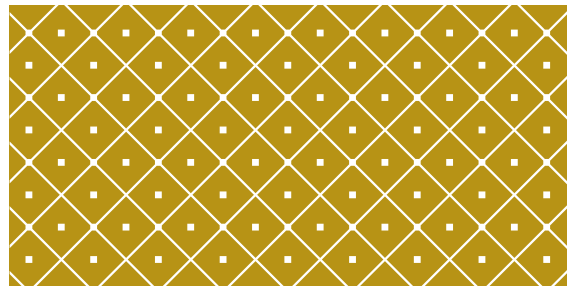


PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE

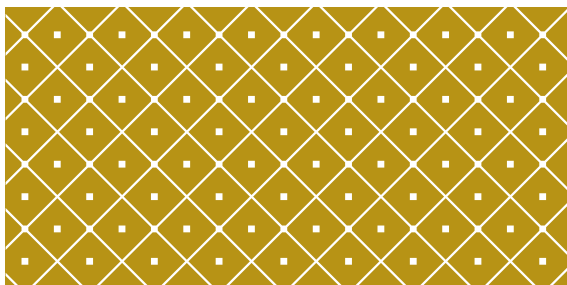
Wykład dla kierunku:
Matematyka stosowana
i technologie
informatyczne



PROGRAMOWANIE PROCESORÓW.

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 2



TRYBY ADRESOWANIA INSTRUKCJE PRZESYŁANIA

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 3

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

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 4

Tryby adresowania - rejstrowy

Argumentem instrukcji jest rejestr:

```
push ebx  
  
mov edx,ebx  
  
inc ecx  
  
dec r9
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 5

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
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 6

Tryby adresowania - bezpośredni

Argumentem instrukcji jest adres w pamięci (wskaźnik):

```
mov al, [1234ec5fh]

mov edi, tabela ;pobiera pierwszy element

mov zmienna, rdx
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 7

Tryby adresowania - pośredni - rejestrowy

Argumentem instrukcji jest rejestr – wskaźnik

```
mov al, [rcx]

mov edi, [ebx]

mov [edi], edx
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 8

Tryby adresowania - pośredni - bazowy

Argumentem instrukcji jest wskaźnik

```
mov al, [ebx+5]

mov edi, [ebx+tablica]

mov [rbp+8], rdx
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 9

Tryby adresowania - pośredni - indeksowy

Argumentem instrukcji jest rejestr – wskaźnik

```
mov al, [esi]

mov edi, [esi*4+tablica]

mov [rdi*8+tablica], rdx
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 10

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
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 11

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]
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 12

Przedrostki segmentowe

Można podać segment do danych ch:

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

Przy porządkowanie rejestrów

esp, ebp: ss

eax, ebx, ecx, edx, edi, esi: ds.

eip: cs

Analogicznie rejestry 16 i 64 bitowe.

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 13

Instrukcje przesyłania

MOV	przesyła dane między rejestrami, pamięcią
XCHG	zamień
BSWAP	zamień bajty
XADD	wymień i dodaj
PUSH	wyślij na stos
POP	zdejmij ze stosu
PUSHF/PUSHFD/PUSHFQ	wyślij na stos flagi
POPF/POPFD/POPFQ	zdejmij ze stosu flagi
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 quadword w RAX
MOVSX/MOVSXD	prześlij i rozszerz znakiem
MOVZX	prześlij i rozszerz zerem

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 14

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
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 15

Wpływa na flagi: -

Instrukcja XCHG

```
xchg cel, źródło
```

Zamienia zawartość źródła i celu.

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

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 16

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
----	----	----	----

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 17

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
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 18

Wpływa na flagi: -

Instrukcja PUSH

`push arg`

Przesyła zawartość argumentu na stos.

```

push    eax
push    rdx
push    ds
push    1234

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 19

Wpływa na flagi: -

Instrukcja POP

`pop cel`

Przesyła zawartość stosu do celu.

```

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

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 20

Wpływa na flagi: -

Instrukcja PUSHF/PUSHFD/PUSHFQ

`pushf/pushfd/pushfq`

Przesyła zawartość Flag/Eflag/Rflag na stos.

```

pushf
pushfd
pushfq

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 21

Wpływa na flagi: OSZAPC

Instrukcja POPF/POPFD/POPFQ

`popf/popfd/popfq`

Pobiera zawartość Flag/Eflag/Rflag ze stosu.

```

popf
popfd
popfq

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 22

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

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 23

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

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 24

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
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 25

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
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 26

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
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 27

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
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 28

Przykład

Przepisanie wartości int (32 bity) z tabeli tab1 do tabeli tab2 z konwersją na int64. rcx – adrestab1, rdx - adrestab2, r8 – liczba elementów.

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

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 29

Przykład

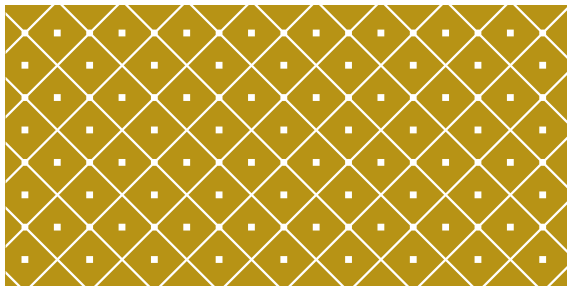
Zamiana zawartości zmiennych a i b typu int64.

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

push   a
push   b
pop    a
pop    b
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 30



INSTRUKCJE ARYTMETYCZNE

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 31

Instrukcje arytmetyczne

ADD	dodawanie całkowitoliczbowe
ADC	dodawanie z przeniesieniem
ADCX	dodawanie z przeniesieniem bez znaku
ADOX	dodawanie z przeniesieniem bez znaku
SUB	odejmowanie
SBB	odejmowanie z pożyczką
MUL	mnożenie bez znaku
MULX	mnożenie bez znaku
IMUL	mnożenie ze znakiem
DIV	dzielenie bez znaku
IDIV	dzielenie ze znakiem
INC	inkrementacja (zwiększenie)
DEC	dekrementacja (zmniejszenie)
NEG	zmiana znaku
CMP	porównanie

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 32

Wpływa na flagi: OSZAPC

Instrukcja ADD

```
add cel, źródło
```

Dodaje zawartość źródła i celu, sumę umieszcza w miejscu przeznaczenia (cel).

```
cel := cel + źródło
```

```
add eax, zmienna
add edx, [ebx+esi*4]
add rcx, rbx
```

Uwaga:

Jeśli źródło jest adresowane w trybie prostym może mieć do 32 bitów.

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 33

Przykład – oblicz sumę krawędzi prostokątnianu

```
mov eax, a           ;wczytaj a
add eax, b           ;dodaj b
add eax, c           ;dodaj c
add eax, eax         ; *2
add eax, eax         ; *2
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 34

Wpływa na flagi: OSZAPC

Instrukcja ADC

```
adc cel, źródło
```

Dodaje zawartość źródła, celu i przeniesienia, sumę umieszcza w miejscu przeznaczenia (cel).

```
cel := cel + źródło + CF
```

```
adc eax, zmienna
adc edx, [ebx+esi*4]
add rcx, rbx
```

Uwaga:

Jeśli źródło jest adresowane w trybie prostym może mieć do 32 bitów.

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 35

Przykład – oblicz sumę liczb całkowitych 128-bitowych

```
mov esi, offset b   ;adres zmiennej b
mov edi, offset a   ;adres zmiennej a
mov eax, [esi]      ;najmłodsze podwójne słowo b
add [edi], eax      ;dodajemy do najmłodszego a
mov eax, [esi+4]    ;drugie podwójne słowo b
adc [edi+4], eax    ;do drugiego a plus przeniesienie
mov eax, [esi+8]    ;trzecie podwójne słowo b
adc [edi+8], eax    ;do trzeciego a plus przeniesienie
mov eax, [esi+12]   ;czwarte podwójne słowo b
adc [edi+12], eax   ;do czwartego a plus przeniesienie
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 36

Przykład – oblicz sumę liczb całkowitych 128-bitowych

```

mov rsi, offset b ;adres zmiennej b
mov rdi, offset a ;adres zmiennej a
mov rax, [rsi] ;najmłodsze poczwórne słowo b
add [rdi], rax ;dodajemy do najmłodszego a
mov rax, [rsi+8] ;drugie poczwórne słowo b
adc [rdi+8], rax ;do drugiego a plus przeniesienie

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 37

Wpływa na flagi: -----C
Wymagane: ADX

Instrukcja ADCX

```

adcx cel, źródło

```

Dodaje bez znaku zawartość źródła, celu i przeniesienia, sumę umieszcza w miejscu przeznaczenia (cel – rejestr 32|64 bitowy).

cel := cel + źródło + CF

```

adcx eax, zmienna
adcx edx, [ebx+esi*4]
adcx rcx, rbx

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 38

Instrukcja ADOX

```

adox cel, źródło

```

Dodaje bez znaku zawartość źródła, celu i flagi przepełnienia, sumę umieszcza w miejscu przeznaczenia (cel – rejestr 32|64 bitowy).

cel := cel + źródło + OF

```

adox eax, zmienna
adox edx, [ebx+esi*4]
adox rcx, rbx

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 39

Wpływa na flagi: O-----
Wymagane: ADX

Instrukcja SUB

```

sub cel, źródło

```

Odejmuje zawartość źródła od celu, różnicę umieszcza w miejscu przeznaczenia (cel).

cel := cel - źródło

```

sub ecx, zmienna
sub ebx, [ebx+esi*4]
sub rax, rdx

```

Uwaga:

Jeśli źródło jest adresowane w trybie prostym może mieć do 32 bitów.

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 40

Wpływa na flagi: OSZAPC

Instrukcja SBB

```

sbb cel, źródło

```

Odejmuje zawartość źródła od celu z uwzględnieniem pożyczki, różnicę umieszcza w miejscu przeznaczenia (cel).

cel := cel - (źródło + CF)

```

sbb edx, zmienna
sbb eax, [ebx+esi*4]
sbb rax, rdx

```

Uwaga:

Jeśli źródło jest adresowane w trybie prostym może mieć do 32 bitów.

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 41

Wpływa na flagi: OSZAPC

Instrukcja MUL

```

mul źródło

```

Mnoży bez znaku zawartość akumulatora (al, ax, eax, rax) i źródła, wynik umieszcza w miejscu przeznaczenia (ax, dx:ax, edx:eax, rdx:rax). Flagi CF i OF są zerem, jeśli starsza połowa bitów wyniku jest równa zero.

wynik := acc * źródło

```

mul zmienna
mul word ptr [ebx + esi*4]

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 42

Wpływa na flagi: OxxxxC

Przykład – oblicz pole prostokąta o bokach a, b

```

mov  eax, a           ;a
mul  b                ; a * b
jc   poza_int        ;jeśli przekroczony zakres
mov  pole, eax        ;zapisz
...
...
poza_int: ...

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 43

Wpływa na flagi: -----
Wymaga BMI2

Instrukcja MULX

```

mulx cel1, cel2, źródło

```

Mnoży bez znaku zawartość rejestru `edx` `rdx` i źródła, wynik umieszcza w rejestrach celu (`cel1:cel2`).

$$\text{cel1:cel2} := e(r)dx * \text{źródło}$$

```

mulx ecx, ebx, [tab + esi*4]
mulx rcx, rbx, rax

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 44

Wpływa na flagi: OxxxxC

Instrukcja IMUL - 1

```

imul źródło

```

Mnoży ze znakiem zawartość akumulatora (`al`, `ax`, `eax`, `rax`) i źródła, wynik umieszcza w miejscu przeznaczenia (`ax`, `dx:ax`, `edx:eax`, `rdx:rax`). Flagi `CF` i `OF` są zerem, jeśli iloczyn mieści się w młodszej połowie bitów wyniku.

$$\text{wynik} := \text{acc} * \text{źródło}$$

```

imul zmienna
imul ecx

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 45

Wpływa na flagi: OxxxxC

Instrukcja IMUL - 2

```

imul cel, źródło

```

Mnoży ze znakiem zawartość rejestru celu (16|32|64 bity) przez źródło, wynik umieszcza w miejscu przeznaczenia (`cel`). Flagi `CF` i `OF` są zerem, jeśli iloczyn mieści się w rejestrze celu.

$$\text{cel} := \text{cel} * \text{źródło}$$

```

imul eax, zmienna
imul rdx, rcx

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 46

Wpływa na flagi: OxxxxC

Instrukcja IMUL - 3

```

imul cel, źródło1, źródło2

```

Mnoży ze znakiem zawartość źródła1 (16|32|64 bity) przez źródło2 (stała), wynik umieszcza w miejscu przeznaczenia (`cel`). Flagi `CF` i `OF` są zerem, jeśli iloczyn mieści się w rejestrze celu.

$$\text{cel} := \text{źródło1} * \text{źródło2}$$

```

imul eax, zmienna, 5
imul cx, dx, 77

```

Uwaga:

Jeśli `źródło2` jest adresowane w trybie prostym może mieć do 32 bitów.

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 47

Przykład – oblicz sumę krawędzi prostopadłościanu

```

mov  eax, a           ;wczytaj a
add  eax, b           ;dodaj b
add  eax, c           ;dodaj c
imul eax, eax, 4      ; *4

```

(C) KISI d.KIK PCz 2023

Programowanie wektorowe i równoległe

48

Wpływa na flagi: xxxxxx

Instrukcja DIV

div źródło

Dzieli bez znaku zawartość AX, DX:AX, EDX:EAX, RDX:RAX przez źródło, iloraz umieszcza w AL, AX, EAX, RAX a resztę w AH, DX, EDX, RDX.

div byte ptr zmienna

div ebx

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 49

Wpływa na flagi: xxxxxx

Instrukcja IDIV

idiv źródło

Dzieli ze znakiem zawartość AX, DX:AX, EDX:EAX, RDX:RAX przez źródło, iloraz umieszcza w AL, AX, EAX, RAX a resztę w AH, DX, EDX, RDX.

idiv byte ptr zmienna

idiv ebx

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 50

Wpływa na flagi: OSZAP

Instrukcja INC

inc cel

Zwiększa zawartość celu o 1.

inc zmienna

inc edx

inc rcx

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 51

Wpływa na flagi: OSZAP

Instrukcja DEC

dec cel

Zmniejsza zawartość celu o 1.

dec zmienna

dec edx

dec r8

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 52

Wpływa na flagi: OSZAPC

Instrukcja NEG

neg cel

Zmienia znak celu w kodzie U2.

cel := -cel

neg ax

neg byte ptr[ebx+esi*4]

neg r11

Flaga CF=0, tylko dla argumentu=0.

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 53

Wpływa na flagi: OSZAPC

Instrukcja CMP

cmp źródło1, źródło2

Porównuje zawartość źródła1 i źródła2, wyniknie jest zapamiętywany, tylko są ustawiane flagi.

źródło1-źródło2

cmp ax, zmienna

cmp edx, [ebx+esi*4]

cmp rcx, 123

Uwaga:

Jeśli źródło2 jest adresowane w trybie prostym może mieć do 32 bitów.

(C) KISI d.KIK PCz 2023

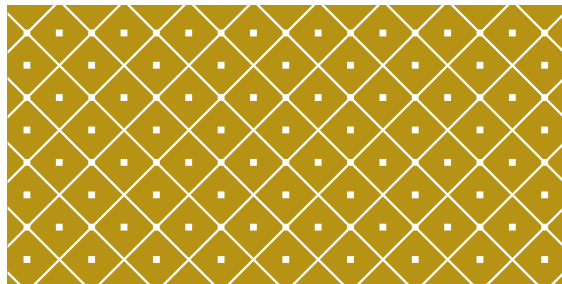
PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 54

Przykład – oblicz sumę kwadratów liczb w tablicy

```

mov  eax, 0           ;wartość początkowa sumy
mov  esi, eax         ;indeks
mov  ebx, offset wektor ;tablica liczb całkowitych
mov  ecx, 123        ;licznik
petla: mov  edx, [ebx+esi*4]
      imul  edx, edx   ;kwadrat liczby
      add  eax, edx    ;suma
      inc  esi
      dec  ecx
      jnz  petla      ;wynikw eax

```



INSTRUKCJE LOGICZNE, PRZESUNIĘĆ I ROTACJI

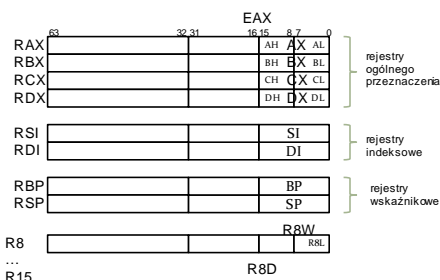
(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 55

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 56

EM64T/Intel 64 - rejestry



(C) KISI d.KIK PCz 2023

Programowanie wektorowe i równoległe

57

Instrukcje logiczne

- AND bitowa funkcja AND
- ANDN bitowa funkcja AND z negacją
- OR bitowa funkcja OR
- XOR bitowa funkcja OR
- NOT bitowa funkcja NOT

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 58

Wpływa na flagi: OSZAPC OSZxP0

Instrukcja AND

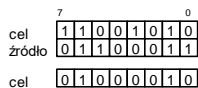
```
and cel, źródło
```

Wyznacza iloczyn logiczny zawartości źródła i celu (bit po bicie), wynik umieszcza w miejscu przeznaczenia (cel).

```
cel := cel and źródło
```

```
and  eax, zmienna
and  edx, [ebx+esi*4]
and  rax, rdx
```

Uwaga: Jeśli źródło jest adresowane w trybie prostym może mieć do 32 bitów.



(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 59

Wpływa na flagi: OSZAPC OSZx00 Wy magane BMI1

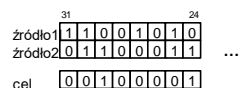
Instrukcja ANDN

```
andn cel, źródło1, źródło2
```

Wyznacza iloczyn logiczny zanegowanego źródła1 i źródła2 (bit po bicie), wynik umieszcza w miejscu przeznaczenia (cel). Cel i źródło1 są rejestrami 32/64 bitowymi.

```
cel := (not źródło1) and źródło2
```

```
andn  edx, eax, zmienna
andn  eax, edx, [ebx+esi*4]
andn  r8, rax, rdx
```



(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 60

Wpływa na flagi: OSZAPC
OSZxP0

Instrukcja OR

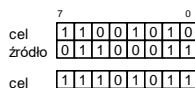
or cel, źródło

Wyznacza sumę logiczną zawartości źródła i celu (bit po bicie), wynik umieszcza w miejscu przeznaczenia (cel).

cel := cel or źródło

or eax, zmienna
or edx, [ebx+esi*4]
or rax, r9

Uwaga:
Jeśli źródło jest adresowane w trybie prostym może mieć do 32 bitów.



(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 61

Wpływa na flagi: OSZAPC
OSZxP0

Instrukcja XOR

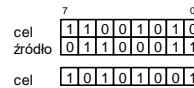
xor cel, źródło

Wyznacza, bit po bicie, sumę modulo 2 zawartości źródła i celu wynik umieszcza w miejscu przeznaczenia (cel).

cel := cel xor źródło

xor eax, zmienna
xor edx, [ebx+esi*4]
xor rax, r9

Uwaga:
Jeśli źródło jest adresowane w trybie prostym może mieć do 32 bitów.



(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 62

Wpływa na flagi: -

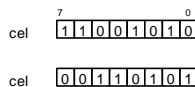
Instrukcja NOT

not cel

Wyznacza negację logiczną zawartości i celu (bit po bicie), wynik umieszcza w miejscu przeznaczenia (cel).

cel := not cel

not eax
not byte ptr [ebx+esi*4]
not rdx



(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 63

If ((!a&&b)||{c&&d&&e}) {...}else {...};

```

mov al, a          mov al, a          mov di, c
not al             andn al, b         mov al, a
and al, b         jnz then_1         and di, d
mov di, c         mov di, c         andn al, al, b
and di, d         and di, d         and di, e
and di, e         and di, e         or al, di
or al, di         or al, di         jz else_1
jz else_1         jz else_1         then_1:
then_1:           then_1:

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 64

Wpływa na flagi: OSZAPC
(0x)SZxPC

Instrukcje przesunięć i rotacji

- SAR przesunięcie arytmetyczne w prawo
- SHR przesunięcie logiczne w prawo
- SAL przesunięcie arytmetyczne w lewo
- SHL przesunięcie logiczne w lewo
- SARX przesunięcie arytmetyczne w prawo
- SHRX przesunięcie logiczne w prawo
- SHLX przesunięcie logiczne w lewo
- SHRD przesunięcie w prawo double
- SHLD przesunięcie w lewo double
- ROR rotacja w prawo
- RQL rotacja w lewo
- RCR rotacja w prawo przez przeniesienie
- RCL rotacja w lewo przez przeniesienie
- RORX rotacja w prawo

(C) KISI d.KIK PCz 2023

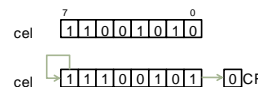
PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 65

Instrukcja SAR

sar cel, ile

Przesunięcie arytmetyczne celu w prawo o ile bitów. Ile=1, ci lub wartość 0-31|63.

sar eax, 1
sar [ebx+esi*4], cl
sar rdx, cl



(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 66

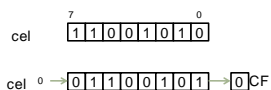
Wpływa na flagi: OSZAPC (0x)SZxPC

Instrukcja SHR

```
shr cel, ile
```

Przesunięcie logiczne w prawo o ile bitów. Ile=1, cl lub wartość 0-31|63.

```
shr eax, 1
shr [ebx+esi*4], cl
shr rdx, cl
```



(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 67

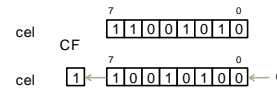
Wpływa na flagi: OSZAPC (0x)SZxPC

Instrukcja SAL

```
sal cel, ile
```

Przesunięcie arytmetyczne celu w lewo o ile bitów. Ile=1, cl lub wartość 0-31|63.

```
sal eax, 1
sal [ebx+esi*4], cl
sal rdx, cl
```



(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 68

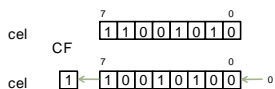
Wpływa na flagi: OSZAPC (0x)SZxPC

Instrukcja SHL

```
shl cel, ile
```

Przesunięcie logiczne celu w lewo o ile bitów. Ile=1, cl lub wartość 0-31|63.

```
shl eax, 1
shl [ebx+esi*4], cl
shl rdx, cl
```



(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 69

Przykład – oblicz sumę krawędzi prostopadłościanu

```
mov eax, a ;wczytaj a
add eax, b ;dodaj b
add eax, c ;dodaj c
sal eax, 2 ; *4
```

(C) KISI d.KIK PCz 2023

Programowanie wektorowe i równoległe

70

Wpływa na flagi: -----
Wymaga BMI2

Instrukcja SARX

```
sarx cel, źródło, ile
```

Przesunięcie arytmetyczne źródła w prawo o ile bitów i zapisanie w celu. Cel i ile są rejestrami 32|64 bitowymi.

```
sarx eax, zmienna, edx
sarx eax, [ebx+esi*4], ecx
sarx rdx, rax, rcx
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 71

Wpływa na flagi: -----
Wymaga BMI2

Instrukcja SHRX

```
shrx cel, źródło, ile
```

Przesunięcie logiczne źródła w prawo o ile bitów i zapisanie w celu. Cel i ile są rejestrami 32|64 bitowymi.

```
shrx eax, zmienna, edx
shrx eax, [ebx+esi*4], ecx
shrx rdx, rax, rcx
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 72

Wpływa na flagi: -----
Wymaga BMI2

Instrukcja SHLX

shl x cel, źródło, ile

Przesunięcie logiczne źródła w lewo ile bitów i zapisanie w celu. Cel i ile są rejestrami 32|64 bitowymi.

```
shl x  eax, zmienna, edx
shl x  eax, [ebx+esi*4], ecx
shl x  rdx, rax, rcx
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 73

Wpływa na flagi: OSZAPC
(Ox)SZxPC

Instrukcja SHRD

shrd cel, źródło, ile

Przesunięcie źródła:celu w prawo o ile bitów. Ile=cl lub wartość 0-31|64. Rejestr źródła (16,32,64) pozostaje bez zmian.

```
shrd  eax, ecx, 15
shrd  [ebx+esi*4], edx, cl
shrd  zmienna, rax, cl
```



(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 74

Wpływa na flagi: OSZAPC
(Ox)SZxPC

Instrukcja SHLD

shld cel, źródło, ile

Przesunięcie źródła:celu w lewo o ile bitów. Ile=d lub wartość 0-31|63. Rejestr źródła (16,32,64) pozostaje bez zmian.

```
shld  eax, ecx, 15
shld  [ebx+esi*4], edx, cl
shld  zmienna, rax, cl
```



(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 75

Przykład – podzielić 256-bitową liczbę a przez 64

```
mov  rcx, offset a ;w czytaj adres a
mov  rax, [rcx+8]   ;w czytaj drugie 64 bity
shrd [rcx], rax, 6  ;przesuń pierwsze
mov  rax, [rcx+16] ;w czytaj trzecie 64 bity
shrd [rcx+8], rax, 6 ;przesuń drugie
mov  rax, [rcx+24] ;w czytaj czwarte 64 bity
shrd [rcx+16], rax, 6 ;przesuń trzecie
shr  rax, 6         ;przesuń czwarte
mov  [rcx+24], rax ;zapisz
```

(C) KISI d.KIK PCz 2023

Programowanie wektorowe i równoległe

76

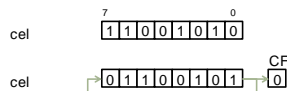
Wpływa na flagi: OSZAPC
(0x)----C

Instrukcja ROR

ror cel, ile

Rotacja (obrót) celu w prawo o ile bitów. Ile=1, cl lub wartość 0-31|63.

```
ror  eax, 1
ror  [ebx+esi*4], cl
ror  rdx, cl
```



(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 77

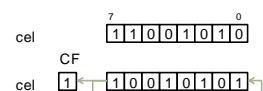
Wpływa na flagi: OSZAPC
(0x)----C

Instrukcja ROL

rol cel, ile

Rotacja (obrót) celu w lewo o ile bitów. Ile=1, cl lub wartość 0-31|63.

```
rol  eax, 1
rol  [ebx+esi*4], cl
rol  rdx, cl
```



(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 78

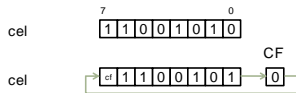
Wpływa na flagi: OSZAPC (0x)----C

Instrukcja RCR

```
rcr cel, ile
```

Rotacja (obrót) przez przeniesienie celu w prawo o ile bitów. Ile=1, cl lub wartość 0-31|63.

```
rcr eax, 1
rcr [ebx+esi*4], cl
rcr rdx, cl
```



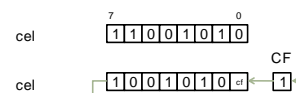
Wpływa na flagi: OSZAPC (0x)----C

Instrukcja RCL

```
ror cel, ile
```

Rotacja (obrót) przez przeniesienie celu w lewo o ile bitów. Ile=1, cl lub wartość 0-31|63.

```
rcl eax, 1
rcl [ebx+esi*4], cl
rcl rdx, cl
```



Wpływa na flagi: -----
Wymaga BMI2

Instrukcja RORX

```
rorx cel, źródło, ile
```

Rotacja źródła w prawo o ile bitów i zapisanie w celu. Cel jest rejestrem 32|64 bitowym. Ile przyjmuje wartość 0-31|63.

```
rorx eax, zmienna, 12
rorx eax, [ebx+esi*4], 7
rorx rdx, rax, 44
```

Przykłady

```
and eax, 0ffffh ;zeruje 12 bit eax
Or ax, 01000h ;ustawia 12 bit ax
xor rax, 01000h ;neguje 12 bit rax
and eax, eax ;m. in. zeruje CF
xor eax, eax ;zeruje eax
```

Przykłady

```
and eax, 070h ;maska
sar eax, 4 ;eax - 3bitowa liczba

mov ah, al ;zamienia al na jego
and ax, 0f00fh ;szesnastkową reprezentację
shr ah, 4 ;ASCII w ah, al.
add ax, 3030h
```

Przykłady - * 4,25

```
movsxd rax, zmienna ;typu int - 32 bity
mov rdx, rax ;powiel
sal rax, 2 ;x4
sar rdx, 2 ;/4
adc rax, rdx ;x4¼
mov zmienna, eax ;zapisz

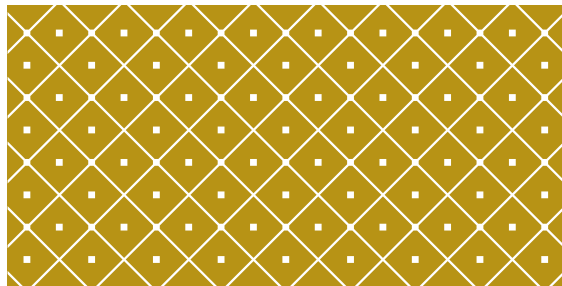
movsxd rdx, eax ; test
cmp rax, rdx ;porów naj
jnz blad ;przekroczenie zakresu
```

Przykłady

```

;odwracanie bitów z rcx; rdx liczba odwracanych bitów 1-64
invb proc
    xor     rax, rax           ;rax:=0
    dec    rdx               ;eliminacja zera
    js     @e
    and    rdx, 3fh          ;ograniczenie zakresu
@p:      shr    rcx, 1         ;lsb->CF
        rcl    rax, 1         ;CF->msb
    dec    rdx               ;licznik--
    jns   @p                 ;następny jeśli nieujemne
@e:      ret
invb endp

```



INSTRUKCJE WARUNKOWE I SKOKU

Rejestr flag



Rejestr	flag	w architekturze	Intel x86
bit	Skroty/wartość	opis	typ
0	CF	flaga przeniesienia (carry)	S
2	PF	flaga parzystości (parity)	S
4	AF	flaga wyroczenia (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 kontroli
 X: Znacznik systemowy
 (C) KISI d.KIK PCz 2023

Warunki dotyczące flag

- E/Z equal/ zero ZF=1
- NE/NZ not equal/ not zero ZF=0
- C carry CF=1
- NC not carry CF=0
- O overflow OF=1
- NO not overflow OF=0
- S sign (negative) SF=1
- NS not sign (non-negative) SF=0
- P/PE parity/parity even PF=1
- NP/PO not parity/parity odd PF=0

Wpływa na flagi: -

Warunki porównania liczb

- E/Z equal/ zero ZF=1
 - NE/NZ not equal/ not zero ZF=0
- Dla liczb bez znaku:
- A/NBE above/ not below or equal CF=0 i ZF=0
 - AE/NB above or equal/ not below CF=0
 - B/NAE below/ not above or equal CF=1
 - BE/NA below or equal/ not above CF=1 lub ZF=1
- Dla liczb ze znakiem
- GNLE greater/ not less or equal ZF=0 i SF=OF
 - GE/NL greater or equal/ not less SF=OF
 - L/NGE less/ not greater or equal SF<->OF
 - LE/NG less or equal/ not greater ZF=1 lub SF<->OF

Instrukcja CMOVcc

CMOVcc cel, źródło

Jeśli jest spełniony warunekcc, przesyła źródło do miejsca przeznaczenia (rejestr 16, 32 lub 64 bitowy). Instrukcja wprowadzona w procesorach rodziny P6!

if cc then cel:=źródło

```

cmovz eax, zmienna
cmovgeedx, [ebx+esi*4]
cmovnarax, rdx

```

Instrukcje CMOVcc

CMOVB/CMOVZ	Prześlij jeśli equal/ zero
CMOVNB/CMOVNZ	Prześlij jeśli not equal/ not zero
CMOVA/CMOVNB	Prześlij jeśli above/ not below or equal
CMOVAE/CMOVNB	Prześlij jeśli above or equal/ not below
CMOVNB/CMOVNAE	Prześlij jeśli below/ not above or equal
CMOVBE/CMOVNA	Prześlij jeśli below or equal/ not above
CMOVGB/CMOVNLE	Prześlij jeśli greater/ not less or equal
CMOVGE/CMOVNLE	Prześlij jeśli greater or equal/ not less
CMOVLE/CMOVNGE	Prześlij jeśli less/ not greater or equal
CMOVLE/CMOVNG	Prześlij jeśli less or equal/ not greater
CMOVC	Prześlij jeśli carry
CMOVNC	Prześlij jeśli not carry
CMOVO	Prześlij jeśli overflow
CMOVNO	Prześlij jeśli not overflow
CMOVS	Prześlij jeśli sign (negative)
CMOVNS	Prześlij jeśli not sign (non-negative)
CMOVPI/CMOVPE	Prześlij jeśli parity/ parity even
CMOVNPI/CMOVPO	Prześlij jeśli not parity/ parity odd

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 91

Przykład

```
MyMax64 proc
movsxd rax, ecx
movsxd rdx, edx
cmp rax, rdx
cmovl rax, rdx
ret
MyMax64 endp
```

(C) KISI d.KIK PCz 2023

```
function TForm1.MyMax(x,y:integer):integer;
asm
mov eax, x
cmp eax, y
jnc @@exit
mov eax, y
@@exit:
end;

function TForm1.MyMax2(x,y:integer):integer;
asm
mov eax, x
cmp eax, y
cmovc eax, y ;cmovb eax,y
end;
```

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 92

Wplywa na flagi: -

Skoki warunkowe Jcc

JEJZ	Skocz jeśli equalzero
JNEJNZ	Skocz jeśli not equalnot zero
JAEJNBE	Skocz jeśli above/not below or equal
JAEJNB	Skocz jeśli above or equalnot below
JBEJNAE	Skocz jeśli below/not above or equal
JBEJNA	Skocz jeśli below or equalnot above
JGJNLE	Skocz jeśli greater/not less or equal
JGEJNL	Skocz jeśli greater or equalnot less
JLJNGE	Skocz jeśli less/not greater or equal
JLEJNG	Skocz jeśli less or equalnot greater
JC	Skocz jeśli carry
JNC	Skocz jeśli not carry
JO	Skocz jeśli overflow
JNO	Skocz jeśli not overflow
JS	Skocz jeśli sign (negative)
JNS	Skocz jeśli not sign (non-negative)
JPEJNP	Skocz jeśli parity oddnot parity
JPEJP	Skocz jeśli parity evenparity

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 93

Instrukcja JZ

jz przesunięcie
Przeskakuje do podanej etykiety (adres jest względny 16/32bitowy).
EIP := EIP + przesunięcie

```
jz dalej
...
dalej: ...
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 94

Przykład

Sortowanie

```
procedure MySort1(t:array of integer;n:integer);
asm
push edi ;zabezpiecz rejestry
push esi
mov esi, n ;liczba
dec esi ;esi ostatni element
mov edx, t ;adres tablicy
@p: mov edi, esi
dec edi ;poprzedzający element
mov eax, [edx+esi*4] ;ostatni do eax
@w: cmp eax, [edx+edi*4] ;czy >=
jae @@a
xchg eax, [edx+edi*4] ;zamień elementy
mov [edx+esi*4], eax
@@a: dec edi ;pętla wewnętrzna
jns @w
dec esi ;pętla główna
jnz @p
pop esi ;przywróć rejestry
pop edi
end;
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 95

Instrukcje sterujące przebiegiem programu +

JMP	Skok bezwarunkowy
JCZ/JECXZ/JRCX	Skok jeśli zero w rejestrze CX/ECX/RCX
LOOP	Pętla z licznikiem CX/ECX/RCX
LOOPZ/LOOPE	Pętla z licznikiem CX/ECX/RCX i zero/equal
LOOPNZ/LOOPNE	Pętla z licznikiem CX/ECX/RCX i not zero/not equal
CALL	Wywołanie podprogramu
RET	Powrót z podprogramu sprawdzenie ograniczeń indeksu tablicy
ENTER	Powrót z podprogramu sprawdzenie ograniczeń utworzenia ramy stosu
LEAVE	Wywołanie podprogramu – usunięcie ramy stosu

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE 96

Wpływa na flagi: -

Instrukcja JMP

jmp adres

Przeskakuje do podanej etykiety (adres jest w zględny 8/16/32bitowy lub bezwzględnny).

EIP := EIP + przesunięcie(adres)

CS := segment(adres); EIP := EIP + przesunięcie(adres)

jmp dalej
jmp eax
jmp [esi]
jmp lib1:dalej1

(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 97

Wpływa na flagi: -

Instrukcja JCXZ/JECXZ/JRCXZ

JCXZ/JECXZ/JRCXZ przesunięcie

Skok jeśli zero w rejestrze CX/ECX/RCX do podanej etykiety (adres jest w zględny 16/32/64 bitowy).

EIP := EIP + przesunięcie

petla: ...
jeczx dalej
...
jmp petla

dalej: ...

(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 98

Wpływa na flagi: -

Instrukcja LOOP

loop przesunięcie

Pętla z licznikiem CX/ECX/RCX. Zmniejsza CX/ECX/RCX o 1 i jeśli nie uzyskano zera przeskakuje do podanej etykiety (adres jest w zględny 8 bitowy).

EIP := EIP + przesunięcie(adres)

petla: ...
...
loop petla

(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 99

Przykład

Sinia

```
function sinia(n:integer):integer;
asm
  mov  ecx, eax
  dec  ecx
  @:p: imul eax, ecx
  dec  ecx
  jz   @:p
end;

function sinia1(n:integer):int eger;
asm
  mov  ecx, eax
  @:p: dec  ecx
  jecz @:e
  imul eax, ecx
  jmp  @:p
  @:e:
end;

function sinia2(n:integer):int eger;
asm
  mov  ecx, eax
  dec  ecx
  @:p: imul eax, ecx
  loop @:p
end;
```

(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 100

Wpływa na flagi: -

Przykład

```
for_petla: xor    rcx,rcx
           cmp    rcx,n
           jnb   for_end
           ...
           ...
for_cort:  inc    rcx
           jmp   for_petla
for_end:
```

```
for(int i=0;i<n;i++)
{
  ...
}
```

```
for_petla: xor    rcx,rcx
           ...
           ...
for_cort:  inc    rcx
           cmp    rcx,n
           jb    for_petla
for_end:
```

```
for_petla: mov    rcx,n
           ...
           ...
for_cort:  dec    rcx
           jnz   for_petla
for_end:
```

(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 101

Instrukcja LOOPZ/LOOPE

loopz/loope przesunięcie

Pętla z licznikiem CX/ECX/RCX i zero/equal. Zmniejsza CX/ECX/RCX o 1 i jeśli nie uzyskano zera w CX/ECX/RCX i flaga ZF=1 przeskakuje do podanej etykiety (adres jest w zględny 8 bitowy).

EIP := EIP + przesunięcie(adres)

petla: ...
...
loopz petla

(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 102

Wpływa na flagi: -

Instrukcja LOOPNZ/LOOPNE

loopnz/loopne przesunięcie

Pętla z licznikiem CX/ECX/RCX i nie zero/equal. Zmniejsza CX/ECX/RCX o 1 i jeśli nie uzyskano zera i flaga ZF=0 przeskakuje do podanej etykiety (adres jest względny 8 bitowy).

EIP := EIP + przesunięcie(adres)

```

petla: ...
...
loopnz petla

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 103

Wpływa na flagi: -

Instrukcja CALL

call adres

Instrukcja call wywołuje podprogram, wysyła na stos adres powrotu EIP|RIP lub CS:EIP|RIP. Parametr adres wpisuje do EIP|RIP lub CS:EIP|RIP.

push e(r)ip; (push cs);
E(R)IP := przesunięcie adres; (CS := selektor segmentu)

```

call procedura
call eax
call [esi]

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 104

Wpływa na flagi: -

Instrukcja RET

ret (ile)

Instrukcja ret wraca z podprogramu, pobiera ze stosu adres powrotu do EIP|RIP lub CS:EIP|RIP. Jeśli posiada parametr ile, to dodatkowo usuwa ze stosu ile bajtów (niepotrzebne już parametry aktualne wywołania).

pop e(r)ip; (pop cs); (e(r)sp:=e(r)sp+ile)

```

ret
ret 6

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 105

Przykład

Sinia

```

function sinia3(n:integer):int eger;
asm
and     eax, eax
cmovz  eax, one
jz      @e
push   eax
dec    eax
call   sinia3
pop    edx
imul  eax, edx
@e:
entz

one db 1,0,0,0,0,0,0,0
sinia4 proc;
cmp    rcx, 1
cmovbe rax, one
jbe    @e
push  rcx
dec   rcx
call  sinia4
pop   rcx
imul rax, rcx
@e: ret
sinia4 endp;

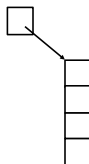
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 106

Przykład

Tablice - wektory



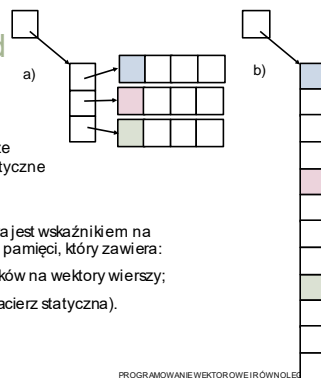
Zmienna tablicowa jest wskaźnikiem na początek obszaru pamięci, który zawiera tablicę.

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 107

Przykład

Tablice – Macierze Dynamiczne i statyczne



Zmienna tablicowa jest wskaźnikiem na początek obszaru pamięci, który zawiera:
a) wektor wskaźników na wektory wierszy;
b) całą tablicę (macierz statyczna).

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 108

Wpływa na flagi: -

Przykład

Oblicz sumę elementów tablicy typu int (32)

```

;rcx – wskaźnik do macierzy dynamicznej
;rdx – liczba wierszy
;r8 – liczba kolumn
suma_et proc;
xor rax, rax ;suma=0
@pz: mov r9, r8 ;liczba kolumn
mov r10, [rcx+8*rdx-8];adr. wiersza
@pw: movsxd r11, [r10+4*r9-4]
add rax, r11
dec r9 ;liczba kolumn--
jnz @pw
dec rdx ;liczba wierszy--
jnz @pz
ret
suma_et endp;

```

Instrukcja ENTER

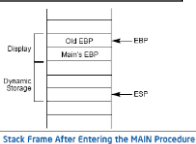
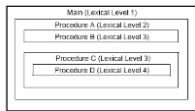
```

enter storage, level
Tworzy ramę stosu w podprogramie z uwzględnieniem poziomu zagnieżdżenia
podprogramów lokalnych (level) i rozmiaru w bajtach zmiennych lokalnych (storage). Na
stosie umieszcza wskaźniki ram stosu; podprogramu wywołującego, wszystkich poziomów
nadrzędnych i własny.

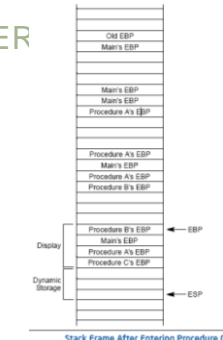
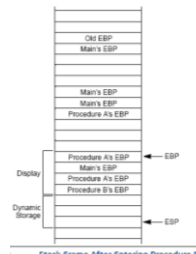
PUSH EBP;
FRAME_PTR ← ESP;
IF LEVEL > 0 THEN
DO (LEVEL - 1) times
EBP ← EBP - 4;
PUSH Pointer(EBP); (* doubleword wskazywane przez EBP *)
OD;
PUSH FRAME_PTR;
FI;
EBP ← FRAME_PTR;
ESP ← ESP + STORAGE;

```

Instrukcja ENTER



Instrukcja ENTER



Wpływa na flagi: -

Instrukcja LEAVE

```
leave
```

Usuwa ramę stosu (utworzoną instrukcją ENTER) przed wyjściem z podprogramu. Przypisuje do wskaźnika stosu rejestr bazowy, następnie zdejmuję rej. basowy ze stosu.

```
esp:=ebp; pop ebp
```

```
leave
```

Przykład

Sortowanie

```

procedure Sort(var A: array of Integer);
procedure QuickSort(var A: array of Integer; Lo, Hi: Integer);
var
Lo, Hi, Mid, T: Integer;
begin
Lo := Lo;
Hi := Hi;
Mid := A[(Lo + Hi) div 2];
repeat
while A[Lo] < Mid do Inc(Lo);
while A[Hi] > Mid do Dec(Hi);
if Lo <= Hi then
begin
T := A[Lo];
A[Lo] := A[Hi];
A[Hi] := T;
Inc(Lo);
Dec(Hi);
end;
until Lo > Hi;
if Hi > Lo then QuickSort(A, Lo, Hi);
if Lo < Hi then QuickSort(A, Lo, Hi);
end;
begin
QuickSort(A, Low(A), High(A));
end;

```

```

procedure mysortq(var Aarray of Integer;Lo,Hi:Integer);
asm
  push edi
  push esi
  push lo
  push hi
  call @s
  jmp @e

@b: enter 0,0 /abortowanie eax=<A[ebp+6]-hi;
[ebp+12]=lo
mov esi[ebp+8] /esi=hi-hi
mov edi[ebp+12] /edi=lo-lo
mov ecx,esi /ecx=Md > A[Lo + Hi] div 2
addi ecx,edi
sar ecx,1
mov ecx,[eax+ecx*4] /mid
@r: /repeat
  cmp [eax+edi*4]jcx /while A[Lo] < Md do hc(Lo)
  jge @1
  inc edi
  jmp @r
@1: cmp [eax+esi*4]jcx /while A[Hi] > Md do Dec(Hi)
  je @2
  dec esi
  jmp @1
@2: cmp edi,esi /if Lo <= Hi then
  jle @3
  mov edx,[eax+esi*4] /A[Lo] <-> A[Hi]

```

(C) KISI d.KIK PCz 2023

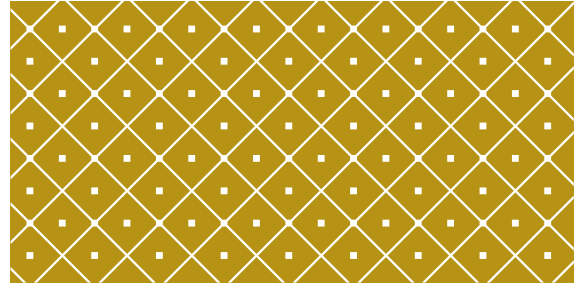
```

  xchg edx,[eax+edi*4]
  mov [eax+esi*4]edx
  inc edi
  dec esi
  @3: cmp edi,esi /until Lo > Hi;
  je @r
  cmp esi[ebp+12] /if Hi > Lo then QuickSort(A, Lo, Hi)
  jng @4
  push edi /zabezpieczenie lo
  push [ebp+12]
  push esi
  call @s
  pop edi
  @4: cmp edi[ebp+8] /if Lo < Hi then QuickSort(A, Lo, Hi)
  jle @5
  push edi
  push [ebp+8]
  call @s
  @5: leave
  ret 8

@e: pop esi
  pop edi
end.

```

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 115



OPERACJE NA ZNACZNIKACH, BITACH I BAJTACH

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 116

Rejestr flag



Rejestr	flag	w architekturze	Intel	x86
0	CF	znacznik przeniesienia	(carry)	S
2	PF	znacznik parzystości	(parity)	S
4	DF	znacznik kierunku	(direction)	S
6	ZF	znacznik zera	(zero)	S
7	SF	znacznik znaku	(sign)	S
10	DF	znacznik kierunku	(direction)	C
11	OF	znacznik przepełnienia	(overflow)	S

S: Znacznik stanu
 C: Znacznik kontroli
 X: Znacznik systemowy
 (C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 117

Operacje na flagach

- STC Ustawienie CF
- CLC Zerowanie CF
- CMC Zanegowanie CF
- CLD Zerowanie DF – flagi kierunku
- STD Ustawienie DF
- LAHF Przesłanie flag do rejestru AH
- SAHF Przesłanie rejestru AH do flag
- PUSHF/PUSHFD/ Wysłanie flag na stos
- PUSHFQ
- POPF/POPCD/POPFQ Pobranie flag ze stosu

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 118

Wpływ na na flagi: C

Wpływ na na flagi: C

Instrukcja STC

stc
 Ustawienie flagi CF.
 CF := 1

stc

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 119

Instrukcja CLC

clc
 Zerowanie flagi CF.
 CF := 0

clc

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 120

Wpływa na flagi: C

Instrukcja CMC

cmc

Zanegowanie flagi CF.

CF := not CF

cmc

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 121

Wpływa na flagi: D

Instrukcja STD

std

Ustawienie flagi kierunku DF. Jeżeli DF=1 instrukcje tańcuchowe zmniejszają rejestr ESI lub EDI.

DF := 1

std

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 122

Wpływa na flagi: D

Instrukcja CLD

cld

Zerowanie flagi kierunku DF. Jeżeli DF=0 instrukcje tańcuchowe zwiększają rejestr ESI lub EDI.

DF := 0

cld

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 123

Wpływa na flagi: -

Instrukcja LAHF

lahf

Przesłanie flag do rejestru AH

AH := lo(FLAGS)

lahf

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 124

Wpływa na flagi: SZAPC

Instrukcja SAHF

sahf

Przesłanie rejestru AH do flag. Bity 1,3,5 są ignorowane.

lo(FLAGS) := AH

sahf

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 125

Wpływa na flagi: -

Instrukcja PUSHF/PUSHFD/PUSHFQ

pushf/pushfd/pushfq

Przesyła zawartość Flag/Eflag/Rflag na stos.

pushf

pushfd

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 126

Wpływa na flagi: OSZAPC

Instrukcja POPF/POPFD/POPFQ

popf/popfd/popfq

Pobiera zawartość Flag/EFlag ze stosu.

popf
popfd

(C) KISI d.KIK PCz 2023

PROGRAMOWANIEWEKTOROWEIRÓWNOLEGLE 127

Operacje na bitach

BT	Testowanie bitu
BTS	Testowanie bitu z ustawianiem
BTR	Testowanie bitu z zerowaniem
BTC	Testowanie bitu z negacją
TEST	Porównanie logiczne
BSF	Przeszukiwanie bitów w przód
BSR	Przeszukiwanie bitów wstecz
LZCNT	Zlicza zerowe bity od najstarszego
TZCNT	Zlicza zerowe bity od najmłodszego
BEXTR	Wycina ciąg bitów
BLSI	Kopiuje najmłodszy ustawiony bit
BLSR	Zeruje najmłodszy ustawiony bit
BLSMSK	Tworzy maskę do bitu=0
BZHI	Zeruje starsze bity

(C) KISI d.KIK PCz 2023

PROGRAMOWANIEWEKTOROWEIRÓWNOLEGLE 128

Wpływa na flagi: OSZAPC
xxxxxC

Instrukcja BT

bt baza, nr

Wyznacza wartość bitu nr (rejestr lub wartość) w bazie (rejestr lub zmienna) i umieszcza ją w CF.

CF := bit bazy numer nr

bt zmienna, eax
bt edx, 12
bt rcx, 37

(C) KISI d.KIK PCz 2023

PROGRAMOWANIEWEKTOROWEIRÓWNOLEGLE 129

Wpływa na flagi: OSZAPC
xxxxxC

Instrukcja BTS

bts baza, nr

Wyznacza wartość bitu nr (rejestr lub wartość) w bazie (rejestr lub zmienna) i umieszcza ją w CF. Następnie ustawia badany bit.

CF := bit bazy numer nr

bit bazy numer nr:=1

bts zmienna, eax
bts edx, 12
bts rcx, 37

(C) KISI d.KIK PCz 2023

PROGRAMOWANIEWEKTOROWEIRÓWNOLEGLE 130

Wpływa na flagi: OSZAPC
xxxxxC

Instrukcja BTR

btr baza, nr

Wyznacza wartość bitu nr (rejestr lub wartość) w bazie (rejestr lub zmienna) i umieszcza ją w CF. Następnie zeruje badany bit.

CF := bit bazy numer nr

bit bazy numer nr := 0

btr zmienna, eax
btr edx, 12
btr rcx, 37

(C) KISI d.KIK PCz 2023

PROGRAMOWANIEWEKTOROWEIRÓWNOLEGLE 131

Wpływa na flagi: OSZAPC
xxxxxC

Instrukcja BTC

btc baza, nr

Wyznacza wartość bitu nr (rejestr lub wartość) w bazie (rejestr lub zmienna) i umieszcza ją w CF. Następnie neguje badany bit.

CF := bit bazy numer nr

bit bazy numer nr := not bit bazy numer nr

btc zmienna, eax
btc edx, 12
btc rcx, 37

(C) KISI d.KIK PCz 2023

PROGRAMOWANIEWEKTOROWEIRÓWNOLEGLE 132

Wpływa na flagi: OSZAPC
OSZxP0

Instrukcja TEST

test cel, źródło

Wyznacza iloczyn logiczny (bit po bicie) zawartości celu i źródła (rejestr lub wartość), wynik jest pominięty, ustawia flagi.

cel and źródło

```
test  eax, zmienna
test  edx, [ebx+esi*4]
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 133

Wpływa na flagi: OSZAPC
xxZxxx

Instrukcja BSF

bsf cel, źródło

Przeszukiwanie bitów w przód. Szuka w rejestrze lub zmiennej źródła najmłodszego bitu=1, jego indeks umieszcza w rejestrze celu (ZF = 0). Jeśli źródło = 0, wówczas ZF = 1, a cel jest niezdefiniowany

cel := indeks najmłodszego bitu = 1 źródła

```
bsf  eax, zmienna
bsf  edx, esi
bsf  rcx, rdx
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 134

Wpływa na flagi: OSZAPC
xxZxxx

Instrukcja BSR

bsr cel, źródło

Przeszukiwanie bitów wstecz. Szuka w rejestrze lub zmiennej źródła najstarszego bitu=1, jego indeks umieszcza w rejestrze celu (ZF = 0). Jeśli źródło = 0, wówczas ZF = 1, a cel jest niezdefiniowany

cel := indeks najstarszego bitu = 1 źródła

```
bsr  eax, zmienna
bsr  edx, esi
bsr  rcx, rdx
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 135

Wpływa na flagi: OSZAPC
xxZxxx
Wymaga LZCNT

Instrukcja LZCNT

lzcnt cel, źródło

Zlicza starsze (wiodące) zerowe bity źródła (16|32|64) i ilość zapisuje do rejestru celu. Dla celu = 0 ZF = 1. Dla celu = rozmiarowi źródła CF = 1.

cel := liczba wiodących zer w źródle

```
lzcnt eax, zmienna
lzcnt edx, esi
lzcnt rcx, rdx
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 136

Wpływa na flagi: OSZAPC
xxZxxx
Wymaga BMI1

Instrukcja TZCNT

tzcnt cel, źródło

Zlicza od najmłodszego zerowe bity źródła (16|32|64) i ilość zapisuje do rejestru celu. Dla celu = 0 ZF = 1. Dla celu = rozmiarowi źródła CF = 1.

cel := liczba końcowych zer w źródle

```
tzcnt eax, zmienna
tzcnt edx, esi
tzcnt rcx, rdx
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 137

Wpływa na flagi: OSZAPC
0xZxxx
Wymaga BMI1

Instrukcja BEXTR

bextr cel, źródło, st_ile

Wycina z rejestru | zmiennej źródła (32|64) ciąg bitów i umieszcza w rejestrze celu. Początkowy bit określa rejestr st_ile[7:0], a ilość bitów st_ile[15:8]. Jeśli cel = 0, wówczas ZF = 1.

cel := źródło[start + ile - 1: start]

```
bextr eax, zmienna, edx
bextr edx, esi, eax
bextr rcx, rdx, rax
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 138

Wpływa na flagi: OSZAPC
OSZxxC
Wymaga BMI1

Instrukcja BLSI

blsi cel, źródło

Izoluje z rejestru lub zmiennej źródła najmłodszy bit=1 i umieszcza w rejestrze celu (CF = 1). Zeruje pozostałe bity. Jeśli źródło = 0, wówczas CF = 0, a cel = 0.

cel := (źródło) and źródło

blsi eax, zmienna

blsi edx, esi

blsi rcx, rdx

(C) KISI d.KIK PCz 2023

PROGRAMOWANIEWEKTOROWEIRÓWNOLEGŁE 139

Wpływa na flagi: OSZAPC
OSZxxC
Wymaga BMI1

Instrukcja BLSR

blsr cel, źródło

Kopiuje bity z rejestru lub zmiennej źródła (32|64) i umieszcza w rejestrze celu, zeruje najmłodszy bit = 1 (CF = 0). Jeśli źródło = 0, wówczas CF = 1, a cel = 0.

cel := (źródło - 1) and źródło

blsr eax, zmienna

blsr edx, esi

blsr rcx, rdx

(C) KISI d.KIK PCz 2023

PROGRAMOWANIEWEKTOROWEIRÓWNOLEGŁE 140

Wpływa na flagi: OSZAPC
OS0xxC
Wymaga BMI1

Instrukcja BLSMSK

blsmsk cel, źródło

Ustawia młodsze bity rejestru celu (32|64) na 1 aż do numeru najmłodszego bitu=1 z rejestru lub zmiennej źródła włącznie (CF = 0). Zeruje pozostałe bity. Jeśli źródło = 0, wówczas CF = 1, a cel = not 0.

cel := (źródło - 1) xor źródło

blsmsk eax, zmienna

blsmsk edx, esi

blsmsk rcx, rdx

(C) KISI d.KIK PCz 2023

PROGRAMOWANIEWEKTOROWEIRÓWNOLEGŁE 141

Wpływa na flagi: OSZAPC
OSZxxC
Wymaga BMI2

Instrukcja BZHI

bzhi cel, źródło, idx

Kopiuje bity z rejestru lub zmiennej źródła (32|64) do rejestru celu (32|64) i kasuje starsze bity od numeru z rejestru idx (CF = 0). Jeśli idx > 31|63, wówczas CF = 1.

cel := źródło; cel[rozmiar - 1: idx] = 0

bzhi eax, zmienna, edx

bzhi edx, esi, eax

bzhi rcx, rdx, rax

(C) KISI d.KIK PCz 2023

PROGRAMOWANIEWEKTOROWEIRÓWNOLEGŁE 142

Wpływa na flagi: -

Instrukcja SETcc

SETcc cel

Jeśli jest spełniony warunekcc, ustawia bajt na 1, w przeciwnym wypadku na 0.

```
if cc then cel := 1
else cel := 0
```

sets al

setge [esi+8]

(C) KISI d.KIK PCz 2023

PROGRAMOWANIEWEKTOROWEIRÓWNOLEGŁE 143

Instrukcje SETcc

SETE/SETZ	Ustaw bajt jeśli equal/ zero
SETNE/SETNZ	Ustaw bajt jeśli not equal/ not zero
SETS	Ustaw bajt jeśli sign (negative)
SETNS	Ustaw bajt jeśli not sign (non-negative)
SETO	Ustaw bajt jeśli overflow
SETNO	Ustaw bajt jeśli not overflow
SETPE/SETP	Ustaw bajt jeśli parity even/ parity
SETPO/SETNP	Ustaw bajt jeśli parity odd/ not parity
SETA/SETNBE	Ustaw bajt jeśli above/ not below or equal
SETAE/SETNB/SETNC	Ustaw bajt jeśli above or equal/ not below/ not carry
SETB/SETNAE/SETC	Ustaw bajt jeśli below/ not above or equal/ carry
SETBE/SETNA	Ustaw bajt jeśli below or equal/ not above
SETG/SETNLE	Ustaw bajt jeśli greater/ not less or equal
SETGE/SETNL	Ustaw bajt jeśli greater or equal/ not less
SETL/SETNGE	Ustaw bajt jeśli less/ not greater or equal
SETLE/SETNG	Ustaw bajt jeśli less or equal/ not greater

(C) KISI d.KIK PCz 2023

PROGRAMOWANIEWEKTOROWEIRÓWNOLEGŁE 144

Wpływa na flagi: -

Przykład – int na bin(string)

```

procedure s2b(var s:string; i:integer);
asm
    push ebx
    bsr edx,ecx //w edx nr najstarszej 1
    jnz @1
    mov word ptr [eax],$3001
    jmp @e
@1: inc edx
    mov [eax],dl //długość
    dec edx
    inc eax
@p: bt ecx,edx //testuj
    setc bl
    add bl,$30
    mov [eax],bl //zapisz znak
    inc eax
    dec edx
    jns @p
@e: pop ebx
end;

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 145

Instrukcja RDPID

rdpid cel

Czyta 32-bitowy identyfikator procesora do rejestru celu.

cel=PID

rdpid eax

rdpid rdx

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 146

Wpływa na flagi: -

Instrukcja RDTSC

rdtsc

Read Time Stamp Counter. Czyta 64-bitowy licznik do rejestrów EDX: EAX.

EDX: EAX := licznik

rdtsc

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 147

Przykład – funkcja pomiaru czasu

function Pomiar(a:integer):integer;

var Cykle_H,Cykle_L:integer;

```

asm
    rdtsc
    mov Cykle_H,edx
    mov Cykle_L,eax
    ...
    ...
    rdtsc
    sub eax,Cykle_L
    sbb edx,Cykle_H
    sub EAX,9 ; odliczenie 9?
end;

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 148

Wpływa na flagi: 00000C

Przykład – funkcja random

```

function MyRandom0(a:integer):integer; overload;
asm
    push ebx
    xor ebx,ebx
    imul edx,[ebx+MySeed],$08088405
    inc edx
    mov [ebx+MySeed],edx
    mul edx
    mov eax,edx
    pop ebx
end;

function MyRandom1(a:integer):integer; overload;
asm
    push ebx
    rdtsc
    bswap eax
    pop edx
    mul edx
    mov eax,edx
end;

```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 149

Instrukcja RDRAND

rdrand cel

Czyta 16|32|64-bitową liczbę losową (deterministic random bit generator) do rejestru celu wg normy NIST SP 800-90A. Jeśli CF = 1 wartość jest prawidłowa.

rdrand rax

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 150

Wpływa na flagi: 00000C

Instrukcja RDSEED

rdseed cel

Czyta 32|64-bitową liczbę losową (non-deterministic random bit generator) do rejestru celu wg norm NIST SP 800-90B i NIST SP800-90C. Jeśli CF = 1 wartość jest prawidłowa.

rdseed rax

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 151

Przykład

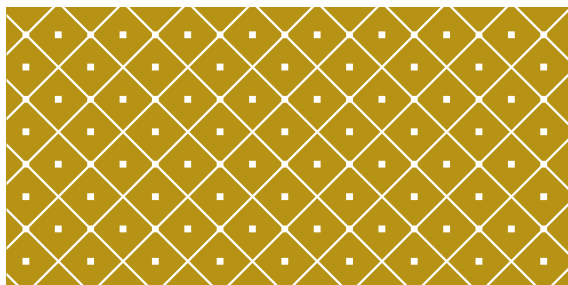
```

;rcx – wskaźnik do liczbyin64
random proc;
xor rax, rax ;nieudane
mov rdx, 10 ;liczba powtórzeń
@p: rdseed r8 ;czytaj liczbę
jc @ok
dec rdx ;licznik-
jnz @pz
ret
@ok: mov [rcx], r8 ;zapisz wartość
inc rax ;udane
ret
random endp;

```

(C) KISI d.KIK PCz 2023

Programowanie wektorowe i równoległe 152



OPERACJE NA ŁAŃCUCHACH

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 153

Operacje na łańcuchach

MOVS/MOVSb/MOVSW/MOVSd/MOVSQ Prześlij łańcuch/bajtów/słów/podwójnych słów/poczwońnych słów

CMPS/CMPSb/CMPSW/CMPSd/CMPSQ Porównaj łańcuch/bajtów/słów/podwójnych słów/poczwońnych słów

SCAS/SCASb/SCASW/SCASd/SCASQ Skanuj łańcuch/bajtów/słów/podwójnych słów/poczwońnych słów

LODS/LODSb/LODSW/LODSd/LODSQ Ładuj łańcuch/bajtów/słów/podwójnych słów/poczwońnych słów

STOS/STOSb/STOSW/STOSd/STOSQ Zapamiętaj łańcuch/bajtów/słów/podwójnych słów/poczwońnych słów

REP Powtarzaj dopóki ECX nie jest zerem

REPE/REPZ Powtarzaj dopóki equal/zero

REPNE/REPZ Powtarzaj dopóki not equal/not zero

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 154

Wpływa na flagi: -

Instrukcja MOVS/MOVSb

movs byte ptr [(r)e]di, [(r)e]si
movsb

Przesyła bajt z pamięci ds:(r)e:si do pamięci es:(r)e:di. Rejestry (r)e:di/(r)e:si są zwiększane/zmniejszane o 1 w zależności od flagi DF (0/1).

[(r)e]s:edi := [ds:(r)e]si
(r)e]di := (r)e]di ±1
(r)e]si := (r)e]si ±1

movsb

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 155

Wpływa na flagi: -

Instrukcja MOVS/MOVSW

movs word ptr [(r)e]di, [(r)e]si
movsw

Przesyła słowo z pamięci ds:(r)e:si do pamięci es:(r)e:di. Rejestry (r)e:di/(r)e:si są zwiększane/zmniejszane o 2 w zależności od flagi DF (0/1).

[es:(r)e]di := [ds:(r)e]si
(r)e]di := (r)e]di ±2
(r)e]si := (r)e]si ±2

movsw

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 156

Wpływa na flagi: -

Instrukcja MOVS/MOVSQ

```
movs  dword ptr [(r)e]di, [(r)e]si
movsq
```

Przesyła podwójne słowo z pamięci ds:(r)e)si do pamięci es:(r)e)di. Rejestry (r)e)di/(r)e)si są zwiększane/zmniejszane o 4 w zależności od flagi DF (0/1).

```
[es:(r)e]di := [ds:(r)e]si
(r)e)di := (r)e)di ±4
(r)e)si := (r)e)si ±4
```

```
movsd
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 157

Wpływa na flagi: -

Instrukcja MOVS/MOVSQ

```
movs  qword ptr [(r)e]di, [(r)e]si
movsq
```

Przesyła poczwójne słowo z pamięci ds:(r)e)si do pamięci es:(r)e)di. Rejestry (r)e)di/(r)e)si są zwiększane/zmniejszane o 8 w zależności od flagi DF (0/1).

```
[es:(r)e]di := [ds:(r)e]si
(r)e)di := (r)e)di ±8
(r)e)si := (r)e)si ±8
```

```
movsq
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 158

Wpływa na flagi: OSZAPC

Instrukcja CMPS/CMPSB

```
cmps  byte ptr [(r)e]si, [(r)e]di
cmpsb
```

Porównuje bajt z pamięci ds:(r)e)si i z pamięci es:(r)e)di. Rejestry (r)e)di/(r)e)si są zwiększane/zmniejszane o 1 w zależności od flagi DF (0/1).

```
[ds:(r)e]si - [es:(r)e]di
(r)e)di := (r)e)di ±1
(r)e)si := (r)e)si ±1
```

```
cmpsb
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 159

Wpływa na flagi: OSZAPC

Instrukcja CMPS/CMPSW

```
cmps  word ptr [(r)e]si, [(r)e]di
cmpsw
```

Porównuje słowo z pamięci ds:(r)e)si i z pamięci es:(r)e)di. Rejestry (r)e)di/(r)e)si są zwiększane/zmniejszane o 2 w zależności od flagi DF (0/1).

```
[ds:(r)e]si - [es:(r)e]di
(r)e)di := (r)e)di ±2
(r)e)si := (r)e)si ±2
```

```
cmpsw
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 160

Wpływa na flagi: OSZAPC

Instrukcja CMPS/CMPSD

```
cmps  dword ptr [(r)e]si, [(r)e]di
cmpsd
```

Porównuje podwójne słowo z pamięci ds:(r)e)si i z pamięci es:(r)e)di. Rejestry (r)e)di/(r)e)si są zwiększane/zmniejszane o 4 w zależności od flagi DF (0/1).

```
[ds:(r)e]si - [es:(r)e]di
(r)e)di := (r)e)di ±4
(r)e)si := (r)e)si ±4
```

```
cmpsd
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 161

Wpływa na flagi: OSZAPC

Instrukcja CMPS/CMPSQ

```
cmps  qword ptr [(r)e]si, [(r)e]di
cmpsq
```

Porównuje poczwójne słowo z pamięci ds:(r)e)si i z pamięci es:(r)e)di. Rejestry (r)e)di/(r)e)si są zwiększane/zmniejszane o 8 w zależności od flagi DF (0/1).

```
[ds:(r)e]si - [es:(r)e]di
(r)e)di := (r)e)di ±8
(r)e)si := (r)e)si ±8
```

```
cmpsq
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 162

Wpływa na flagi: OSZAPC

Instrukcja SCAS/SCASB

```
scas  byte ptr [(r)e]di
```

```
scasb
```

Porównuje bajt akumulatora AL i pamięci es:(r)e]di. Rejestr (r)e]di jest zwiększany/zmniejszany o 1 w zależności od flagi DF (0/1).

```
AL - [es:(r)e]di
```

```
(r)e]di := (r)e]di ± 1
```

```
scasb
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 163

Wpływa na flagi: OSZAPC

Instrukcja SCAS/SCASW

```
scas  word ptr [(r)e]di
```

```
scasw
```

Porównuje słowo akumulatora AX i pamięci es:(r)e]di. Rejestr (r)e]di jest zwiększany/zmniejszany o 2 w zależności od flagi DF (0/1).

```
AX-[es:(r)e]di
```

```
(r)e]di := (r)e]di ± 2
```

```
scasw
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 164

Wpływa na flagi: OSZAPC

Instrukcja SCAS/SCASD

```
scas  dword ptr [(r)e]di
```

```
scasd
```

Porównuje podwójne słowo akumulatora EAX i pamięci es:(r)e]di. Rejestr (r)e]di jest zwiększany/zmniejszany o 4 w zależności od flagi DF (0/1).

```
EAX - [es:(r)e]di
```

```
(r)e]di := (r)e]di ± 4
```

```
scasd
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 165

Wpływa na flagi: OSZAPC

Instrukcja SCAS/SCASQ

```
scas  qword ptr [(r)e]di
```

```
scasq
```

Porównuje początkowe słowo akumulatora RAX i pamięci es:(r)e]di. Rejestr (r)e]di jest zwiększany/zmniejszany o 8 w zależności od flagi DF (0/1).

```
RAX-[es:(r)e]di
```

```
(r)e]di := (r)e]di ± 8
```

```
scasq
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 166

Wpływa na flagi: -

Instrukcja LODS/LODSB

```
lods  byte ptr [(r)e]si
```

```
lodsb
```

Czyta bajt do akumulatora AL z pamięci ds:(r)e]si. Rejestr (r)e]si jest zwiększany/zmniejszany o 1 w zależności od flagi DF (0/1).

```
AL := [ds:(r)e]si
```

```
(r)e]si := (r)e]si ± 1
```

```
lodsb
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 167

Wpływa na flagi: -

Instrukcja LODS/LODSW

```
lods  word ptr [(r)e]si
```

```
lodsw
```

Czyta słowo do akumulatora AX z pamięci ds:(r)e]si. Rejestr (r)e]si jest zwiększany/zmniejszany o 2 w zależności od flagi DF (0/1).

```
AX = [ds:(r)e]si
```

```
(r)e]si := (r)e]si ± 2
```

```
lodsw
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 168

Wpływa na flagi: -

Instrukcja LODS/LODSD

```
lods  dword ptr [(r)e:si]
lodsd
```

Czyta podwójne słowo do akumulatora EAX z pamięci ds:(r)e:si. Rejestr (r)e:si jest zwiększany/zmniejszany o 4 w zależności od flagi DF (0/1).

```
EAX = [ds:(r)e:si]
(r)e:si := (r)e:si ± 4
```

```
lodsd
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 169

Wpływa na flagi: -

Instrukcja LODS/LODSQ

```
lods  qword ptr [(r)e:si]
lodsq
```

Czyta początkowe słowo do akumulatora RAX z pamięci ds:(r)e:si. Rejestr (r)e:si jest zwiększany/zmniejszany o 8 w zależności od flagi DF (0/1).

```
RAX = [ds:(r)e:si]
(r)e:si := (r)e:si ± 8
```

```
lodsq
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 170

Wpływa na flagi: -

Instrukcja STOS/STOSB

```
stos  byte ptr [(r)e:di]
stosb
```

Zapisuje bajt z akumulatora AL do pamięci es:(r)e:di. Rejestr (r)e:di jest zwiększany/zmniejszany o 1 w zależności od flagi DF (0/1).

```
[es:(r)e:di] := AL
(r)e:di := (r)e:di ± 1
```

```
stosb
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 171

Wpływa na flagi: -

Instrukcja STOS/STOSW

```
stos  word ptr [(r)e:di]
stosw
```

Zapisuje słowo z akumulatora AX do pamięci es:(r)e:di. Rejestr (r)e:di jest zwiększany/zmniejszany o 2 w zależności od flagi DF (0/1).

```
[es:(r)e:di] := AX
(r)e:di := (r)e:di ± 2
```

```
stosw
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 172

Wpływa na flagi: -

Instrukcja STOS/STOSD

```
stos  dword ptr [(r)e:di]
stosd
```

Zapisuje podwójne słowo z akumulatora EAX do pamięci es:(r)e:di. Rejestr (r)e:di jest zwiększany/zmniejszany o 4 w zależności od flagi DF (0/1).

```
[es:(r)e:di] := EAX
(r)e:di := (r)e:di ± 4
```

```
stosd
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 173

Wpływa na flagi: -

Instrukcja STOS/STOSQ

```
stos  qword ptr [(r)e:di]
stosq
```

Zapisuje początkowe słowo z akumulatora RAX do pamięci es:(r)e:di. Rejestr (r)e:di jest zwiększany/zmniejszany o 8 w zależności od flagi DF (0/1).

```
[es:(r)e:di] = RAX
(r)e:di := (r)e:di ± 8
```

```
stosq
```

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 174

Prefiks REP REPZ/REPNE REPZ/REPE

Wpływa na flagi: -

Powoduje powtórzenie (R|E)CX razy następującej po nim instrukcji łańcuchowej, jeśli spełniony jest warunek (repnz powtarza dopóty ZF = 0, jeśli ZF = 1 powtarzanie jest przerywane itd.). Jeżeli (R|E)CX = 0, to instrukcja nie zostanie wykonana.

(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 175

Prefiks REP REPZ/REPNE REPZ/REPE

Wpływa na flagi: -

- rep movsb
- rep lodsd
- rep stosq
- rep_W cmpsw
- rep_W scasb

(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 176

Przykład

```
mov ecx, 100
mov esi, bufor1
mov edi, bufor2
rep movsb
```

Kopiuje zawartość bufora1 do bufora2.

```
mov rax, 0
mov rcx, 100
mov rdi, bufor
rep ds:stosq
```

Zeruje zawartość bufora (800B).

```
mov al, 77
mov ecx, 100
mov edi, bufor
repnz ds:scasb
```

Szuka wartości 77 w buforze. ZF = 1 oznacza znalezienie żądanej wartości.

```
mov al, 0
mov rcx, 100
mov rdi, bufor
rep ds:scasb
```

Szuka wartości <0 w buforze. ZF = 0 oznacza znalezienie żądanej wartości.

(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 177

Inne operacje

- LOCK Powoduje niepodzielne wykonanie następnjej instrukcji
- LEA Ładowanie adresu efektywnego
- NOP Nie wykonuje żadnego działania
- MOVBE Przesłanie po zamianie kolejności bajtów
- CPUID Identyfikacja procesora

(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 178

Prefiks LOCK

Wpływa na flagi: -

lock

Powoduje wstawienie sygnału LOCK procesora i wykonanie w sposób niepodzielny instrukcji:

add, adc, and, brc, btr, bts, cmpxchg, cmpxch8b, cmpxch16b, dec, inc, neg, not, or, sbb, sub, xor, xadd i xchg,

jeśli argument celu jest w pamięci.

lock btr

(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 179

Przykład

```
mov     zu, 1 ;inicjalizacja      mov     zu, 1
...
...
xor     ax, ax ;ax=0
@p:
lock   xchg  zu, ax ;al<->zu      lock   btr   zu, 0 ;zajmij
      test  ax, ax ;czy 0        jnc   @p
      jz   @p ;akt. czekanie
...
...
...
mov     zu, 1 ;uwolnij
...
...
mov     zu, 1 ;oddaj 1 do zu
```

(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 180

Wpływa na flagi: -

Instrukcja LEA

lea cel, źródło

Wczytanie wyznaczonego adresu źródła do rejestru celu.

cel := adres źródła

lea eax, [edx+esi*4+12]; eax = edx + esi * 4 + 12

lea rax, [rdx+rsi*4+12] ; rax = rdx + rsi * 4 + 12

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 181

Przykład

lea rax, [rdx] ;kopiuje rejestr

lea rbx, [rcx + rdx] ;suma rejestrów

lea rdx, [rax + 10] ;suma rejestru i stałej

lea rax, [rax + 1] ;inkrementacja

lea rax, [rbx+8*rsi + 3] ;suma trzech wartości

lea rax, [rax + 4*rax] ;mnożenie przez 5

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 182

Wpływa na flagi: -

Instrukcja NOP

nop

Nic nie robi.

nop

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 183

Wpływa na flagi: -

Instrukcja MOVBE

movbe cel, źródło

Przesłanie po zamianie kolejności bajtów. Jeden z argumentów musi być rejestrem (16, 32, 64).

cel := zamiern(źródło)

movbe eax, zmienna

przed

12	c4	7f	de
----	----	----	----

po

de	7f	c4	12
----	----	----	----

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 184

Wpływa na flagi: -

Rejestr flag

Rejestr	flaga w architekturze Intel x86	Symbol/wartość	typ
0	CF	flaga przeniesienia (carry)	S
1	PF	zaczerniowany	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
8	TF	flaga umożliwiająca krokowe wykonywanie (trap)	X
9	IF	flaga zezwolenia na przerwanie (interrupt enable)	X
10	DF	flaga kierunku (direction)	C
11	OF	flaga przepełnienia (overflow)	S
12, 13	IOPL	poziom uprawnień we/wy (I/O privilege level - od 286)	X
14	NT	nested task flag (od 286)	X
16	RF	flaga wznowienia (resume, od 386)	X
17	VM	flaga trybu Virtual 8086 (od 386)	X
18	AC	alignment check (od 486SX)	X
19	VIF	Virtual interrupt flag (od Pentium)	X
20	VIP	Virtual interrupt pending (od Pentium)	X
21	ID	identification (od Pentium)	X
3, 5, 15, 22-31	0	zaczerniowany	

S: Znacznik stanu
C: Znacznik kontroli
X: Znacznik systemowy

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 185

Instrukcja CPUID

cpuid

Identyfikacja procesora jest możliwa, jeśli bit 21 flaga ID w rejestrze flag może być zmieniana. Na podstawie EAX (czasem też ECX) podaje w EAX, EBX, ECX i EDX różne informacje o procesorze.

cpuid

(C) KISI d.KIK PCz 2023

PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 186

Test ID

```

xor     eax, eax ;0
pushfd
pop     edx
bits   edx, 21 ;ustaw
push   edx
popfd
pushfd
pop     edx
btr    edx, 21 ;skasuj
rcl    eax, 1 ;1b
push   edx
popfd
pushfd
pop     edx
bits   edx, 21 ;ustaw
push   edx
popfd
pushfd
pop     edx
bt     edx, 21 ;sprawdz
rcl    eax, 1 ;101b
cmp    eax, 5
je     cpuidOK

```

(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 187

Przykład

```

mov eax, 0
cpuid

```

Zwraca wartość maksymalną dla cpuid oraz identyfikator producenta:

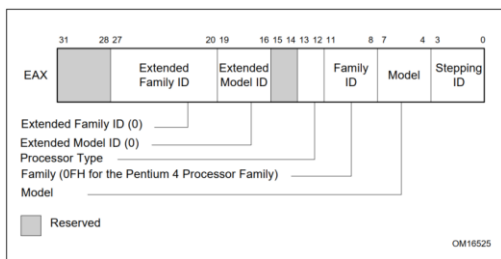
```

eax = max
ebx = 'Genu'
ecx = 'ntel'
edx = 'inel'

```

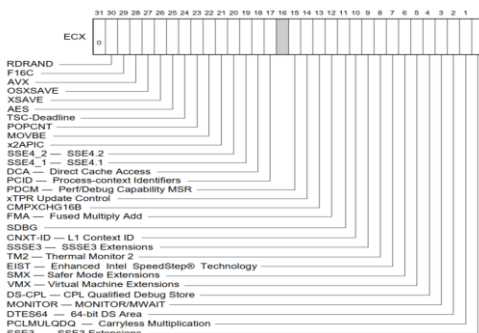
(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 188

CPUID EAX=1



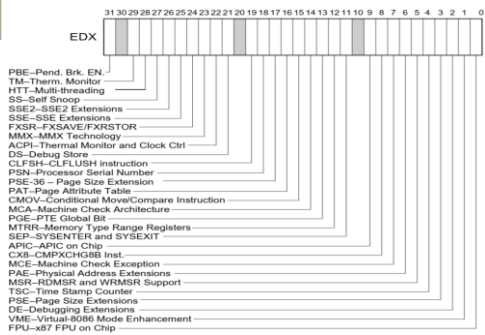
(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 189

CPUID EAX=1



(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 190

CPUID EAX=1



(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 191

Przykład

Sprawdzenie, czy procesor posiada technologię MMX.

```

mov EAX, 1 ; żądanie informacji o właściwościach
CPUID ; 0FH, 0A2H instrukcja CPUID
test EDX, 00800000H ; sprawdzenie bitu technologii MMX (bit 23 w EDX)
jnz ; znaleziono technologię MMX

```

(C) KISI d.KIK PCz 2023 PROGRAMOWANIE WEKTOROWE I RÓWNOLEGLE 192

Wpływa na flagi: -

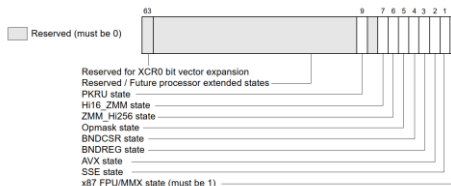
Instrukcja XGETBV

xgetbv

Czyta do edx:eax zawartość rozszerzonego rejestru kontrolnego o indeksie ecx.

xgetbv

XCR0



Przykład - Sprawdzenie, czy procesor posiada technologię AVX.

```

supports_AVX proc
    mov     eax, 1
    cpuid
    and    ecx, 018000000H
    cmp    ecx, 018000000H ; sprawdź flagi OSXSAVE i AVX
    jne    not_sup
; procesor wspiera inst. AVX i XGETBV jest włączone przez OS
    mov    ecx, 0 ; wybierz rejestr XCR0
    XGETBV ; wynik w EDX:EAX
    and    eax, 06H
    cmp    eax, 06H ; czy OS włączył obsługę XMM i YMM
    jne    not_sup
    mov    eax, 1 ; wspiera AVX
    jmp    done
not_sup: mov    eax, 0 ; nie wspiera AVX
done:   ret
endp

```