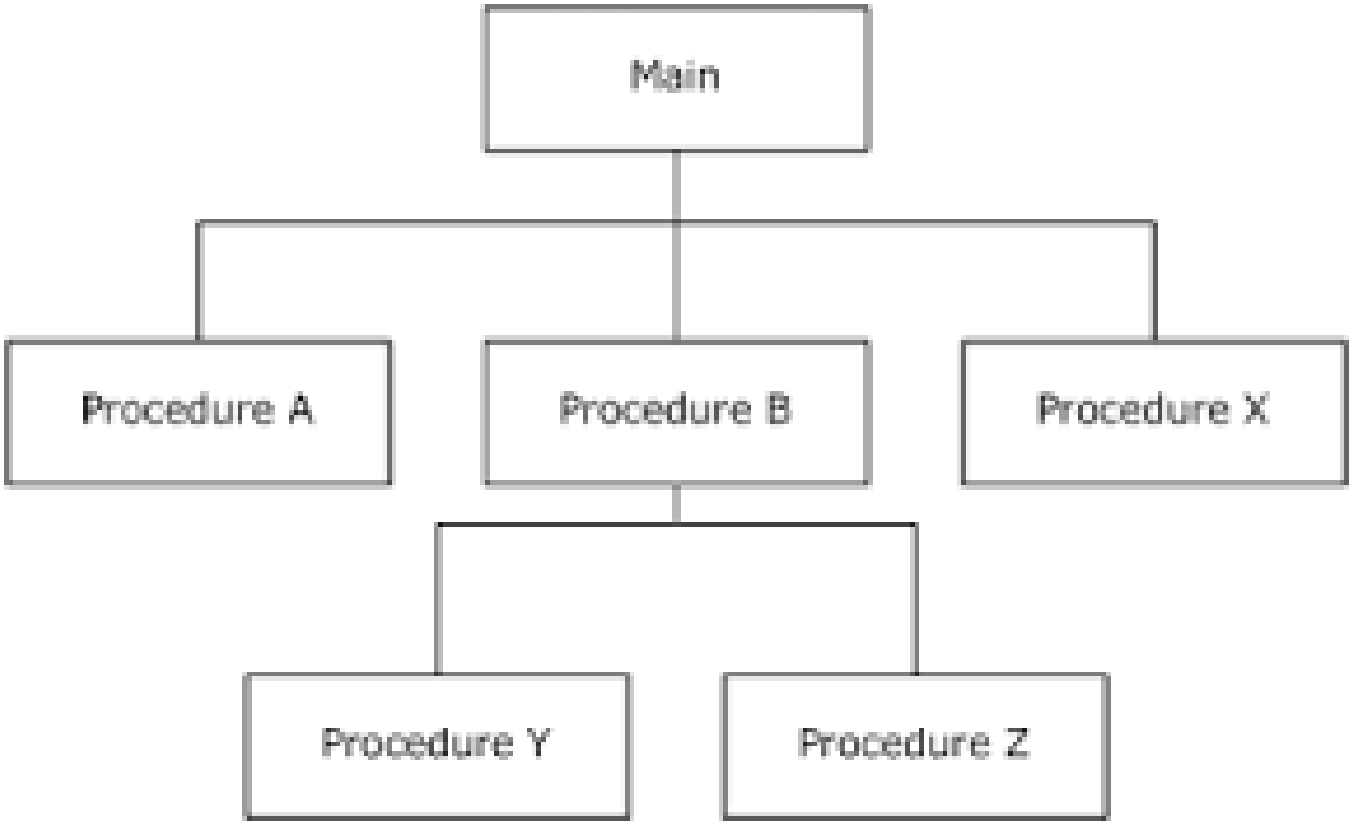
Wykład 4

Programowanie proceduralne

Programowanie proceduralne



Programowanie proceduralne

0) pij z filiżanki

1) chwyć filiżankę za ucho

- 1.1) rozewrzyj palce
- 1.2) sięgnij ręką do ucha filiżanki
- 1.3) zaciśnij palce na uchu

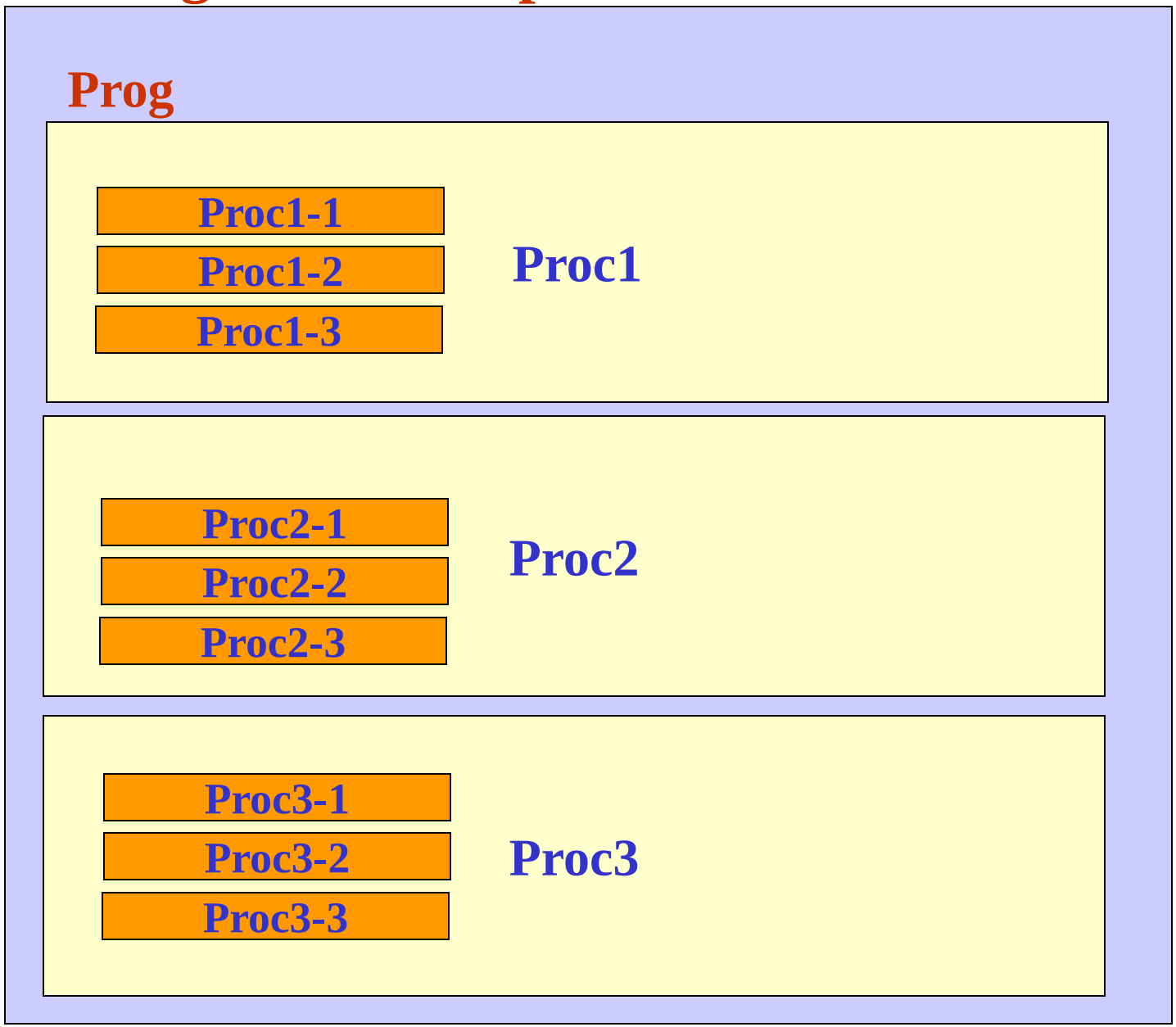
2) podnieś filiżankę do ust

- 2.1) unieś rękę do góry
- 2.2) przysuń rękę poziomo do ust

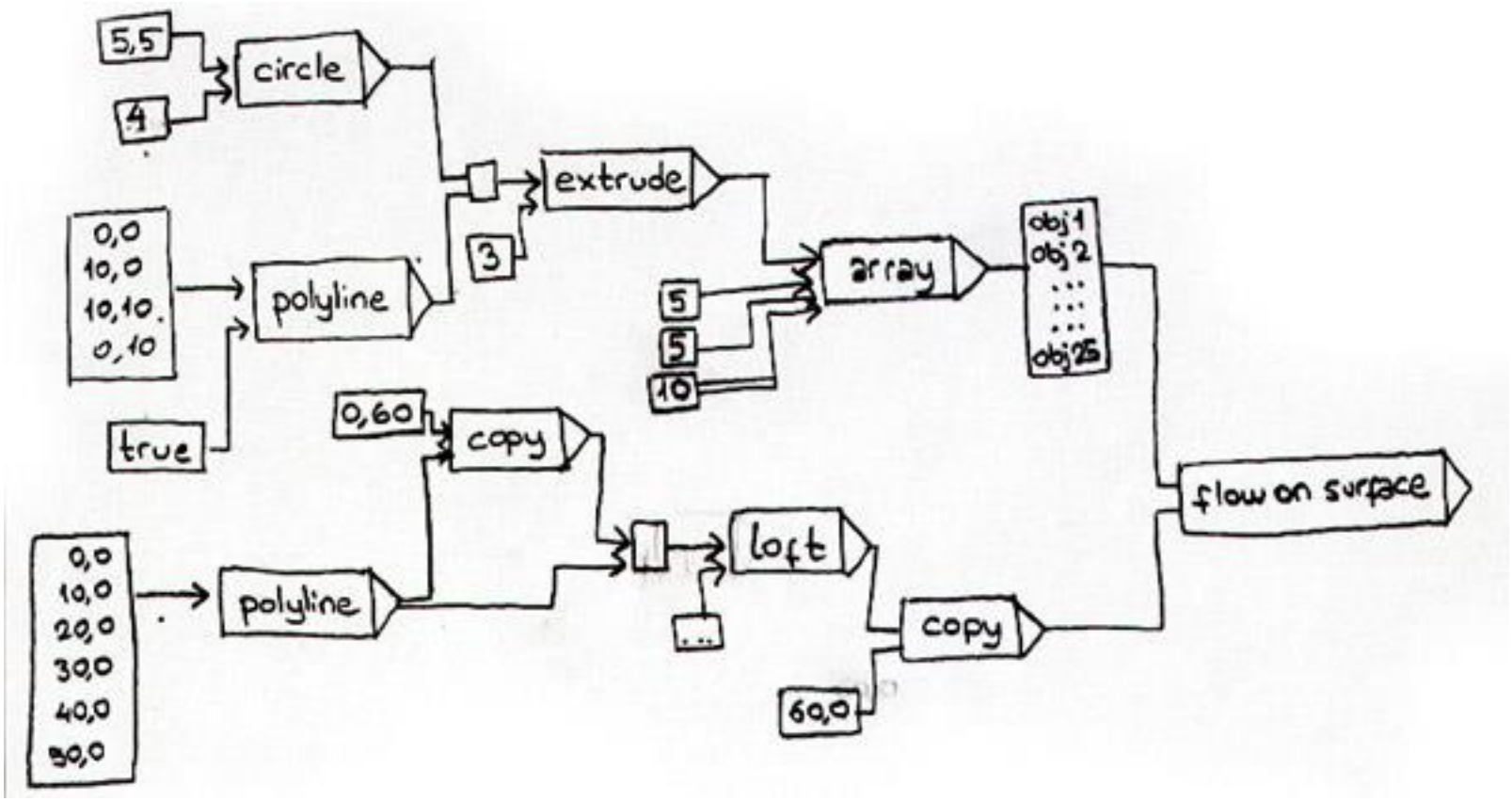
3) wlej zawartość do gardła

- 3.1) otwórz usta
- 3.2) przechyl filiżankę
- 3.3) POŁYKAJ PŁYN!!!

Programowanie proceduralne



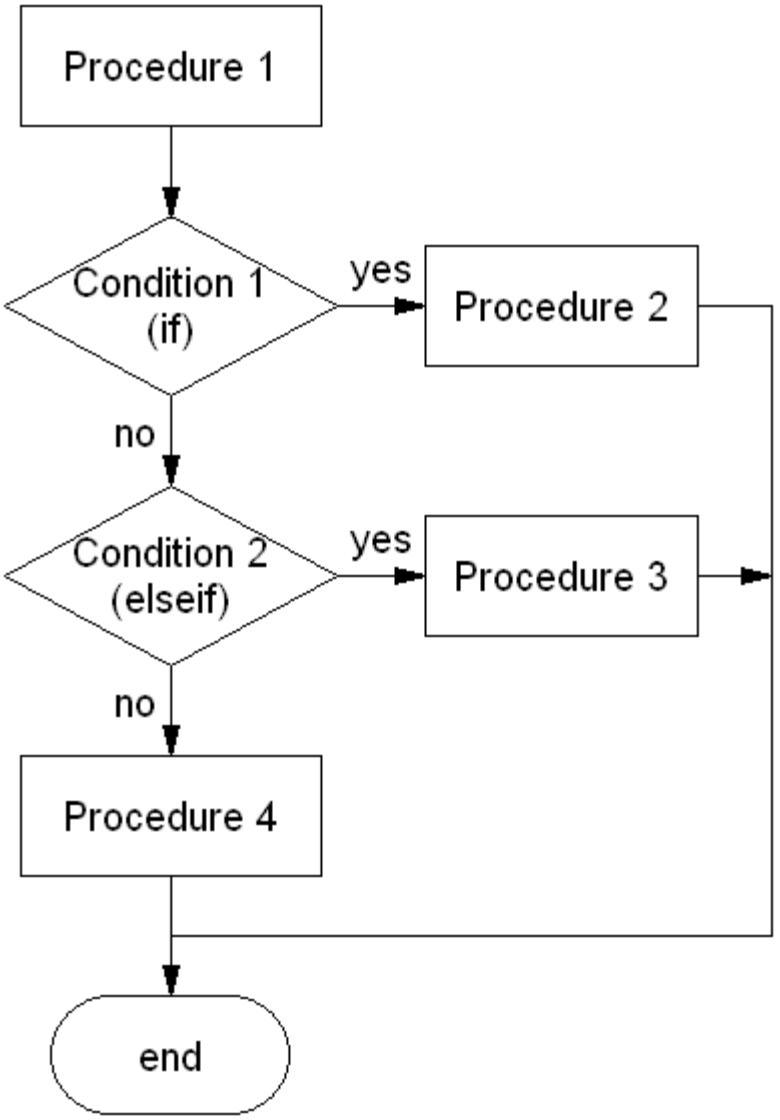
Programowanie proceduralne



Przetwarzanie danych w programie proceduralnym

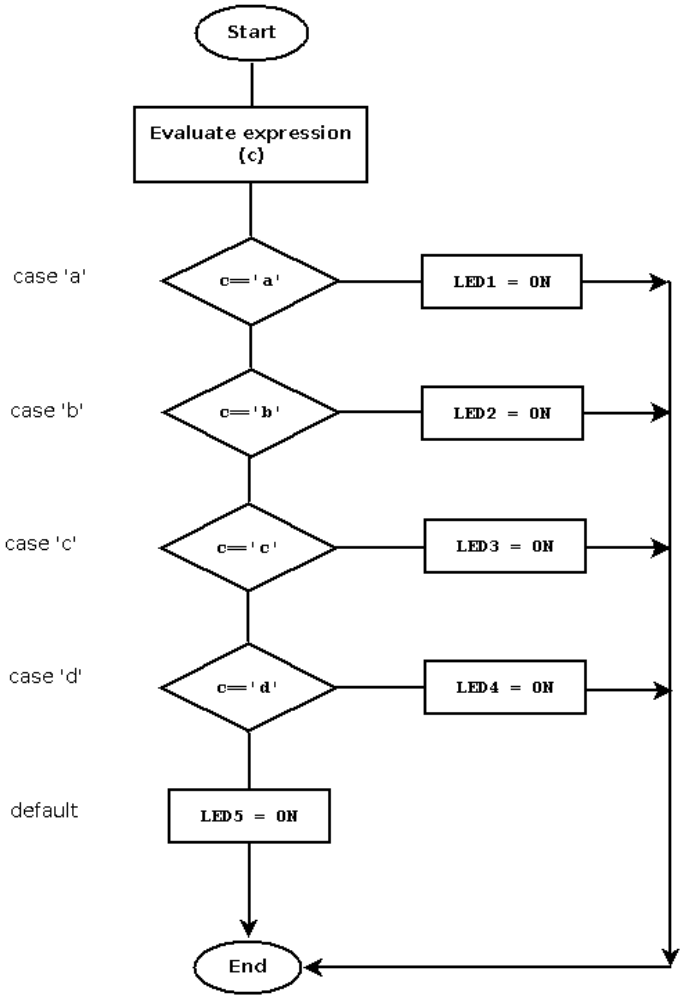
Programowanie proceduralne

Struktura programu na dowolnym poziomie dekompozycji:



Programowanie proceduralne

Przykładowa struktura podprogramu na niskim poziomie dekompozycji:



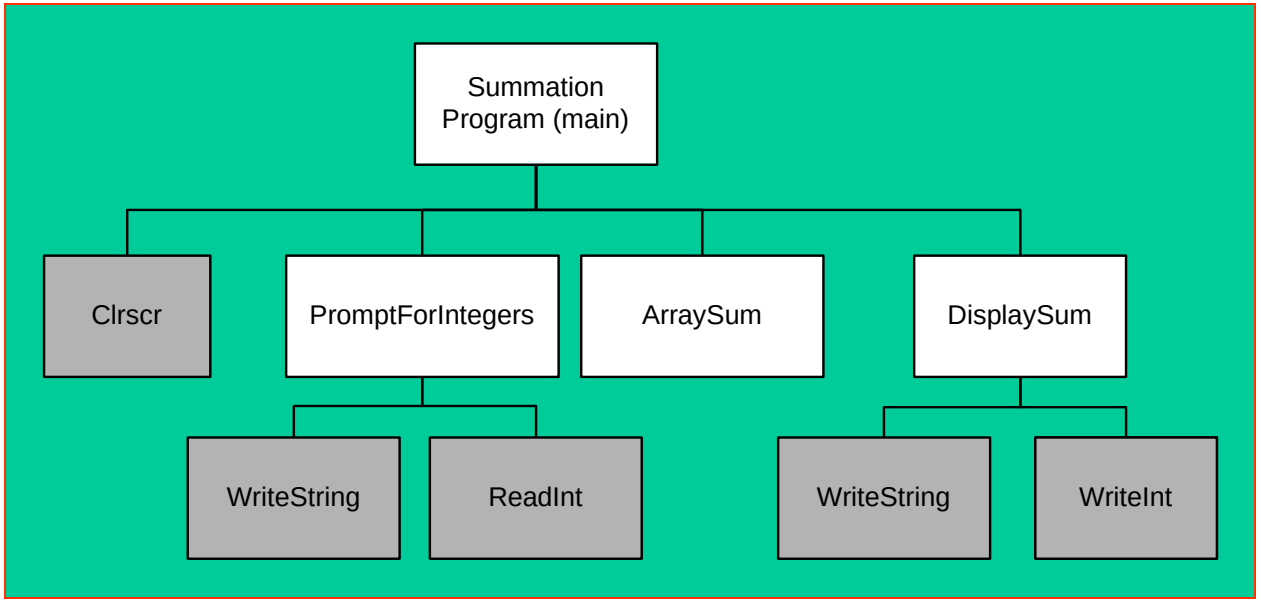
Programowanie proceduralne

Konstrukcja procedur:

- **Jakie dane wejściowe?**
- **Jakie są potrzebne rezultaty przetwarzania danych?**
- **Jakie działania należy wykonać na danych**

Herarchiczna struktura programu proceduralnego

- **Dekompozycja funkcjonalna (od góry do dołu) :**
  - **zaprojektuj program przed programowaniem**
  - **podziel duże zadania na mniejsze**
  - **użyj struktury hierarchicznej opartej na procedurach**
  - **poszczególne procedury testuj oddzielnie**



Parametery procedury

- **Dobra procedura powinna być użyteczna w wielu różnych programach**
  - **nie będzie, jeżeli odwołuje się do ustalonych nazw zmiennych**
- **Parametry umożliwiają elastyczność procedur – wartości parametrów mogą zmieniać się w czasie wykonywania programu**

Parametery procedury

Procedura ArraySum oblicza sumę 32-bitowych wyrazów tablicy (liczby całkowite). Odwołuje się do 2 nazw zmiennych: myArray, theSum

```
ArraySum PROC
        mov esi,0                       ; wskaźnik tablicy
        mov eax,0                       ; wstępnie suma=0
        mov ecx,LENGTHOF myArray  ;liczba elementów tablicy

L1:  add eax,myArray[esi]               ; dodaj każdy wyraz do sumy
        add esi,4                       ; wskaż następny wyraz
        loop L1                         ; powtarzaj size

        mov theSum,eax                  ; wynik do theSum
        ret
ArraySum ENDP
```

Procedura nieprzydatna, gdy trzeba ją wykonywać dla wielu tablic!!!

Klasyfikacja parametrów

- **Parametr wejściowy przekazywany do procedury:**
  - **Wywoływana procedura nie będzie modyfikować zmiennej odpowiadającej temu parametrowi (ew. modyfikacja nie będzie widzialna poza procedurą).**

- **Parametr wyjściowy utworzony jest poprzez przekazanie wskaźnika do zmiennej wraz z wywołaniem procedury:**
  - **Procedura nie wykorzystuje wartości zmiennej; przed zakończeniem wykonywania procedura wpisuje do tej zmiennej wynikową wartość.**

- **Parametr wejściowo-wyjściowy jest wskaźnikiem do zmiennej zawierającej wartość, która zarówno będzie używana jak i modyfikowana przez procedurę:**
  - **Zmienna przekazana do procedury jest modyfikowana przez procedurę.**

Przykład

Procedura Swap wymienia wartości dwóch 32-bitowych zmiennych całkowitych. pValX i pValY (wskaźniki do zmiennych) nie zmieniają wartości. Zmianie podlegają zmienne wskazywane przez wskaźniki:

```
Swap PROC USES eax esi edi,
    pValX:PTR DWORD,          ; wskaźnik do pierwszej zmiennej
    pValY:PTR DWORD           ; wskaźnik do drugiej zmiennej

    mov esi,pValX             ; pobierz wskaźniki
    mov edi,pValY
    mov eax,[esi]             ; pobierz wart. pierwszej zmiennej
    xchg eax,[edi]            ; zamień z drugą
    mov [esi],eax             ; zastąp wart. pierwszej zmiennej
    ret
Swap ENDP
```

Przekazywanie parametrów do i z procedury

5 podstawowych mechanizmów przekazywania danych

- **poprzez wartość** — **tylko wejściowe; procedura nie zmienia wartości parametrów – kopiowanie wewnątrz procedury**
- **poprzez odwołanie** — **wskaźnik do zmiennej (adres)**
- **poprzez zwróconą wartość** — **przekazywany wskaźnik do zmiennej, wartość zmiennej kopiowana do zmiennej wewn. (mogą mieć różną wart.). Wyjście - wartość wyniku.**
- **poprzez rezultat** — **jw.; przekazywanie wyłącznie parametru wyjściowego (nie potrzebne kopiowanie przy wejściu).**
- **poprzez nazwę** — **Wskaźnik do funkcji obliczającej adres zmiennej**

Przekazywanie parametrów do procedury

- rejestry
- komórki pamięci (globalne)
- blok
- kod programu
- stos

Mała liczba rejestrów!!!

```
.............
BLOCK1_ADDR EQU 00AA9988H
BLOCK2_ADDR EQU 00AA9988H
.............
                    MOV BX, #BLOCK1_ADDR
                    CALL ZEROBYTES
.............
                    MOV BX, #BLOCK1_ADDR
                    CALL ZEROBYTES
.................
END

ZEROBYTES:   XOR    AX, AX
             MOV    CX, 128
ZEROLOOP:    MOV    [BX], AX
             ADD    BX, 2
             LOOP   ZEROLOOP
             RET
```

Przekazywanie parametrów do procedury

Ta wersja ArraySum wyznacza sumę 32-bitowych elementów tablicy, której adres podany jest w ESI. Wynik sumy w EAX.

```
ArraySum PROC
; Wejścia: ESI wskaźnik tabeli słów podwójnej długości,
; ECX = liczba elementów tablicy
; Wyjście: EAX = suma
;-----------------------------------------------
      mov eax,0                    ; wstępnie suma=0

L1:   add eax,[esi]                ; dodaj każdy wyraz do sumy
      add esi,4                    ; wskaż następny wyraz
      loop L1                      ; powtarzaj

      ret
ArraySum ENDP
```

Można wielokrotnie wywoływać dla różnych wartości parametrów

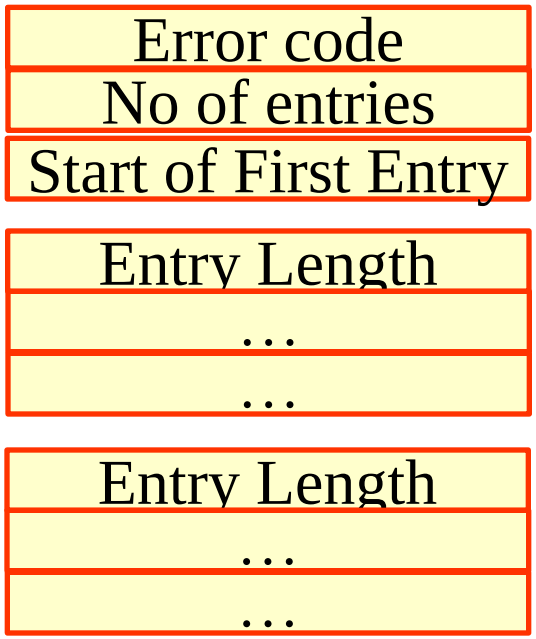
Przekazywanie parametrów do procedury

- Rejestry
- Komórki pamięci (globalne)
- Blok
- Kod programu
- Stos

```
.............
BLOCK1_ADDR EQU 00AA9988H
BLOCK2_ADDR EQU 00AA9988H
.............
                    MOV PARAM, #BLOCK1_ADDR
                    CALL ZEROBYTES
.............
                    MOV PARAM, #BLOCK2_ADDR
                    CALL ZEROBYTES
.................
END
PARAM: DW     ?
ZEROBYTES:    MOV     BX,PARAM
              XOR     AX, AX
              MOV     CX, 128
ZEROLOOP:     MOV     [BX], AX
              ADD     BX, 2
              LOOP    ZEROLOOP
              RET
```

Przekazywanie parametrów do procedury

- Rejestry
- Komórki pamięci (globalne)
- Blok (tablicę)
- Kod programu
- Stos



Parametry umieszczone w bloku komórek pamięci (jak dla przekazywania w pamięci). Wskaźnik do bloku przekazany dowolną metodą (np. w rejestrze, na stosie)

Przekazywanie parametrów do procedury

- Rejestry
- Komórki pamięci (globalne)
- Blok
- Kod programu
- Stos

Możliwe zmienne o zmiennej długości

```
        Call disp
        DB “This parameter is in the code stream.”,0
        ..............
Disp PROC NEAR
        push bp
        mov bp, sp
        ..... (m.in. zapisz rejestry na stos)
        mov bx, [bp+2] ;adr powrotu do BX
        ..... (BX zmienia się, aż wskazuje
                adres za końcem danych)
        mov [bp+2], bx ;nowy adr. powrotu na stos
        ..... (m.in. czytaj rejestry ze stosu)
        pop bp
        ret
disp endp
```

Przekazywanie parametrów do procedury

- Rejestry
- Komórki pamięci (globalne)
- Blok
- Kod programu
- Stos

Parametry na stosie czy w rejestrach?

- Parametry w rejestrach wymagają przeznaczenia po rejestrze na parametr. Użycie stosu jest wygodniejsze.
- Przykład dwóch różnych sposobów wywołania procedury DumpMem:

```
pushad
mov esi,OFFSET array
mov ecx,LENGTHOF array
mov ebx,TYPE array
call DumpMem
popad
```

```
push OFFSET array
push LENGTHOF array
push TYPE array
call DumpMem
```

prostszy

Specyfika architektury z segmentacją pamięci

Procesory z segmentacją (Intel x86)

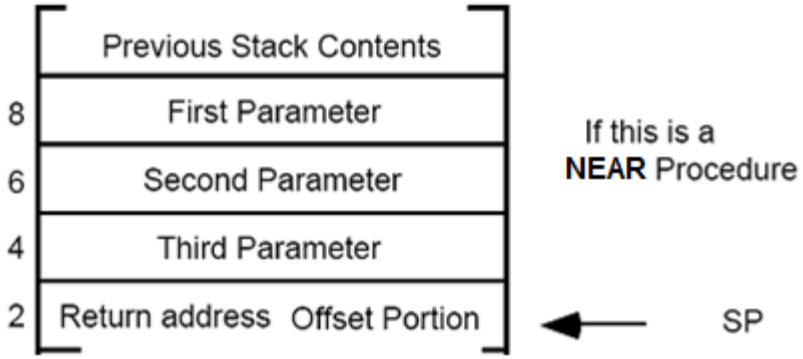
CALL
RETN

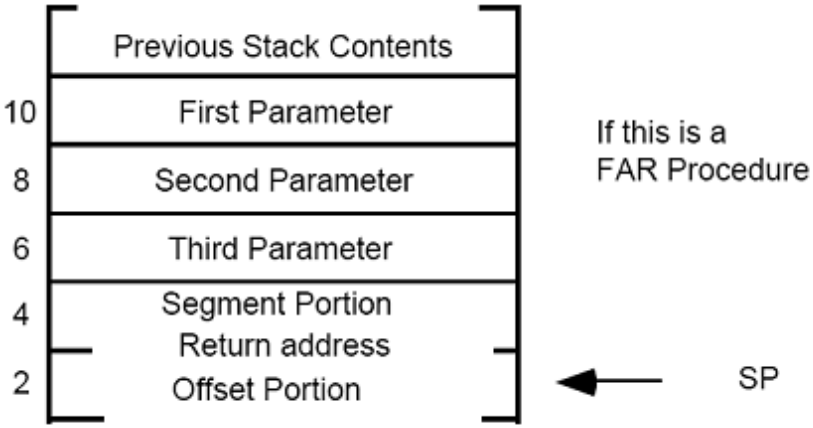
na stosie tylko przesunięcie

(przesunięcie:segment)

CALL FAR PTR symbol
RETF

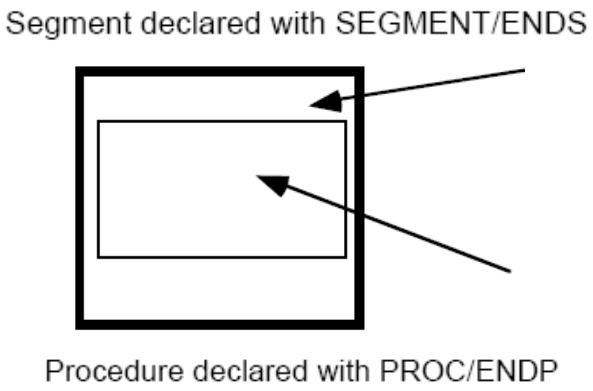
na stosie CS i przesunięcie





Specyfika architektury z segmentacją pamięci

Procesory z segmentacją (Intel x86)




```
cseg segment
.............
MyProc proc near
            Ret
MyProc   endp
.............
cseg       ends
```

Parametry na stosie

Przykład: procedura z 3-ma parametrami:

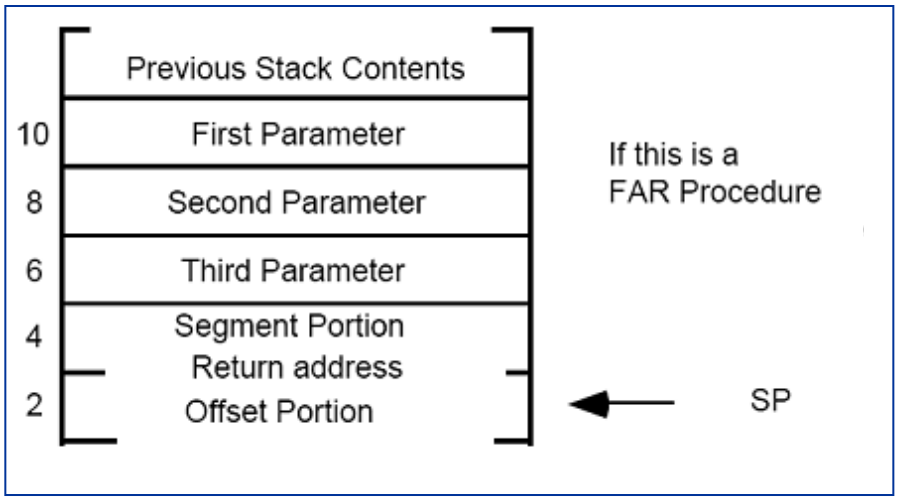
MyProc(par1,par2,par3);

Wywołanie procedury:

```
push par1
push par2
push par3
call MyProc
```

Odczytanie parametrów w procedurze

ramka



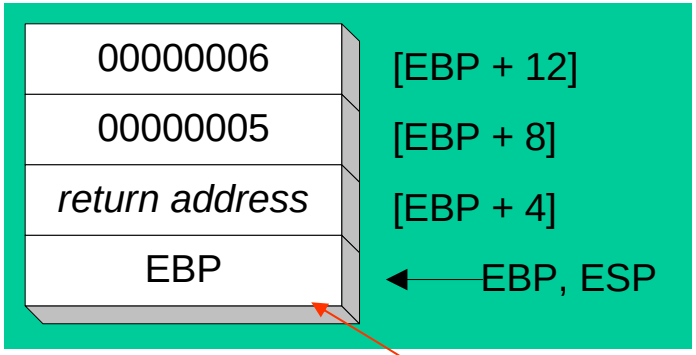

```
MyProc  proc far
                pop RtnAdrs32
                pop Par3
                pop Par2
                pop Par1
                push RtnAdrs32
                .
                .
                .
                retf
CallProc        endp
```

Bezpośredni dostęp do parametrów na stosie

- **bezpośredni dostęp do parametrów na stosie z użyciem przesunięcia w stosunku do EBP (BP – tryb rzeczywisty).**
  - **np: [ebp + 8]**
- **EBP – wskaźnik bazowy (wskaźnik ramki) = adres bazowy ramki na stosie.**
- **EBP nie zmienia wartości w czasie procedury.**
- **Na zakończenie procedury przywrócić wejściową wartość EBP.**

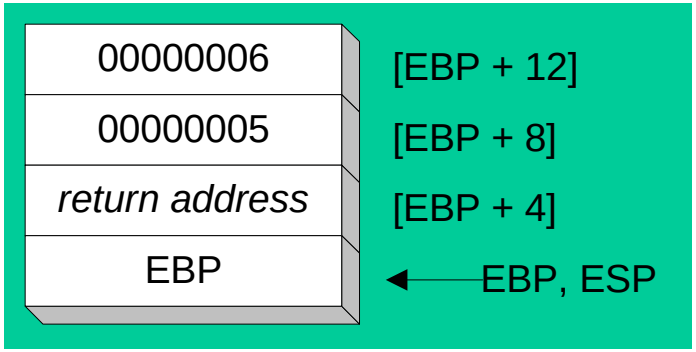
Przykład:

```
.data
sum DWORD ?
.code
        push 6          ; drugi argument
        push 5          ; pierwszy argument
        call AddTwo     ; EAX = suma
        mov  sum,eax     ; zapisz sumę
```




```
AddTwo PROC
        push ebp
        mov  ebp,esp
        .
        . (dostęp do stosu z użyciem ebp)
        .
```

Ramka na stosie – cd przykładu



- **Odczytanie parametrów ze stosu**
- **Przywrócenie wartości esp i ebp**
- **Usunięcie ramki ze stosu**

```
AddTwo: PROC
        push ebp
        mov ebp,esp             ; adres bazowy ramki
        mov eax,[ebp + 12]      ; drugi argument (6)
        add eax,[ebp + 8]       ; pierwszy argument (5)
        mov  esp,ebp
        pop  ebp
        ret
AddTwo: ENDP                    ; EAX zawiera sumę
```

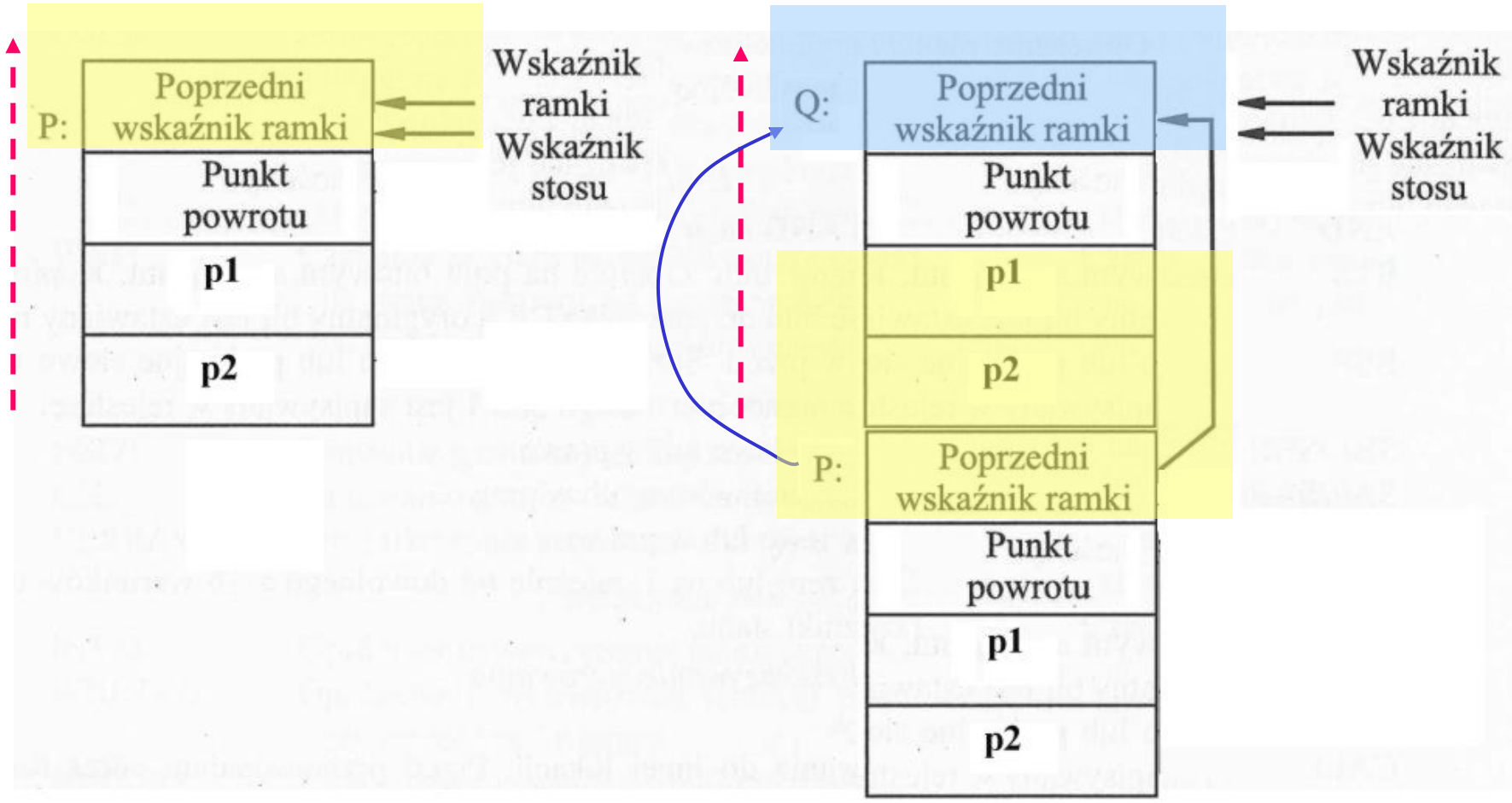
↔ instrukcja *leave*

8× { inc esp
.....
inc esp

↔ Instrukcja *ret 8*

Przekazywanie parametrów do procedury

Ramka z parametrami procedury



Zmienne (lokalne, wewnętrzne) procedury

1) Wykonanie procedury może wymagać użycia zmiennych (pomocniczych)
- rejestry procesora

Inne
rejes

Zmienna lokalna jest tworzona, używana i likwidowana w ramach pojedynczej procedury (jest tymczasowa).

2) Wykonanie procedury wymaga użycia rejestrów procesora (wykonywanie rozkazów)

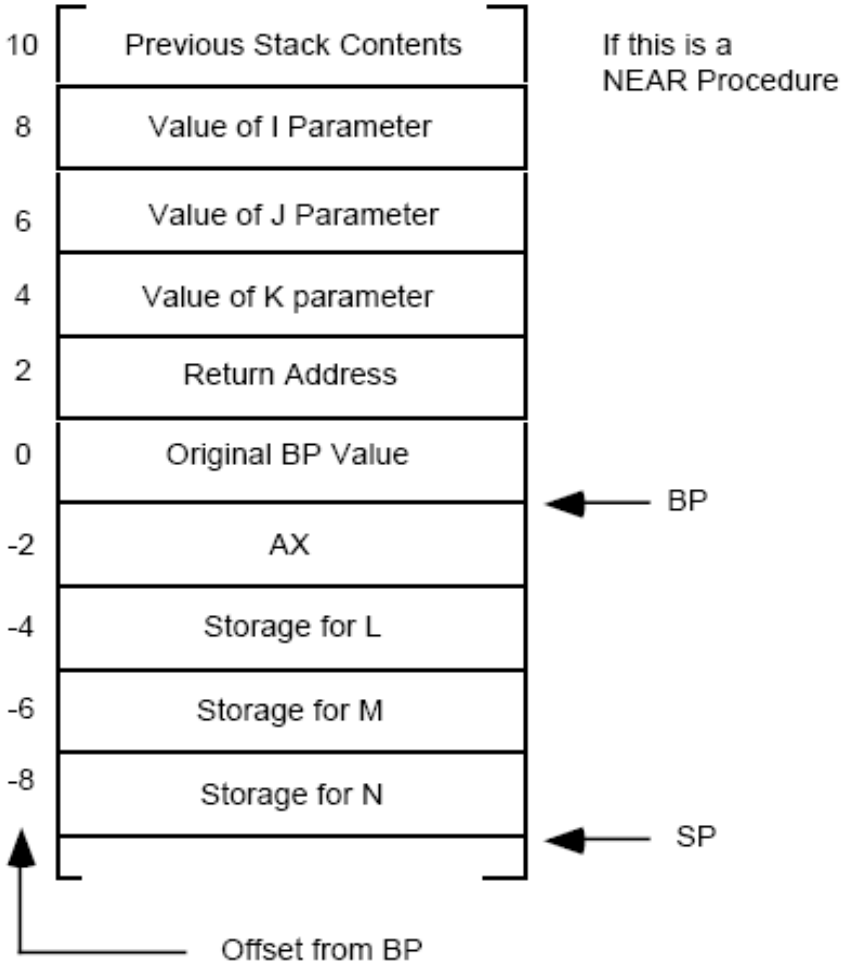
- te same rejestry mogą być w użyciu w programie wywołującym procedurę - konflikt!!!

- gdzie w pamięci zlokalizować zmienne procedury?
- jak „powielić” lokalizacje zmiennych dla kolejnych wywołań procedury?
- jak odwoływać się do zmiennych procedury?

Ramka na stosie – przykład Intel

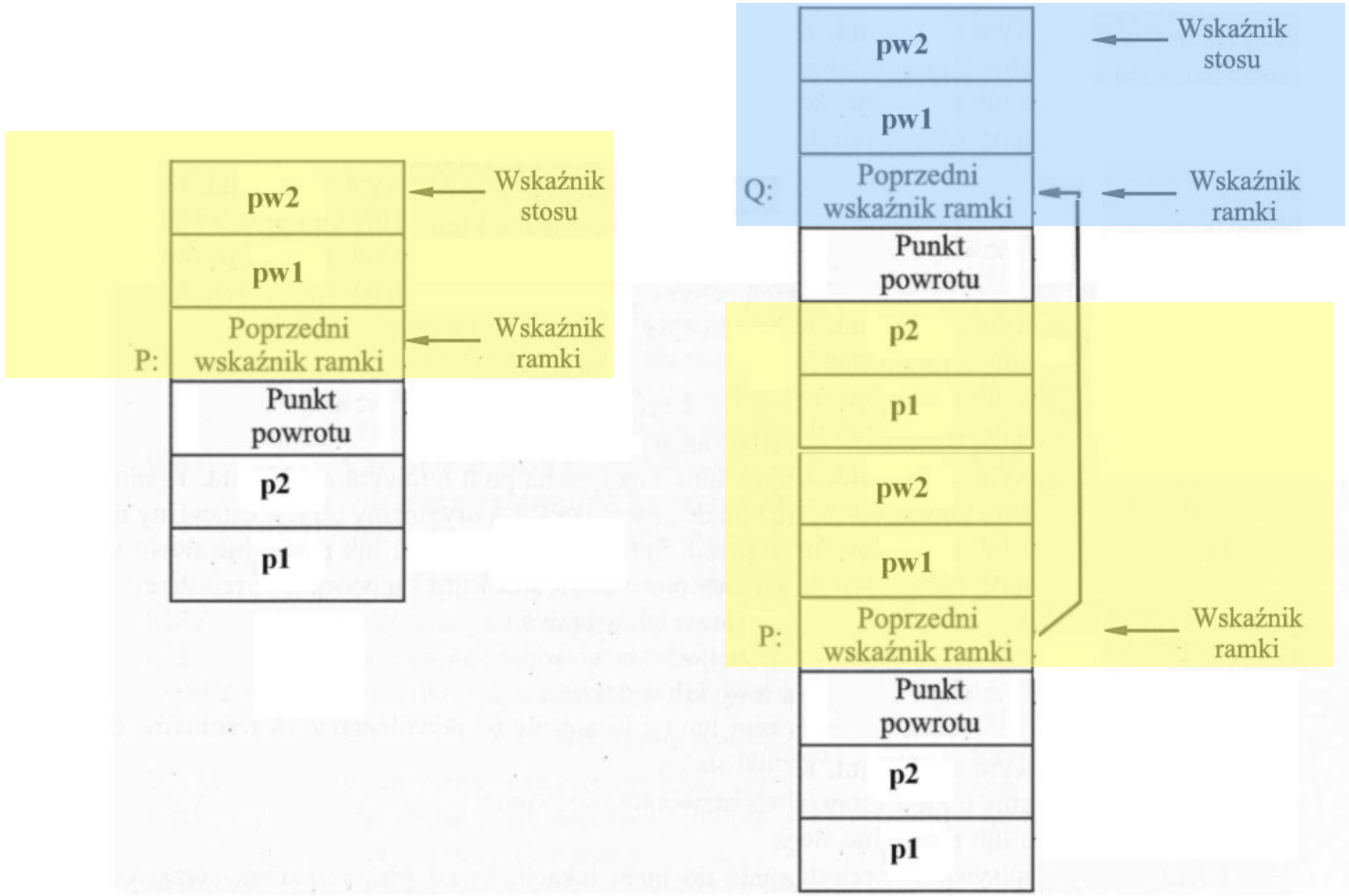
- **Obszar na stosie przeznaczony na: adres powrotu, przekazywane parametry, zachowane rejestry, zmienne lokalne**

- **Tworzony w następujących krokach:**
  - **–program wywołujący umieszcza argumenty na stosie i wywołuje procedurę.**
  - **–Wywołana procedura zapisuje EBP na stos i przepisuje ESP do EBP.**
  - **–Jeżeli potrzebne są zmienne lokalne odpowiednia liczba całkowita odejmowana jest od ESP, aby utworzyć miejsce na stosie.**



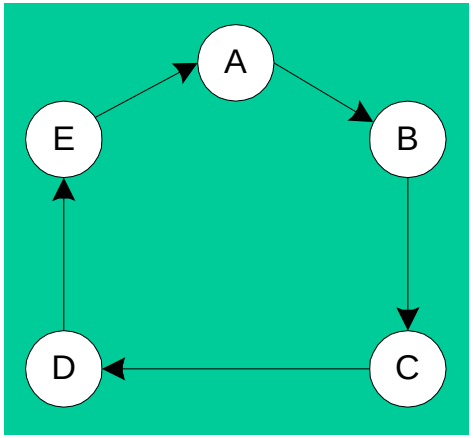
Zmienne wewnętrzne procedury

Ramka ze zmiennymi wewnętrznymi procedury



Przykład: zmienne wewnętrzne procedury - rekursja

- **Proces powstający gdy . . .**
  - **procedura wywołuje samą siebie**
  - **Procedura A wywołuje procedurę B, która wywołuje procedurę A**
- **Rekursja tworzy cykl (patrz ilustracja graficzna)**



Problem: rekursywne wywołanie – równocześnie używane te same parametry

Rekursja – ogólne zasady

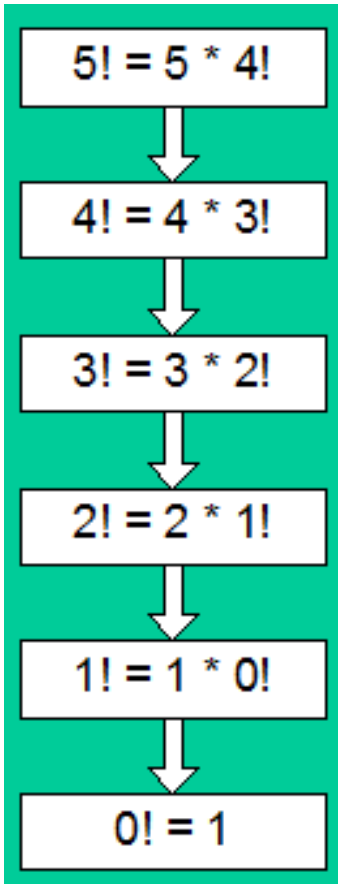
```
Function  Proc
          Push      eBp
          Mov       eBp, eSp

  .IF     Terminating Condition is true
          base case result
   .ELSE  recursive case result
          Push      Parameters
          Call      Function
          Add       eSp, <Number of Parameter bytes>
  .ENDIF
          Pop       eBp
          Ret
Function Endp
```

Rekursja – przykład obliczania silni

Poniższa funkcja rekursywnie oblicza silnię liczby całkowitej *n*. W każdym kroku nowa wartość *n* zapisywana jest na stos.

```
int function factorial(int n)
{
    if(n == 0)
      return 1;
    else
      return n * factorial(n-1);
}
```

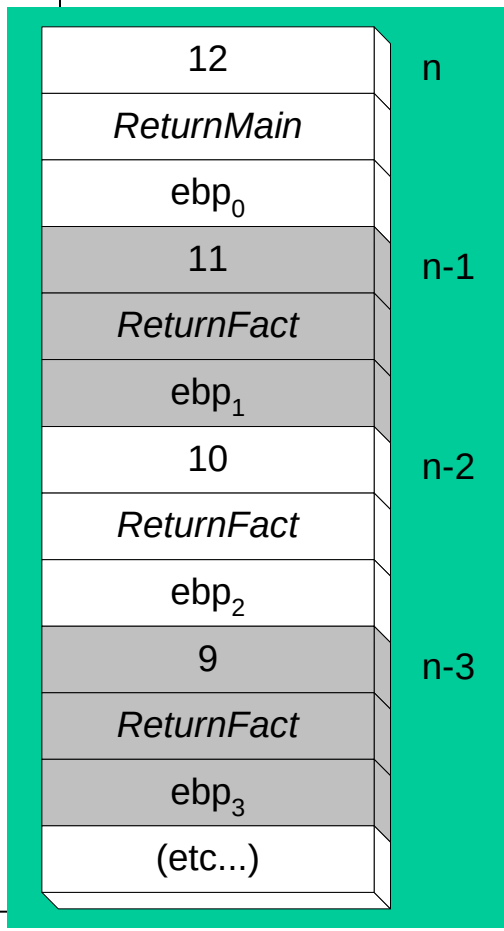
Każde wywołanie dostarcza wynik w postaci poprzedniej wartości pomnożonej przez *n*.



Rekursja – przykład obliczania silni

```
Factorial PROC
    push ebp
    mov  ebp,esp
    mov  eax,[ebp+8]                ; pobierz n
    cmp  eax,0                      ; n < 0?
    ja   L1                         ; tak: kontynuuj
    mov  eax,1                      ; nie: zwróć 1
    jmp  L2

L1: dec  eax
    push eax                        ; Silnia (n-1)
    call Factorial

;  Poniższe instrukcje wykonywane przy powrocie z
;  wywołań rekursywnych

ReturnFact:
    mov  ebx,[ebp+8]                ; pobierz n
    mul  ebx                        ; ax = ax * bx

L2: pop  ebp                        ; zwróć EAX
    ret  4                          ; czyść stos
Factorial ENDP
```

Ramka na stosie:



Ramka na stosie – przykład Intel

Instrukcja: ENTER (Make Stack Frame, 80286+)

Instrukcja tworzy ramkę na stosie dla procedury używającej parametrów własnych zlokalizowanych na stosie.

Składnia:

ENTER *immed16, 0*

ENTER *immed16, 1*

ENTER *immed16, immed8*

Liczba bajtów do zarezerwowania.
Immed16=0 ↔ ENTER = push bp, mov bp,sp.

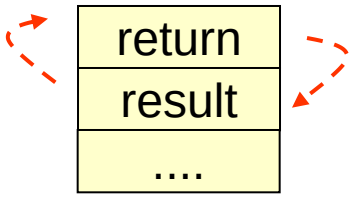
Maksymalny poziom zagnieżdżenia

Parametry wyników z procedury

- **Rejestry**
- **Stos**
- **Pamięć (blok)**

Use	First	Last
Bytes:	al, ah, dl, dh, cl, ch, bl, bh	
Words:	ax, dx, cx, si, di, bx	
Double words:	dx:ax	On pre-80386
	eax, edx, ecx, esi, edi, ebx	On 80386 and later.
16-bitOffsets:	bx, si, di, dx	
32-bit Offsets	ebx, esi , edi, eax, ecx, edx	
Segmented Pointers:	es:di, es:bx, dx:ax, es:si	Do not use DS.

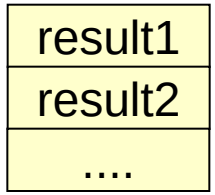
przed opuszczeniem procedury



stos

zwykle wskaźnik do bloku (rejestr lub stos) + tablica w pamięci

ptr



Dokumentacja procedury

Dokumentacja procedury

Dokumentacja procedury powinna zawierać:

- **Opis wszystkich zadań procedury**
- **Opis wejść: lista parametrów wejściowych, określenie ich użycia i wymagań**
- **Opis wyników: opis wartości zwracanych przez procedurę**
- **Opis wymagań: opcjonalna lista wymagań wstępnych, które muszą być spełnione przed wywołaniem procedury**

Brak gwarancji, że procedura wywołana bez spełnienia warunków wstępnych będzie działać.

Dokumentacja procedury - przykład

```
;-----------------------------------------------------
SumOf PROC
;
; Obl. i zwraca wynik sumy trzech 32-bitowych liczb całk.
; wejścia: EAX, EBX, ECX – trzy liczby całkowite. Mogą być
; ze znakiem lub bez.
; wyjścia: EAX = suma, flagi statusu (Carry,
; Overflow, itd.) zmienione.
; Wymagania: nic
;-----------------------------------------------------
      add eax,ebx
      add eax,ecx
      ret
SumOf ENDP
```

Dokumentacja procedury - przykład

```
;-----------------------------------------------------
SumOf PROC
;
; Obl. i zwraca wynik sumy trzech 32-bitowych liczb całk.
; wejścia: EAX, EBX, ECX – trzy liczby całkowite. Mogą być
; ze znakiem lub bez.
; wyjścia: EAX = suma, flagi statusu (Carry,
; Overflow, itd.) zmienione.
; Wymagania: nic
;-----------------------------------------------------
      add eax,ebx
      add eax,ecx
      ret
SumOf ENDP
```