Przetwarzanie informacji

rozkaz – niezależna jednostka syntaktyczna oraz semantyczna

- działania na danych (w celu wytworzenia wyniku)
  - *kopiowanie* – niszczące, nieodwracalne
  - *zmiana formatu* – przemieszczanie pól (rekordów)
  - *konwersja kodu* – dołączanie lub usuwanie bitów
  - *działania logiczne* – jednobitowe, równocześnie na wielu bitach słowa
  - *działania arytmetyczne* – niezbędna kontrola poprawności wyniku

program (algorytm) = sekwencja instrukcji

- sterowanie – ustalenie sekwencji przetwarzania
  - tworzenie warunków – przygotowanie przesłanek
  - identyfikacja warunków – wybór przesłanek
  - wybór ścieżki przetwarzania – decyzja

Przepływ sterowania

powiązanie instrukcji tworzących program

• *funkcjonalne* – jaka jest następna instrukcja

konstrukcje: **repeat** (powtarzaj), **execute** (wykonaj).

• *lokacyjne* – gdzie jest następna instrukcja

- *sekwencyjne* (ang. *sequential*) – wyklucza sterowanie,
  - ❖ domniemana lokacja kolejnego rozkazu (porządek liniowy)
    → krótszy kod
- *łańcuchowe* (ang. *chained*) – umożliwia sterowanie,
  - ❖ konieczne wskazanie lokacji kolejnego rozkazu
    → dłuższy kod (musi zawierać wskazanie kolejnego rozkazu)

Kompromis:

- domniemany liniowy porządek instrukcji → *powiązanie sekwencyjne*
- sterowanie (zmiana porządku instrukcji) → *powiązanie łańcuchowe*

Realizacja decyzji

Sterowanie – realizacja decyzji (**jeśli** warunek WX **podejmij decyzję** XX)

Każda *decyzja* :może być

→ określona jako *wybór jednej spośród rozłącznych decyzji elementarnych* (case)

$$AA \cap BB \cap \ldots \cap NN = \phi, \quad AA \cup BB \cup \ldots \cup NN = \Omega,$$

będących skutkiem spełnienia jednej z rozłącznych przesłanek

$$WA \cap WB \cap \ldots \cap WN = \phi, \quad WA \cup WB \cup \ldots \cup WN = \Omega$$

→ *dekomponowana* na *sekwencję rozłącznych decyzji binarnych*: (if ... then ... else...)

jeśli $WA \cup WB \cup WC \cup WD = True$ **wtedy** wykonaj:
  jeśli $WA \cup WB = True$ **wtedy** wykonaj
    jeśli $WA = True$ **wtedy** wykonaj AA
      w przeciwnym razie wykonaj B
  w przeciwnym razie ($WC \cup WD = True$) wykonaj
    jeśli $WC = True$ **wtedy** wykonaj CC
      w przeciwnym razie wykonaj DD
w przeciwnym razie ($WA \cup WB \cup WC \cup WD = False$) wykonaj ...

Rozgałęzienia (w programie)

- *wytworzenie warunków* – wykonanie działania, np. porównanie
- *wybór warunku* – jawne lub implikowane wskazanie przesłanki
- *użycie warunku – decyzja* (*zmiana porządku* lub *sposobu wykonania*)
  - jawne wykonanie rozgałęzienia (instrukcja), ominięcie
    - *rozgałęzienie zwykłe* (*branch*), *skok warunkowy* (**j***war* adres / **b***war* adres)

      if *warunek* **then goto** adres
    - *rozgałęzienie ze śladem* (*branch & link*) ((**call***war* adres /**bl***war* adres)

      if *warunek* **then goto** adres and **link**
  - warunkowe wykonanie instrukcji

    if *warunek* **then** *polecenie*
    - pułapka (*trap*) – warunkowe wykonanie funkcji

      if *warunek* **then call** *exception*
  - przechowanie stanu logicznego warunku (**set***war* zmienna)

    if *warunek* **then** zmienna:= *TRUE* **else** zmienna:= *FALSE*

Techniki wykonywania rozgałęzień

- każde rozgałęzienie można odnieść do alternatywnego warunku

alternatywa

if *warunek* (PRAWDZIWY) **then** *polecenie*(*P*) (FAŁSZYWY) **else** *polecenie*(*F*)

```
                  bwar  adr_pol(P)          bnot–war  adr_pol(F)
                  polecenie(F)              polecenie(P)
                  bra  dalej                bra  dalej
adr_pol(P):  polecenie(P)        adr_pol(F):  polecenie(F)
      dalej:                           dalej:
```

ominięcie

if *war*=***true*** **then** *polecenie* (**else** *kontynuacja*)

```
                  bwar  adr_pol             bnot–war  dalej
                  bra  dalej                polecenie
adr_pol:  polecenie              dalej:  kontynuacja
    dalej:  kontynuacja
```

if *war*=***false*** **then** *kontynuacja* **else** *polecenie*

Unikanie rozgałęzień

instrukcje wykonywane warunkowo (IA-32)

cmov*war arg, zmienna* ; **if** *war*=***true*** **then** *arg*=*zmienna*

cmpxchg (*acc*), *zm*1, *zm*2 ; **if** *acc*=*zm*1 **then** *zm*1=*zm*2,
; *acc*=*zm*1

alternatywne podstawienie

if *warunek* **then** *X*=*A* **else** *X*=*B*

TRUE=00…01,	TRUE=11…11,	
FALSE=00…00	FALSE=00…00	
sub X, A, B	**sub** X, A, B	X:=A−B
set *not-war* Z	**set***war* Z	
sub X, A, B	—	Z:= 11…11 if *war*
and X, X, Z	**and** X, X, Z	X:=(A−B)&Z
add X, X, B	**add** X, X, B	X:=B+Z&(A−B)

Warunki – architektura CISC

			IA-32		MC 680x0
	Warunek	Funkcja	J*cc*	Funkcja	B*cc* / DB*cc*
status	*przeniesienie*	~CF / CF	**NC / C**	~C / C	**CC / CS**
	nadmiar U2	~OF / OF	**NO / O**	~V / V	**VC / VS**
	znak	~SF / SF	**NS / S**	~N / N	**PL / MI**
	zero	~ZF / ZF	**NZ / Z**	~Z / Z	**NE / EQ**
identyczność	≠ / =	~ZF / ZF	**NE / E**	~Z / Z	**NE / EQ**
porządek liczb	≻=	~CF	**AE ∣ NB**	~C	**CC***(HS)
naturalnych	≺	CF	**B ∣ NAE**	C	**CS***(LO)
	≻	~CF&~ZF	**A ∣ NBE**	~C&~Z	**HI**
	≺=	CF∨ZF	**BE ∣ NA**	C∨Z	**LS**
porządek liczb	≥	SF≡OF	**GE ∣ NL**	N≡V	**GE**
całkowitych	<	SF⊕OF	**L ∣ NGE**	N⊕V	**LT**
	>	~ZF&(SF≡OF)	**G ∣ NLE**	~Z(N≡V)	**GT**
	≤	ZF∨SF⊕OF	**LE ∣ NG**	Z∨N⊕V	**LE**

Uwaga: *Asembler Motoroli nie przewiduje notacji HS oraz LO dla tych instrukcji.

Warunki – architektura RISC

	MIPS R2000			**PowerPC 601**	
Warunek	**b***cc* rA, rB, adr	**bc[..]** CR*n*/*cc*, adr	CR*n*/*cc*	**tw***cc* rA, op2	*kod pułapki*
=	**eq / eqz**	**CR*n*/eq=T**	CR_{4n+2}	**eq**	00100
≠	**ne / nez**	**CR*n*/eq=F**	$\sim CR_{4n+2}$	**ne**	11000
≺	**ltu**	**CR*n*/lt=T**	CR_{4n+0}	**llt**	00010
≻=	**gequ**	**CR*n*/lt=F**	$\sim CR_{4n+0}$	**lge**	01010
≻	**gtu**	**CR*n*/gt=T**	CR_{4n+1}	**lgt**	00001
≺=	**lequ**	**CR*n*/gt=F**	$\sim CR_{4n+1}$	**lle**	00110
<	**lt / ltz**	**CR*n*/lt=T**	CR_{4n+0}	**lt**	10000
≥	**ge / gez**	**CR*n*/lt=F**	$\sim CR_{4n+0}$	**ge**	01100
>	**gt / gtz**	**CR*n*/gt=T**	CR_{4n+1}	**gt**	01000
≤	**le**	**CR*n*/gt=F**	$\sim CR_{4n+1}$	**le**	10100
nadmiar	**(trapv)**	**CR*n*/so=T**	CR_{4n+3}	—	—

Uwaga: W procesorze PowerPC 601 numer bitu CR odpowiadającego warunkowi CR*n*/*cond* określa pole BI kodu rozkazu, typ warunku *type* i reakcję na stan licznika CTR określa pole BO kodu rozkazu **bc**, zapisywanego też jako **bc[..]** BO, BI, adr.

Instrukcje warunkowe – architektura CISC

Wytworzenie warunku – aktualizacja rejestru flag / kodów warunkowych

- instrukcje arytmetyczne
- porównanie – jak odejmowanie
- instrukcje logiczne i przesunięcia – w ograniczonym zakresie
- stan rejestru zliczającego

	IA-32		**MC 680x0**	
Wytworzenie	**cmp** op1, op2; **(alu)** op1, op2;	*cc* → F *cc* → F	**CMP** op1, op2; **(ALU)** op1, op2;	*cc* → CCR *cc* → CCR
Rozgałęzienie	**j***cc* adres **loop** adres	*cc*(*F*) ? e(cx)=0	**B***cc* adres **DB***cc* D#, adres	*cc*(*CCR*) ? D#=0
Zapamiętanie	**set***cc* zmienna		**S***cc* zmienna	
Kopiowanie	**cmov***cc* op1, op2			
Pułapka	**into**		**TRAPV**	

Inne instrukcje –działanie gdy zgodność

- porównaj i wymień / dodaj
- testuj i ustaw

Instrukcje warunkowe – architektura RISC

Wytworzenie warunku – aktualizacja rejestru flag / kodów warunkowych

- porównanie przy założonej interpretacji kodu (NB / U2)
- instrukcje arytmetyczne – z ograniczeniami
- stan rejestru zliczającego

	MIPS R2000	**PowerPC**	
Wytworzenie	—	**cmp[l|i] crf**D, rA, op *ex*(**alu**) → XER / CR0 *ex*(**fpu**) → FXER / CR1	
Rozgałęzienie	**b***cc* op1, op2, adr	**bc[a][l]** *cond & type*, adr **bc[lr|ctr][l]** *cond & type*	/ **bc[a][l]** BO, BI, adr / **bc[lr|ctr][l]** BO, BI
Pułapka	**trapv**	**tw***cc*	

Inne instrukcje –działanie gdy zgodność

- porównaj i wyzeruj
- testuj i ustaw

Kompilacja instrukcji warunkowych (1)

rozgałęzienie

if (A>B **and** C<D) **then** *polecenie* **else** *inne*

MC 680x0*	IA-32	PowerPC 601	MIPS R2000
MOVE.L B, D#	**movl** A, %eax	**lwz** rA, r0, A	**lw** rA, A
CMP.L A, D#	**cmpl** B, %eax, B	**lwz** rB, r0, B	**lw** rB, B
BLE alt	**jle** alt	**lwz** rC, r0, C	**lw** rC, C
MOVE.L C, D#	**movl** D, %eax	**lwz** rD, r0, D	**lw** rD, D
CMP.L D#, D	**cmpl** C, %eax, B	**cmp** rA, rB	**ble** rA, rB, alt
BGE alt	**jge** alt	**bc** CR0/gt=F, alt	**bge** rC, rD, alt
polecenie	*polecenie*	**cmp** rC, rD	*polecenie*
BRA continue	**jmp** continue	**bc** CR0/lt=F, alt	**b** continue
alt:	alt:	*polecenie*	alt:
inne	*inne*	**b** continue	*inne*
continue:	continue:	alt:	continue:
		inne	
		continue:	

zmienne A-D w pamięci

Kompilacja instrukcji warunkowych (2)

ominięcie – **if** (A>B) **then** *polecenie*

MC 680x0*	IA-32	PowerPC 601	MIPS R2000
MOVE.L B, D#	**movl** A, %eax	**lwz** rA, r0, A	**lw** rA, A
CMP.L A, D#	**cmpl** B, %eax	**lwz** rB, r0, B	**lw** rB, B
BLE alt	**jle** alt	**cmp** rA, rB	**ble** rA, rB, alt
polecenie	*polecenie*	**bc** CR0/gt=F, alt	*polecenie*
alt:	alt:	*polecenie*	alt:
		alt:	

przypisanie warunku – B:= (X ≤ V) AND (Z > Y)

MC 680x0	IA-32	PowerPC 601	MIPS R2000
MOVE.L X, D#	**movl** X, %eax	**xor** rB, rB, rB	**sle** rB, rX, rV
CMP.L V, D#	**cmpl** V, %eax	**cmp** rX, rV	**sgt** rC, rZ, rY
SLE B	**setle** B	**bc** CR0/gt=T, hop	**and** rB, rB, rC
MOVE.L Z, D#	**movl** Z, %eax	**cmp** rZ, rY	
CMP.L Y, D#	**cmpl** Y, %eax	**bc** CR0/gt=F, hop	
SGT D#	**setgt** %ebx	**nand** rB, rB, rB	
AND.L D#, B	**andl** B, %ebx	hop:	

Kompilacja instrukcji warunkowych (3)

alternatywne podstawienie

if (A>B) **then** $X=P$ **else** $X=Q$

MC 680x0*	IA-32	PowerPC 601	MIPS R2000
MOVE A, D1	**movl** A, %eax	**lwz** rA, r0, A	**lw** rA, A
CMP D1, B	**cmp** B, %eax	**lwz** rB, r0, B	**lw** rB, B
SLE D2	**setle** %ebx	**subf** rZ, rB, rA	**sle** rZ, rA, rB
MOVE Q, D3	**decl** %ebx	**subfe** rZ, rZ, rZ	**lw** rP, P
MOVE P, D4	**movl** Q, %ecx	**lwz** rQ, r0, Q	**lw** rQ, Q
SUB D3, D4	**movl** P, %edx	**lwz** rP, r0, P	**sub** rX, rP, rQ
AND D2, D4	**subl** %ecx, %edx	**subf** rX, rP, rQ	**and** rX, rZ, rX
ADD D3, D4	**andl** %ebx, %edx	**and** rX, rZ, rX	**add** rX, rQ, rX
	addl %edx, %ecx	**add** rX, rQ, rX	
	lub		
	movl Q, %ecx		
	movl A, %eax		
	cmpl B, %eax		
true = 11…11 (–1)	**cmovgt** P, %edx		
false = 00…00	*true* = 00…01 (+1)	*false* = 00…00	

Kompilacja instrukcji warunkowych (4)

wybór wielowariantowy

case $i, n, \{(i \leq n) \Rightarrow$ *polecenie* $[i], (i > n) \Rightarrow$ *polecenie* $[0]\}$

MC 680x0	**IA-32**	**PowerPC 601**	**MIPS R2000**
CMPA.L A#, *N* **BLS** sel **SUBA.L** A#, A# sel: **LSL** A#, 2 **JSR** (A#)	**cmpl** *N*, %ebx **jbe** sel **xorl** %ebx, %ebx sel: **shl** 2, %ebx **call** (%ebx))* *skok odległy (far)*	**xor** rB, rB, rB **ori** rB, rB, 2 **cmpu** rA, *N* **bc** CR0/gt=F, sel **xor** rA, rA, rA sel: **sle** rA, rA, rB **ld** rC, 0, rA **mtspr** LR, rC **bclr**	**bgtu** rA, N, case0 **beq** rA, 1, case1 ... **beq** rA, N, caseN

IA-32: 80386 … Pentium IV: skok pośredni (wyznaczony): jmp [bx] / call [bx]
Nie daje się prognozować, należy go unikać!

Kompilacja instrukcji warunkowych (5)

instrukcja pętli indeksowanej

for *i*:= *st* **step** –1 **until** *end* **do** *polecenie* (*i*)

MC 680x0	**IA-32**	**PowerPC 601**	**MIPS R2000**
MOVE.L D#, *end*	**movl** *end*, %ecx	**xor** rC, rC, rC	**li** rX, 1
SUB.L *st*, D#	**subl** *st*–1, %ecx	**ori** rC, rC, *end*–*st*	**li** rC, *end*–*st*
powt:	powt:	**mtspr** CTR, rC	powt:
polecenie (*i*)	*polecenie* (*i*)	powt:	*polecenie* (*i*)
i:=*i*+1	*i*:=*i*+1	*polecenie* (*i*)	*i*:=*i*+1
DBF D#, powt	**loop** powt	*i*:=*i*+1	**sub** rC, rC, rX
		bc CTR≠0, powt	**bgtz** rC, powt

Uwaga: Założono, że wszystkie parametry są liczbami całkowitymi.

Kompilacja instrukcji warunkowych (6)

uogólniona instrukcja pętli indeksowanej

for *i*:= *st* **step** *kr* **until** *end* **do** *polecenie* (*i*)

MC 680x0	**IA-32**	**PowerPC 601**	**MIPS R2000**
MOVE.L *st*, D#	**movl** *st*, %ebx	**xor** rC, rC, rC	**li** rX, 1
MOVE CCR, SP	**pushf**	**ori** rC, rC, *end–st*	**li** rC, *end–st*
powt:	powt:	**div** rC, *kr*	**div** rC, *kr*
MOVE SP, CCR	**popf**	**mtspr** CTR, rC	powt:
Polecenie (D#)	*polecenie* (%ebx)	powt:	*polecenie* (*i*)
MOVE CCR, SP	**pushf**	*polecenie* (*i*)	**sub** rC, rC, rX
ADD.L *kr*, D#	**addl** *kr*, %ebx	**bc** CTR≠0, powt	**bgtz** rC, powt
CMP.L D#, *end*	**cmpl** *end*, %ebx		
BLE powt	**jgt** powt		
MOVE SP, CCR	**popf**		

Kompilacja instrukcji warunkowych (7)

instrukcja warunkowego powtórzenia

repeat *polecenie* (*A*, *B*) **until** *A*>*B*

MC 680x0	**IA-32**	**PowerPC 601**	**MIPS R2000**
start: *polecenie* (*A*, *B*) **CMPM.L** A, B **BGT** start	start: *polecenie* (*A*, *B*) **movl** A, %eax **cmpl** B, %eax **jgt** start	start: *polecenie* (*A*, *B*) **cmp** rA, rB **bc** CR0/gt=T, start	start: *polecenie* (*A*, *B*) **bgt** rA, rB, start

Kompilacja instrukcji warunkowych (8)

instrukcje powtórzenia z warunkiem zakończenia

while *A*>*B* **do** *polecenie* (*A, B*)

MC 680x0	**IA-32**	**PowerPC 601**	**MIPS R2000**
start: **CMPM.L** *A, B* **BLE** skip *polecenie* (*A, B*) **BRA** start skip:	start: **movl** A, %eax **cmpl** B, %eax **jle** skip *polecenie* (*A, B*) **jmp** start skip:	start: **cmp** rA, rB **bc** CR0/gt=F, skip *polecenie* (rA, rB) **b** start skip:	start: **bgt** rA, rB, skip *polecenie* (rA, rB) **b** start skip:

Funkcje i procedury

Funkcja – działanie złożone, którego wynikiem jest wartość

- *uaktywnienie funkcji – wywołanie* (*function call*)
- *zakończenie – zwrot sterowania* (*return*) *do miejsca wywołania*
- *parametry formalne i parametry bieżące* (*aktualne*), *przekazywane*
  - przez odniesienie (*by reference*) – wskaźnik obiektu
  - przez wartość (*by value*) – obiekt

Procedura – funkcja, która jawnie nie zwraca wartości.

	FUNKCJA	ROZKAZ
parametry	lista zmiennych	lista argumentów
formalne	nazwy zmiennych	nazwy rejestrów i adresy zmiennych
aktualne	wartości	zawartość rejestrów i zmiennych
przekazanie	umieszczenie na liście	tryb adresowania
odniesienie	nazwa	adresowanie pośrednie
wartość	wartość	adresowanie bezpośrednie
uaktywnienie	powiązania	—
	wywołanie	wykonanie
zakończenie	return	PC++

Powiązania i aktywacja

Powiązania (*binding*):

- parametrów formalnych z parametrami faktycznymi (ang. *actual*)
- parametrów faktycznych (bieżących) z wartościami
- nazw stałych z wartościami i nazw zmiennych z lokacjami (adresami).

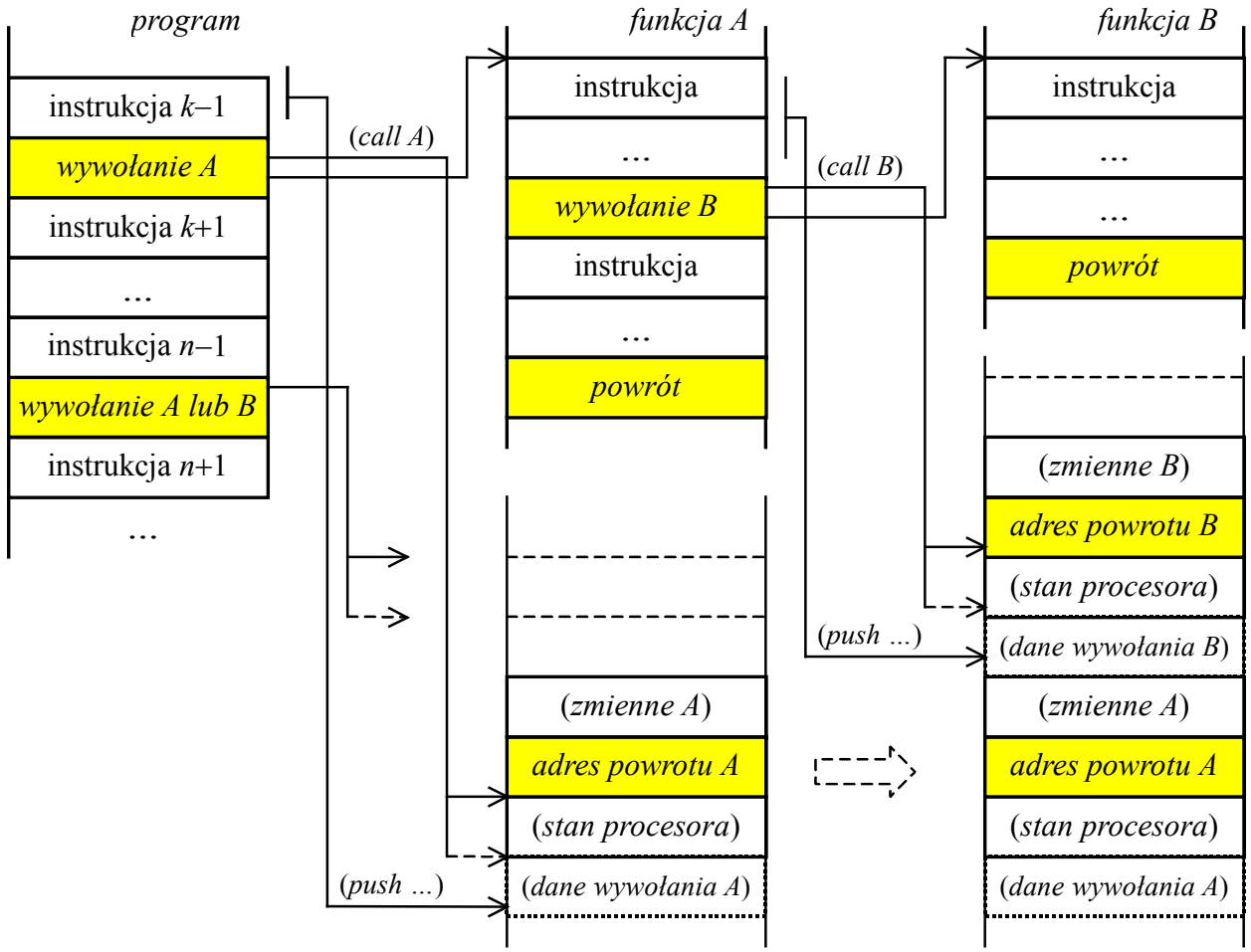
Środowisko wykonania (*run-time environment*), *kontekst* funkcji:
blok aktywacji (*activation record*).

- parametry *bezpośrednie* (*explicit*) przekazane przez funkcję wywołującą
- parametry *implikowane* (*implicit*) tworzone automatycznie
- dane lokalne (*local data*) lub wskaźniki danych strukturalnych.

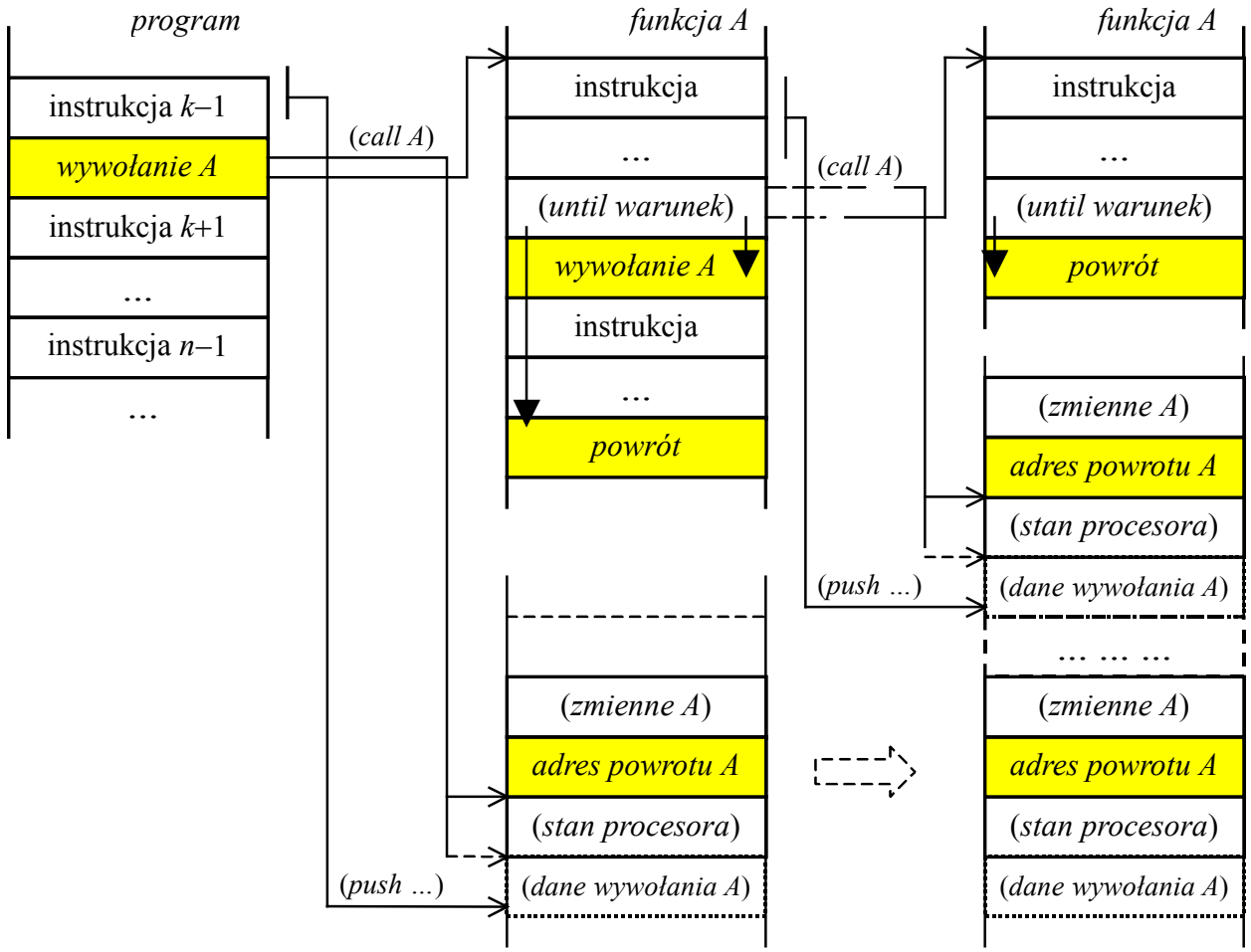
Alokacja bloku aktywacji

- *alokacja statyczna* – podczas kompilacji (wyklucza rekurencję, także pośrednią, oraz jednoczesne udostępnienie funkcji różnym procesom)
  - *powiązania statyczne* (*leksykalne*) – podczas kompilacji (*wczesne*)
- *alokacja dynamiczna* – na czas wykonania, unieważniana po zakończeniu
  - *powiązania dynamiczne* – podczas wykonywania (tylko zmienne)
    - *przysłonięcie* (*shadowing*) zmiennych – zmienne lokalne
    - *nakładanie* (*overlapping*) zmiennych – zmienne robocze

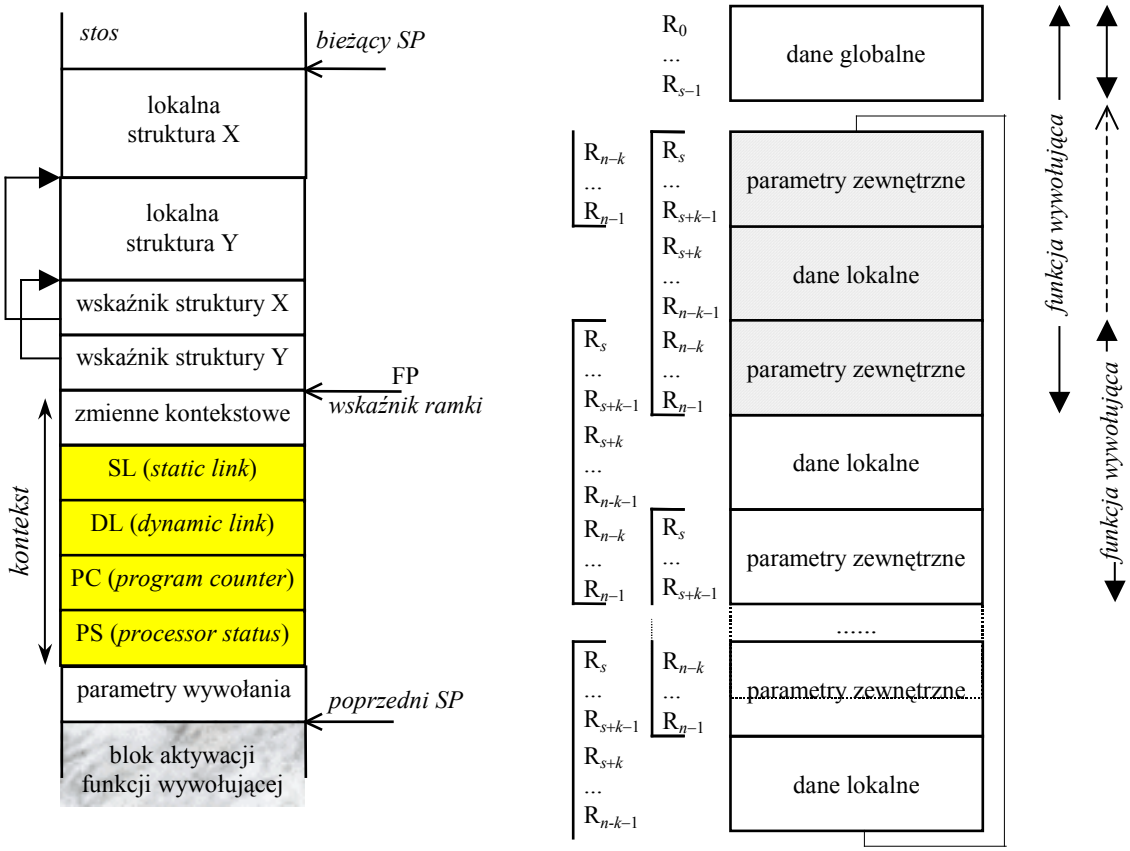
Wywołanie i zagnieżdżanie funkcji (procedury)



Rekurencja

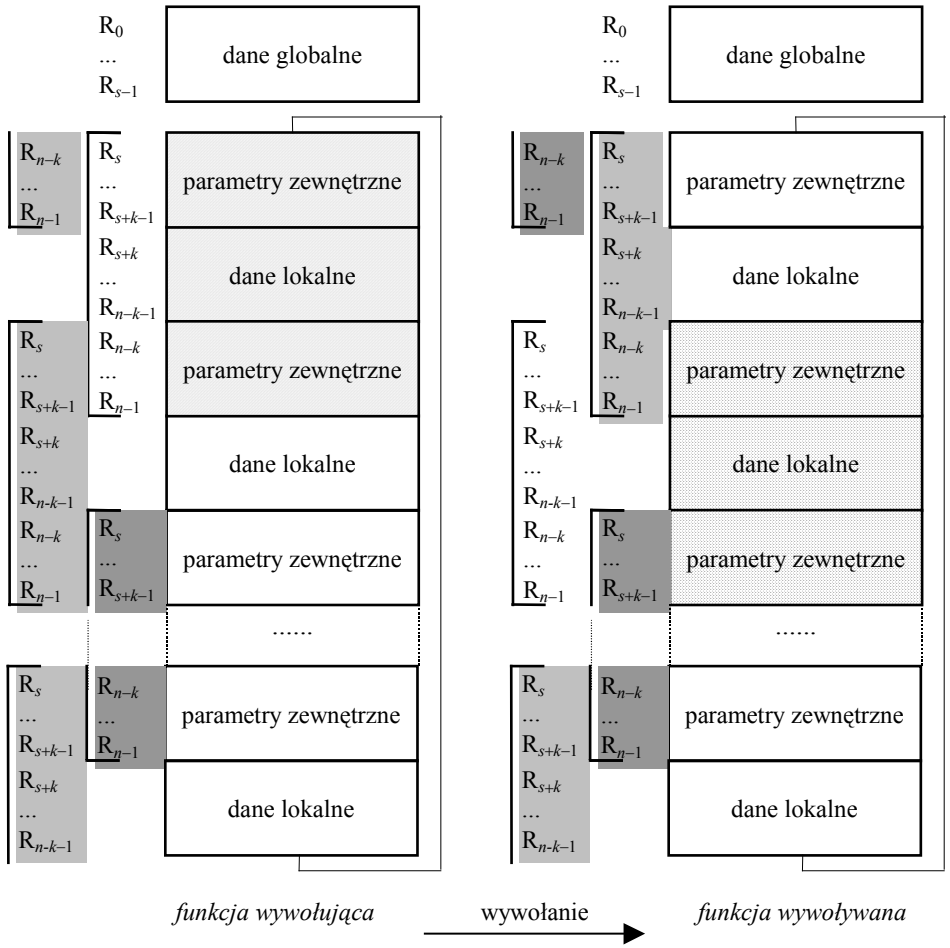

program
instrukcja k–1
wywołanie A
instrukcja k+1
…
instrukcja n–1
…
(call A)
(push …)
funkcja A
instrukcja
…
(until warunek)
wywołanie A
instrukcja
…
powrót
(zmienne A)
adres powrotu A
(stan procesora)
(dane wywołania A)
funkcja A
instrukcja
…
(until warunek)
powrót
(call A)
(push …)
(zmienne A)
adres powrotu A
(stan procesora)
(dane wywołania A)
… … …
(zmienne A)
adres powrotu A
(stan procesora)
(dane wywołania A)

Kontekst funkcji i blok aktywacji



Blok aktywacji: a) w obszarze stosu, b) okna rejestrowe (SL – powiązanie do zmiennych statycznych, DL – powiązanie do poprzedniego bloku akt.)

Blok aktywacji w oknie rejestrowym



Konwencje wywołania

Konwencja wywołania
– sposób przekazywania parametrów do i z funkcji (ang. *calling conventions*)
Proces przekazywania parametrów – powiązanie procedury (ang. *subroutine linkage*)

Zagnieżdżanie funkcji
– rejestr powiązań (ang. *link register*) lub adres powrotu na stosie

Konwencje:
– przez rejestry procesora
metoda szybka, liczba parametrów ograniczona
– przez obszar powiązania danych (ang. *subroutine linkage*)
adres obszaru przekazywany przez rejestr procesora
– przez stos programowy
w obszarze stosu tworzona ramka dla parametrów (kontekst/blok aktywacji)
adresowana wskaźnik ramki (ang. *frame pointer*)

Mechanizm okien rejestrowych – przekazywanie parametrów i powiązania przez okno rejestrowe

Przekazywanie parametrów

Blok aktywacji na stosie:

- zmienne przekazywane do funkcji
  - wskaźniki (parametry przekazywane przez referencję – adresy)
  - wartości (parametry przekazywane przez wartość – stałe wywołania)
- kontekst stabilny (chroniony podczas wywołania)
  - stan (wybranych) rejestrów procesora
- kontekst dynamiczny (określany podczas wywołania)
  - słowo stanu procesora i stan licznika rozkazów (adres powrotu)
  - wskaźnik powiązań statycznych (poprzedni wskaźnik stosu)
  - wskaźnik powiązań dynamicznych (zagnieżdżanie i rekurencja)
- wskaźniki lokalnych struktur danych
  - rozmiar i liczba zmiennych struktury

Okna rejestrowe

- CALL / RET $\rightarrow$ przełączenie okna – nie ma dynamicznej zmiany kontekstu
- parametry zewnętrzne – zmienne przekazywane
- parametry lokalne – zmienne robocze i wskaźniki lokalnych struktur
- ograniczona liczba poziomów wywołania

Korzyści z mechanizmu funkcji i problemy

Korzyści:

- *redukcja rozmiaru kodu (usunięcie powtarzalnych fragmentów programu)*
- *redukcja zapotrzebowania na pamięć*
- *ukrycie szczegółów implementowanego algorytmu i struktury danych*
  - możliwość modyfikacji algorytmu bez zmiany sposobu użycia
- *łatwa implementacja wyższego poziomu abstrakcji – maszyny wirtualnej*
  - funkcja = makrorozkaz $\rightarrow$
    $\rightarrow$ lista makrorozkazów = architektura witrualna

Problemy:

- *naruszenie sekwencyjności rozkazów podczas wywołania*
- *nieprzewidywalność lokalizacji kolejnego rozkazu podczas powrotu*

!! *UWAGA:*

Funkcja – makrorozkaz, złożone polecenie wykonywane przez procesor

Makro – tekstowy opis w pliku źródłowym (*makrodefinicja*), przetwarzany przez kompilator (*makrowywołanie*) na sekwencję instrukcji podczas generowania kodu