Instrukcje warunkowe i skoku

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

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

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

G/NLE	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

Wpływa na flagi: -

Instrukcja CMOVcc

CMOVcc cel, źródło

Jeśli jest spełniony warunek cc, 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
cmovge   edx, [ebx+esi*4]
cmovna   rax, rdx
```


Instrukcje CMOVcc

CMOVE/CMOVZ	Prześlij jeżeli equal/ zero
CMOVNE/CMOVNZ	Prześlij jeżeli not equal/ not zero
CMOVA/CMOVNBE	Prześlij jeżeli above/ not below or equal
CMOVAE/CMOVNB	Prześlij jeżeli above or equal/ not below
CMOVB/CMOVNAE	Prześlij jeżeli below/ not above or equal
CMOVBE/CMOVNA	Prześlij jeżeli below or equal/ not above
CMOVG/CMOVNLE	Prześlij jeżeli greater/ not less or equal
CMOVGE/CMOVNL	Prześlij jeżeli greater or equal/ not less
CMOVL/CMOVNGE	Prześlij jeżeli less/ not greater or equal
CMOVLE/CMOVNG	Prześlij jeżeli less or equal/ not greater
CMOVC	Prześlij jeżeli carry
CMOVNC	Prześlij jeżeli not carry
CMOVO	Prześlij jeżeli overflow
CMOVNO	Prześlij jeżeli not overflow
CMOVS	Prześlij jeżeli sign (negative)
CMOVNS	Prześlij jeżeli not sign (non-negative)
CMOVP/CMOVPE	Prześlij jeżeli parity/ parity even
CMOVNP/CMOVPO	Prześlij jeżeli not parity/ parity odd

Przykład

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

```
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;
```


Skoki warunkowe Jcc

- JE/JZ — Skocz jeśli equal/zero
- JNE/JNZ — Skocz jeśli not equal/not zero
- JA/JNBE — Skocz jeśli above/not below or equal
- JAE/JNB — Skocz jeśli above or equal/not below
- JB/JNAE — Skocz jeśli below/not above or equal
- JBE/JNA — Skocz jeśli below or equal/not above
- JG/JNLE — Skocz jeśli greater/not less or equal
- JGE/JNL — Skocz jeśli greater or equal/not less
- JL/JNGE — Skocz jeśli less/not greater or equal
- JLE/JNG — Skocz jeśli less or equal/not 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)
- JPO/JNP — Skocz jeśli parity odd/not parity
- JPE/JP — Skocz jeśli parity even/parity

Wpływa na flagi: -

Instrukcja JZ

jz przesunięcie

Przeskakuje do podanej etykiety (adres jest względny 16/32bitowy).

EIP := EIP + przesunięcie

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


Przykład

Sortowanie

```
procedure MySort1(t:array of integer;n:integer);
asm
        push edi            ;zabezpiecz rejestry
        push esi
        mov esi, n          ;ilość
        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;
```


Instrukcje sterujące przebiegiem programu +

- JMP — Skok bezwarunkowy
- JCXZ/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
- IRET — Powrót z podprogramu obsługi przerwania
- INT — Przerwanie programowe
- INTO — Przerwanie przy przekroczeniu zakresu
- BOUND — sprawdzenie ograniczeń indeksu tablicy
- ENTER — Wysokopoziomowe wejście do podprogramu – utworzenie ramy stosu
- LEAVE — Wysokopoziomowe wyjście z podprogramu – usunięcie ramy stosu

Wpływa na flagi: -

Instrukcja JMP

jmp adres

Przeskakuje do podanej etykiety (adres jest względny 8/16/32bitowy lub bezwzględny).

EIP := EIP + przesunięcie(adres)

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

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

Wpływa na flagi: -

Instrukcja JCXZ/JECXZ/JRCXZ

JCXZ/JECXZ/JRCXZ przesunięcie

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

EIP := EIP + przesunięcie

```
petla: ...
       jecxz  dalej
       ...
       jmp    petla
dalej: ...
```


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 względny 8 bitowy).

EIP := EIP + przesunięcie(adres)

```
petla: ...
       ...
       loop   petla
```


Przykład

Silnia

```
function silnia(n:integer):integer;
asm
  mov   ecx, eax
  dec   ecx
@p: imul eax, ecx
  dec   ecx
  jnz @p
end;
function silnia1(n:integer):integer;
asm
  mov   ecx, eax
@p: dec   ecx
  jecxz  @e
  imul   eax, ecx
  jmp    @p
@e:
end;
function silnia2(n:integer):integer;
asm
  mov   ecx, eax
  dec   ecx
@p: imul eax, ecx
  loop   @p
end;
```


Wpływa na flagi: -

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 względny 8 bitowy).

EIP := EIP + przesunięcie(adres)

```
petla: ...
       ...
       loopz  petla
```


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
```


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:=segment adres)

```
call   procedura
call   eax
call   [esi]
```

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

Przykład

Silnia

```
function silnia3(n:integer):integer;
asm
  and   eax,eax
  cmovz eax,one
  jz    @e
  push  eax
  dec   eax
  call  silnia3
  pop   edx
  imul  eax,edx
@e:
end;

one db  1,0,0,0,0,0,0,0
silnia4 proc;
  cmp   rcx,1
  cmovbe rax, one
  jbe   @e
  push  rcx
  dec   rcx
  call  silnia4
  pop   rcx
  imul  rax, rcx
@e: ret
silnia4 endp;
```


Wpływa na flagi: -

Instrukcja INT

int nr

Wywołuje przerwanie programowe o numerze nr (0-255). Numery z zakresu 0-31 są zarezerwowane. Instrukcja int działa podobnie do instrukcji call, jednak dodatkowo wysyła na stos flagi i wchodząc do podprogramu część z nich zeruje.

int 21h

Przerwania i wyjątki zarezerwowane

Wektor Nr.	Mnemonik	Opis	Źródło
0	#DE	Błąd dzielenia	Instrukcje DIV i IDIV.
1	#DB	Debuger	Dowolne odwołanie do kodu i danych.
2		Przerwanie NMI	Przerwanie niemaskowalne Non-maskable external.
3	#BP	Breakpoint	Instrukcja INT 3.
4	#OF	Overflow	Instrukcja INTO.
5	#BR	BOUND przekroczony zakres	Instrukcja BOUND.
6	#UD	Błędny kod	Instrukcja UD2 lub zarezerwowany kod.1
7	#NM	Urządzenie nie dostępne (Brak koprocesora)	Instrukcje zmiennoprzecinkowe lub WAIT/FWAIT.
8	#DF	Błąd Double	Dowolna instrukcja generująca wyjątek, NMI lub INTR.
9	#MF	CoProcessor Segment Overrun (reserved)	Instrukcje zmiennoprzecinkowe.2
10	#TS	Błędny TSS	Przełączanie zadań lub dostęp do TSS.
11	#NP	Segment nieobecny	Ładowanie rejestrów segmentowych lub dostęp do segmentów systemowych.
12	#SS	Segment stosu uszkodzony	Operacje na stosie I ładowanie rejestru SS.
13	#GP	Ogólna ochrona	Dowolne odwołanie do pamięci i inne zabezpieczenia.
14	#PF	Błąd strony	Dowolne odwołanie do pamięci.
15		Zarezerwowane	
16	#MF	Błąd zmiennoprzecinkowy (błąd matematyczny)	Instrukcje zmiennoprzecinkowe lub WAIT/FWAIT.
17	#AC	Kontrola wyrównania	Dowolne odwołanie do danych w pamięci.3
18	#MC	Kontrola maszyny	Kod błędu (o ile występuje) i źródło zależą od modelu.4
19	#XM	Wyjątek SIMD	Instrukcje zmiennoprzecinkowe SIMDs
20-31		Zarezerwowane	
32-255		Przerwania maskowalne	Przerwania zewnętrzne INTR lub instrucje INT *n*.

1. Instrukcja UD2 została wprowadzona w procesorze Pentium Pro. 2. Procesory IA-32 po procesorze Intel386 nie generują tego wyjątku. 3. Wyjątek wprowadzony w procesorze Intel486. 4. Wyjątek wprowadzony w procesorze Pentium i poprawiony w procesorach rodziny P6. 5. Wyjątek wprowadzony w procesorze Pentium III.

Wpływa na flagi: -

Instrukcja INTO

into

Wywołuje przerwanie programowe (4) w przypadku ustawienia flagi OF (nadmiaru).

into

Wpływa na flagi: -

Instrukcja IRET/IRETD/IRETQ

iret/iretd/iretq

Instrukcja iret/iretd/iretq wraca z podprogramu obsługi przerwania, pobiera ze stosu adres powrotu do CS:EIP|RIP oraz flagi zachowane przy wywołaniu przerwania.

pop e(r)ip; pop cs; pop (e(r)flags)

iret

Wpływa na flagi: -

Instrukcja BOUND

bound idx, gr

Sprawdza, czy indeks tablicy (wartość 16/32 bitowa ze znakiem) zawarty w rejestrze idx nie przekracza jej granic określonych przez strukturę w pamięci gr złożoną z granicy dolnej i górnej. W przypadku przekroczenia granicy generowany jest wyjątek przekroczenia granicy tablicy.

bound eax, granice1

Wpływa na flagi: -

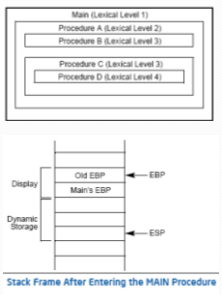
Instrukcja ENTER

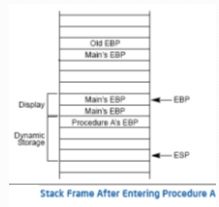
enter storage, level

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.

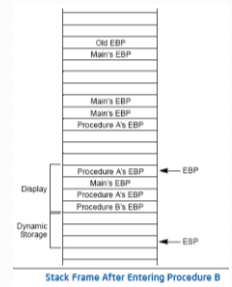
```
PUSH EBP;
FRAME_PTR ← ESP;
IF LEVEL > 0 THEN
        DO (LEVEL – 1) times
                EBP ← EBP – 4;
                PUSH Pointer(EBP); (* doubleword wskazywane przez EBP *)
        OD;
        PUSH FRAME_PTR;
FI;
EBP ← FRAME_PTR;
ESP ← ESP – STORAGE;
```

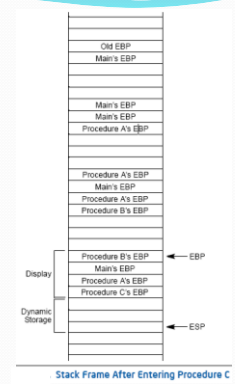

Instrukcja ENTER





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 zdejmuje 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; iLo, iHi: Integer);
  var
    Lo, Hi, Mid, T: Integer;
  begin
    Lo := iLo;
    Hi := iHi;
    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 > iLo then QuickSort(A, iLo, Hi);
    if Lo < iHi then QuickSort(A, Lo, iHi);
  end;

  begin
    QuickSort(A, Low(A), High(A));
  end;
```

```
procedure mysortq(var A:array of integer;iLo,iHi:integer);
asm
    push edi
    push esi
    push ilo
    push ihi
    call @s
    jmp @e

@s: enter 0,0          //sortowanie eax=A; [ebp+8]=ihi; [ebp+12]=ilo
    mov esi,[ebp+8]    //esi=hi:=ihi
    mov edi,[ebp+12]   //edi=lo:=ilo
    mov ecx,esi        //ecx=Mid := A[(Lo + Hi) div 2]
    add ecx,edi
    sar ecx,1
    mov ecx,[eax+ecx*4]   //mid
@r:                       //repeat
    cmp [eax+edi*4],ecx   //while A[Lo] < Mid do Inc(Lo)
    jge @1
    inc edi
    jmp @r
@r: cmp [eax+esi*4],ecx   //while A[Hi] > Mid do Dec(Hi)
    jle @2
    dec esi
    jmp @1
@2: cmp edi,esi           //if Lo <= Hi then
    jnle  @3
    mov edx,[eax+esi*4]   //A[Lo] <-> A[Hi]
    xchg edx,[eax+edi*4]
    mov [eax+esi*4],edx
    inc edi               //Inc(Lo)
    dec esi               //Dec(Hi)
@3: cmp edi,esi           //until Lo > Hi;
    jle @r
    cmp esi,[ebp+12]      //if Hi > iLo then QuickSort(A, iLo, Hi)
    jng @4
    push edi              //zabezpieczenie lo
    push [ebp+12]
    push esi
    call @s
    pop edi
@4: cmp edi,[ebp+8]       //if Lo < iHi then QuickSort(A, Lo, iHi)
    jnl @5
    push edi
    push [ebp+8]
    call @s
@5: leave
    ret 8

@e: pop esi
    pop edi
end;
```



```
SDiMAIN.PAS.147: begin
0046ACFC 55          push ebp
0046ACFD 8BEC        mov ebp,esp
0046ACFF 83C4F4      add esp,-$0c
0046AD02 53          push ebx
0046AD03 56          push esi
0046AD04 894DF8      mov [ebp-$08],ecx
0046AD07 8955FC      mov [ebp-$04],edx
0046AD0A 8BF0        mov esi,eax
0046AD0C 8B5DF8      mov ebx,[ebp-$08]   // Lo := iLo;
0046AD0F 8B4508      mov eax,[ebp+$08]   // Hi := iHi;
0046AD12 8D1418      lea edx,[eax+ebx]   // Mid := A[(Lo + Hi) div 2];
0046AD15 D1FA        sar edx,1
0046AD17 7903        jns $0046ad1c
0046AD19 83D200      adc edx,$00
0046AD1C 8B1496      mov edx,[esi+edx*4]
0046AD1F EB01        jmp $0046ad22
0046AD21 43          inc ebx        //while A[Lo] < Mid do Inc(Lo);
0046AD22 3B149E      cmp edx,[esi+ebx*4]
0046AD25 7FFA        jnle $0046ad21
0046AD27 EB01        jmp $0046ad2a
0046AD29 48          dec eax        //while A[Hi] > Mid do Dec(Hi);
0046AD2A 3B1486      cmp edx,[esi+eax*4]
0046AD2D 7CFA        jl $0046ad29
0046AD2F 3BC3        cmp eax,ebx          //if Lo <= Hi then
0046AD31 7C14        jl $0046ad47
0046AD33 8B0C9E      mov ecx,[esi+ebx*4]    //T := A[Lo];
0046AD36 894DF4      mov [ebp-$0c],ecx
0046AD39 8B0CB6      mov ecx,[esi+eax*4]    // A[Lo] := A[Hi];
0046AD3C 890C9E      mov [esi+ebx*4],ecx
0046AD3F 8B4DF4      mov ecx,[ebp-$0c]   // A[Hi] := T;
0046AD42 890CB6      mov [esi+eax*4],ecx
0046AD45 43          inc ebx              //Inc(Lo);
0046AD46 48          dec eax              //Dec(Hi);
0046AD47 3BC3        cmp eax,ebx          //until Lo > Hi;
0046AD49 7DD7        jnl $0046ad22
0046AD4B 3B45F8      cmp eax,[ebp-$08]    // if Hi > iLo then QuickSort(A, iLo, Hi);
0046AD4E 7E13        jle $0046ad63
0046AD50 8B550C      mov edx,[ebp+$0c]
0046AD53 52          push edx
0046AD54 50          push eax
0046AD55 8BC6        mov eax,esi
0046AD57 8B45F8      mov ecx,[ebp-$08]
0046AD5A 8B55FC      mov edx,[ebp-$04]
0046AD5D E89AFFFFFF  call QuickSort
0046AD62 59          pop ecx
0046AD63 3B5D08      cmp ebx,[ebp+$08]    // if Lo < iHi then QuickSort(A, Lo, iHi);
0046AD66 7D15        jnl $0046ad7d
0046AD68 8B450C      mov eax,[ebp+$0c]
0046AD6B 50          push eax
0046AD6C 8B4508      mov eax,[ebp+$08]
0046AD6F 50          push eax
0046AD70 8BC6        mov eax,esi
0046AD72 8BCB        mov ecx,ebx
0046AD74 8B55FC      mov edx,[ebp-$04]
0046AD77 E880FFFFFF  call QuickSort
0046AD7C 59          pop ecx
0046AD7D 5E          pop esi     //end;
0046AD7E 5B          pop ebx
0046AD7F 8BE5        mov esp,ebp
0046AD81 5D          pop ebp
0046AD82 C20400      ret $0004
```


Mnożenie BCD

```
  222
  5678
*    3
------
17034
```

```
   5678
*     3
-------
   5814
+1122
-------
17034
```


Mnożenie BCD

```
void m_BCD(char*a,char*b ,int n, int m, char*w) {
__asm{
    pushad
    pushf
    mov     ecx,m       //ilość cyfr mnożnika - m
    mov     edi,b       //adres mnożnika
    add     edi,ecx
    mov     edx,w       //adres wyniku
    add     edx,ecx
po_m:
    push    ecx         //zabezpiecz m ilość
                        //pozostałych cyfr mnożnika
    dec     edx         //m do o cyfra wyniku
    dec     edi         //m do o cyfra mnożnika
    mov     bl,[edi]    //do bl
    mov     ecx,n       //ilość cyfr mnożnej - m
    mov     esi,a       //adres mnożnej
iloczyn:
    mov     al,[esi]    //cyfra mnożnej - od najstarszej
    mul     bl          //razy cyfra mnożnika
    aam
    push    ax          //ah - przeniesienie z
                        //mnożenia, al - wynik na stos
    inc     esi
    Loop    iloczyn
    mov     cx,n
    pop     ax          //najmłodsze ze stosu
    add     al,[edx+ecx]  //i dodajemy cyfrę do wyniku
    aaa
    mov     [edx+ecx],al
    dec     ecx
    jz      one_n
SumaBCD:
    mov     bl, ah      //przeniesienie do bl
    pop     ax          //następna cyfra i przeniesienie ze stosu
    add     al,[edx+ecx]  //dodajemy kolejną cyfrę wyniku
    aaa
    add     al,bl       //plus poprzednie przeniesienie
    aaa
    mov     [edx+ecx],al  //i do wyniku
    loop    SumaBCD
one_n:
    mov     [edx],ah    //ostatnie przeniesienie do wyniku
    pop     ecx         //ilość pozostałych cyfr mnożnika
    loop    po_m
    popf
    popad
    }
}
```