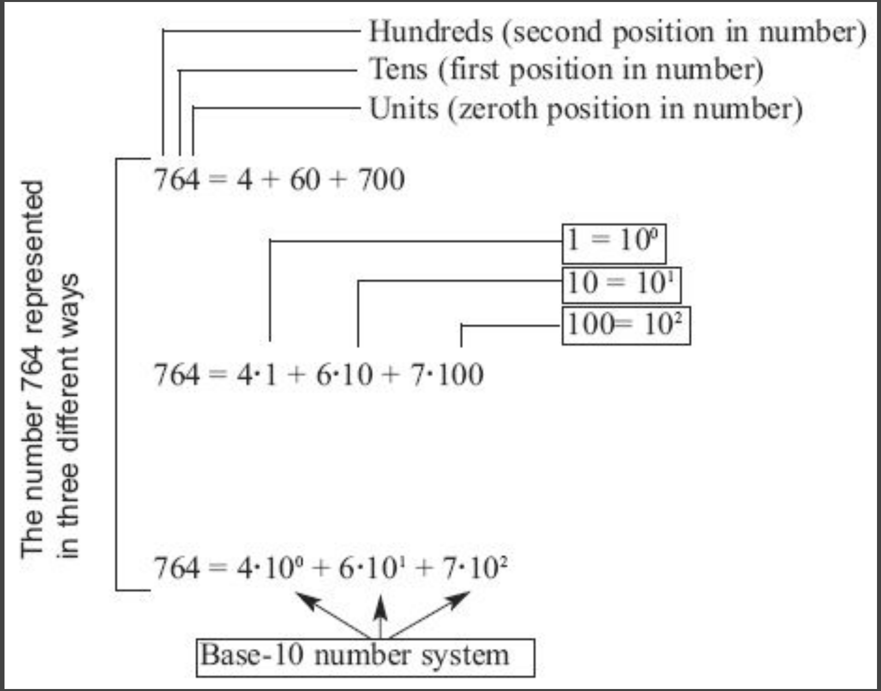
Introduction to 8086 Assembly

Lecture 4

Binary, Decimal and Hex Integer representation, signed integers, x86 flags, extending bit size

Decimal numbers



Hundreds (second position in number)

Tens (first position in number)

Units (zeroth position in number)

$764 = 4 + 60 + 700$

$1 = 10^0$

$10 = 10^1$

$100 = 10^2$

$764 = 4 \cdot 1 + 6 \cdot 10 + 7 \cdot 100$

$764 = 4 \cdot 10^0 + 6 \cdot 10^1 + 7 \cdot 10^2$

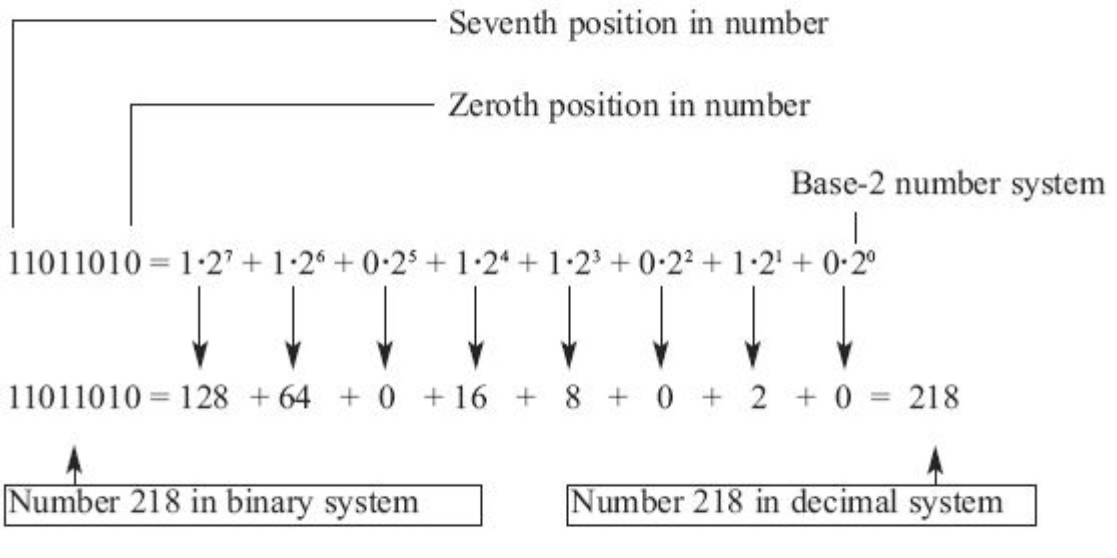
Base-10 number system

The number 764 represented in three different ways

https://learn.mikroe.com/ebooks/picmicrocontrollersprogramminginassembly/front-matter/introduction-to-the-world-of-microcontrollers/

Binary numbers

```
mov al, 218
mov al, 11011010b
```



Seventh position in number

Zeroth position in number

Base-2 number system

$$11011010 = 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$$

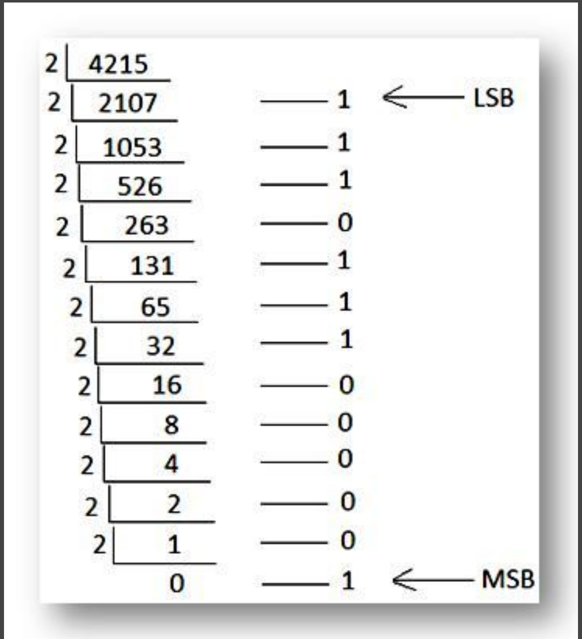
$$11011010 = 128 + 64 + 0 + 16 + 8 + 0 + 2 + 0 = 218$$

Number 218 in binary system

Number 218 in decimal system

https://learn.mikroe.com/ebooks/picmicrocontrollersprogramminginassembly/front-matter/introduction-to-the-world-of-microcontrollers/

Decimal to binary conversion



2	4215		
2	2107	1	← LSB
2	1053	1	
2	526	1	
2	263	0	
2	131	1	
2	65	1	
2	32	1	
2	16	0	
2	8	0	
2	4	0	
2	2	0	
2	1	0	
	0	1	← MSB

```
mov eax, 4215
mov eax, 1000001110111b
```

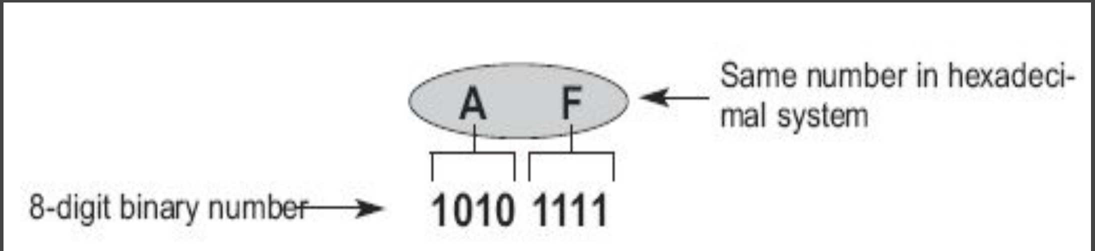
http://www.math-only-math.com/conversion-of-numbers.html

Hexadecimal numbers (hex)

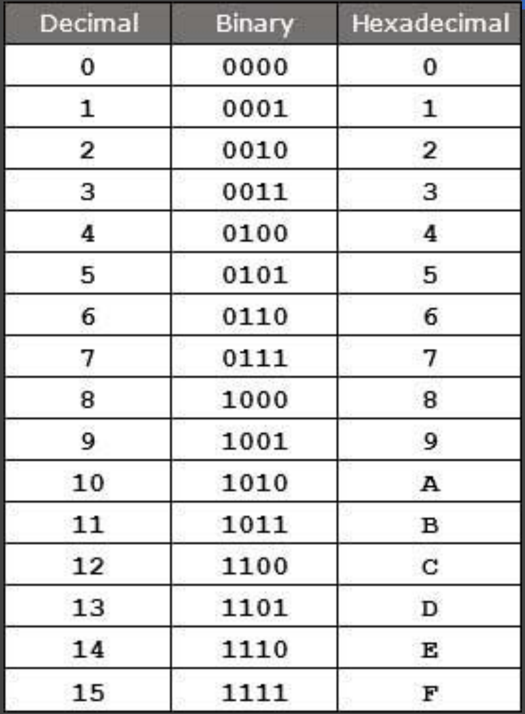
```
mov ax, 0A9E2h
mov ax, 0xA9E2
```

Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Convert hex to/from binary



https://learn.mikroe.com/ebooks/picmicrocontrollersprogramminginassembly/front-matter/introduction-to-the-world-of-microcontrollers/

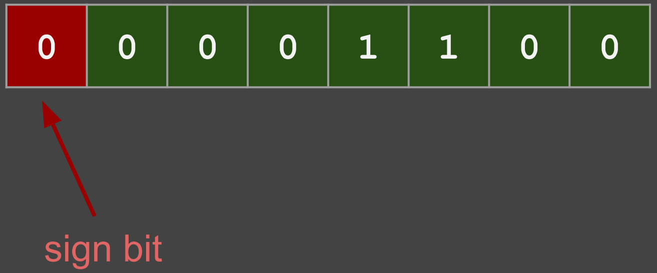


Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

https://codegolf.stackexchange.com/questions/53001/hexadecimal-counter

Signed integers

- Sign bit
  - $12 = 1100_2$

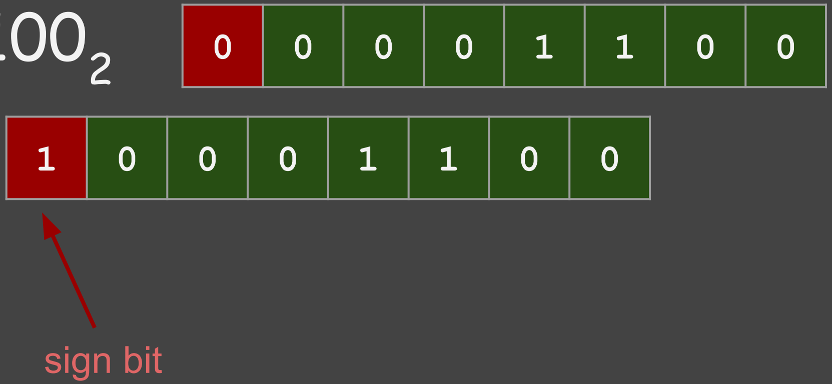


0	0	0	0	1	1	0	0

sign bit

Signed integers

- Sign bit
  - 12 = 1100_2
  - -12



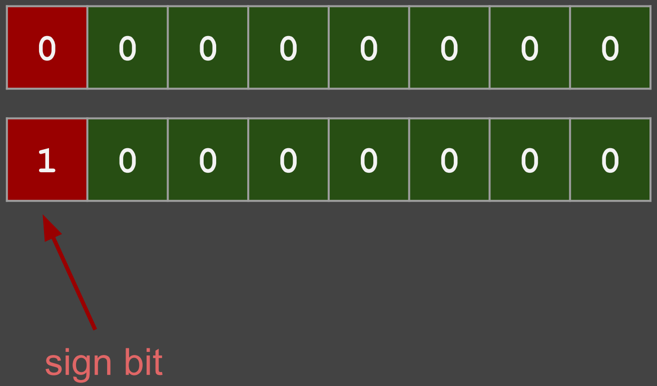
0	0	0	0	1	1	0	0

1	0	0	0	1	1	0	0

sign bit

Signed integers

- Sign bit
  - ○ +0
  - ○ -0



0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0

sign bit

One's complement

- Sign bit
  - 12 = 1100_2

0	0	0	0	1	1	0	0

  - -12

1	1	1	1	0	0	1	1

`mov al, 12`	0	0	0	0	1	1	0	0	AL
`not al`	1	1	1	1	0	0	1	1	AL

One's complement

- Sign bit
  - +0
  - -0

+0	0	0	0	0	0	0	0	0
-0	1	1	1	1	1	1	1	1

two's complement

8 bits

	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
128	10000000	80	128
129	10000001	81	129
130	10000010	82	130
:	:	:	:
253	11111101	FD	253
254	11111110	FE	254
255	11111111	FF	255

```
mov al, 255
add al, 1

al=?
```

```
  11111111
+        1
```

two's complement

8 bits

	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
128	10000000	80	128
129	10000001	81	129
130	10000010	82	130
:	:	:	:
253	11111101	FD	253
254	11111110	FE	254
255	11111111	FF	255

```
mov al, 255
add al, 1
```

```
al=0
```

```
  11111111
+        1
```

two's complement

K. N. Toosi
University of Technology

8 bits

	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
128	10000000	80	128
129	10000001	81	129
130	10000010	82	130
:	:	:	:
253	11111101	FD	253
254	11111110	FE	254
255	11111111	FF	255

```
mov al, 255
add al, 3

al=?
```

```
  11111111
+        1
```

two's complement

8 bits

	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
128	10000000	80	128
129	10000001	81	129
130	10000010	82	130
:	:	:	:
253	11111101	FD	253
254	11111110	FE	254
255	11111111	FF	255

```
mov al, 255
add al, 3
```

```
al=2
```

$$\begin{array}{r} 11111111 \\ +\qquad\quad 1 \end{array}$$

two's complement

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
128	10000000	80	128
129	10000001	81	129
130	10000010	82	130
:	:	:	:
253	11111101	FD	253
254	11111110	FE	254
-1	11111111	FF	255

```
mov al, 255
add al, 1
al=0

mov al, 255
add al, 3
al=2
```

two's complement

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
128	10000000	80	128
129	10000001	81	129
130	10000010	82	130
:	:	:	:
253	11111101	FD	253
254	11111110	FE	254
-1	11111111	FF	255

```
mov al, 254
add al, 2
al=0
```

two's complement

K. N. Toosi
University of Technology

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
128	10000000	80	128
129	10000001	81	129
130	10000010	82	130
:	:	:	:
253	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

```
mov al, 254
add al, 2
al=0
```

two's complement

K. N. Toosi
University of Technology

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
128	10000000	80	128
129	10000001	81	129
130	10000010	82	130
:	:	:	:
-3	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

two's complement

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
128	10000000	80	128
129	10000001	81	129
-126	10000010	82	130
:	:	:	:
-3	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

two's complement

K. N. Toosi
University of Technology

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
128	10000000	80	128
129	10000001	81	129
-126	10000010	82	130
:	:	:	:
-3	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

→ where to put the boundary?

two's complement

K. N. Toosi
University of Technology

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
128	10000000	80	128
-127	10000001	81	129
-126	10000010	82	130
:	:	:	:
-3	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

→ where to put the boundary?

two's complement

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
-128	10000000	80	128
-127	10000001	81	129
-126	10000010	82	130
:	:	:	:
-3	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

→ where to put the boundary?

two's complement

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
-129	01111111	7F	127
-128	10000000	80	128
-127	10000001	81	129
-126	10000010	82	130
:	:	:	:
-3	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

→ where to put the boundary?

K. N. Toosi
University of Technology

two's complement

K. N. Toosi
University of Technology

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
-130	01111110	7E	126
-129	01111111	7F	127
-128	10000000	80	128
-127	10000001	81	129
-126	10000010	82	130
:	:	:	:
-3	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

→ where to put the boundary?

two's complement

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
-128	10000000	80	128
-127	10000001	81	129
-126	10000010	82	130
:	:	:	:
-3	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

→ What is special about -128 ≡ 1000000?

two's complement

- `mov AL, 0xFF`
    - How do we know if AL stores a signed integer or an unsigned integer?
    - How do we know if AL=-1 or AL=255?

two's complement

- `mov AL, 0xFF`
  - How do we know if AL stores a signed integer or an unsigned integer?
  - How do we know if AL=-1 or AL=255?
  - How do we know if AL stores
    - A signed integer with signed bit?
    - A 1's complement signed integer?
    - A 2's complement signed integer?
    - An unsigned integer?
    - ASCII code of a character?

two's complement

- `add AL, BL`
  - signed or unsigned addition?
- `sub EDI, ESI`
  - signed or unsigned subtraction?

two's complement

- `add AL, BL`
  - signed or unsigned addition?
- `sub EDI, ESI`
  - signed or unsigned subtraction?
- Does not matter when 2's complement signed integers are used

two's complement

- `add AL, BL`
  - signed or unsigned addition?
- `sub EDI, ESI`
  - signed or unsigned subtraction?
- Does not matter when 2's complement signed integers are used
- Not the case for multiplication and division

two's complement

- 8 bits: -128 to 127 $(-2^7$ to $2^7-1)$
- 16 bits: -2^{15} to $2^{15}-1$
- 32 bits: -2^{31} to $2^{31}-1$
- n bits: -2^{n-1} to $2^{n-1}-1$

two's complement

- 8 bits: -128 to 127 (-2^7 to 2^7-1)
- 16 bits: -2^{15} to 2^{15}-1
- 32 bits: -2^{31} to 2^{31}-1
- n bits: -2^{n-1} to 2^{n-1}-1

- `mov eax, 0xFFFFFFFF`
- eax=?
  - signed 2's complement
  - unsigned

two's complement

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
-128	10000000	80	128
-127	10000001	81	129
-126	10000010	82	130
:	:	:	:
-3	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

two's complement

K. N. Toosi
University of Technology

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
-128	10000000	80	128
-127	10000001	81	129
-126	10000010	82	130
:	:	:	:
-3	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

two's complement

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
-128	10000000	80	128
-127	10000001	81	129
-126	10000010	82	130
:	:	:	:
-3	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

two's complement

K. N. Toosi
University of Technology

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
-128	10000000	80	128
-127	10000001	81	129
-126	10000010	82	130
:	:	:	:
-3	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

two's complement

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
-128	10000000	80	128
-127	10000001	81	129
-126	10000010	82	130
:	:	:	:
-3	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

```
not al
inc al
```

two's complement

8 bits

signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
-128	10000000	80	128
-127	10000001	81	129
-126	10000010	82	130
:	:	:	:
-3	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

```
not al
         ≡    neg al
inc al
```

two's complement

- `neg eax`
- `neg bx`
- `neg cl`
- `neg dh`

8086 FLAGS register

K. N. Toosi
University of Technology

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟩	🟥	🟩	🟥
				Overflow Flag (OF)	Direction Flag (DF)	Interrupt Flag (IF)	Trap Flag (TF)	Sign Flag (SF)	Zero Flag (ZF)		Auxiliary Carry Flag (AF)		Parity Flag (PF)		Carry Flag (CF)

CF: carry flag
OF: overflow flag
SF: sign flag
ZF: zero flag

PF: parity flag
DF: direction flag
IF: interrupt flag

8086 FLAGS register

K. N. Toosi
University of Technology

CLC (clear carry, set CF=0)
STC (set carry, set CF=1)
CMC (complement carry, set CF= ~CF)
CLD, STD, CLI, STI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟩	🟥	🟩	🟥
				Overflow Flag (OF)	Direction Flag (DF)	Interrupt Flag (IF)	Trap Flag (TF)	Sign Flag (SF)	Zero Flag (ZF)		Auxiliary Carry Flag (AF)		Parity Flag (PF)		Carry Flag (CF)

8086 FLAGS register

```
 8086 : FLAGS  (16 bits)
 80386: EFLAGS (32 bits)
x86-64: RFLAGS (64 bits)
```

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟩	🟥	🟩	🟥
				Overflow Flag (OF)	Direction Flag (DF)	Interrupt Flag (IF)	Trap Flag (TF)	Sign Flag (SF)	Zero Flag (ZF)		Auxiliary Carry Flag (AF)		Parity Flag (PF)		Carry Flag (CF)

Overflow - unsigned integers

- add eax, ebx
  - when there is carray
  - carry flag (CF) is set
- sub eax, ebx
  - when there is borrow
  - carry flag (CF) is set

Overflow - signed integers

- add eax, ebx
  - when POSITIVE+POSITIVE=NEGATIVE
  - when NEGATIVE+NEGATIVE=POSITIVE
  - overflow flag (OF) is set

Overflow

- carry flag (CF): unsigned
- overflow flag (OF): signed

Decreasing bit size - unsigned

0	0	A	2		A	2
4	E	A	2		A	2
F	F	A	2		A	2
F	F	F	3		F	3

Decreasing bit size - signed

0	0	7	2		7	2
0	0	A	2		A	2
4	E	A	2		A	2
F	F	7	2		7	2
F	F	F	F		F	F

Extending bit size - unsigned

7	2

A	2

7	2

F	F

Extending bit size - unsigned

7	2	0	0	7	2
A	2	0	0	A	2
7	2	0	0	7	2
F	F	0	0	F	F

Extending bit size - unsigned

- `AX  <- AL`   `mov ah, 0`
- `EAX <- AX`   `movzx eax, ax`
- `EAX <- AL`   `movzx eax, al`
- `AX  <- AL`   `movzx ax, al`
- `EAX <- BX`   `movzx eax, bx`
- `RAX <- EAX`   `movzx rax, eax`   (64 bit)

Extending bit size - signed

0	2

8	2

7	2

F	F

Extending bit size - signed

0 2	0 0 0 2
8 2	F F 8 2
7 2	0 0 7 2
F F	F F F F

Extending bit size - signed

K. N. Toosi
University of Technology

- AX <- AL CBW (convert Byte to Word)
- EAX <- AX CWDE (convert Word to double word extended)
- RAX <- EAX CDQE (convert Double to Quad extended,64 bit)

- DX:AX <- AX CWD (convert Word to Double word)
- EDX:EAX <- EAX CDQ (convert Double word to Quad word)
- RDX:RAX <- RAX CQO (convert Quad word to Oct Word, 64 bit)

Data size names

- Byte (8 bit)
- Word (2 bytes)
- Double word (4 bytes)
- Quad word   (8 bytes)
- Oct word   (16 bytes)

Extending bit size - signed

- `EAX <- AX` `movsx eax, ax`
- `EAX <- AL` `movsx eax, al`
- `AX  <- AL` `movsx ax, al`
- `EAX <- BX` `movsx eax, bx`
- `RAX <- EAX` `movsx rax, eax` (64 bit)

ADC and SBB

- ADC: add with carry
    - ADC dest, src   dest = dest + src + CF
- SBB: subtract with borrow
    - SBB dest, src   dest = dest - src - CF

- Example:
    - edx:eax = edx:eax + ecx:ebx

ADC and SBB

- ADC: add with carry
  - ADC dest, src   dest = dest + src + CF
- SBB: subtract with borrow
  - SBB dest, src   dest = dest - src - CF

- Example:
  - edx:eax = edx:eax + ecx:ebx
    - add eax, ebx
    - adc edx, ecx