

32 bit	SRC	Dest
IMUL	mem, Reg32	Reg32 *= mem
	Reg16, Reg32	Reg32 *= Reg16
	Reg32s, Reg32d	Reg32d *= Reg32s
	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 ctd before Div

DivB opnd %AL ← %AX / opnd Quotient
 %AH ← %AX %opnd Remainder

DivW opnd %AX ← %DX : %AX / opnd
 %DX ← %DX : %AX %opnd

DivL opnd %EAX ← %EDX : %EAX / opnd
 %EDX ← %EDX : %EAX %opnd

For signed integers,
 IDIB
 IDIW
 IDIL
 IDIQ

DivQ opnd %RAX ← %RDX : %RAX / opnd
 %RDX ← %RDX : %RAX %opnd

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, ...);

→ Typed

Set max. size of string

Problem: output may be longer than string args

Format Specifiers

Characters %c %[-][width]c

char

%c

A

%3c

AAA

Right Justified

%-3c

AAA

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

ABCDEFGHJKLM

%10s

ABCDEFGHJKLM...

(Does not stop! weird! No padding)

%-10.5s

ABCDE.....

Left Justified

%10.5s

.....ABCDE

Right Justified

Signed integers

%d

%[-][+][width][.][L]d

int x

%d

43

↑ lowercase L

%td

+43

%ld

43

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

Unsigned integers

unsigned int

unsigned long

void *p

%u → unsigned

%o → octal

%x → lowercase hex

%X → hex in uppercase

%[-][+][width][L][L]^u_x

Floating point

float

double

%f 3.14159

%e 3.14159e0

%E 3.14159E0

$\left\{ \begin{array}{l} \%g \\ \%G \end{array} \right\}$ $\left\{ \begin{array}{l} \%f \text{ if exponent is small,} \\ \%e \text{ if exponent is large.} \end{array} \right.$ ^{ABS} _{ABS}

%[-][+][#][width][.precision] f

for trailing zeros
and decimal point

e
E
g
G

* %%. 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;

strlen (yw) == 12

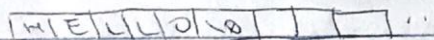
sizeof (pyw) == 8

or

4 (byte)

↳ (byte) on
64 bit intel

↳ on 32 bit
intel



char array[50] = "Hello"

sizeof (array) = 50

strlen (array) = 5

char hello[] = "Hello"

sizeof (hello) = 6

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)
{
    printf ("x is %d\n", X);
}
```

↑ skip white space

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

Jump LABEL

Indirect Jump

Jump *EAX

Jump *(%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 ②

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
```

```
}
```

```
    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
```

Assembly
language