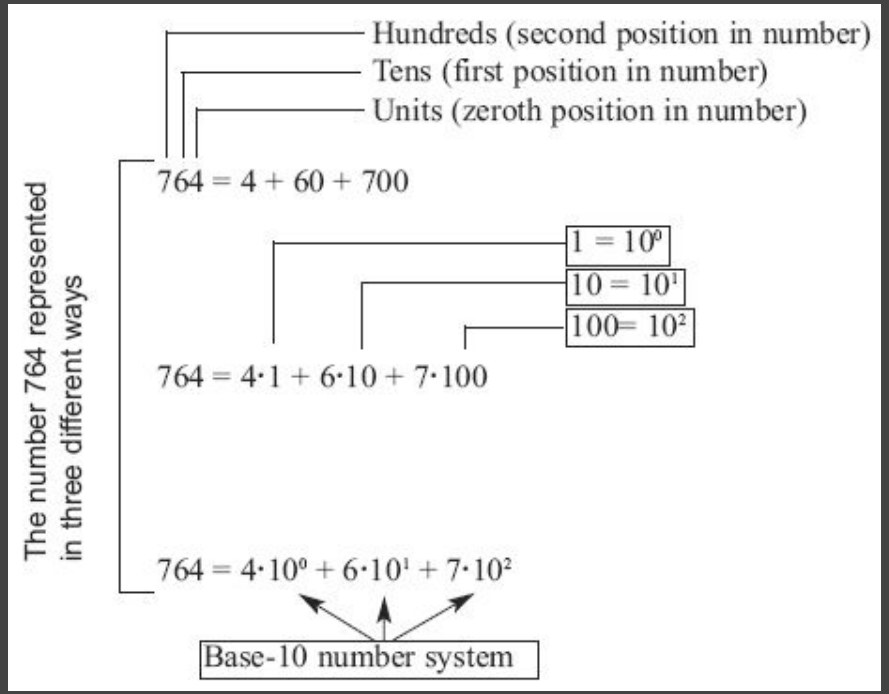
Introduction to 8086 Assembly

Lecture 4

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

Decimal numbers

K. N. Toosi
University of Technology

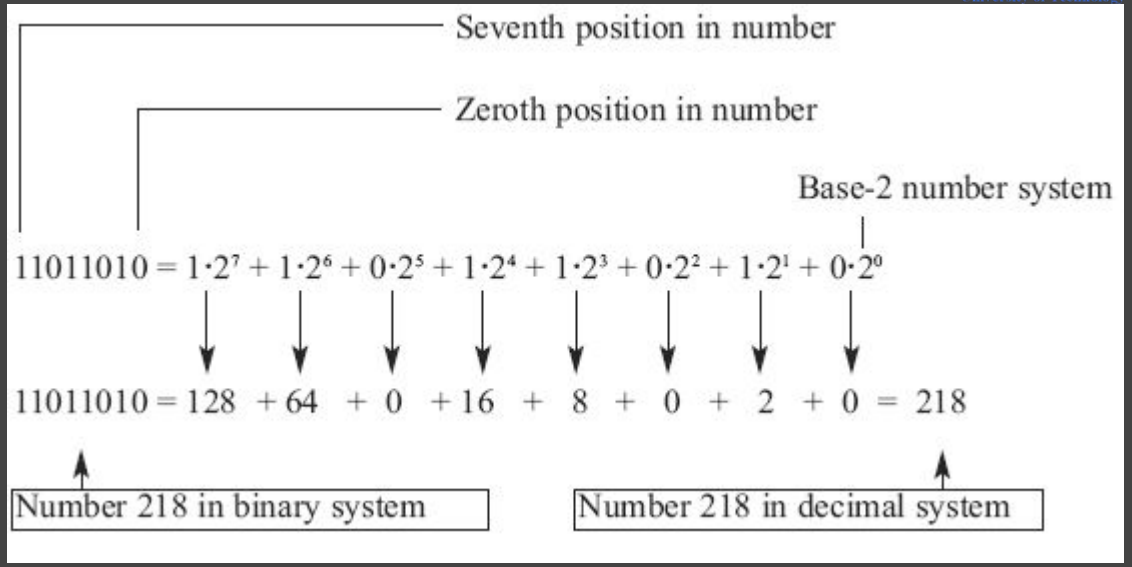


https://learn.mikroe.com/ebooks/picmicrocontrollersprogramminginassembly/front-matter/introduction-to-the-world-of-microcontrollers/

Binary numbers

K. N. Toosi
University of Technology

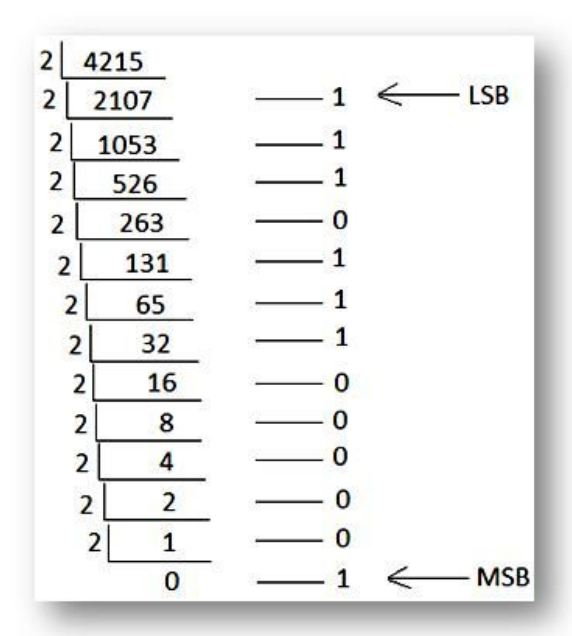
```
mov al, 218
mov al, 11011010b
```



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

Decimal to binary conversion

K. N. Toosi
University of Technology




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

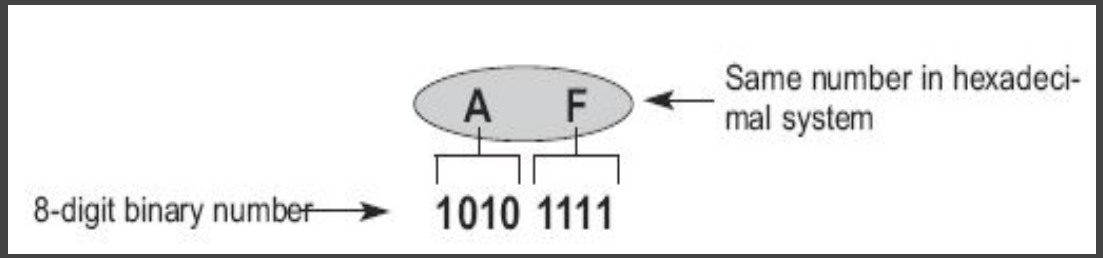
http://www.math-only-math.com/conversion-of-numbers.html

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

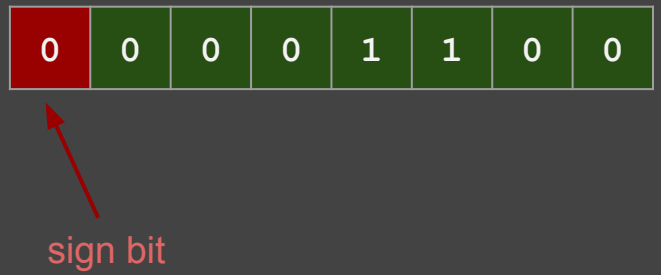


https://learn.mikroe.com/ebooks/picmicrocontrollersprogramminginassembly/front-matter/introduction-to-the-world-of-microcontrollers/

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

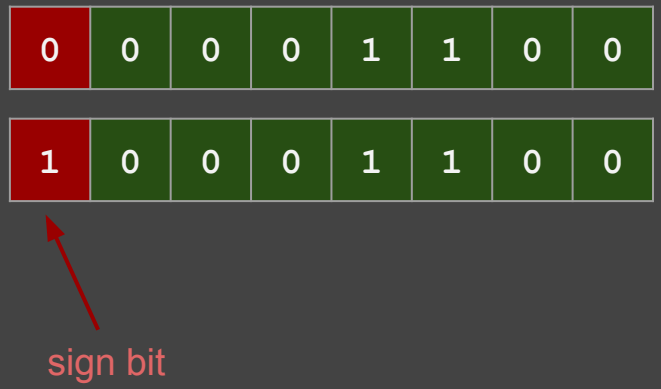
Signed integers



- Sign bit
  - $12 = 1100_2$

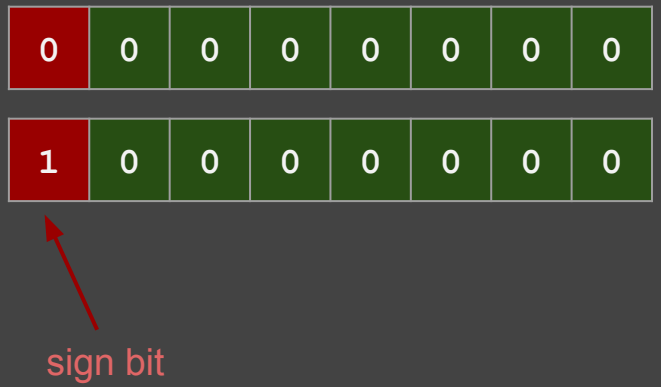
Signed integers

- Sign bit
  - $12 = 1100_2$
  - -12

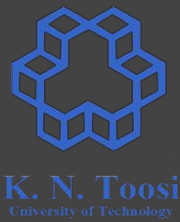


Signed integers

- Sign bit
  - +0
  - -0



One's complement



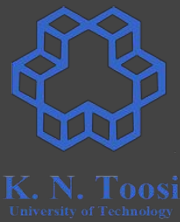
- Sign bit
  - $12 = 1100_2$
  - -12

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

```
mov al, 12
not al
```

0	0	0	0	1	1	0	0	AL
1	1	1	1	0	0	1	1	AL

One's complement



- Representing 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

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
+       11
```

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
```

```
  11111111
+       11
```

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

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

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
```

$x + 1 = 0$

$x + 3 = 2$

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

$x = ?$

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
```

$x + 1 = 0$

$x + 3 = 2$

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

$x = ?$

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

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

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

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

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?

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
-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?

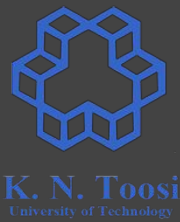
two's complement

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

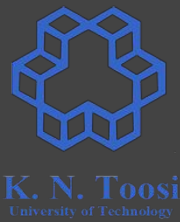
→ What is special about -128 ≡ 1000000?

two's complement



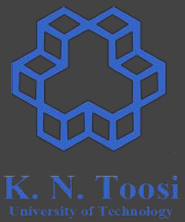
- `mov AL, 0xFF`
  - How do we know if AL stores a signed integer or an unsigned integer?
  - How do we know if AL=-1 or AL=255?

two's complement



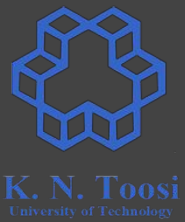
- `mov AL, 0xFF`
  - How do we know if AL stores a signed integer or an unsigned integer?
  - How do we know if AL=-1 or AL=255?
  - How do we know if AL stores
    - A signed integer with signed bit?
    - A 1's complement signed integer?
    - A 2's complement signed integer?
    - An unsigned integer?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

two's complement



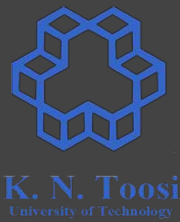
- `add AL, BL`
  - signed or unsigned addition?
- `sub EDI, ESI`
  - signed or unsigned subtraction?

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

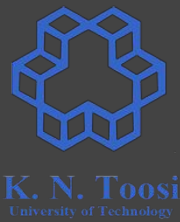


- `add AL, BL`
  - signed or unsigned addition?
- `sub EDI, ESI`
  - signed or unsigned subtraction?
- Does not matter when 2's complement signed integers are used
- Not the case for multiplication and division

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

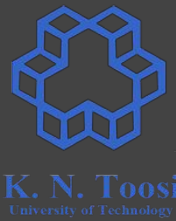
signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
-128	10000000	80	128
-127	10000001	81	129
-126	10000010	82	130
:	:	:	:
-3	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

two's complement

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

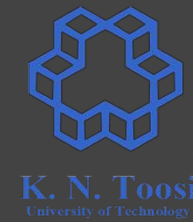
signed	binary	hex	unsigned
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
:	:	:	:
125	01111101	7D	125
126	01111110	7E	126
127	01111111	7F	127
-128	10000000	80	128
-127	10000001	81	129
-126	10000010	82	130
:	:	:	:
-3	11111101	FD	253
-2	11111110	FE	254
-1	11111111	FF	255

two's complement

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
```

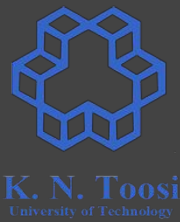
two's complement

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

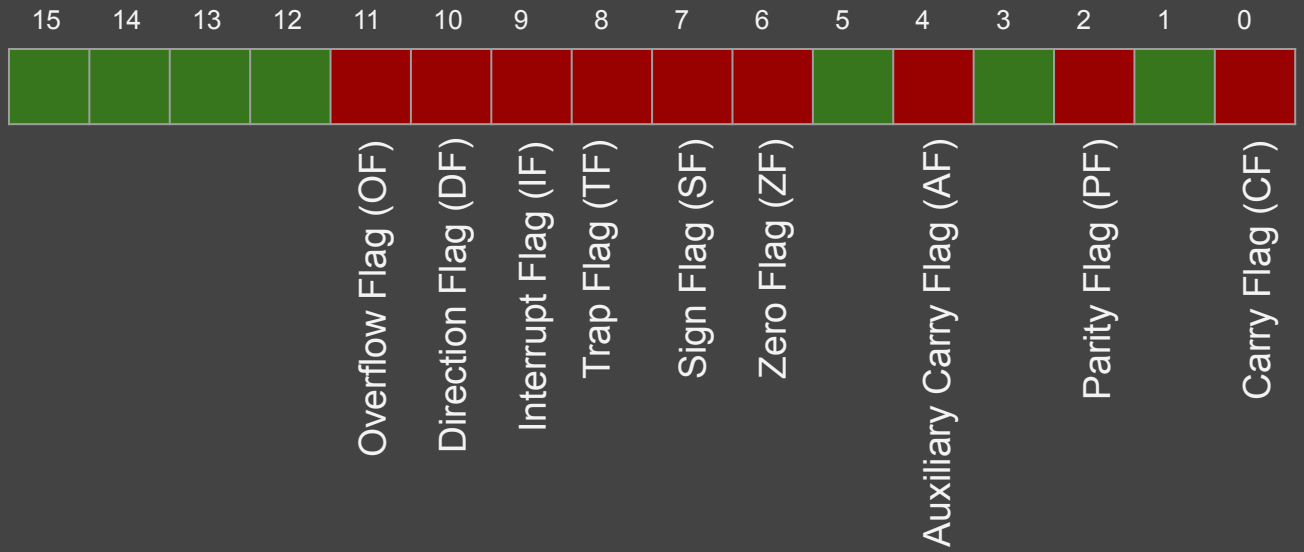
```
not al
inc al   ≡   neg al
```

two's complement



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

8086 FLAGS register



K. N. Toosi
University of Technology

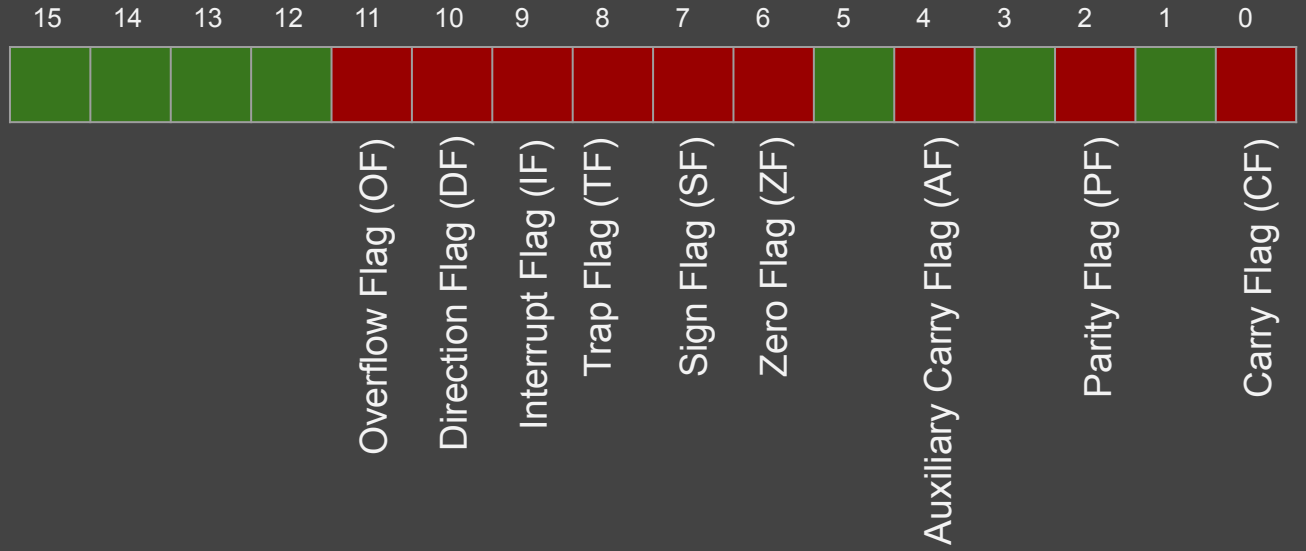
CF: carry flag
OF: overflow flag
SF: sign flag
ZF: zero flag

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

8086 FLAGS register

K. N. Toosi
University of Technology

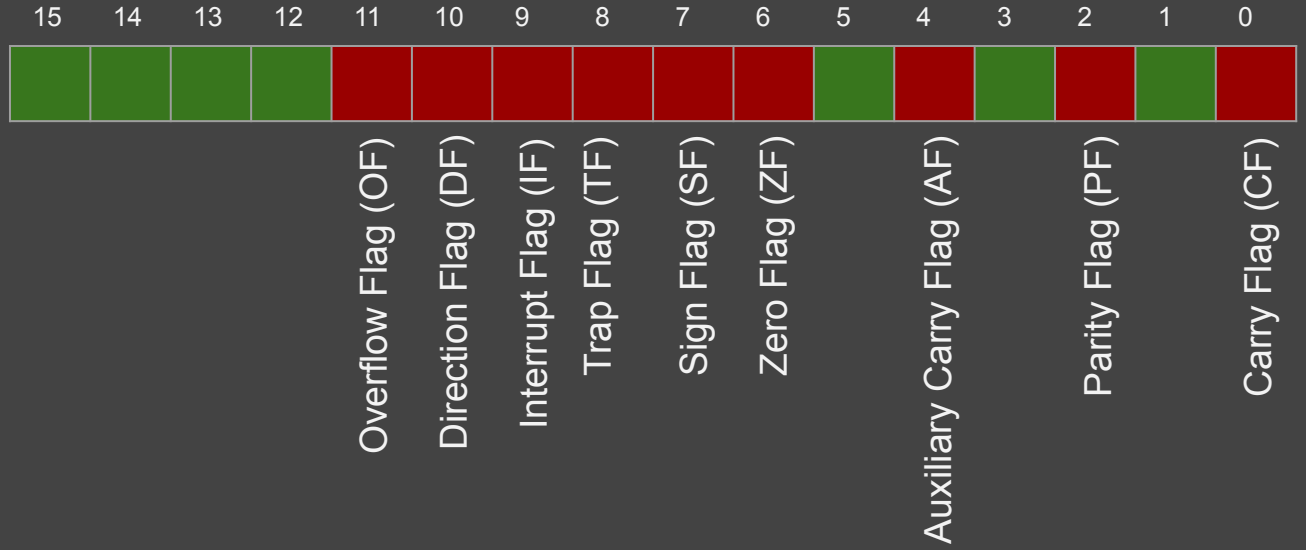
CLC (clear carry, set CF=0)
STC (set carry, set CF=1)
CMC (complement carry, set CF= ~CF)
CLD, STD, CLI, STI



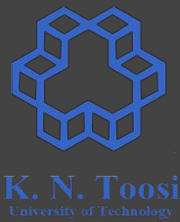
8086 FLAGS register

K. N. Toosi
University of Technology

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



Overflow - unsigned integers

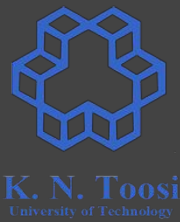


- `add eax, ebx`
  - when there is carray
  - carry flag (CF) is set
- `sub eax, ebx`
  - when there is borrow
  - carry flag (CF) is set

Overflow - signed integers

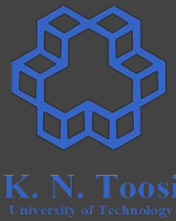
- `add eax, ebx`
  - when POSITIVE+POSITIVE=NEGATIVE
  - when NEGATIVE+NEGATIVE=POSITIVE
  - overflow flag (OF) is set

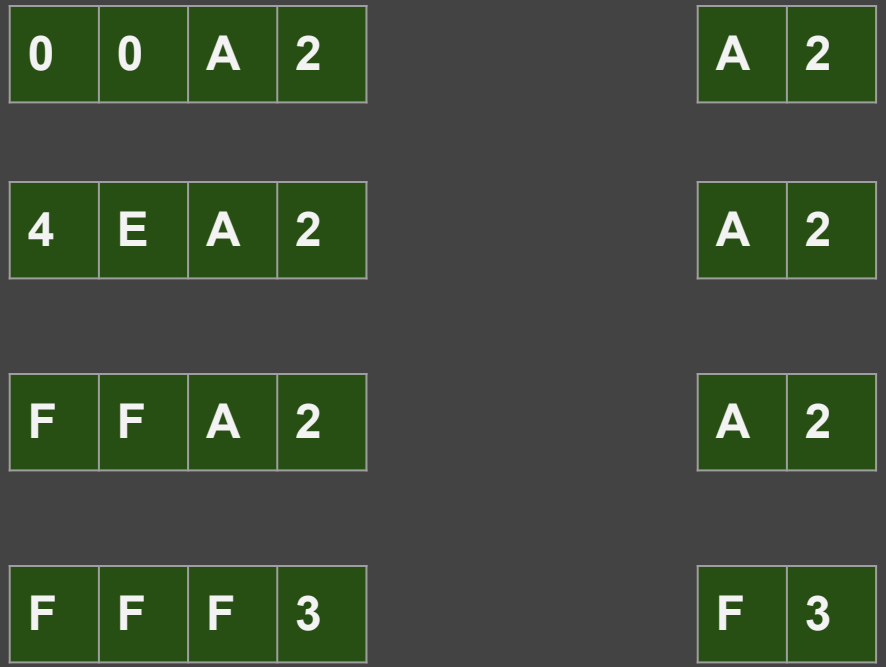
Overflow



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

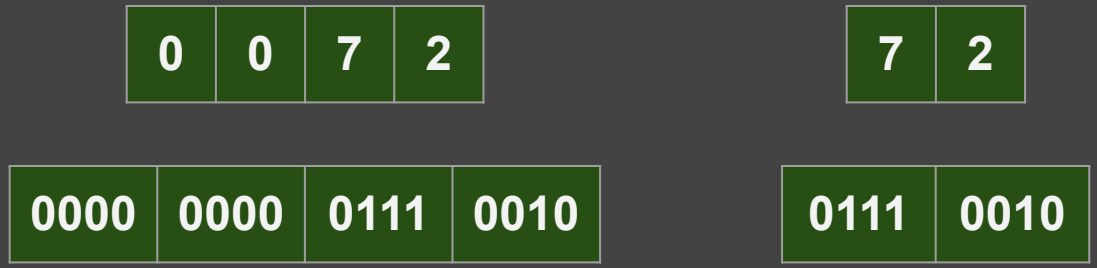
Decreasing bit size - unsigned



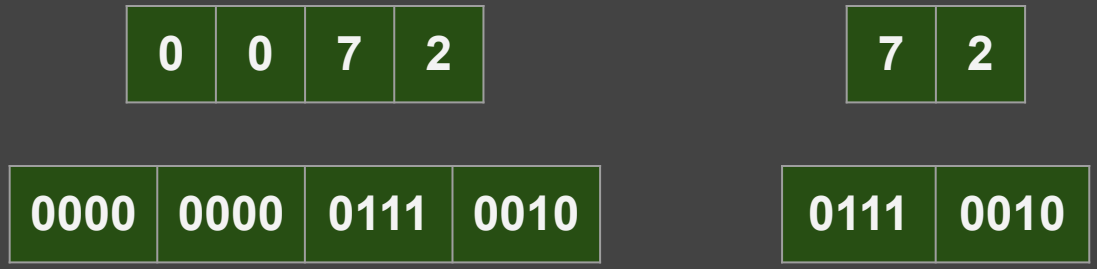


Decreasing bit size - signed

Decreasing bit size - signed



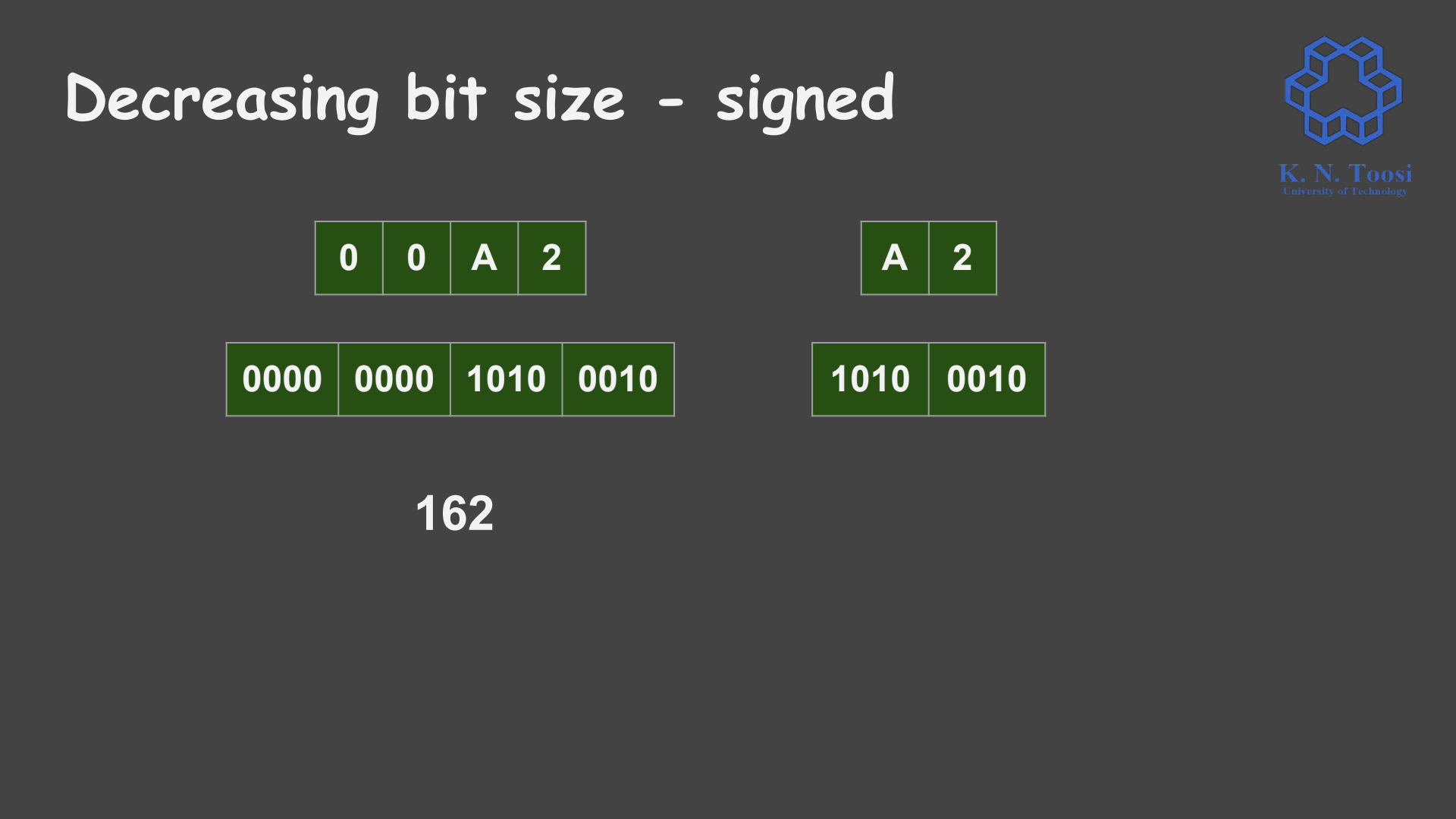
Decreasing bit size - signed



Decreasing bit size - signed

Decreasing bit size - signed

Decreasing bit size - signed



Decreasing bit size - signed

0	0	A	2

0000	0000	1010	0010

162

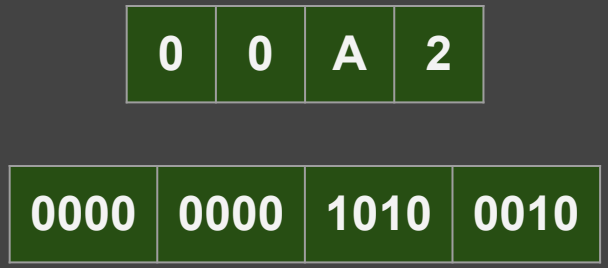
A	2

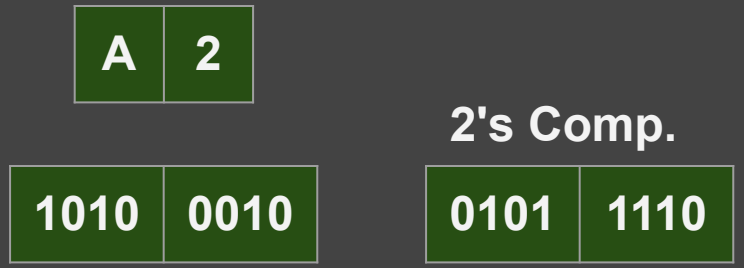
1010	0010

2's Comp.

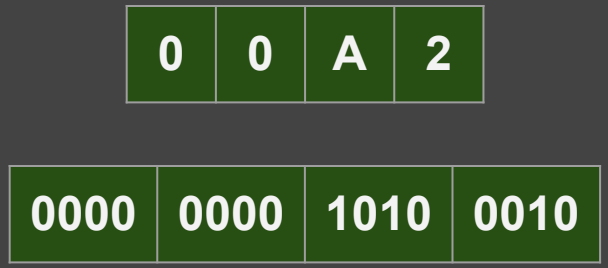
0101	1110

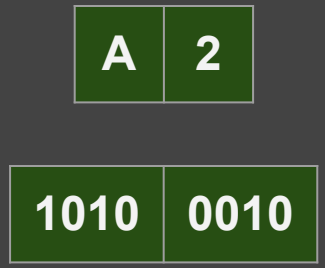
Decreasing bit size - signed





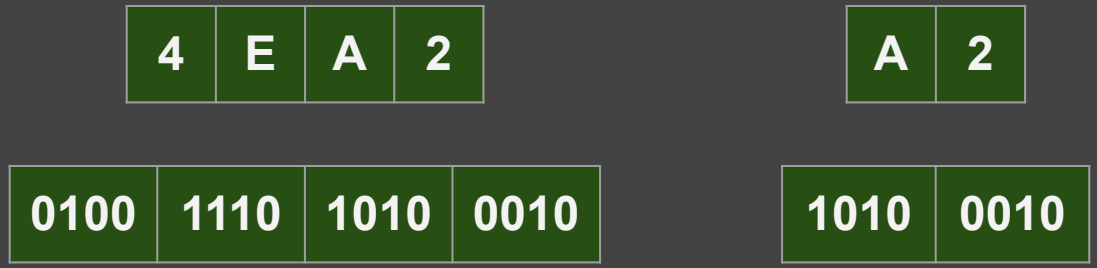
Decreasing bit size - signed





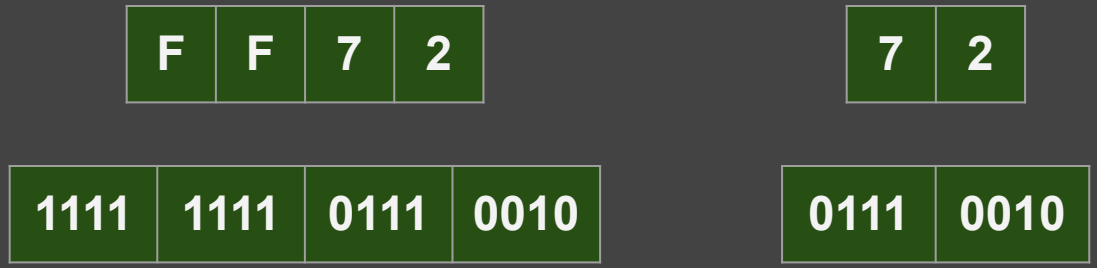
Decreasing bit size - signed

Decreasing bit size - signed

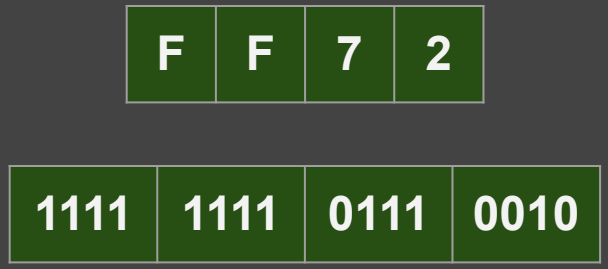


Decreasing bit size - signed

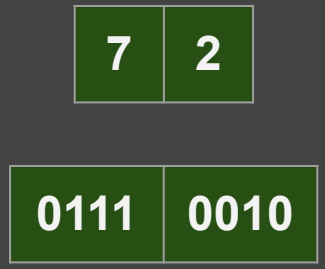
Decreasing bit size - signed



Decreasing bit size - signed



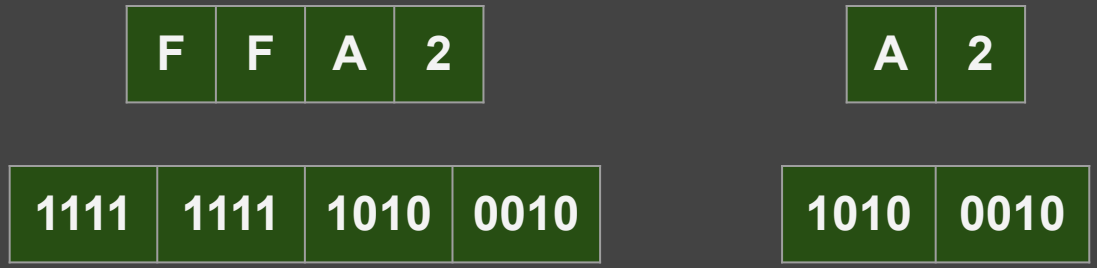
< 0



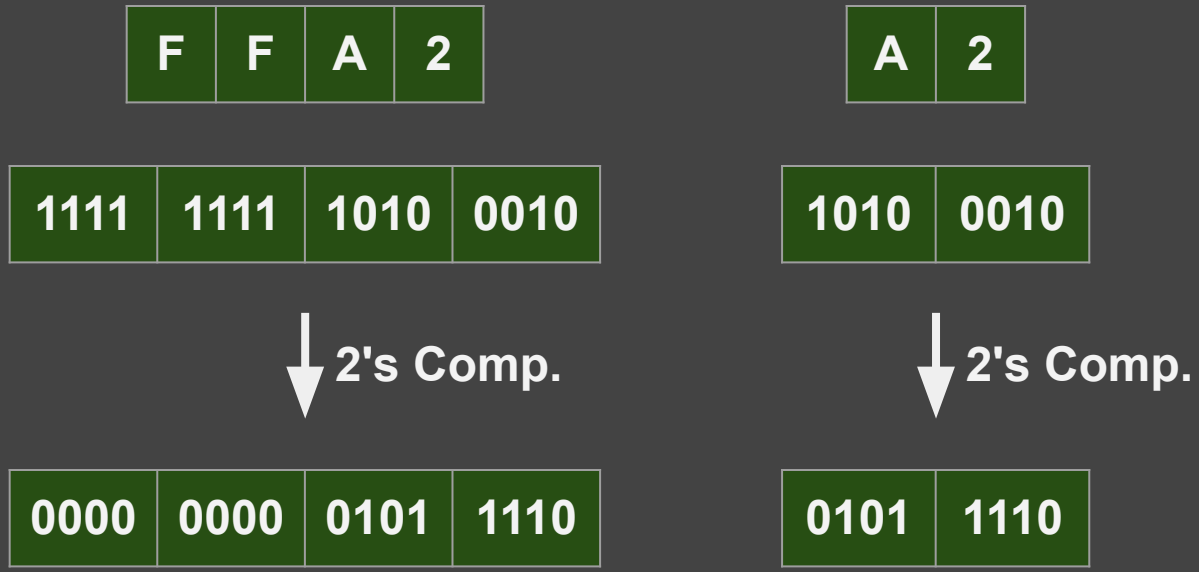
> 0

Decreasing bit size - signed

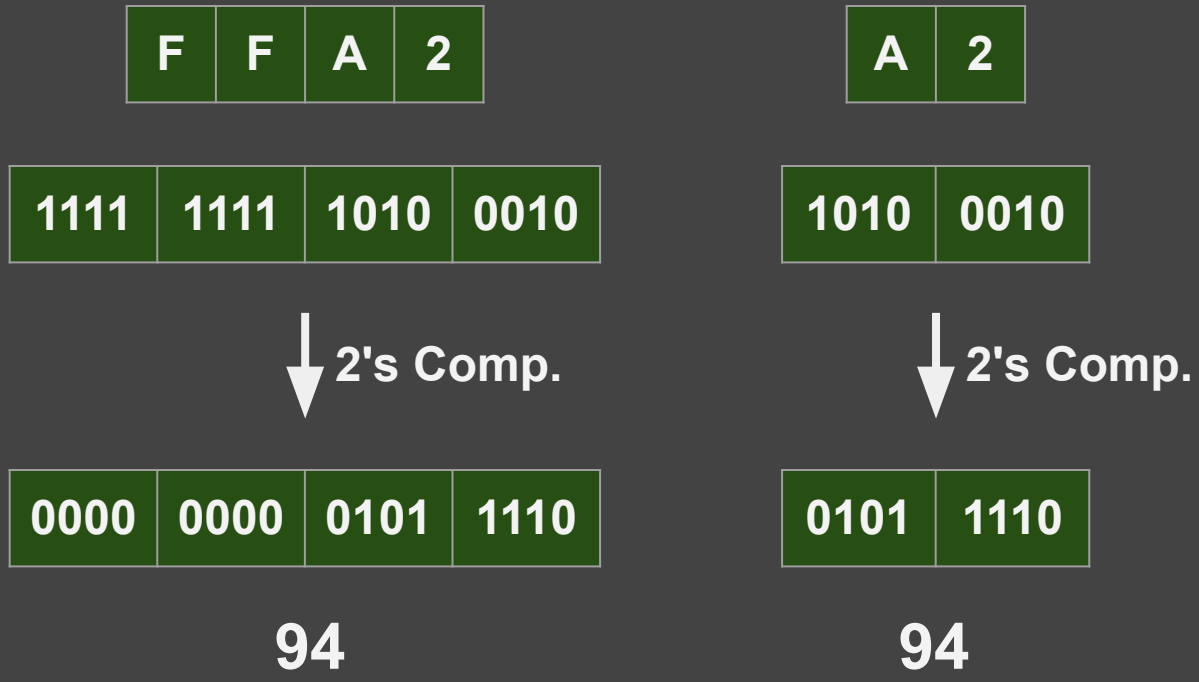
Decreasing bit size - signed



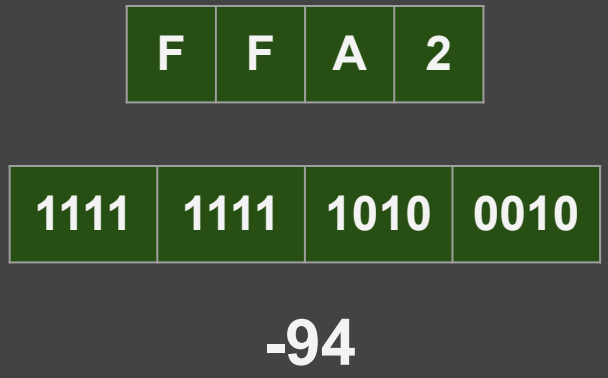
Decreasing bit size - signed

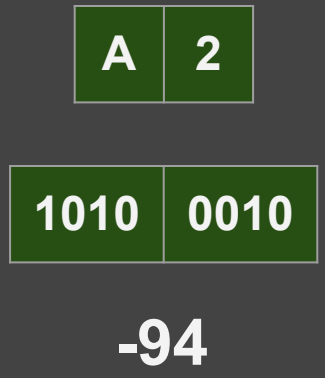


Decreasing bit size - signed



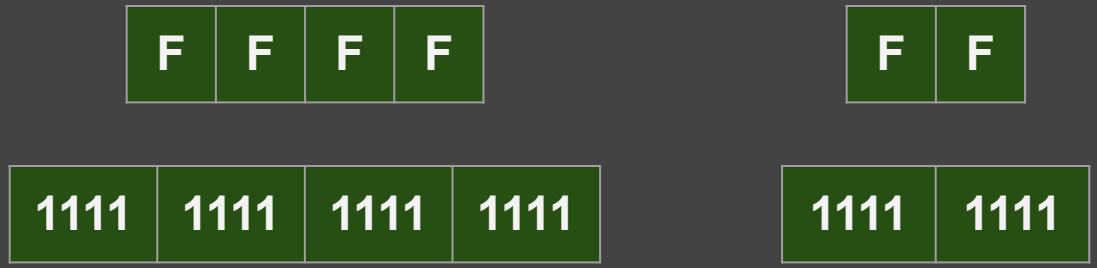
Decreasing bit size - signed



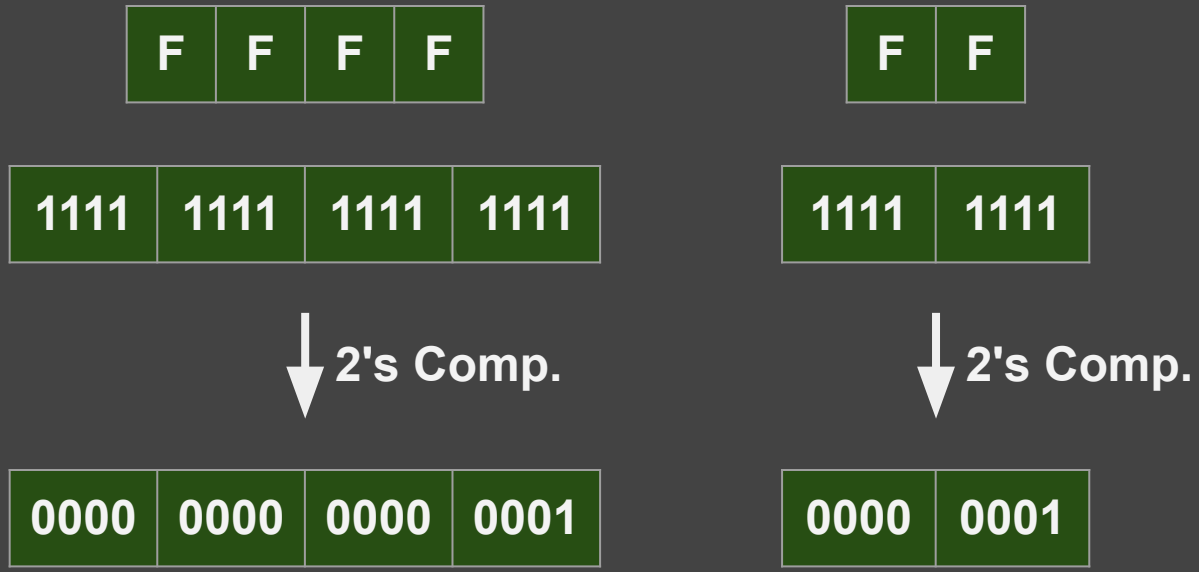


Decreasing bit size - signed

Decreasing bit size - signed

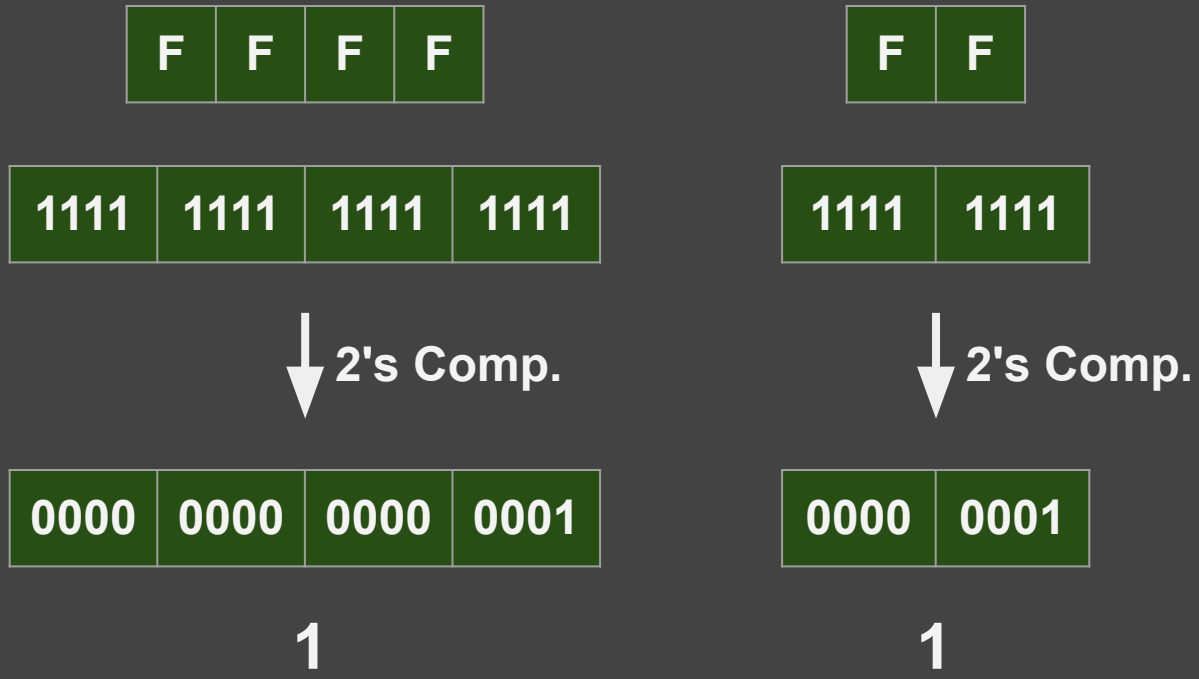


Decreasing bit size - signed

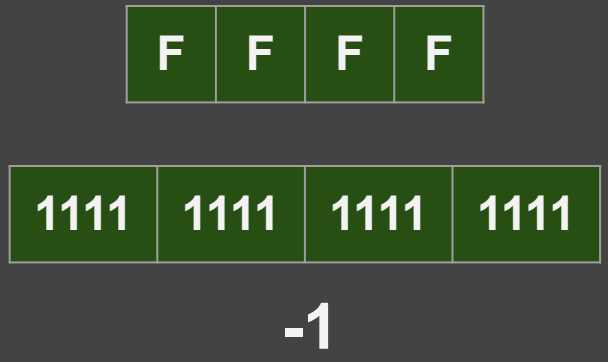


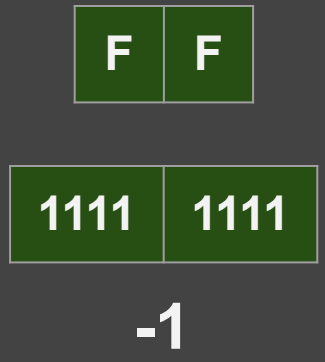
Decreasing bit size - signed

K. N. Toosi
University of Technology

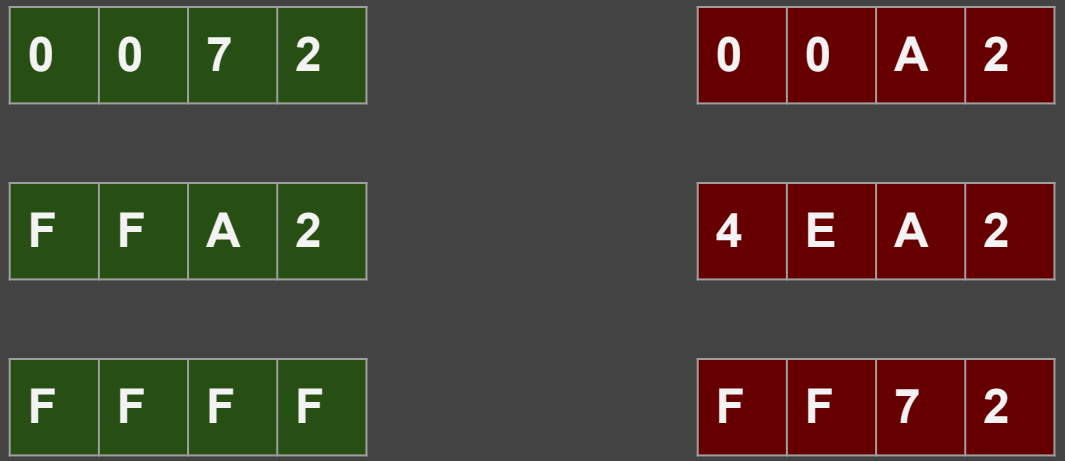


Decreasing bit size - signed

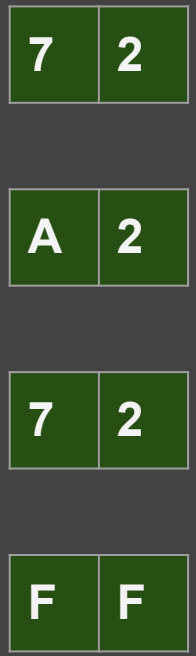




Decreasing bit size - signed



Extending bit size - unsigned

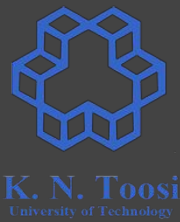


Extending bit size - unsigned

7	2
A	2
7	2
F	F

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

Extending bit size - unsigned



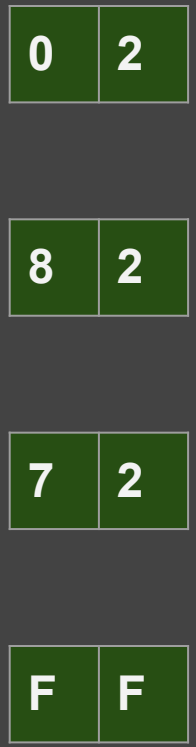
- AX <- AL `mov ah, 0`
- EAX <- AX `movzx eax, ax`
- EAX <- AL `movzx eax, al`
- AX <- AL `movzx ax, al`
- EAX <- BX `movzx eax, bx`
- RAX <- EAX `movzx rax, eax` (64 bit)

Extending bit size - signed

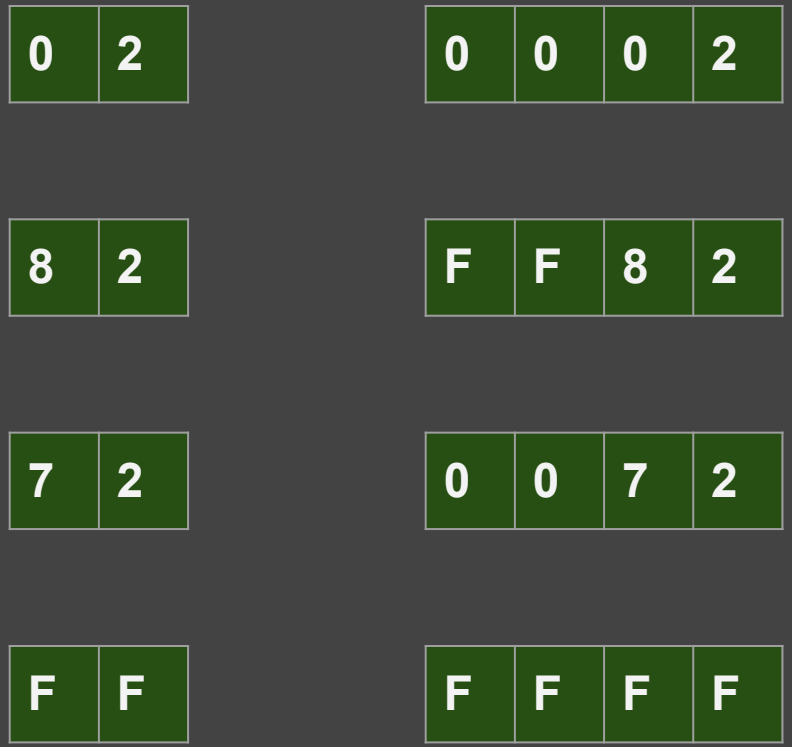
Extending bit size - signed

repeat the sign bit

Extending bit size - signed

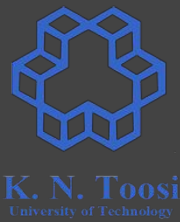


Extending bit size - signed



K. N. Toosi
University of Technology

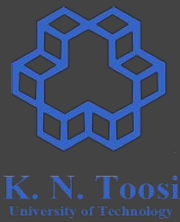
Extending bit size - signed



- 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)

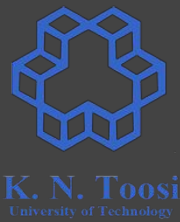
- DX:AX <- AX CWD (convert Word to Double word)
- EDX:EAX <- EAX CDQ (convert Double word to Quad word)
- RDX:RAX <- RAX CQO (convert Quad word to Oct Word, 64 bit)

Data size names



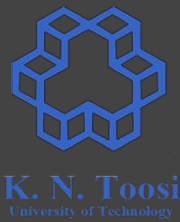
- Byte (8 bit)
- Word (2 bytes)
- Double word (4 bytes)
- Quad word (8 bytes)
- Oct word (16 bytes)

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)`

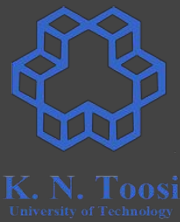
ADC and SBB



- 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