BigNumber,1
rcl dword ptr BigNumber + 4,1
```

rcr register/memory, number/CL (rotate right through carry)

Bit set

To store a set in bits we need as many bits as there are all possible elements of the set.

- Example:
  Let variable `setX` is a bit set representing a subset of all integers $\in \langle 0, 31 \rangle$. Further, let the AL register store number $a \in \langle 0, 31 \rangle$. Add number a to the set `setX`.