Arithmetic instructions
Comparison instructions
Flow Control instructions

1

Arithmetic instructions
Basic instructions

ADD *destination,source*; (destination=destination+source)

destination and *source* have the same rules of mov

INC *destination*; (destination=destination+1)

INC is faster than ADD

SUB *destination,source* (destination=destination-source)

destination and *source* have the same rules of mov

DEC *destination*; (destination=destination-1)

DEC is faster than SUB

2

Comparison instruction

CMP *destination,source* (temporary=destination-source)

destination and *source* have the same rules than mov

destination is unchanged

FLAGS are affected according the subtraction operation

Note: *The Flag register is a Special Purpose Register. Depending upon the value of result after any arithmetic and logical operation the flag bits become set (1) or reset (0).*

3

Flow control instructions
Unconditional jump

JMP label

Performs an unconditional jump to label
label can be put before or after the jmp line
label must be unique in the file code

Example

```
next:   ADD EAX,2
        JMP next
```

This code performs an infinite loop

4

Flow control instructions conditional jumps

Conditional jumps control the program according conditions

The jump is made just if condition is verified

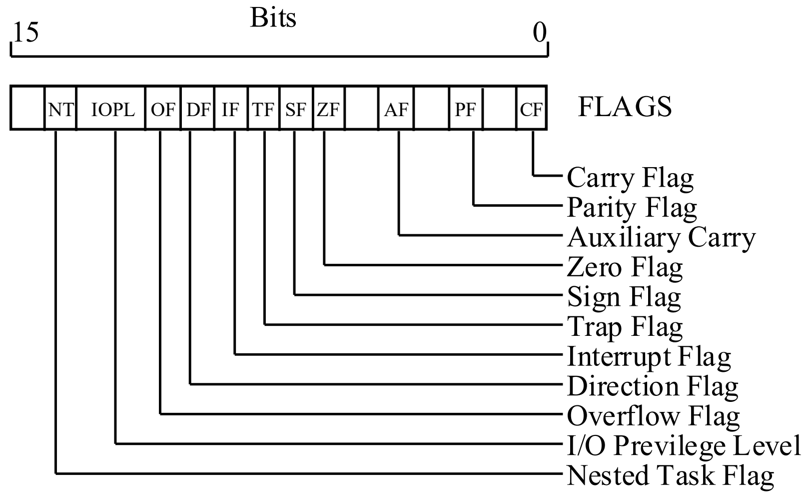
If jump isn't made the next line instruction is executed

Condition is verified by the contends of FLAGS

Conditional jumps are affected by any instruction that change the FLAGS

There are different conditions for signed or unsigned numbers

5



6

Flow control instructions
conditional jumps (most used at red)

Opcode	Description	CPU Flags
JA	Above	CF = 0 and ZF = 0
JAE	Above or equal	CF = 0
JB	Bellow	CF
JBE	Bellow or equal	CF or ZF
JC	Carry	CF
JE	Equality	ZF
JG	Greater(s)	ZF = 0 and SF = OF
JGE	Greater of equal(s)	SF = OF
JL	Less(s)	SF ≠ OF
JLE	Less equal(s)	ZF or SF ≠ OF
JNA	Not above	CF or ZF
JNAE	Neither above nor equal	CF
JNB	Not bellow	CF = 0
JNBE	Neither bellow nor equal	CF = 0 and ZF = 0

Opcode	Description	CPU Flags
JNC	Not carry	CF = 0
JNE	Not equal	ZF = 0
JNG	Not greater	ZF or SF ≠ OF
JNGE	Neither greater nor equal	SF ≠ OF
JNL	Not less	SF = OF
JNLE	Not less nor equal	ZF = 0 and SF = OF
JNO	Not overflow	OF = 0
JNP	Not parity	PF = 0
JNS	Not negative	SF = 0
JNZ	Not zero	ZF = 0
JO	Overflow(s)	OF
JP	Parity	PF
JPE	Parity	PF
JPO	Not parity	PF = 0
JS	Negative(s)	SF
JZ	Null	ZF

7

Instruction	Description	Flags tested
JE/JZ	Jump Equal or Jump Zero	ZF
JNE/JNZ	Jump not Equal or Jump Not Zero	ZF
JG/JNLE	Jump Greater or Jump Not Less/Equal	OF, SF, ZF
JGE/JNL	Jump Greater or Jump Not Less	OF, SF
JL/JNGE	Jump Less or Jump Not Greater/Equal	OF, SF
JLE/JNG	Jump Less/Equal or Jump Not Greater	OF, SF, ZF

8

Instruction	Description	Flags tested
JE/JZ	Jump Equal or Jump Zero	ZF
JNE/JNZ	Jump not Equal or Jump Not Zero	ZF
JA/JNBE	Jump Above or Jump Not Below/Equal	CF, ZF
JAE/JNB	Jump Above/Equal or Jump Not Below	CF
JB/JNAE	Jump Below or Jump Not Above/Equal	CF
JBE/JNA	Jump Below/Equal or Jump Not Above	AF, CF

9

Instruction	Description	Flags tested
JXCZ	Jump if CX is Zero	none
JC	Jump If Carry	CF
JNC	Jump If No Carry	CF
JO	Jump If Overflow	OF
JNO	Jump If No Overflow	OF
JP/JPE	Jump Parity or Jump Parity Even	PF
JNP/JPO	Jump No Parity or Jump Parity Odd	PF
JS	Jump Sign (negative value)	SF
JNS	Jump No Sign (positive value)	SF

10

Flow control instructions
loop implementation

```
        mov ecx,0            ;initiate the loop counter
repeat: ...
        ...
        inc ecx
        cmp ecx,100
        jne repeat
        ... (next line is execute when ecx reached 100)
```

Loop lines code executed 100 times

11

Flow control instructions
loop implementation

```
        mov ecx,100;   start the counter at max value
repeat: ...
        ...
        dec ecx
        jnz repeat
        ... (next line is execute when ecx reached 0)
```

Loop lines code executed 100 times

dec instruction change the flags
cmp instruction isn´t necessaire

12

Flow control instructions
if implementation

C code

```
if(x>0){
    x--;
}
y=x;
```

Assembly code

```
        mov eax,x
        cmp eax,0
        jle end_if   ;jump less or equal (condition for not if)
        dec eax      ;if code
end_if: mov y,eax    ;put the value of x (eax) in y
```

13

Flow control instructions
if then else implementation

C code

```
if(x>0){
    x--;
}
else{
    x++;
}
y=x;
```

Assembly code

```
        mov eax,x
        cmp eax,0
        jle else      ;jump less or equal (condition for else)
        dec eax       ;if code
        jmp end_if    ;this jump prevents the execution of the
                      else code when the if code is executed
else:   inc eax       ;else code
end_if: mov y,eax     ;put the value of x (eax) in y
```

If and else code are mutually exclusive

14

MUL/IMUL Instruction

There are two instructions for multiplying binary data.

1. The MUL (Multiply) instruction handles unsigned data
2. The the IMUL (Integer Multiply) handles signed data.

Both instructions affect the Carry and Overflow flag.

15

Arithmetic instructions

MUL *source*

MUL multiply **unsigned** integers

It was only on parameter *(source)* that can be memory or register

The others parameters are fixed

- 2 → Multiplicand
- × 8 → Multiplier
- 16 → Product

16

When two bytes are multiplied

- The multiplicand is in the AL register
- The multiplier is a byte in the memory or in another register.
- The product is in AX.
- High-order 8 bits of the product is stored in AH and the low-order 8 bits are stored in AL.

17

When two one-word values are multiplied

The multiplicand should be in the AX register.
The multiplier is a word in memory or another register.

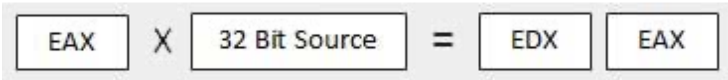
MUL DX, you must store the multiplier in DX and the multiplicand in AX.

The resultant product is a doubleword, which will need two registers.
The high-order (leftmost) portion gets stored in DX and the lower-order (rightmost) portion gets stored in AX.

18

When two doubleword values are multiplied

- The multiplicand should be in EAX
- The multiplier is a doubleword value stored in memory or in another register.
- The product generated is stored in the EDX:EAX registers
- The high order 32 bits gets stored in the EDX
- The low order 32-bits are stored in the EAX register.



19

Arithmetic instructions

Examples

MOV AL, 10

MOV BL,5

MUL BL; AX=AL*BL=10*5=50

MOV AX,100

MOV BX,50

MUL BX; DX:AX=100*50=5000

MOV EAX,1000

MOV EBX,100

MUL EBX; EDX:EAX=1000*100=100000

20

Arithmetic instructions

IMUL *source*

IMUL multiply **signed** integers

It was only on parameter *(source)* that can be memory or register

The other parameters are fixed

21

Arithmetic instructions
Examples

MOV AL, 10

MOV BL,5

IMUL BL; AX=AL*BL=10*5=50

MOV AX,100

MOV BX,-50

IMUL BX; DX:AX=100*(-50)=-5000

MOV EAX,-1000

MOV EBX,100

IMUL EBX; EDX:EAX=(-1000)*100=-100000

22

Arithmetic instructions

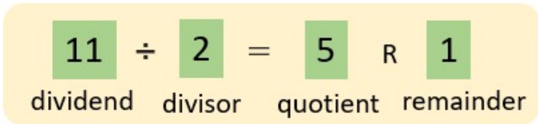
DIV *source*

DIV divide **<u>unsigned</u>** integers

It was only on parameter *(source)* that can be memory or register

The other parameters are fixed

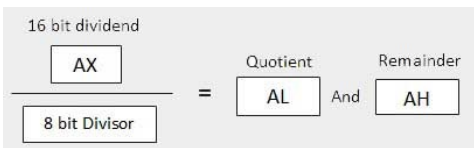
For each situation if the result doesn't fit in AL, AX or EAX, an exception is generated, and the program stops.



23

When the divisor is 1 byte

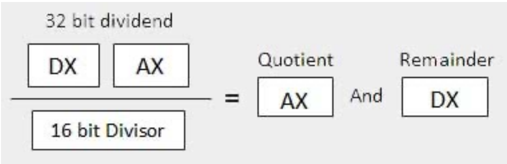
- The dividend is assumed to be in the AX register (16 bits).
- After division, the quotient goes to the AL register and the remainder goes to the AH register.



24

When the divisor is 1 word

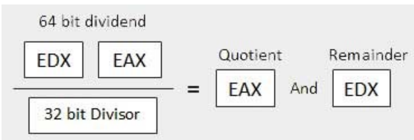
- The dividend is assumed to be 32 bits long and in the DX:AX registers.
- The high-order 16 bits are in DX
- The low-order 16 bits are in AX.
- After division, the 16-bit quotient goes to the AX register and the 16-bit remainder goes to the DX register.



25

When the divisor is doubleword

- The dividend is assumed to be 64 bits long and in the EDX:EAX registers.
- The high-order 32 bits are in EDX and the low-order 32 bits are in EAX.
- After division, the 32-bit quotient goes to the EAX register and the 32-bit remainder goes to the EDX register.



26

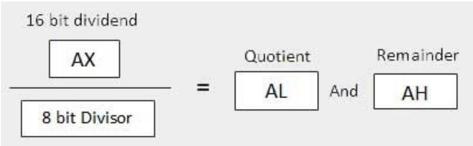
Arithmetic instructions
Examples

MOV AX, 102

MOV BL,5

DIV BL   AL=AX/BL=102/5=20

              AH=AX%BL=102%5=2



27

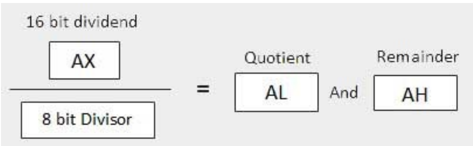
Arithmetic instructions
Examples

MOV AX,1001

MOV BL,2

DIV BL;

This instruction produce a run time error because the result is grater than 255



28

Arithmetic instructions
Solution to divide 1001 by 2

MOV AX,1001

MOV DX,0

MOV BX,2

DIV BX

In this case the source (BX) is 16 bits, and the result is

AX=DX:AX/BX=1001/2=500

DX=DX:AX%BX=1001%2=1

29

Arithmetic instructions

IDIV *source*

IDIV divide **signed** integers

It was only on parameter *(source)* that can be memory or register

The others parameters are fixe

For each situation if the result doesn't fit in AL, AX or EAX, an exception is generated and the program stops.

30

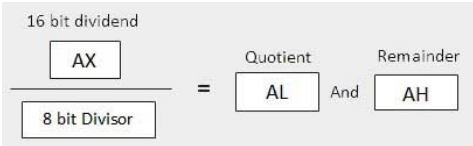
Arithmetic instructions
Examples

MOV AX,401

MOV BL,2

IDIV BL;

This instruction produce a run time error because the result is grater than 127



31