

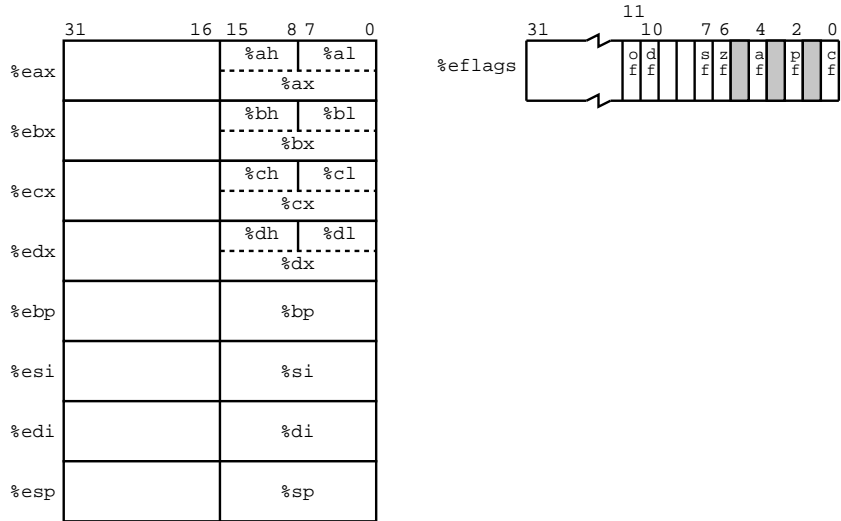
1 Intel x86 reference

1.1 Registers

The IA-32 architecture contains 8 user-accessible 32-bit general purpose registers. These are listed in the table below, complete with the special uses of them.

- `%eax` Accumulator register, destination of `mul` etc.
- `%ebx` (Sometimes pointer to data)
- `%ecx` Counter register
- `%edx` Extra accumulator, destination of `mul` etc.
- `%edi` Source for string ops
- `%esi` Destination for string ops
- `%ebp` Base/frame pointer. Points out the start of the stack frame.
- `%esp` Stack pointer. Points out the top of the stack.

- `%eip` Instruction pointer/program counter. Not accessible.
- `%eflags` Flag register, accessible through special instructions.



2 Instruction set reference

2.1 Memory references

A memory address is referenced through the following syntax:

```
offset(%base_reg, %idx_reg, scale)
```

Where the address is `%base_reg + %index_reg*scale + offset`. `offset` is a 32-bit constant and `scale` is one of 1,2,4,8. An example is shown below.

```
movl %eax, 8(%edi, %eax, 4)
  offset-^ | | ^-scale
  base_register  index_register
```

Where the value of `%eax` is written to `%edi + %eax*4 + 8`.

2.2 Common instructions

Name	Arguments	Operation	Description	Transfer	
mov	SRC, DST	DST = SRC	Copy	Transfer	
xchg	SRC, DST	DST = SRC, SRC = DST	Exchange		
push	SRC	(%esp) = SRC; %esp -= 4;	Push on stack		
pop	DST	DST = (%esp); %esp += 4;	Pop stack		
logical					
xor	SRC, DST	DST = DST ^ SRC	Bitwise xor	logical	
or	SRC, DST	DST = DST SRC	Bitwise or		
and	SRC, DST	DST = DST & SRC	Bitwise and		
arithm.					
cmp	A, B	EFLAGS = B - A	sub (only flags)	arithm.	
test	A, B	EFLAGS = B & A	and (only flags)		
inc	DST	DST++	Increment		
dec	DST	DST-	Decrement		
add	SRC, DST	DST = DST + SRC	Add		
sub	SRC, DST	DST = DST - SRC	Subtract		
mul	SRC	%edx:%eax = %eax * SRC	Multiply (unsigned)		
imul	SRC	%edx:%eax = %eax * SRC	Multiply (signed)		
div	SRC	%edx = %eax MOD SRC; %eax = %eax / SRC;	Divide (unsigned)		
idiv	SRC	%edx = %eax MOD SRC; %eax = %eax / SRC;	Divide (signed)		
jumps					
jmp	LABEL		Unconditional jump		jumps
je	LABEL		Jump if equal		
jne	LABEL		Jump if not equal		
jg	LABEL		Jump if greater then		
jge	LABEL		Jump if greater or equal		
jl	LABEL		Jump if less then		
jle	LABEL		Jump if less or equal		
call	LABEL		Call subroutine		
loop	LABEL	%ecx-	dec %ecx, jump to label if non-zero		
loope	LABEL	if -%ecx != 0 && ZERO flag: %eip = LABEL			
loopz	LABEL	if -%ecx != 0 && ZERO flag: %eip = LABEL			
loopnz	LABEL				
misc					
int	INT_NR	Cause software interrupt number INT_NR		misc	