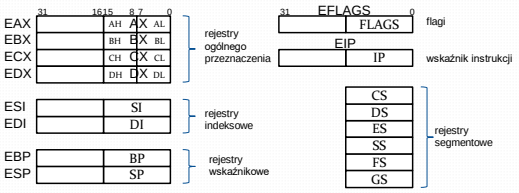
Tryby adresowania
Instrukcje przesyłania

IA32- rejestry



Rejestr flag

Rejestr flag w architekturze Intel x86			
bit	Skrót/wartość	opis	typ
0	CF	flaga przeniesienia (carry)	S
2	PF	flaga parzystości (parity)	S
4	AF	flaga wyrównania (adjust)	S
6	ZF	flaga zera (zero)	S
7	SF	flaga znaku (sign)	S
10	DF	flaga kierunku (direction)	C
11	OF	flaga przepełnienia (overflow)	S

S: Znacznik stanu
C: Znacznik kontrolny
X: Znacznik systemowy

Format rozkazów

```
etykieta:    mnemonik    argument1, argument2    ;komentarz
etykieta:    mnemonik    cel, źródło             ;komentarz

etykieta:    mnemonik
             mnemonik    argument1
             mnemonik    argument1, argument2
Np.:
        ret
        pop     eax
        mov     edx,ecx  ;zapamiętaj licznik
        mov     rax,1001
```


Tryby adresowania - rejestrowy

Argumentem instrukcji jest rejestr:

```
push   ebx

mov    edx,ebx

inc    ecx

dec    r9
```


Tryby adresowania - prosty - natychmiastowy

Argumentem instrukcji jest wartość (zawiera się w kodzie rozkazu):

```
mov    al, 5

mov    r10d,32

mov    edi, offset tabela

jnz    petla
```

Tryby adresowania - bezpośredni

Argumentem instrukcji jest adres w pamięci (wskaźnik):

```
mov   al, [1234ec5fh]
mov   edi, tabela    ;pobiera pierwszy element
mov   zmienna, rdx
```


Tryby adresowania - pośredni - rejestrowy

Argumentem instrukcji jest rejestr – wskaźnik:

```
mov   al, [rcx]
mov   edi, [ebx]
mov   [edi], edx
```


Tryby adresowania - pośredni - bazowy

Argumentem instrukcji jest wskaźnik:

```
mov   al, [ebx+5]
mov   edi, [ebx+tablica]
mov   [rbp+8], rdx
```


Tryby adresowania - pośredni - indeksowy

Argumentem instrukcji jest rejestr – wskaźnik:

```
mov   al, [esi]
mov   edi, [esi*4+tablica]
mov   [rdi*8+tablica], rdx
```


Tryby adresowania - pośredni – bazowo-indeksowy

Argumentem instrukcji jest wskaźnik:

```
mov   al, [ebx+esi+3]
mov   edi, [ebx+eax*4]
mov   [rbp+rdi*8+tablica], rdx
```


Wielkość danych

Można określić wielkość stosowanych danych:

```
mov   al, byte ptr [ebx+esi+3]
mov   cx, word ptr [ebx+eax*4]
mov   dword ptr [ebp+edi*4+tablica], edx
mov   qword ptr [rbp+rdi*8+tablica], rdx

inc    byte ptr [ebx+esi+3]
dec    word ptr [ebx+eax*4]
inc    dword ptr [ebp+edi*4+tablica]
```

Przedrostki segmentowe

Można podać segment do danych:

```
mov   al, es:byte ptr [ebx+esi+3]
mov   cx, cs:word ptr [ebx+eax*4]
mov   ss:[ebp+4], edx
```

Przyporządkowanie rejestrów
esp, ebp: ss
eax, ebx, ecx, edx, edi, esi: ds.
eip: cs
Analogicznie rejestry 16 i 64 bitowe.

Instrukcje przesyłania

MOV	przesyła dane między rejestrami, pamięcią
XCHG	zamień
BSWAP	zamień bajty
XADD	wymień i dodaj
CMPXCHG	porównaj i wymień
CMPXCHG8(16)B	porównaj i wymień 8(16) bajtów
PUSH	wyślij na stos
POP	zdejmij ze stosu
PUSHF/PUSHFD/PUSHFQ	wyślij na stos flagi
POPF/POPFD/POPFQ	zdejmij ze stosu flagi
PUSHA/PUSHAD	wyślij rejestry na stos
POPA/POPAD	zdejmij rejestry ze stosu
CWD/CDQ/CQO	konwertuj word na dword/dword na qword
CBW/CWDE /CDQE	konwertuj byte na word/word na doubleword w rejestrze EAX/ doubleword na qouadword w RAX
MOVSX/MOVSXD	prześlij i rozszerz znakiem
MOVZX	prześlij i rozszerz zerem

Wpływa na flagi: -

Instrukcja MOV

```
mov   cel, źródło
```

Przesyła zawartość źródła do miejsca przeznaczenia (cel).

```
mov   al, bl
mov   [ebp+4], edx
mov   zmienna, eax
mov   rcx,licznik
mov   [ebp+edi*4+tablica], edx
```


Wpływa na flagi: -

Instrukcja XCHG

```
xchg      cel, źródło
```

Zamienia zawartość źródła i celu.

```
xchg      ax, zmienna
xchg      ecx, [epb+4]
xchg      rax, r12
```


Wpływa na flagi: -

Instrukcja BSWAP

```
bswap     rejestr
```

Zamienia bajty w argumencie – 32/64 bity.

```
bswap     eax
bswap     rdx
```

przed

12	c4	7f	de

po

de	7f	c4	12

Wpływa na flagi: OSZAPC

Instrukcja XADD

```
xadd   cel, źródło
```

Zamienia zawartość źródła i celu(8/16/32/64 bity), a ich sumę umieszcza w miejscu przeznaczenia (cel).

```
xadd   al,bl
xadd   eax,zmienna
xadd   edx,[ebx+esi*4]
xadd   rcx,r8
```

Wpływa na flagi: OSZAPC

Instrukcja CMPXCHG

CMPXCHG arg1, arg2

Działanie:

if acc = arg1 then
    arg1 = arg2
else
    acc = arg1

acc = al, ax, eax, rax
arg2 - rejestr

Wpływa na flagi: OSZAPC

Instrukcja CMPXCHG8(16)B

CMPXCHG8(16)B cel

Działanie:

if (E(R)DX:E(R)AX = cel) then
    cel = e(r)cx:e(r)bx
else
    e(r)dx:e(r)ax = cel

Wpływa na flagi: -

Instrukcja PUSH

push arg

Przesyła zawartość argumentu na stos.

```
push    eax
push    rdx
push    ds
push    1234
```


Wpływa na flagi: -

Instrukcja POP

pop cel

Przesyła zawartość stosu do celu.

```
pop    bx
pop    ecx
pop    rdx
pop    [edx+edi+4]
```


Wpływa na flagi: -

Instrukcja PUSHF/PUSHFD/PUSHFQ

pushf/pushfd/pushfq

Przesyła zawartość Flag/Eflag/Rflag na stos.

```
pushf
pushfd
pushfq
```


Wpływa na flagi: OSZAPC

Instrukcja POPF/POPFD/POPFQ

popf/popfd/popfq

Pobiera zawartość Flag/Eflag/Rflag ze stosu.

```
popf
popfd
popfq
```

Wpływa na flagi: -

Instrukcja PUSHA/PUSHAD

pusha/pushad

Przesyła zawartość di, si, bp, bx, dx, cx, ax / edi, esi, ebp, ebx, edx, ecx, eax na stos.

Instrukcja nie działa w trybie 64-bitowym.

```
pusha
pushad
```


Wpływa na flagi: -

Instrukcja POPA/POPAD

popa/popad

Przesyła zawartość stosu do di, si, bp, bx, dx, cx, ax / edi, esi, ebp, ebx, edx, ecx, eax.

Instrukcja nie działa w trybie 64-bitowym.

```
popa
popad
```


Wpływa na flagi: -

Instrukcja CWD/CDQ/CQO

CWD/CDQ/CQO

Konwertuje z zachowaniem znaku word na doubleword / doubleword na quadword / quadword na octaword (ax na dx:ax, eax na edx:eax, rax na rdx:rax).

```
cwd
cdq
cqo
```


Wpływa na flagi: -

Instrukcja CBW/CWDE/CDQE

CBW/CWDE

Konwertuje byte (AL) na word(AX) / word(AX) na doubleword (EAX) / doubleword (EAX) na quadword (RAX) z uwzględnieniem znaku.

```
cbw
cwde
cdqe
```


Wpływa na flagi: -

Instrukcja MOVSX/MOVSXD

movsx cel, źródło

Przesyła zawartość źródła do rejestru celu z uwzględnieniem znaku. Cel posiada 2/4/8 razy więcej bitów.

```
movsx     eax, bl
movsx     cx, al
movsxd    r8, edx     ;movsxd tylko dla 32 na 64
```


Wpływa na flagi: -

Instrukcja MOVZX

movzx cel, źródło

Przesyła zawartość źródła do rejestru celu z dopisaniem na starszych bitach zer. Cel posiada 2/4/8 razy więcej bitów. Źródło 8/16 bitów.

```
movzx     eax, bl
movzx     cx, al
```

Przykład

Wypełnienie wartościami od 0 do 255 tabeli bajtów

```
        mov   ecx, 256
        mov   eax, 0
        mov   edi, 0
p1:     mov   [edi+tabela], al
        inc   edi
        inc   eax
        dec   ecx
        jnz   p1
```


Przykład

Przepisanie wartości integer (32 bity) z tabeli tab1 do tabeli tab2.

```
        mov   rcx, 65536
        mov   rdi, 0
p1:     mov   eax, [tab1+4*rdi]
        mov   [tab2+4*rdi], eax
        inc   rdi
        dec   rcx
        jnz   p1
```


Przykład

Przepisanie wartości int (32 bity) z tabeli tab1 do tabeli tab2 z konwersją na int64. rcx – adres tab1, rdx - adres tab2, r8 – liczba elementów.

```
p1:     movsxd    rax, dword ptr[rcx+4*r8-4]
        mov       [rdx+8*r8-8], rax
        dec       r8
        jnz       p1
```


Przykład

Zamiana zawartości zmiennych a i b typu int64.

```
        mov   rax, a
        xchg  rax, b
        mov   a, rax

        push  a
        push  b
        pop   a
        pop   b
```