General memory reference	
`disp(base, index, scale)` $disp + base + index \cdot scale$	
disp	Label or constant, relative.
base	Register.
index	Register.
scale	Constant.

Note: no dollar signs before constants.

Example memory references
`$M` Location of M in memory.
`(M)` Value at M in memory.
`(%eax)` Value at (value of eax) in memory.
`M(%eax)` Value at (location of M + eax).
`M(,%eax,4)` Value at (location of M + eax * 4).
`(%eax,%ebx)` Value at (value of eax + value of ebx).
`M(%eax,%ebx)` Value at (location of M + eax + ebx).
`M(%eax,%ebx,4)` Value at (location of M + eax + ebx * 4).

8-bit	16-bit	32-bit	Register
AL+AH	AX	EAX	**Accumulator**
CL+CH	CX	ECX	**Counter**
DL+DH	DX	EDX	**Data**
BL+BH	BX	EBX	**Base**
	SP	ESP	**Stack pointer**
	BP	EBP	**Base pointer**
	SI	ESI	**Source index**
	DI	EDI	**Dest. index**

Function call stack	X(%ebp)
(stack bottom)	***high mem.***
…	
argument 3	**16**
argument 2	**12**
argument 1	**8**
return address	**4**
previous EBP	**0**
local variable 1	**-4**
local variable 2	**-8**
local variable 3	**-12**
…	
(stack top)	***low mem.***

Function start

```
.global functionName
functionName:
pushl %ebp
movl %esp, %ebp
```

Returning a value

```
movl <value>,%eax
```

Function end

```
movl %ebp, %esp
popl %ebp
ret
```

Calling a function fn(arg1,arg2,arg3)

```
...
pushl arg3
pushl arg2
pushl arg1
call fn
add <numargs * 4>, %esp
```

Exiting the program

```
movl <return value>, (%esp)
call exit
```

Arithmetic instructions

`ADD* src, dest`	Add.
`DEC* value`	Decrease by one.
`DIV* value`	Unsigned division. $EAX \ / \ value \ \rightarrow \ EAX$
`IDIV* value`	Signed division. $EAX \ / \ value \ \rightarrow \ EAX$
`IMUL* value`	Signed multiplication. $EAX \cdot value \rightarrow EAX$
`INC* value`	Increase by one.
`MUL* value`	Unsigned multiplication. $EAX \cdot value \rightarrow EAX$
`NEG* value`	Two's complement negation.
`SUB* src, dest`	Subtract.

Bitwise instructions

`AND* src, dest`	Bitwise AND.
`NOT* value`	One's complement negation.
`OR* src, dest`	Bitwise OR.
`RCL* count, value`	Bitwise rotate through carry left.
`RCR* count, value`	Bitwise rotate through carry right.
`ROL* count, value`	Bitwise rotate left.
`ROR* count, value`	Bitwise rotate right.
`SHL* count, value`	Bitwise shift left.
`SHR* count, value`	Bitwise shift right.
`XOR* src, dest`	Bitwise exclusive OR.

Control flow instructions

`CALL target`	Near call.
`JE target`	Jump on condition.
`JG target`	Jump when greather than.
`JGE target`	Jump when greather than or equal.
`JL target`	Jump when less than.
`JLE target`	Jump when less than or equal.
`JMP target`	Near jump.
`JNE target`	Jump when not equal.
`LOOP target`	Jumps when --ECX is not 0.
`RET`	Near return.

Value instructions

`LEA* src, dest`	Load effective address.
`MOV* src, dest`	Moves a value.
`MOVS** src, dest`	Moves a value and sign extends it.
`MOVZ** src, dest`	Moves a value and zero extends it.

Compare instructions

`CMP* src, dest`	Compare (dest <=> src).

Stack frame instructions

`ENTER count, 0`	Enter stack frame.
`LEAVE`	Leaves the stack frame.

*) Operand size specifiers:

Symbol	Bytes	Bits
byte	1 byte	8 bits
word	2 bytes	16 bits
long	4 bytes	32 bits
quad word	8 bytes	64-bits

Stack instructions

`POP* dest`	Pop from stack.
`PUSH* src`	Push to stack.

Other instructions

`NOP`	No operation.
`PAUSE`	No operation in a loop.