

Programowanie systemowe

Dr inż. Zbigniew Lach (pokój E414, budynek Wydziału
Elektrotechniki i Informatyki)

konsultacje:

wtorek, 10:15-11:00

środa, 10:15-11:00

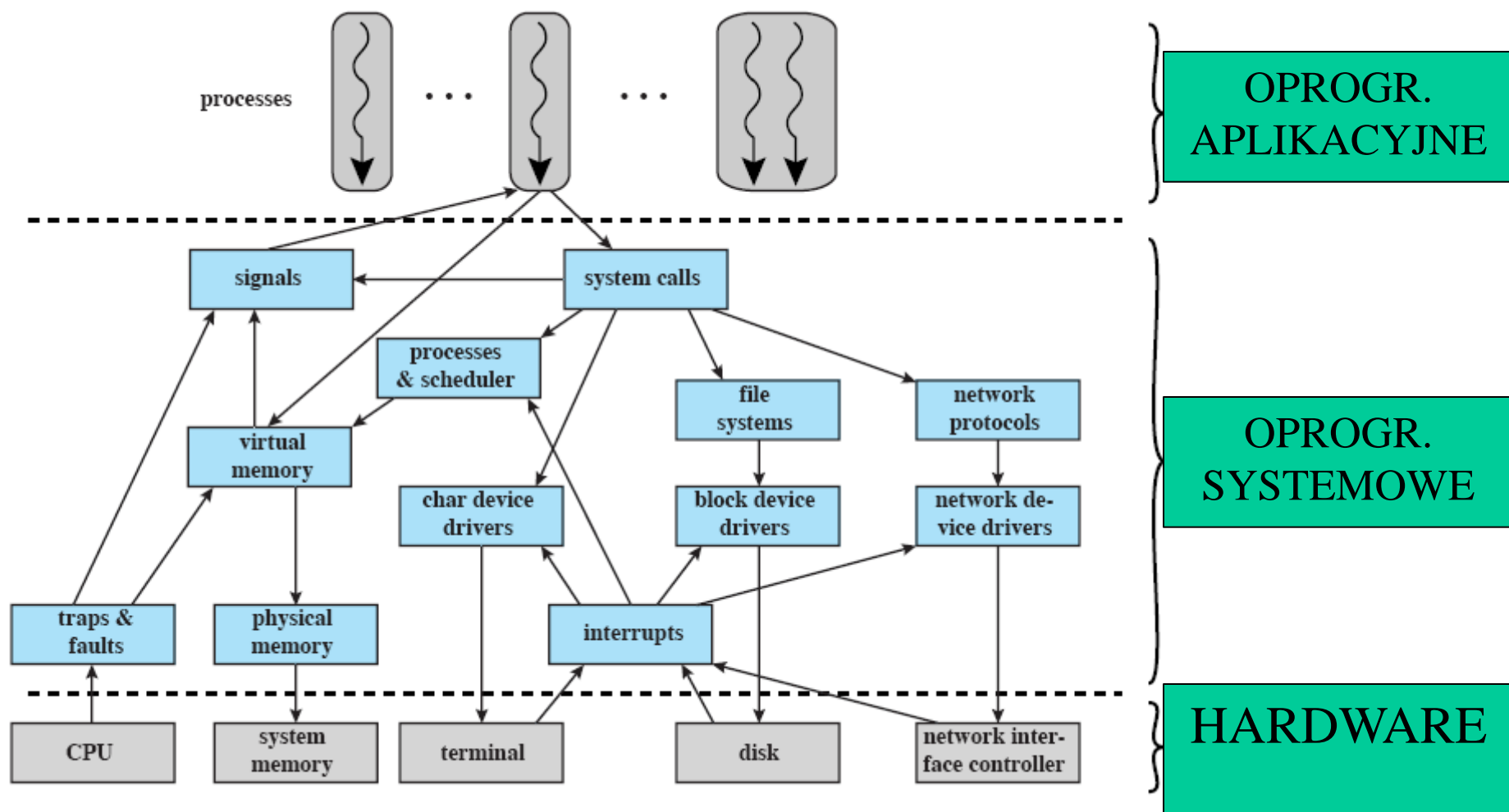
System komputerowy

OPROGRAMOWANIE
UŻYTKOWE

OPROGRAMOWANIE
SYSTEMOWE

HARDWARE

System komputerowy



Programowanie systemowe = rodzaj programowania systemów oprogramowania

oprogramowanie systemowe



- realizuje funkcje konieczne dla działania systemu komputerowego (świadczy usługi sprzętowi komputerowemu)
- w szczególności: steruje i koordynuje pracę komputera zapewniając jego funkcjonalność, zwłaszcza możliwość wykonywania programów aplikacyjnych
- w skład o.s. wchodzi każde oprogramowanie, z którym użytkownik nie ma bezpośredniej styczności
- pojęcie szersze niż systemy operacyjne

Specyfika:

Programista **uwzględnia i aktywnie wykorzystuje cechy sprzętu i inne właściwości systemu komputerowego**, na którym program jest uruchomiony

- o język niskiego poziomu
- o środowisko o limitowanych zasobach
- o bezpośredni i surowy dostęp do pamięci, urządzeń we/wy i kontroli wykonania
- o ma małą bibliotekę uruchomieniową (albo wcale)



→ Program systemowy powinien być bardzo efektywny i mieć małe narzuty uruchomieniowe

Program wykładu

Cechy i mechanizmy języków asemblera

Implementacja struktur sterowania programem

Implementacja działań arytmetycznych i logicznych

Programowanie proceduralne

Obsługa wejścia wyjścia

Programowanie obsługi wyjątków

Programy relokowalne

Programowanie modułowe

Elementy sterowania procesami



Literatura podstawowa:

- 1. W. Stanisławski, D. Raczyński „Programowanie systemowe mikroprocesorów rodziny x86”, PWN Warszawa 2010**
- 2. A. Silberschatz. P.B. Galwin, "Podstawy systemów operacyjnych", WNT Warszawa 2000**
- 3. Gary Syck, "Turbo Assembler. Biblia użytkownika" LT&P, Warszawa 1994**
- 4. W. Stallings, "Organizacja i architektura systemu komputerowego", WNT, Warszawa 2000**

Zaliczenie przedmiotu: min. 50% punktów z dwóch kolokwii:

(I) 27 kwietnia

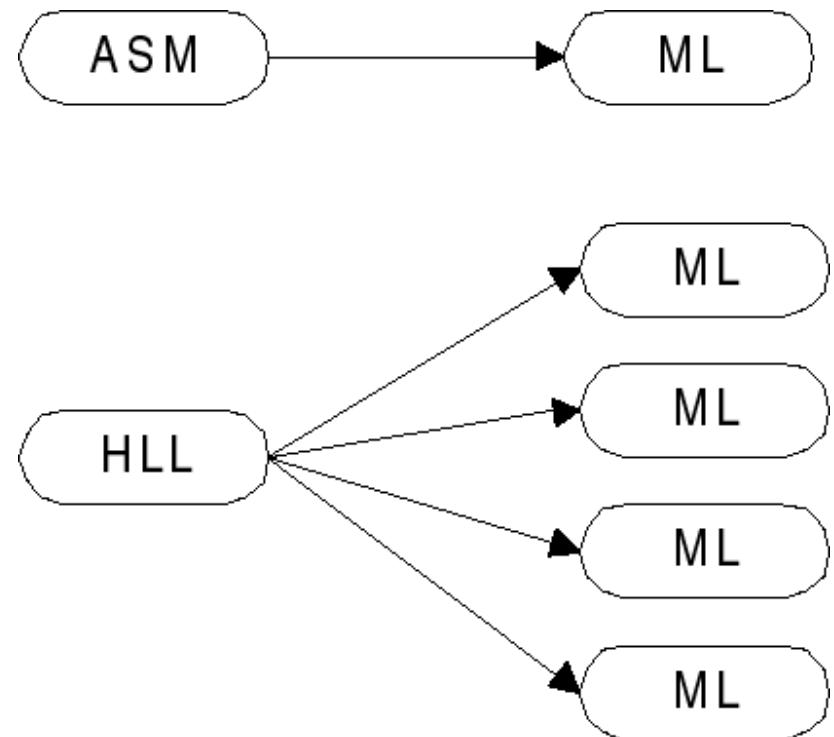
(II) 15 czerwca

Wykład 1

Podstawowe mechanizmy języków asemblera

Język asemblera

- **Niskopoziomowy język programowania związany z konkretnym procesorem ściśle odpowiadający zestawowi instrukcji języka maszynowego**
- **Każda instrukcja ASM odpowiada dokładnie jednej instrukcji ML**



Język asemblera

Poziom	Opis
Asembler (ASM)	Stosuje mnemoniki instrukcji <u>bezpośrednio odpowiadające</u> rozkazom języka maszynowego
Język Maszynowy (ML)	Instrukcje numeryczne i operandy, które mogą być umieszczone w pamięci komputera i bezpośrednio wykonywane przez procesor

Język maszynowy

1	0097	LED	EQU 97H
2	0000: 02 01 00		LJMP START
3	0100:		ORG 100H
4	0100: B2 97	START:	CPL LED
5	0102: 74 0A		MOV A,#10
6	0104: 12 01 09		LCALL DELAY
7	0107: 80 F7		SJMP START
8	0109: 78 FF	DELAY:	MOV R0,#0FFH
9	010B: D8 FE	LOOP:	DJNZ R0, LOOP
10	010D: D5 E0 F9		DJNZ ACC, DELAY
11	0110: 22		RET

Zawartość pliku listingu przykładowego programu

Język asemblera – przykład Atmel AVR

Mnemonic	Description	Mnemonic	Description	Mnemonic	Description
Flow Control		Bit Manipulation		Load/Store	
JMP ◆	Jump absolute (24-bit)	SEC/CLC	Set/clear C flag (carry)	MOV	Copy register to register
RJMP	Branch relative (12-bit)	SEH/CLH	Set/clear H flag (half carry)	LD	Load indirect through X/Y/Z
IJMP ●	Jump indirect (Z)	SEN/CLN	Set/clear N flag (negative)	LD ●	Load indirect with postincrement
RCALL	Call subroutine	SEZ/CLZ	Set/clear Z flag (zero)	LD ●	Load indirect with predecrement
ICALL ●	Call subroutine indirect (Z)	SEI/CLI	Set/clear I flag (interrupt)	LDD ●	Load indirect with 6-bit offset
RET/RETI	Return/from interrupt	SES/CLS	Set/clear S flag (sign)	LDI	Load 8-bit immediate
CP/CPC	Compare/with carry	SEV/CLV	Set/clear V flag (overflow)	LDS ●	Load from 16-bit address
CPI	Compare with 8-bit immediate	SET/CLT	Set/clear T bit	LPS ●	Load from program space
CPSE	Compare, skip if equal	SBR/CBR	Set/clear bit in register	ST	Store indirect through X/Y/Z
SBRs/SBRC	Skip if register bit set/clear	BSET/BCLR	Set/clear bit in status register	ST ●	Store indirect with postincrement
SBIS/SBIC	Skip if I/O bit set/clear	SER/CLR	Set/clear entire register	ST ●	Store indirect with predecrement
BRcc	Conditional branch	SBI/CBI	Set/clear bit in I/O space	STD ●	Store indirect with 6-bit offset
Logical		Arithmetic		STS ●	Store to 16-bit address
AND	Logical AND	ADD/ADC	Add/with carry	IN/OUT	Input/output to/from I/O space
ANDI	Logical AND 8-bit immediate	ADIW ●	Add 6-bit immediate	PUSH/POP	Push/pop stack element
OR	Logical OR	SUB/SUBC	Subtract/with borrow	BLD/BST	Load/store T bit
ORI	Logical OR 8-bit immediate	SBIW ●	Subtract 6-bit immediate	Miscellaneous	
EOR	Logical exclusive-OR	SUBI/SBCI	Subtract 8-bit imm/w borrow	NOP	No operation
LSL/LSR	Logical shift left/right by 1 bit	INC/DEC	Increment/decrement register	SLEEP	Wait for interrupt
ROL/ROR	Rotate left/right by 1 bit	MUL ◆	Multiply $8 \times 8 \rightarrow 16$	WDR	Watchdog reset
ASR	Arithmetic shift right by 1 bit				
COM/NEG	One's/two's complement				
SWAP	Swap nibbles		Can use R16–R31 only	●	Not available on 90S1200, 1220
TST	Test for zero or minus		Can use R24–R31 only	◆	Future enhancement

Język asemblera – przykład Intel 8086

AAA	CMPSB	IRET	JNAE	JP	LOOPZ	PUSHF	SBB
AAD	CMPSW	JA	JNB	JPE	MOV	RCL	SCASB
AAM	CWD	JAE	JNBE	JPO	MOVS	RCR	SCASW
AAS	DAA	JB	JNC	JS	MOVSW	REP	SHL
ADC	DAS	JBE	JNE	JZ	MUL	REPE	SHR
ADD	DEC	JC	JNG	LAHF	NEG	REPNE	STC
AND	DIV	JCXZ	JNGE	LDS	NOP	REPZ	STD
CALL	HLT	JE	JNL	LEA	NOT	REPZ	STI
CBW	IDIV	JG	JNLE	LES	OR	RET	STOSB
CLC	IMUL	JGE	JNO	LODSB	OUT	RETF	STOSW
CLD	IN	JL	JNP	LODSW	POP	ROL	SUB
CLI	INC	JLE	JNS	LOOP	POPA	ROR	TEST
CMC	INT	JMP	JNZ	LOOPE	POPF	SAHF	XCHG
CMP	INTO	JNA	JO	LOOPNE	PUSH	SAL	XLATB
				LOOPNZ	PUSHA	SAR	XOR

Język asemblera – przykład ARM

- Basic data processing instructions

MOV	Move a 32-bit value	MOV Rd,n	$Rd = n$
MVN	Move negated (logical NOT) 32-bit value	MVN Rd,n	$Rd = \sim n$
ADD	Add two 32-bit values	ADD Rd,Rn,n	$Rd = Rn + n$
ADC	Add two 32-bit values and carry	ADC Rd,Rn,n	$Rd = Rn + n + C$
SUB	Subtract two 32-bit values	SUB Rd,Rn,n	$Rd = Rn - n$
SBC	Subtract with carry of two 32-bit values	SBC Rd,Rn,n	$Rd = Rn - n + C - 1$
RSB	Reverse subtract of two 32-bit values	RSB Rd,Rn,n	$Rd = n - Rn$
RSC	Reverse subtract with carry of two 32-bit values	RSC Rd,Rn,n	$Rd = n - Rn + C - 1$
AND	Bitwise AND of two 32-bit values	AND Rd,Rn,n	$Rd = Rn \text{ AND } n$
ORR	Bitwise OR of two 32-bit values	ORR Rd,Rn,n	$Rd = Rn \text{ OR } n$
EOR	Exclusive OR of two 32-bit values	EOR Rd,Rn,n	$Rd = Rn \text{ XOR } n$
BIC	Bit clear. Every '1' in second operand clears corresponding bit of first operand	BIC Rd,Rn,n	$Rd = Rn \text{ AND } (\text{NOT } n)$
CMP	Compare	CMP Rd,n	$Rd - n$ & change flags only
CMN	Compare Negative	CMN Rd,n	$Rd + n$ & change flags only
TST	Test for a bit in a 32-bit value	TST Rd,n	$Rd \text{ AND } n$, change flags
TEQ	Test for equality	TEQ Rd,n	$Rd \text{ XOR } n$, change flags

MUL	Multiply two 32-bit values	MUL Rd,Rm,Rs	$Rd = Rm * Rs$
MLA	Multiple and accumulate	MLA Rd,Rm,Rs,Rn	$Rd = (Rm * Rs) + Rn$

Język asemblera – przykład ARM cd

- Branch instructions

Branch		B{cond} label	R15 := label
with link		BL{cond} label	R14 := R15-4, R15 := label
and exchange	4T	BX{cond} Rm	R15 := Rm, Change to Thumb if Rm[0] is 1
with link and exchange (1)	5T	BLX label	R14 := R15 - 4, R15 := label, Change to Thumb
with link and exchange (2)	5T	BLX{cond} Rm	R14 := R15 - 4, R15 := Rm[31:1] Change to Thumb if Rm[0] is 1

Formalizm języka ASM

1	0097	LED	EQU 97H
2	0000: 02 01 00		LJMP START
3	0100:		ORG 100H
4	0100: B2 97	START:	CPL LED
5	0102: 74 0A		MOV A,#10
6	0104: 12 01 09		LCALL DELAY
7	0107: 80 F7		SJMP START
8	0109: 78 FF	DELAY:	MOV R0,#0FFH
9	010B: D8 FE	LOOP:	DJNZ R0, LOOP
10	010D: D5 E0 F9		DJNZ ACC, DELAY
11	0110: 22		RET

Zawartość pliku listingu przykładowego programu

Formalizm języka ASM

```
EQU 97H
LJMP  START
ORG 100H
CPL LED
MOV A,#10
LCALL DELAY
SJMP START
MOV R0,#0FFH
DJNZ R0, LOOP
DJNZ ACC, DELAY
RET
```

symbolicznie zakodowane
instrukcje procesora

0	0	0	0	0	0	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

LJMP

START

symbolicznie zakodowana instrukcja
procesora (MNEMONIK)

symbolicznie zakodowany argument
instrukcji (ETYKIETA)

HLL i ASM w kontekście programowania systemowego

Specyfika użycia ASM do programowania

- **Zwiększenie szybkości aplikacji**
 - bezpośredni dostęp do hardware'u (np.: bezpośredni dostęp do portów zamiast wywołania systemowego)
 - dobry program ASM jest szybszy i zajmuje mniej miejsca (krytyczne fragmenty programu w ASM)
- **Ograniczenia**
 - Bardzo szybkie i zwarte lecz przystosowane tylko do jednego typu procesora

HLL i ASM w kontekście programowania systemowego

Typ zastosowań	HLL	ASM
Oprogramowanie ogólnego przeznaczenia na <u>jedną platformę</u> komputerową	Struktury formalne języków ułatwiają zorganizowanie programu	Brak struktur formalnych
Oprogramowanie ogólnego przeznaczenia na <u>wiele platform</u> komputerowych	Przenośne	Trudne do wykonania

HLL i ASM w kontekście programowania systemowego

Typ zastosowań	HLL	ASM
Sterownik urządzenia	Złożone techniki programowania	Dostęp do hardware'u bezpośredni i prosty
Systemy wbudowane, gry wymagające bezpośredniego dostępu do urządzeń	Dają w wyniku zbyt długie programy maszynowe; mogą być nieefektywne	Odpowiedni, ponieważ program maszynowy jest krótki i może być wykonywany szybko

Cechy języka ASM

ML a w konsekwencji ASM – nie posiada mechanizmów charakterystycznych dla HLL:

- złożonych konstrukcji sterowania programem (FOR, WHILE, SWITCH)
 - realizacji bloków programu, w tym lokalności zmiennych
 - relokacji programu
 - modułowej budowy programu
 - ... i wielu innych
- realizacja takich mechanizmów przeniesiona na programistę
- możliwe wspomaganie użyciem zaawansowanych mechanizmów asemblera

Cechy zaawansowanego ASM

Podstawa rozwiązania problemu wpływu ograniczeń
ML na cechy ASM →

- 1) Wprowadzenie do języka ASM
dyrektyw
 - polecenia dla asemblera
 - predefiniowane konstrukcje
- 2) Zastosowanie preprocesingu

Program źródłowy



Makro procesor

Program rozwinięty



Asembler (ASM↔ML)



Kod wynikowy

Dyrektywy ASM

Dyrektywy asemblera

- nie odpowiadają instrukcjom ML
- nie są tłumaczone na ML
- służą do sterowania pracą asemblera.

Przykłady dyrektyw:

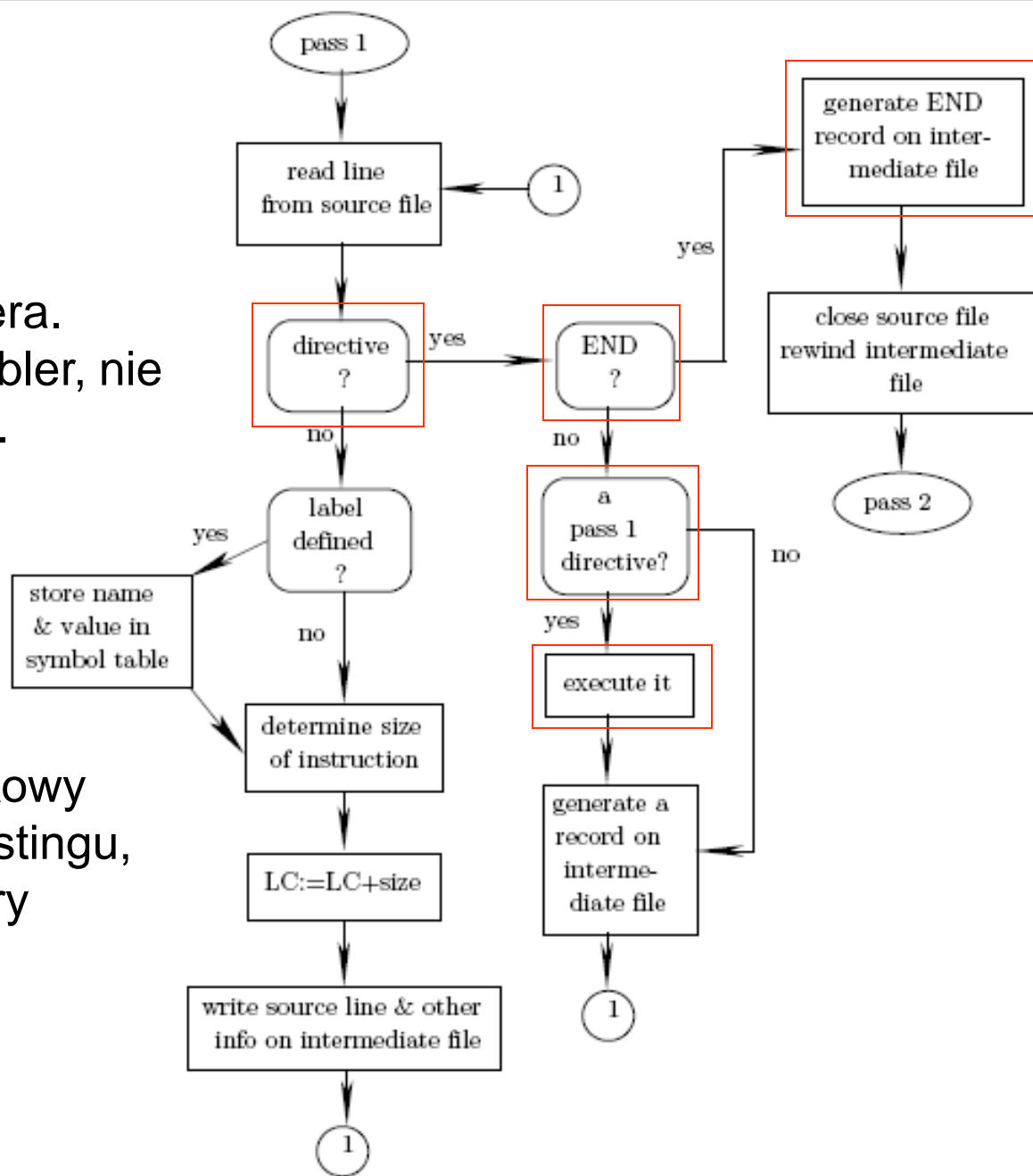
- **ORG**
- **ABS, REL**
- **IF, ENDIF**
- **DB, DW**

Dyrektywy, podobnie jak mnemoniki, posiadają przypisane zwykle kilkuznakowe symbole

Dyrektywy ASM

Rozkazy dla asemblera.
Wykonywane przez asembler, nie
tłumaczone na ML.

Mogą wpływać na wynikowy
kod, tablicę symboli, plik listingu,
wewnętrzne parametry
assemblera.



Funkcje dyrektyw ASM

identyfikacja programu	IDENT
sterowanie programem źródłowym	END, INCLUDE
identyfikacja maszyny	MACHINE, .286, PPU
sterowanie loader'em	LCC
sterowanie trybem	ABS, .RADIX, CODE, QUAL
sterowanie blokami i LC	BEGIN, DS, EVEN, LIMIT, ODD, ORG, OVERLAY, USE
sterowanie segmentami	SEGMENT, ASSUME, GROUP
definiowanie (inicjalizacja) symboli	EQU, MAX, MIN, MICCNT, SET
definiowanie rejestru bazowego	USING, DROP
konsolidacja z podprogramami	CSECT, ENTRY, EXTRN

Funkcje dyrektyw ASM

generacja danych	ASCII, DB, DW, DD, DQ, DT, DATA, DEC, DEF, DIS, LIT, LITORG, PACKED, RECORD, STRUC
sterowanie makroasemblacją	ENDM, IRP, MACRO, REMOVE, SYSLIST, SYSNDX
asemblacja warunkowa	AGO, AIF, ANOP, ELSE, ENDIF, GBLx, IF-ELSE-ENDIF, IFF, IFT, IIF, LCLx, SET
sterowanie reakcją na błędy	ERR, ERRxx
sterowanie listingiem	LIST, TITLE, XREF
definiowanie operacji	OPDEF, PURGDEF
zarządzanie tablicą kodów ML	OPSYN

Makroinstrukcje w ASM

Makroinstrukcja – symbol, któremu przypisano fragment tekstu programu (*w definicji makroinstrukcji*)

Gdziekolwiek w pliku źródłowym znajdzie się symbol zdefiniowanego wcześniej makra, asembler wpisze w jego miejsce w pliku źródłowym tekst przypisany do makra (makro = zmienna asemblera, która może być używana jako “skrót” przypisanego jej tekstu)

Makroinstrukcje w ASM

program

definicja makro

.....

„wywołanie” makro

.....

„wywołanie” makro

Makroinstrukcje w ASM

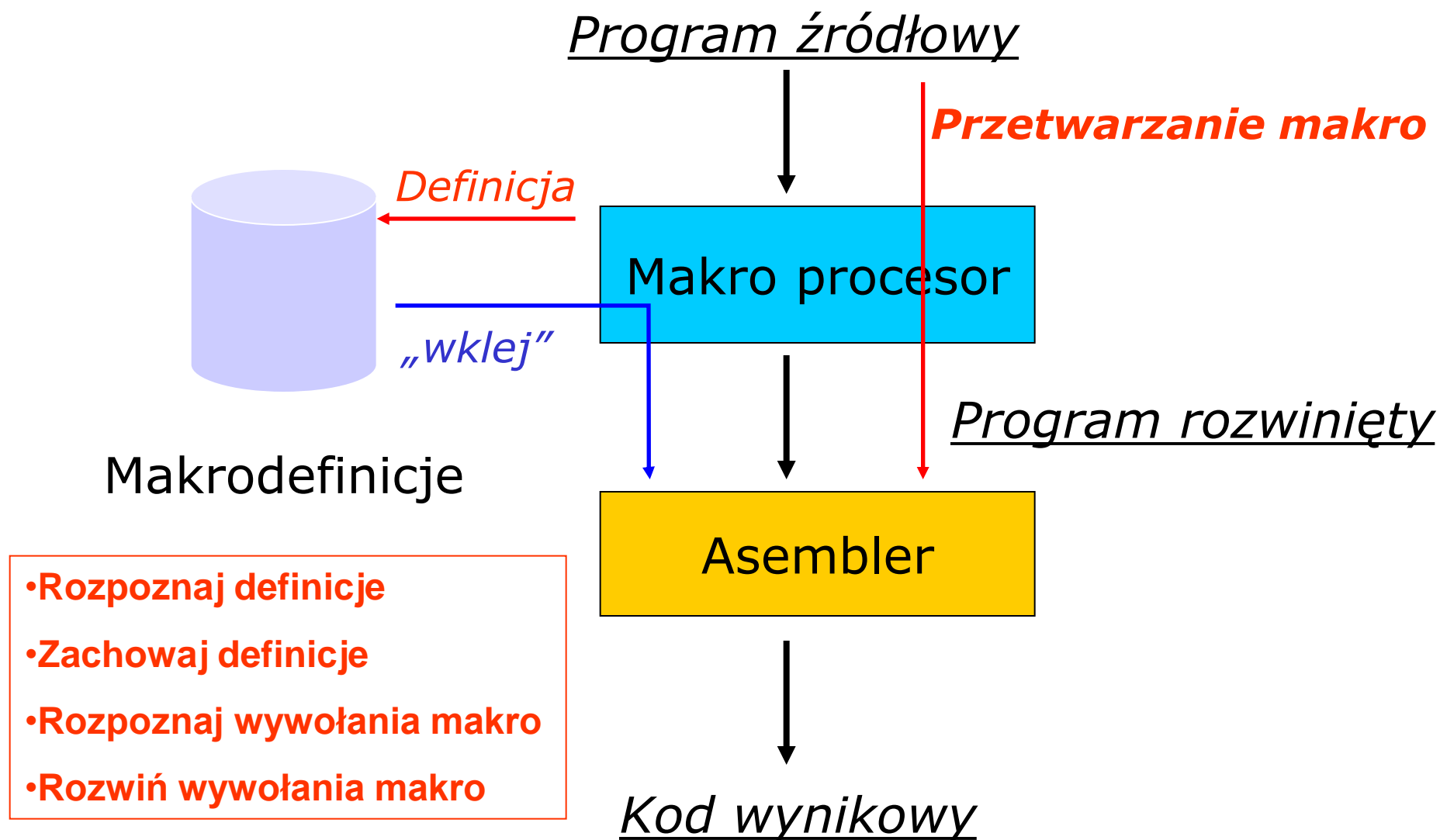
Makrodefinicja:

<etykieta> MACRO [parametry]	;nagłówek
.....	;instrukcje makro
ENDM	;terminator

Makrowywołanie:

<etykieta> [parametry]

Przetwarzanie programu (preprocessing)



Makroinstrukcje w ASM

Korzyści stosowania makroinstrukcji:

- **Redukuje liczbę błędów** powodowanych przez programistę.
- **Umożliwia zdefiniowanie często używanych** w programie sekwencji instrukcji.
- **Wielokrotne użycie tego makra w tekście programu źródłowego** **każdorazowo zapewni ten sam rezultat**
- **Skraca się czas przygotowania** programu źródłowego, a program zyskuje na przejrzystości.
- **Symbole (etykiety) użyte w makro są lokalne** w obrębie makro i nie są mylone przez assembler z identycznymi symbolami używanymi poza nim.

Makroinstrukcje w ASM

```
ADD_AB_R0R1      MACRO
    CLC            ; zeruj bit przeniesienia
    ADD A,R0       ; dodaj młodsze bajty
    ADDC B,R1      ; dod. starsze bajty z uwzględnieniem przen.
    ENDM           ; koniec makro
```

Przykład definicji makra

Zagadnienia makro

- Sposób definiowania makro
- Zastępowanie parametrów
- Lokalność symboli
- Rozwinięcia powtarzane
- Makro z parametrami a procedura z parametrami

Rozwijanie makro

Program źródłowy

```
STRG  MACRO  
      MOV AX, BX  
      MOV BX, AX  
      NOP  
      ENDM
```

```
·  
STRG
```

```
·  
STRG
```

```
·  
·
```

Program rozwinięty

```
·  
·
```

```
{  
  MOV AX, BX  
  MOV BX, AX  
  NOP
```

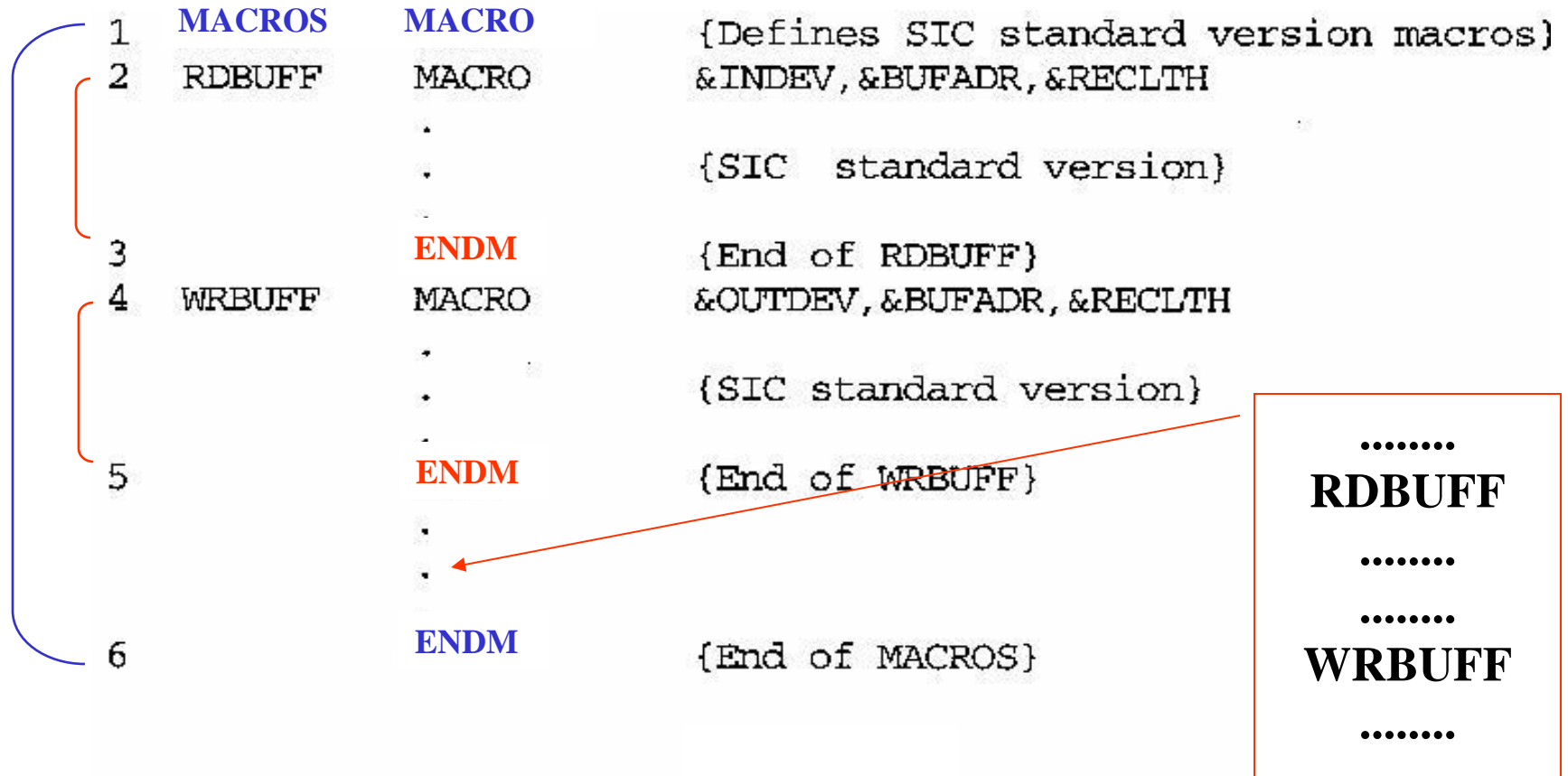
```
·
```

```
{  
  MOV AX, BX  
  MOV BX, AX  
  NOP
```

```
·
```

Makrodefinicje zagnieżdżone

Przetwarzanie makrodefinicji w trakcie rozwijania



Zastępowanie parametrów makro

Program źródłowy

STRG **MACRO** DST, SRC

MOV AX, SRC
MOV DST, AX
NOP

ENDM

·
STRG DATA1, DATA2

·
STRG DATA4, DATA5

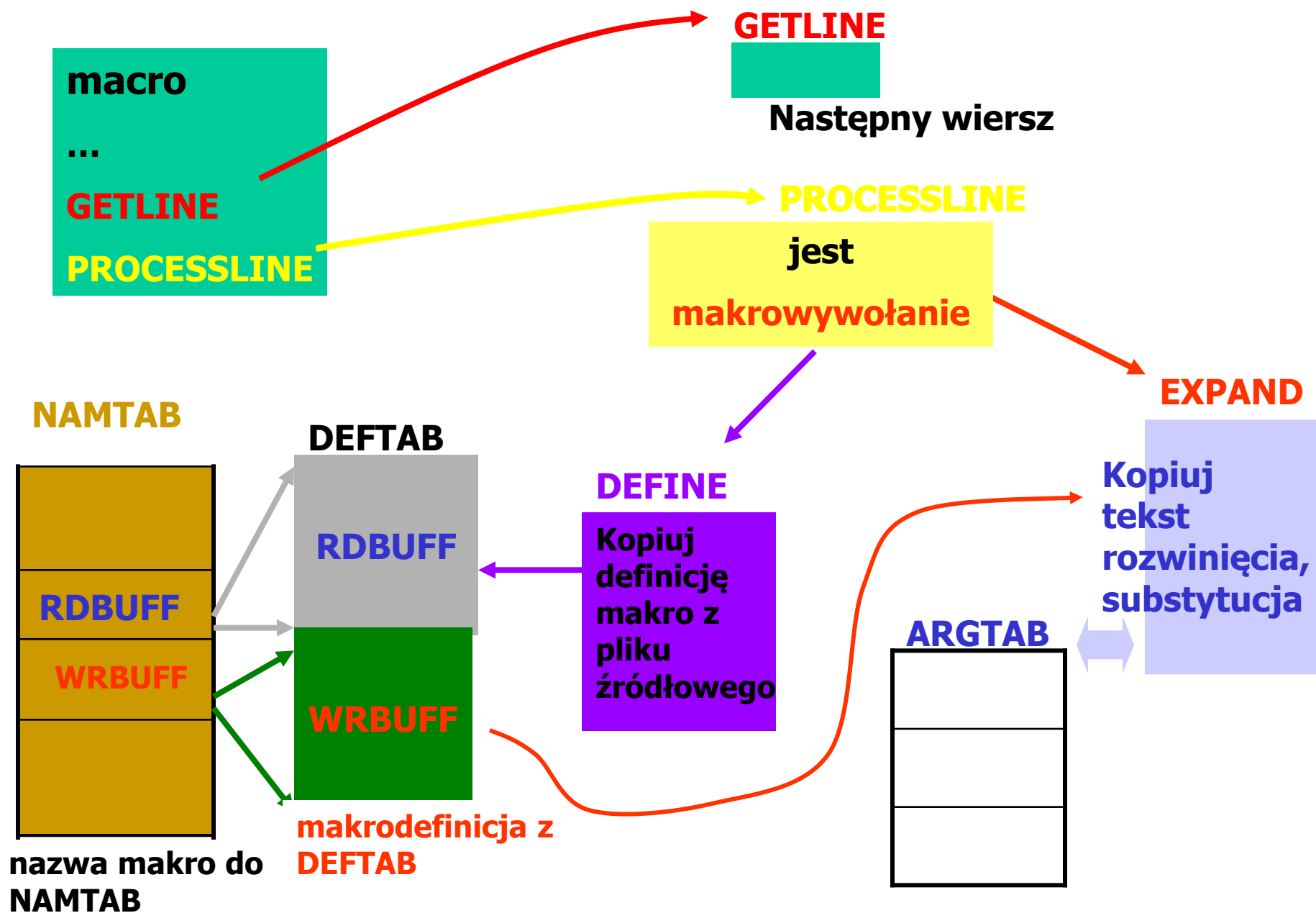
Program rozwinięty

·
·

{
MOV AX, DATA2
MOV DATA1, AX
NOP

{
MOV AX, DATA5
MOV DATA4, AX
NOP

Algorytm jednoprzeglądowego makroprocesora



Lokalność symboli makro

- Podstawowy asembler – brak symboli lokalnych
- **Makroassembler** – jeżeli kopiowałby tekst źródłowy bez zmian symbole w tekście makrodefinicji **zdefiniowane byłyby wielokrotnie**
- **Symbole lokalne** - makroprocesor nadaje unikalne nazwy w czasie wstawiania tekstu makrodefinicji do programu

– Np. **STRG MACRO**

X1 EQU 10H

STRG_1_X1 EQU 10H



Makro – „wywołania” powtarzane

Dyrektywy **FOR**, **REPEAT**, **WHILE**

Efekt użycia: skrócenie tekstu programu:

FOR parametr, <argument [,argument]...>

tekst makro

ENDM

Parametr w tekście – zastąpiony
kolejnym argumentem z listy

REPEAT wyrażenie

tekst makro

ENDM


Powtórz makro „wartość
wyrażenia” razy

Makro – „wywołania” powtarzane

Dyrektywy **FOR, REPEAT, WHILE**

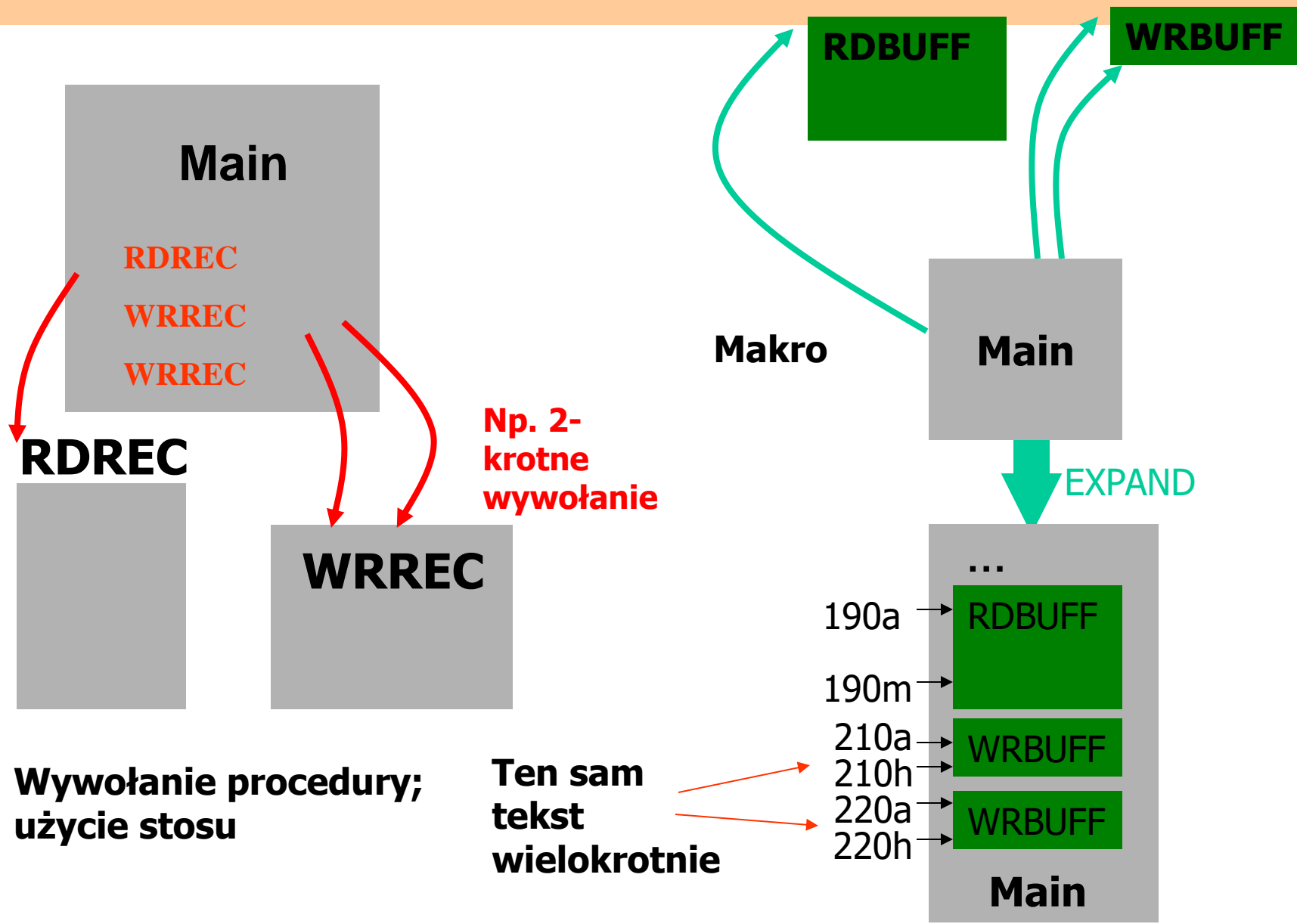
Efekt użycia: skrócenie tekstu programu:

WHILE wyrażenie
tekst makro
ENDM



Powtarzaj tekst makro kiedy wyrażenie
= TRUE (modyfikacja w makro!)

Makro a wywołanie procedury



Cechy makroinstrukcji

Mechanizm „przekazywania” parametrów i lokalność wewnętrznych symboli → makroinstrukcja może służyć jako metoda implementacji bloków programu

Wada rozwiązania: mechanizm statyczny – bloki mają charakter statyczny, ustalona postać na etapie generacji programu źródłowego