32 bit	SRC	Dest
IMul	mem, Reg32	Reg32 *= mem
	Reg16, Reg32	Reg32 *= Reg16
	Reg32$_S$, Reg32$_D$	Reg32$_D$ *= Reg32$_S$
	Imm8, Reg32	Reg32 *= Imm8
	Imm16, Reg32	Reg32 *= Imm16
	Imm32, Reg32	Reg32 *= Imm32

*size of the src, determine size of the dest.

*register is always a dest.

07/17 Intel Division Instructions:

Unsigned integers, one operand

*use cltd before idiv

DivB opnd %AL ∈ %AX / opnd Quotient

%AH ∈ %AX % opnd Remainder

DivW opnd %AX ∈ %DX : %AX / opnd

%DX ∈ %DX : %AX % opnd (32 bit register pair)

DivL opnd %EAX ∈ %EDX : %EAX / opnd

%EDX ∈ %EDX : %EAX % opnd (64 bit register pair)

DivQ opnd %RAX ∈ %RDX : %RAX / opnd

%RDX ∈ %RDX : %RAX % opnd (128 bit register pair)

~~For signed integers,~~
- IDivB
- IDivW
- IDivL
- IDivQ

Comments:

- opnd is register or memory (not Immediate)
- Must set high part of register pair even if numerator fits in one register
- Dividing by 1 still gets you results that fit in two registers.
- Implicit use of %AX, %EAX, %RAX registers
- Dividing by 0 stops execution.

C Library Functions

```
#include <STDIO.H>
int printf(const char * FMT, ...);          // format specifier
int fprintf(File *, const * FMT, ...);      // write on file
int Sprintf(char * string, const char * FMT, ...);   // creates string that can be manipulated
int Snprintf(char *String, size_T size, const char
        * FMT, ...);
```

size_T: typedef — Set max. size of String

Problem: output may be longer than String args.

Format Specifiers

characters %c %.[-][width]c

char

%.c	A	
%.3c	␣␣A	Right Justified
%.-3c	A␣␣	Left Justified

ex) `char c = 'A'; char Str[] = "HELLO"; int i = 3;`

`printf("c is %.c\n", c); printf("c is %c str is %s\n", c, Str);` (address)

Character Strings %s %%[-][width][.Precision]s

`char *s;`

`char Str[];`

%.s	ABCDEFGHIJKLM	
%.10s	ABCDEFGHIJKLM...	(Does not stop! weird) No padding
%.-10.5s	ABCDE␣␣␣␣␣	Left Justified
%.10.5s	␣␣␣␣␣ABCDE	Right Justified

Signed integers %.d %.[-][+␣][width][.][l]d

`int x`

(l: lowercase L)

%.d	43
%.+d	+43
%.␣d	␣43

char → int
short → int
*int no promotion
*long " "

Unsigned integers %[-][+␣][width][l][L]u o x X

unsigned int	%u	→ unsigned
unsigned long	%o	→ octal
void *p	%x	→ lowercase hex
	%X	→ Hex in uppercase

Floating point %[-][+␣][#][width][.precision]f e E g G

float	%f	3.14159
double	%e	3.14159e0
	%E	3.14159E0
	%g	%f if ABS exponent is small,
	%G	%e if ABS exponent is large.

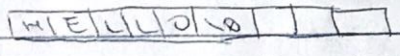
for trailing zeros and decimal point

* %% prints the '%' character

#include <String.H>
int strlen(const char *);
↳ returns number of chars before NUL byte
sizeof(int)
sizeof(x), int x;
sizeof(void *), sizeof(p) ← void *

ex
char yw[] = "Aoue Pilikia";
char *pyw = "Aoue Pilikia";

sizeof(yw) == 13 ; ← plus Null byte strlen(yw) == 12
sizeof(pyw) == 8 (byte) on 64 bit intel or 4 (byte) on 32 bit intel



char array[50] = "Hello"	char hello[] = "Hello"
sizeof(array) = 50	sizeof(hello) = 6
strlen(array) = 5	strlen(hello) = 5

C Library functions

```
#include <STDIO.H>
int scanf (const char * fmt, ...);
int fscanf (file *, const char * fmt, ...);
int sscanf (const char * string, const char * fmt, ...);
```

Format Specifiers:

char	%c	float	%f
string	%s		%e
Integers	%d		%g
	%u	double	%lf
	%o		%le
	%x		%lg

↑ Lowercase L

ex) `char input[30];`
`scanf("%s", input);`
Input: Hello how are you?
↳ Input only gets "Hello\0" ← null

ScanSet: %[ABCDEF]
use with char arrays or char *
✡ %[^\n] get everything other than '\n' newline
ex) `scanf("%[^\n]\n", input);`
Input: Hello how are you? \n

Scanf - Important!
1. Blank in scanf format means skip white space! (i.e. blank, tab, newline, form-feed)
2. Other chars not part of format specifiers are expected in input and skipped.
%[^\n]\n
↳ expect and skip

ex) int A,B,C;
scanf("%d", &A); printf("%d", A);
scanf(" %d", &A);

ex) int x;
while (printf("Enter an integer:"), scanf(" %d", &x) > 0)
{ ↖ skip white space
printf("x is %d\n", x);
}

Comma operator:
- Multiple expressions separated by commas.
- Value is value of rightmost expression
* Good for side effects.

Intel Comparison Instructions:
- cmpB src, DST Non-Destructive DST - SRC
- cmpW src, DST
- cmpL src, DST
- cmpQ src, DST

* Condition codes
Single bit flags in CPU

CF	carry flag	unsigned overflow
ZF	zero flag	Result is zero
SF	sign flag	Result is negative
OF	overflow flag	overflow

CF	1: unsigned < is True, 0: unsigned < is false
ZF	1: values are equal or zero, 0: not equal, not zero.
SF	1: Result is negative, 0: result is not negative
OF	1: 2's complement overflow has occurred, 0: no overflow

*some instructions take action based on condition flag.
*There are other instructions that ignore Cond. flag.

* Some instructions examine cond. flag

Intel Jump Instructions: Use new value at some diff. address
Unconditional

Direct Jump

*Determine at runtime

```
Jmp  LABEL
```

Indirect Jump

```
Jmp  *EAX
Jmp  *(%EAX)
```

(could be any indirect addressing format)

Conditional - based on cond. flag

JE = JZ ... jump if, equal or zero
JNE = JNZ ... jump if, not equal or not zero
JG = JNLE ... jump if, greater or not less than or equal
JA = JNBE ... jump if, above or not below or equal
JGE = JNL ... jump if, greater or equal = not less than
JAE = JNB ... " ", above or equal = not below
JL = JNGE ... " ", less than = not greater or equal
JB = JNAE ... " ", below = not above or equal
JLE = JNG ... " ", less than or equal = not greater than
JBE = JNA ... " ", below or equal = not above
JS ... " ", sign ①
JNS ... " ", not sign ⓪

Condition codes and C comparison Expressions:

Comparison	Cond. code combination	Cond. Jump Instruction
==	ZF	JE, JZ
!=	~ZF	JNE, JNZ
signed >	~(SF^OF) & ~ZF	JG, JNLE
unsigned >	~CF & ~ZF	JA, JNBE
signed >=	~(SF^OF)	JGE, JNL
unsigned >=	~CF	JAE, JNB

Comparison	Cond. code comb.	Cond. Jump Instr.
Signed <	SF ^ OF	JL, JNGE
unsigned <	CF	JB, JNAE
Signed <=	(SF^OF) | ZF	JLE, JNG
unsigned <=	CF | ZF	JBE, JNA
Negative?	SF	JS
Non-negative?	~SF	JNS

* Less/Greater for signed & above/below for unsigned

```
for (I=0, I<10, I++) {
    Body
}
```

Assembly language

```
    movL  $0, %EAX
Top [cmp  $10, %EAX
    [JG   Done
     ;] Body
    Inc   %EAX
    Jmp   Top
Done
```

* Don't have uncond. jump inside a loop (less efficient! If body has to be executed many times)

If-Else

```
    movL  -12(%RBP), %RDX
    cmpL  $3, %RDX
    JNE   Else
              ] True part
    [Jmp  Done
Else
              ] false part
Done
```