Organizacja i architektura komputerów [1]

Wykład 8

Piotr Patronik

17 kwietnia 2015

[1] (Prawie) dokładna kopia slajdów dr hab inż. J. Biernata

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
- identyfikacja warunków
- wybór ścieżki przetwarzania

Przepływ sterowania

program = sekwencja instrukcji $\longrightarrow$ powiązanie instrukcji

- funkcjonalne – jaka jest następna instrukcja
  konstrukcje: repeat (powtarzaj), execute (wykonaj).
- lokacyjne – gdzie jest następna instrukcja
  - sekwencyjne (sequential) – wyklucza sterowanie,
    - domniemana lokacja kolejnego rozkazu (porządek liniowy)
      to krótszy kod
  - łańcuchowe (chained) – umożliwia sterowanie,
    - konieczne wskazanie lokacji kolejnego rozkazu
      $\longrightarrow$ dłuższy kod (musi zawierać wskazanie kolejnego rozkazu)

Kompromis:

- 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 zdekomponowana na:

- wybór jednej spośród rozłącznych możliwych opcji
  $AA \cap BB \cap \ldots \cap NN = \oslash$, $AA \cup BB \cup \ldots \cup NN = QQ$
  będących skutkiem spełnienia jednej z rozłącznych przesłanek
  $WA \cap WB \cap \ldots \cap WN = \oslash$, $WA \cup WB \cup \ldots \cup WN = \Omega$
- sekwencję rozłącznych decyzji binarnych:

jeśli warunek WA∪WB∪…∪WG **wtedy**
… …
… **jeśli** warunek WA∪WB **wtedy**
jeśli warunek WA **podejmij decyzję** AA
jeśli warunek WB **podejmij decyzję** BB
… **jeśli** warunek WC∪WD **wtedy** …
jeśli warunek WC **podejmij decyzję** CC
jeśli warunek WD **podejmij decyzję** DD
jeśli warunek WH∪WI∪…∪WN **wtedy** …

Rozgałęzienia

decyzje $\longrightarrow$ zmiana porządku instrukcji

- wytworzenie warunku – porównanie, wykonanie działania
- wybór warunku – jawne lub implikowane wskazanie przesłanki
- użycie warunku – rozkazy warunkowe
  - jawne wykonanie rozgałęzienia (instrukcja), ominięcie
    - rozgałęzienie zwykłe (branch), skok warunkowy
      if *warunek* **then goto** *adres*
    - rozgałęzienie ze śladem (branch & link)
      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 (spełniony – niespełniony) **if** *warunek* **then** zmienna:= *TRUE* **else** zmienna:= *FALSE*

Techniki wykonywania rozgałęzień

binarne drzewo decyzyjne

- każde rozgałęzienie można wykonać jako sekwencję działań alternatywnych

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 (Pentium)

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, FALSE=00…00	TRUE=11…11, 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

		Intel 80x86 / Pentium		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	$\uparrow$ / =	~ZF / ZF	**NE / E**	~Z / Z	**NE / EQ**
porządek liczb	$\succ=$	~CF	**AE | NB**	~C	**CC***(HS)
naturalnych	$\prec$	CF	**B | NAE**	C	**CS***(LO)
	$\succ$	~CF&~ZF	**A | NBE**	~C&~Z	**HI**
	$\prec=$	CF$\vee$ZF	**BE | NA**	C$\vee$Z	**LS**
porządek liczb	$\geq$	SF$\equiv$OF	**GE | NL**	N$\equiv$V	**GE**
całkowitych	$<$	SF$\oplus$OF	**L | NGE**	N$\oplus$V	**LT**
	$>$	~ZF&(SF$\equiv$OF)	**G | NLE**	~Z(N$\equiv$V)	**GT**
	"	ZF$\vee$SF$\oplus$OF	**LE | NG**	Z$\vee$N$\oplus$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 CRn/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

	Intel 80x86/Pentium		**MC 680x0**	
Wytworzenie	**cmp** op1, op2; **(alu)** op1, op2;	$cc \rightarrow$ F $cc \rightarrow$ F	**CMP** op1, op2; **(ALU)** op1, op2;	$cc \rightarrow$ CCR $cc \rightarrow$ 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 CISC (2)

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)

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

MC 680x0*	**Intel x86/ Pentium**	**PowerPC 601**	**MIPS R2000**
MOVE.L B, D#	**mov** eax, A	**lwz** rA, r0, A	**lw** rA, A
CMP.L A, D#	**cmp** eax, B	**lwz** rB, r0, B	**lw** rB, B
BLE alt	**jle** alt	**lwz** rC, r0, C	**lw** rC, C
MOVE.L C, D#	**mov** eax, D	**lwz** rD, r0, D	**lw** rD, D
CMP.L D#, D	**cmp** eax, C	**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*	Intel x86/ Pentium	PowerPC 601	MIPS R2000
MOVE.L B, D#	**mov** eax, A	**lwz** rA, r0, A	**lw** rA, A
CMP.L A, D#	**cmp** eax, B	**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 – $B := (X \leqslant V)$ AND $(Z > Y)$

MC 680x0	Intel x86/ Pentium	PowerPC 601	MIPS R2000
MOVE.L X, D#	**mov** eax, X	**xor** rB, rB, rB	**sle** rB, rX, rV
CMP.L V, D#	**cmp** eax, V	**cmp** rX, rV	**sgt** rC, rZ, rY
SLE B	**setle** B	**bc** CR0/gt=T, hop	**and** rB, rB, rC
MOVE.L Z, D#	**mov** eax, Z	**cmp** rZ, rY	
CMP.L Y, D#	**cmp** eax, Y	**bc** CR0/gt=F, hop	
SGT D#	**setgt** ebx	**nand** rB, rB, rB	
AND.L D#, B	**and** B, ebx	hop:	

Kompilacja instrukcji warunkowych (3)

alternatywne podstawienie

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

MC 680x0*	Intel x86/ Pentium	PowerPC 601	MIPS R2000
MOVE A, D1	**mov** eax, A	**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	**dec** ebx	**subfe** rZ, rZ, rZ	**lw** rP, P
MOVE P, D4	**mov** ecx, Q	**lwz** rQ, r0, Q	**lw** rQ, Q
SUB D3, D4	**mov** edx, P	**lwz** rP, r0, P	**sub** rX, rP, rQ
AND D2, D4	**sub** edx, ecx	**subf** rX, rP, rQ	**and** rX, rZ, rX
ADD D3, D4	**and** edx, ebx	**and** rX, rZ, rX	**add** rX, rQ, rX
	add ecx, edx	**add** rX, rQ, rX	
	lub		
	mov ecx, Q		
	mov eax, A		
	cmp B, eax		
	cmovgt edx, P		
true = 11…11 (–1)			
false = 00…00	*true* = 00…01 (+1) *false* = 00…00		

Kompilacja instrukcji warunkowych (4)

wybór wielowariantowy

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

MC 680x0	**Intel x86/ Pentium**	**PowerPC 601**	**MIPS R2000**
CMPA.L A#, *N* **BLS** sel **SUBA.L** A#, A# sel: **LSL** A#, 2 **JSR** (A#)	**cmp** bx, *N* **jbe** sel **xor** bx, bx sel: **shl** bx, 2 / 3* **call** ds:[bx])* *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

Kompilacja instrukcji warunkowych (5)

instrukcja pętli indeksowanej

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

MC 680x0	**Intel x86/ Pentium**	**PowerPC 601**	**MIPS R2000**
MOVE.L D#, *end*	**mov** cx, *end*	**xor** rC, rC, rC	**li** rX, 1
SUB.L *st*, D#	**sub** cx, *st*−1	**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

Kompilacja instrukcji warunkowych (6)

uogólniona instrukcja pętli indeksowanej

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

MC 680x0	**Intel x86/ Pentium**	**PowerPC 601**	**MIPS R2000**
MOVE.L *st*, D#	**mov** ebx, *st*	**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#	**add** ebx, *kr*	**bc** CTR↑0, powt	**bgtz** rC, powt
CMP.L D#, *end*	**cmp** *end*, ebx		
BLE powt	**jle** powt	????	
MOVE SP, CCR	**popf**		

Kompilacja instrukcji warunkowych (7)

instrukcje pętli uwarunkowanej

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

MC 680x0	**Intel x86/ Pentium**	**PowerPC 601**	**MIPS R2000**
start: *polecenie* (*A*, *B*) **CMPM.L** A, B **BGT** start	start: *polecenie* (*A*, *B*) **mov** eax, A **cmp** eax, B **jgt** start	start: *polecenie* (*A*, *B*) **cmp** rA, rB **bc** CR0/gt=T, start	start: *polecenie* (*A*, *B*) **bgt** rA, rB, start

while $a > B$ **do** *polecenie*

MC 680x0	**Intel x86/ Pentium**	**PowerPC 601**	**MIPS R2000**
start: **CMPM.L** *A*, *B* **BLE** skip *polecenie* (*A*, *B*) **BRA** start skip:	start: **mov** eax, *B* **cmp** eax, *A* **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 – makrorozkaz, 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 aktualnymi
- parametrów aktualnych z wartościami
- nazw stałych z wartościami i nazw zmiennych z lokacjami (adresami)

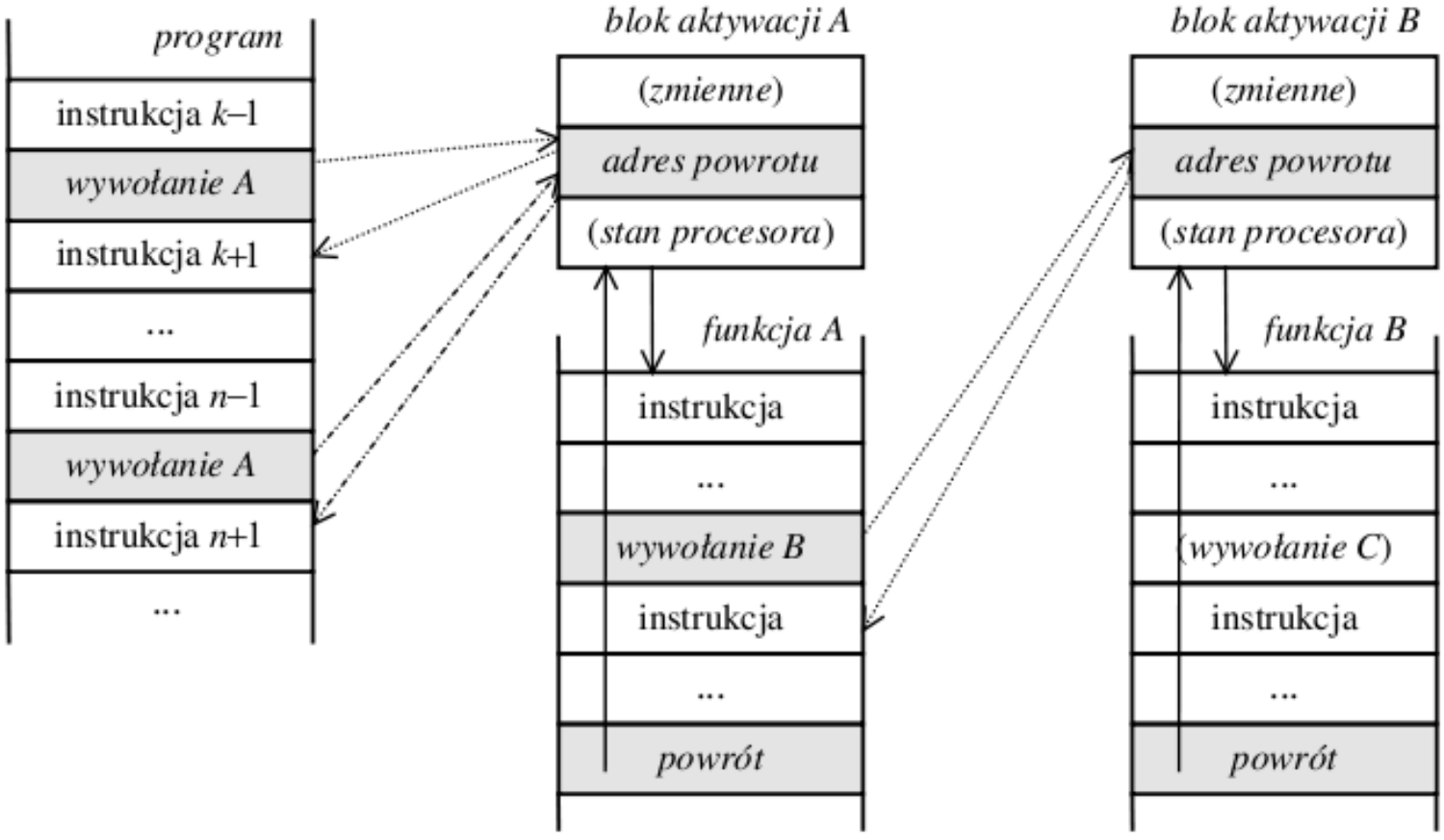
Środowisko wykonania (run-time environment), kontekst funkcji:
blok aktywacji (activation record).

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

Alokacja bloku aktywacji

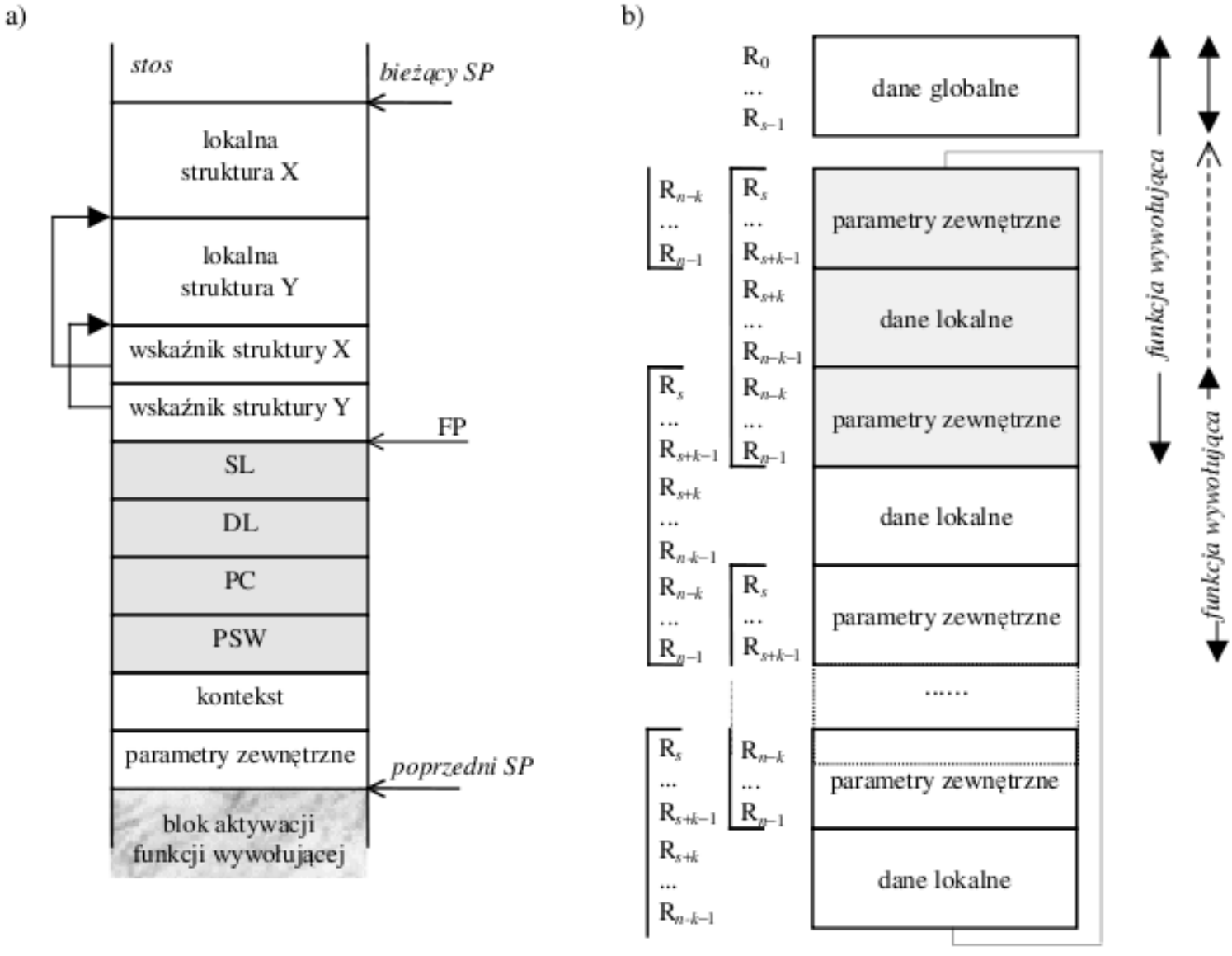
- *alokacja statyczna* – podczas kompilacji (wyklucza rekurencję, także pośrednią, oraz jednoczesne udostępnienie funkcji różnym procesom)
  - *powiązania statyczne (leksykalne)* – podczas kompilacji (*wczesne*)
- *alokacja dynamiczna* – na czas wykonania, unieważniana po zakończeniu
  - powiązania dynamiczne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

Wywołanie i zagnieżdżanie funkcji (procedury)



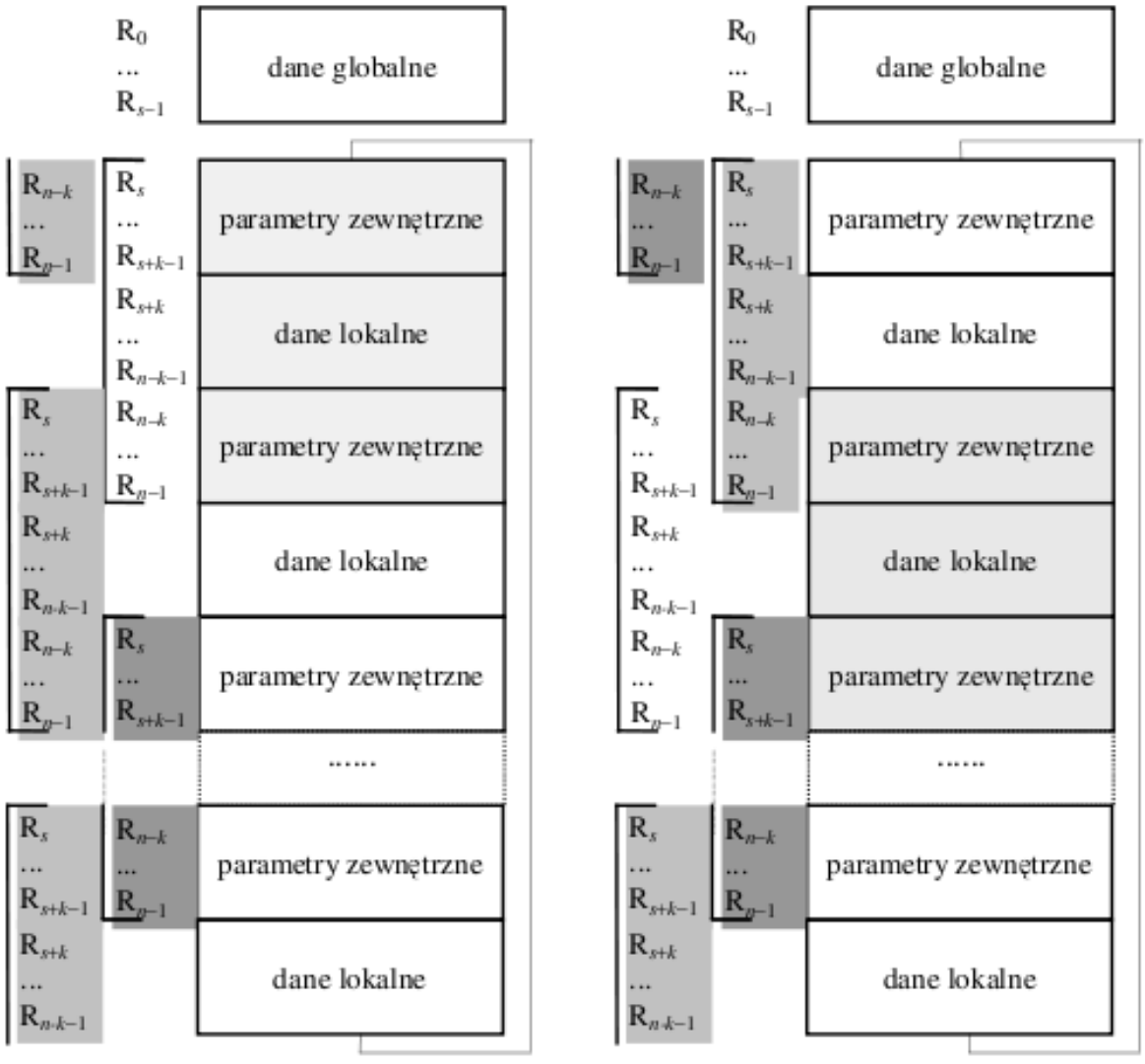
- Powiązania funkcji
  - zagnieżdżanie (nesting) – wywołanie funkcji przez inną funkcję
  - rekurencja (reccurence) – wywołanie funkcji przez nią samą – każde wywołanie ma własny blok aktywacji

Kontekst funkcji i blok aktywacji



Blok aktywacji: a) w obszarze stosu, b) okna rejestrowe

Blok aktywacji w oknie rejestrowym



funkcja wywołująca (wywołanie) $\longrightarrow$ funkcja wywoływana

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 (nie ulegający zmianie podczas wywołania)
    - stan 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 $\longrightarrow$ przełączenie okna – nie ma dynamicznej zmiany kontekstu
  - parametry zewnętrzne – zmienne przekazywane
  - parametry lokalne – zmienne robocze i wskaźniki lokalnych struktur
  - ograniczona liczba poziomów wywołania

Korzyści 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 $\longrightarrow$
      $\longrightarrow$ lista makrorozkazów = architektura wirtualna
- 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