Label opCode operands

NEXT: MOV AX, [CX] ;next element

Move limitations

SEGMENT REGISTER MOVES

Although we do not often directly address **segment registers** it is important to understand the limitations of the **segment register MOV instruction.**

- **Immediate data cannot be moved into a segment register.**
- **CS cannot successfully be loaded with a segment register MOV.**
- **A MOV instruction from segment-to-segment register is not allowed.**

ADDITION

ADD EAX, EBX ;EAX=EAX+EBX

- The ADD instruction is used for binary addition.
- The addition causes the flag bits to change.
- Addition can be 8-, 16-, and 32-bits.

```
MOV AL, 0F5H
ADD AL, 0BH
```

→ AL=F5H+0BH

```
F5H   1111 0101
0BH   0000 1011
100H  0000 0000
```

$ZF = 1$ (result not zero)
$CF = 1$ (no carry)
$AF = 1$ (no half carry)
$SF = 0$ (result is positive)
$PF = 1$ (odd parity)
$OF = 0$ (no overflow)

ADDITION

AL [Counter] 0123H

- **The destination operand can be a register or in memory.**
- **The source operand can be a register or in memory or immediate.**
- **Memory-to-memory operations are never allowed in 80x86 assembly language.** → Add [shift],[Counter]

ADD destination,source

Unsigned Addition.

- The status of **CF**, **ZF**, **SF**, **AF**, and PF are affected.
- Only CF, ZF, an AF are of any use to programmers (other flags are not important)

1. **CF, The carry Flag:** set to 1 whenever there is a **carry out after addition or the borrow after subtraction** , either from d7 after 8-bit operation, or from d15 after 16-bit operation.
2. **PF, The parity flag:** the parity is set to one if the **low order byte has even number of ones** (zero ones is considered even number of ones)
3. **AF, Auxiliary flag:** set to 1 when there is a **half-carry from d3 to d4.**
4. **ZF, The zero flag:** set to 1 if the **result** of an arithmetic or logical operation is **zero**
5. **SF, The sign flag:** indicates the sign of the result (-ve two's complement values has the most significant bit "1', while +ve values have "0")
6. **OF, The overflow flag:** set to 1 if the result of signed number operation is too large (exceed capacity of register)

Example_2

```
MOV DX,126FH   \\DX=126FH
ADD DX,3465H   \\ BH=46D4H (126FH+3465H = 46D4H)
```

```
126F → 0001 0010 0110 1111
+
3465 → 0011 0100 0110 0101
       0100 0110 1101 0100
```

CF = 0 AF = 1 SF = 0 ZF = 0 PF = 1

GUC German University in Cairo

ADD destination, source

Signed Addition.

- The status of OF, ZF, SF are very important to monitor.
- A special attention must be given to the **overflow flag (OF)**

Overflow

- In 2's complement representation, given n bits, the range of numbers that can be represented is calculated as follows $[-2^{n-1}, 2^{n-1}-1]$
- If n = 8 , the range of numbers that can be represented **[-128, +127]**
- Overflow occurs when the available number of bits can not accommodate the number needs to be represented. Ex: trying to represent 135 in 8 bits. **8 bits allow the range [-128, +127]**
- When overflow occurs the OF is set to 1

ADD destination, source

Signed Addition.

- The status of OF, ZF, SF are very important to monitor.
- A special attention must be given to the overflow flag (OF)

Example_1

```
MOV AL,+66   \\AL= +66
MOV CL, +69  \\ CL = +69
ADD CL, AL   \\ CL=CL+AL = 135
```

as Range [-128 to 127]

XOR = 1 (OF) Sign bit

```
+66 → 0100 0010
+
+69 → 0100 0101
      1000 0111   overflow
```

SF = 1 OF = 1 ZF = 0

GUC

Example_2

```
MOV AL, -12   \\AL= -12
MOV CL, +18   \\ CL = +18
ADD CL, AL    \\ CL=CL+AL = +6
```

2's complement

```
+12 → 0000 1100
      1111 0011
           1+
      1111 0100
```

XOR = 0 sign bit

```
-12 → 1111 0100
+
+18 → 0001 0010
     1 0000 0110
```

There is a carry from d6 to d7 **AND** a carry from d7 out

OF = 0 SF = 0 ZF = 0

ADD WITH CARRY

```
ADC AX,DX   ;AX=AX+DX+C
```

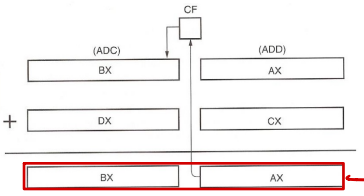
- The ADC instruction adds the carry bit into the sum.
- Used for wide additions (wider than 32-bits) and other reasons.

To add 128 bits using 64 Registers
split 128 to 64 64

ex:
```
ADD RAX,RCX
ADC RDX,RBX
```
Result stored at RDX:RAX

```
ADD AX,CX   ;AX=AX+CX
ADC BX,DX   ;BX=BX+DX+CF
```

Carry flag Register



Result is stored at BX:AX

SUBTRACTION

```
SUB AL,3    ;AL=AL – 3
SUB ECX,ESI ;ECX=ECX – ESI
```

- The **SUB instruction** performs subtraction and the flags change to reflect condition of the result.
- As with other arithmetic and logic instructions, subtraction exists for 8-, 16-, and 32-bit data.

All flags are affected by the subtract instruction, but CF and ZF are very important to monitor.

SUB AL, BL

	CF	ZF	
dest > source	0	0	The result is positive
dest = source	0	1	The result is zero
dest < source	1	0	The result is negative 2's complement

DECREMENT

```
DEC EBX   ;EBX=EBX-1
```

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

- Subtracts 1 from the destination operand
- Format: **DEC** dest ⟶ dest = dest - 1
- All flags are affected by the DEC instruction, except for CF. CF remains unaffected after the execution of the instruction **DEC**

INCREMENT

```
INC BX   ;BX=BX+1
```

- The INC instruction **adds a one** to a register or the contents of a memory location.

Can I do this INC [Row_Counter]

COMPARE

```
CMP AL,3   ;if AL=3 the result to zero (flag)
```

- The **CMP instruction** is a special form of the **SUB instruction**. A compare does not result in a difference that is saved, on the flag bits change to reflect the difference.

It sub the 2 Values [only changes the flags]

CMP Instruction

- It compares two operands of the same size.
- The **source and destination** operands are **not changed.**
- The comparison is performed by **subtracting** the source operand from the destination operand.
- Format: **CMP dest,source** ⟶ Sets flags as if **SUB** dest, source

Signed Numbers

	CF	ZF	SF	OF
dest > source	0	0	0	SF
dest = source	0	1	0	SF
dest < source	1	0	1	$\overline{SF}$

Unsigned Numbers

	CF	ZF
dest > source	0	0
dest = source	0	1
dest < source	1	0

MULTIPLICATION

MUL CL

- The MUL (unsigned) and IMUL (signed) instruction exist to perform 8-, 16-, or 32-bit multiplication.
- The result is always a double wide result.

MPLE

MUL CL → CL is multiplied by AL and the product is stored in AX

MUL BX → BX is multiplied by AX and the product is stored in DX:AX

Write an assembly code to calculate 3225H * 0102H

MUL BX \\ $DX:AX = BX * AX$

```
.data segment
DATA1  DB 25H
DATA2  DB 32H
DATA3  DW 0102H

.code segment
MOV AL,DATA1
MOV AH,DATA2
MOV BX,DATA3
MUL   BX
ret
```

AH	AL	
32	25	
0011 0010	0010 0101	12837

BH	BL	
01	02	
0000 0001	0000 0010	258

0000 0011 0010 1000 1001 0100 1010 — 3311946

DH	DL	AH	AL
00	32	89	4A

GUC German University in Cairo

DIVISION

DIV BX

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

- Divides either the value stored inside **AX** by the source operand. (when the source operand is 8-bits)

OR the value stored in **DX:AX** by the source operand. (when the source operand is 16-bits)

Format: DIV source

- DIV BX \\ $\frac{DX:AX}{BX}$ → Quotient @ AX, Remainder @ DX
- DIV BL \\ $\frac{AX}{BL}$ → Quotient @ AL, Remainder @ AH

- NOT (Logical NOT)
  - Format: **NOT** dest → dest = 1's complement of dest
- OR (Logical OR)
  - Format: **OR** dest,source → dest = dest **OR** source
- AND (Logical AND)
  - Format: **AND** dest,source → dest = dest **AND** source

NEG vs NOT Instructions

- The NEG (negate) instruction 2's complements a number
- The NOT instruction 1's complements a number.