Chapter 5

Arithmetic and Logic Instructions

Addition

- **ADD Operand1,operand2**
- ADD - used for binary addition
- operand1 = operand1 + operand2
- All of the addressing modes are used by addition.
- **ADD   REG, memory**
- **ADD   memory, REG**
- **ADD   REG, REG**
- **ADD   memory, immediate**
- **ADD   REG, immediate**

Ex1: ADD AX,BX   ;AX = AX + BX

Ex 2:   MOV AL, 5 ;

AL = 5

ADD AL, -3 ; AL=?

Ans: AL = 2

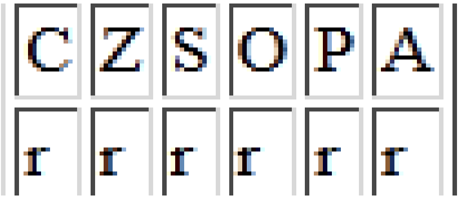
- **ADC Operand1,operand2**
- operand1 = operand1 + operand2 + CF
- **ADC   REG, memory**
- **ADC   memory, REG**
- **ADC   REG, REG**
- **ADC   memory, immediate**
- **ADC   REG, immediate**
- Ex1   ADC AX,DX ;AX = AX + DX + C
- Ex2   **STC ; set CF = 1**

MOV AL, 5 ; AL = 5

ADC AL, 1 ; AL = ?

AL = 7

Causes the flag bits to change.



C	Z	S	O	P	A
r	r	r	r	r	r

Increment / Decrement

INC REG,

INC memory

- operand = operand + 1
- The INC instruction adds a one to a register or the contents of a memory location.

  Ex1. INC BX ;BX = BX + 1

  Ex2: MOV AL, 4

  INC AL ; AL = 5

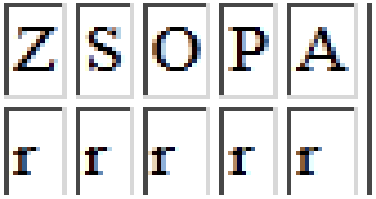
DEC REG,

DEC memory

- operand = operand - 1
- The DEC instruction subtracts one from a register or the contents of a memory location.

  Ex: DEC BX ;BX = BX – 1
- CF- UNCHANGED.

Causes the flag bits to change.



Z	S	O	P	A
r	r	r	r	r

Examples

- ADD AL,74H
- ;Add immediate number 74H to content of AL
- ADC CL,BL
- ;Add contents of BL plus carry status to contents of CL Results in CL
- ADD DX, BX
- ;Add contents of BX to contents of DX
- ADD DX, [SI]
- ;Add contents from memory at offset [SI] in DS {i.e.MA=DS:[SI]} to contents of DX
- ; AX = 7FFFh
  INC AX ;After this instruction AX = 8000h

Subtraction

- **SUB Operand1,operand2**
- **SUB** - used for binary subtraction
- *operand1 = operand1 - operand2*
- All of the addressing modes are used
- **SUB   REG, memory**
- **SUB   memory, REG**
- **SUB   REG, REG**
- **SUB   memory, immediate**
- **SUB   REG, immediate**

Example:

MOV AL, 5

SUB AL, 1 ; AL = 4

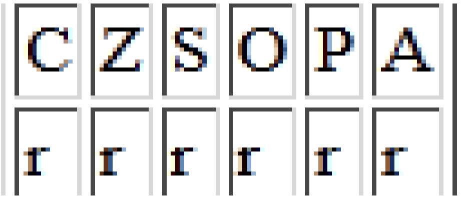
- **SBB Operand1,operand2**
- *operand1 = operand1 - operand2 - CF*
- All of the addressing modes are used by addition.
- **SBB   REG, memory**
- **SBB   memory, REG**
- **SBB   REG, REG**
- **SBB   memory, immediate**
- **SBB   REG, immediate**
- Example:

STC

MOV AL, 5

SBB AL, 3 ; AL = 5 - 3 - 1 = 1

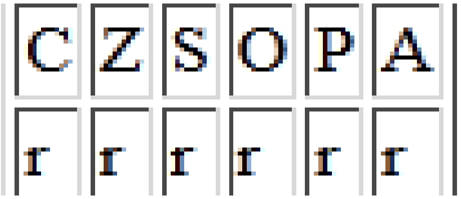
Causes the flag bits to change.



C	Z	S	O	P	A
r	r	r	r	r	r

Compare

Causes the flag bits to change.



C	Z	S	O	P	A
r	r	r	r	r	r

- CMP operand1, operand2
  CMP REG, memory
  CMP memory, REG
  CMP REG, REG
  CMP memory, immediate
  CMP REG, immediate
- operand1 - operand2
- The CMP instruction is a special form of the SUB instruction.
- Result is not stored anywhere,
- Flags are set (OF, SF, ZF, AF, PF, CF) according to result.

Example:

- MOV AL, 5
- MOV BL, 5
- CMP AL, BL ; AL = 5, ZF = 1 (so equal!) so result reflects change in Z flag.

String Compares

- The SCAS and CMPS instruction perform comparisons on blocks of data.
- SCAS is often used to search for a value
- CMPS is often used to compare two blocks.
- Prefix REPE and REPNE are often used to repeat the SCAS or CMPS instructions.

CMPS : Compare –String byte/word.

- CMPSB:
- Compare bytes: ES:[DI] from DS:[SI].
- DS:[SI] - ES:[DI]
- set flags according to result: OF, SF, ZF, AF, PF, CF
- if DF = 0 then
  - SI = SI + 1
  - DI = DI + 1
- else
  - SI = SI - 1
  - DI = DI - 1

•CMPSW:
•Compare words: ES:[DI] from DS:[SI].
DS:[SI] - ES:[DI]

•set flags according to result:
OF, SF, ZF, AF, PF, CF

•if DF = 0 then
  •SI = SI + 2
  •DI = DI + 2
•else
  •SI = SI - 2
  •DI = DI - 2

SCAS

SCASB

- **Compare bytes: AL from ES:[DI].**
- **AL - ES:[DI]**
- **if DF = 0 then**
  - **DI = DI + 1**
- **else**
  - **DI = DI – 1**

SCASW

- **Compare words: AX from ES:[DI].**

  Algorithm:

  AX - ES:[DI]

- **if DF = 0 then**
  - **DI = DI + 2**
- **else**
  - **DI = DI - 2**

C	Z	S	O	P	A
r	r	r	r	r	r

Multiplication

- The instruction exist to perform 8-, 16- multipplication.
- The result is always a double wide result.
- **MUL REG, memory (for unsigned) OR IMUL REG, memory (for signed)**

- when operand is a **byte**: AX = AL * operand.
- when operand is a **word**: (DX AX) = AX * operand.

- Example1: MUL (unsigned)

  MOV AL, 200 ; AL = 0C8h

  MOV BL, 4

  MUL BL ; AX = 0320h (800)
- **CF=OF=0 when high section of the result is zero.**

- Example2: IMUL (signed)
- MOV AL, -2
- MOV BL, -4
- IMUL BL ; AX = 8
- **CF=OF=0 when result fits into operand of IMUL.**
- **The CF carry and OF overflow bits indicate conditions about the multiplication**.

Division

- The DIV (unsigned) and IDIV (signed) instruction exist to perform division on 8-, 16-, or 32-bit numbers.
- Division is always performed o a double wide dividend.
- The result is always in the form of an integer quotient and an integer remainder.

DIV REG, memory

- when operand is a **byte**: AL = AX / operand & AH = remainder (modulus)
- when operand is a **word**: AX = (DX AX) / operand & DX = remainder (modulus) Example:

MOV AX, 203 ; AX = 00CBh

MOV BL, 4

DIV BL ; AL = 50 (32h)(division), AH = 3 (remainder)

IDIV REG, memory

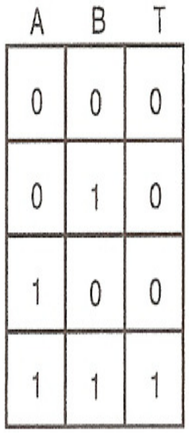
- when operand is a **byte**: AL = AX / operand & AH = remainder (modulus)
- when operand is a **word**: AX = (DX AX) / operand & DX = remainder (modulus) Example:
- MOV AX, -203 ; AX = 0FF35h
- MOV BL, 4
- IDIV BL ; AL = -50 (0CEh), AH = -3 (0FDh)

AND

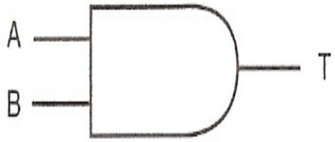
- The AND instruction performs logical multiplication (the AND operation).
- AND REG, memory
  AND memory, REG
  AND REG, REG
  AND memory, immediate
  AND REG, immediate
- Logical AND between all bits of two operands.
- Result is stored in operand1.
- Example:
- MOV AL, 'a' ; AL = 01100001b
- AND AL, 11011111b ; AL = ?
- ; AL = 01000001b ('A')
- Flags affected:

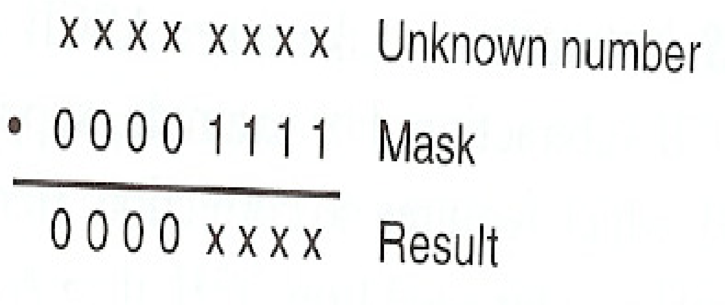
C Z S O P

0 r r 0 r



A	B	T
0	0	0
0	1	0
1	0	0
1	1	1



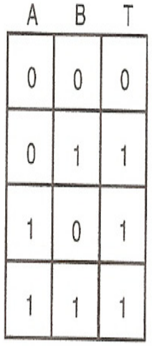


```
  xxxx xxxx   Unknown number
• 0000 1111   Mask
  ---------
  0000 xxxx   Result
```

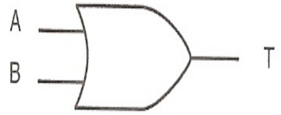
OR

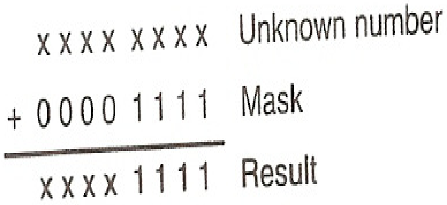
- The OR instruction generates the logical sum (OR operation).
- OR REG, memory
  OR memory, REG
  OR REG, REG
  OR memory, immediate
  OR REG, immediate
- Logical OR between all bits of two operands.
- Result is stored in operand1.
- Example:
- MOV AL, 'A' ; AL = 01000001b
- OR AL, 00100000b ; AL = 01100001b ('a‘)
- Flags affected:

C Z S O P A
0 r r 0 r ?



A	B	T
0	0	0
0	1	1
1	0	1
1	1	1





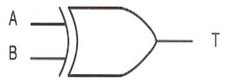
```
  xxxx xxxx  Unknown number
+ 0000 1111  Mask
  xxxx 1111  Result
```

Exclusive OR

- The XOR instruction performs the Exclusive OR operation.
- XOR REG, memory
  XOR memory, REG
  XOR REG, REG
  XOR memory, immediate
  XOR REG, immediate
- Result is stored in operand1.
- Example:
- MOV AL, 00000111b
- XOR AL, 00000010b ; AL = 00000101b
- Flags affected:

C	Z	S	O	P	A
0	r	r	0	r	?

A	B	T
0	0	0
0	1	1
1	0	1
1	1	0



$$\begin{array}{ll} XXXX\ XXXX & \text{Unknown number} \\ \oplus 0000\ 1111 & \text{Mask} \\ \hline XXXX\ \bar{X}\bar{X}\bar{X}\bar{X} & \text{Result} \end{array}$$

NEG and NOT

NEG REG, memory

- Negate: Makes operand negative (two's complement).
- Invert all bits of the operand , then
- Add 1 to inverted operand
- Example:

```
MOV AL, 5 ; AL = 05h
 NEG AL ; AL = 0FBh (-5)
 NEG AL ; AL = 05h (5)
```

- Flags affected:

C	Z	S	O	P	A
r	r	r	r	r	r

NOT REG, memory

- Invert each bit of the operand.

Example:

- MOV AL, 00011011b
- NOT AL ; AL = 11100100b
- Flags affected:
- C Z S O P A ----unchanged

Logical Shift

- The logical shifts reposition the bits in a number. (SHR and SHL)

SHR op1,op2:

SHR memory, immediate
SHR REG, immediate
SHR memory, CL
SHR REG, CL

- Shift operand1 Right. The number of shifts is set by operand2.
- Shift all bits right, the bit that goes off is set to CF.
- Zero bit is inserted to the left-most position.
- Example:
- MOV AL, 00000111b
- SHR AL, 1 ; AL = 00000011b, CF=1.
- Flags affected: C O =r r
- OF=0 if first operand keeps original sign.

SHL

SHL op1,op2:

SHL memory, immediate
SHL REG, immediate
SHLmemory, CL
SHL REG, CL

- Shift operand1 Left. The number of shifts is set by operand2.
- Shift all bits left, the bit that goes off is set to CF.
- Zero bit is inserted to the right-most position.
- Example:MOV AL, 11100000b
- SHL AL, 1 ; AL = 11000000b, CF=1.
- Flags: C O= r r
- OF=0 if first operand keeps original sign.

Arithmetic shift

- The arithmetic shifts multiply or divide signed numbers by powers of two.
- SAR and SAL are arithmetic shifts.

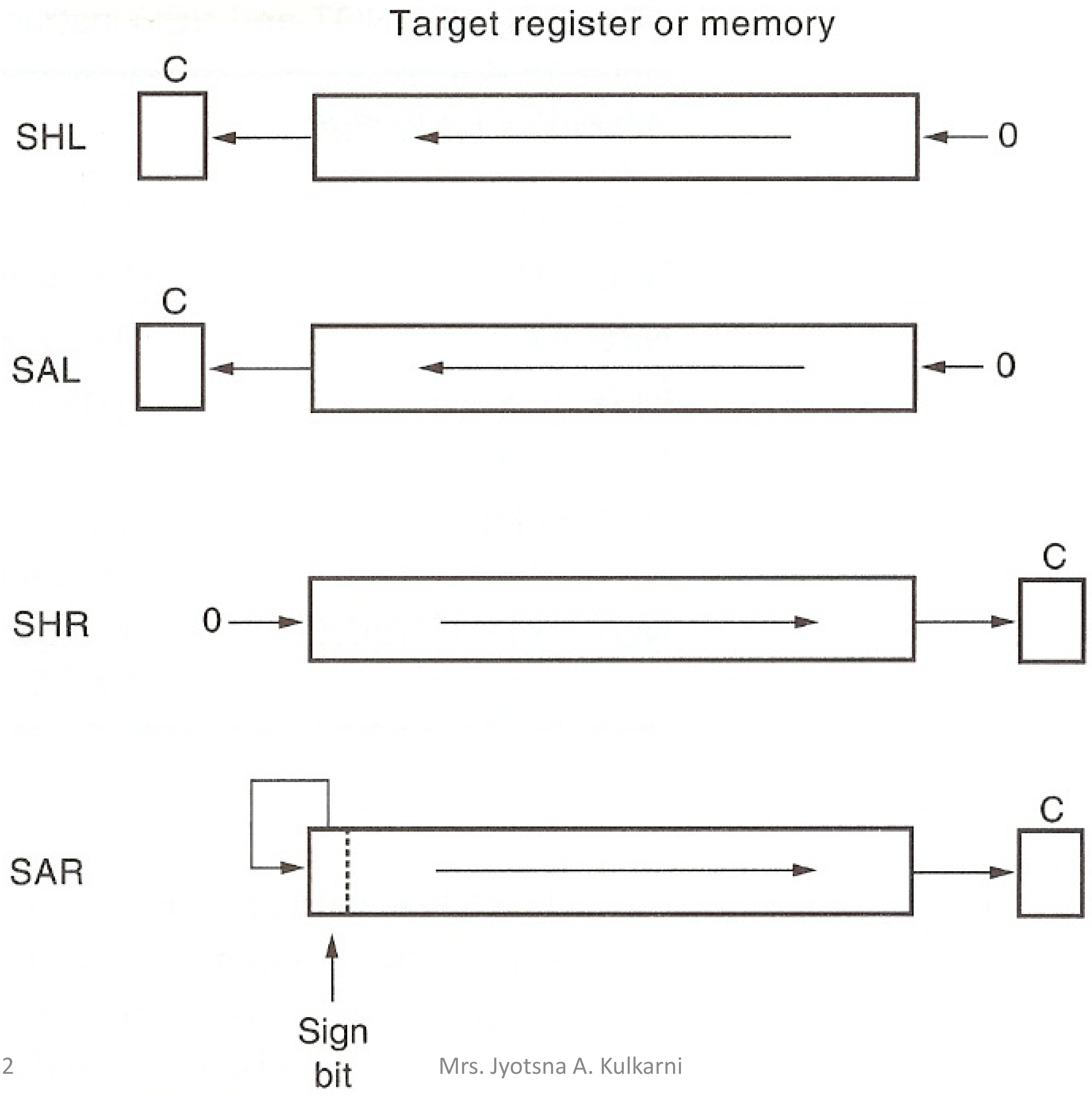
SAR

- SAR memory, immediate
  SAR REG, immediate
  SAR memory, CL
  SAR REG, CL
- Shift Arithmetic operand1 Right. The number of shifts is set by operand2.
- Shift all bits right, the bit that goes off is set to CF.
- The sign bit that is inserted to the left-most position has the same value as before shift.
- Example:MOV AL, 0E0h ; AL = 11100000b
- SAR AL, 1 ; AL = 11110000b, CF=0.
- MOV BL, 4Ch ; BL = 01001100b
- SAR BL, 1 ; BL = 00100110b, CF=0.
- Flags: C O =r r
- OF=0 if first operand keeps original sign.

SAL

- SAL memory, immediate
  SAL REG, immediate

  SAL memory, CL
  SAL REG, CL
- Shift Arithmetic operand1 Left. The number of shifts is set by operand2.
- Shift all bits left, the bit that goes off is set to CF.
- Zero bit is inserted to the right-most position.
- Example:MOV AL, 0E0h ; AL = 11100000b
- SAL AL, 1 ; AL = 11000000b, CF=1.
- Flags: C O = r r
- OF=0 if first operand keeps original sign.


Target register or memory
C
SHL
0
C
SAL
0
SHR
0
C
SAR
C
Sign bit

- .

HLT & NOP