

Wykład 2

Implementacja konstrukcji warunkowych
Implementacja struktur danych

Bloki programu realizowane warunkowo

statycznie (na etapie tłumaczenia)

dynamicznie (wykonania programu)

warunkowe rozgałęzianie programu

bloki wykonywane warunkowo

wsparcie makroasemblera

Definiowanie symboli

Struktury złożone

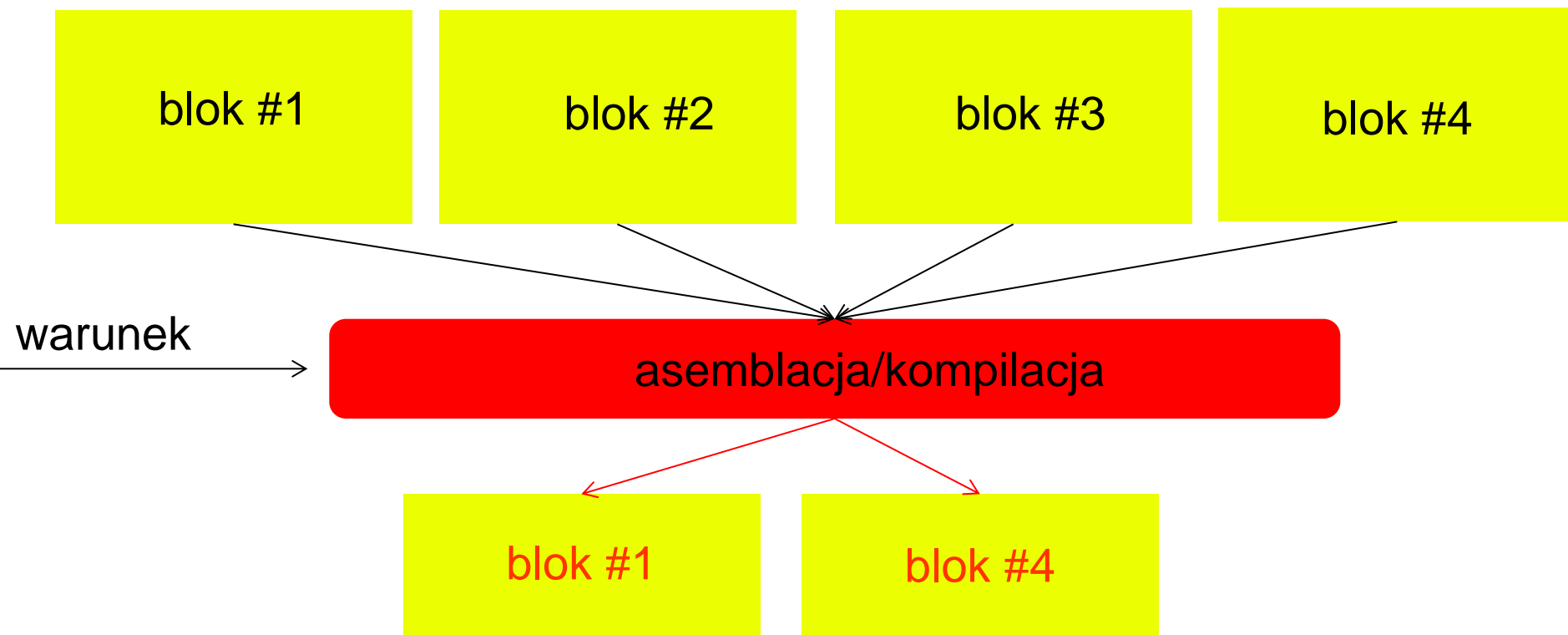
Bloki programu realizowane warunkowo

- **Warunkowa realizacja bloków programu na etapie tłumaczenia (assembly/compile time)**
- **Bloki wykonywane warunkowo (run-time)**

Asemlacja warunkowa – wariantowość programu źródłowego

Jeden program źródłowy – wiele wariantów → generacja innego kodu w zależności od „otoczenia”

- *np. w pliku źródłowym zdefiniowano parametr*



Asemlacja warunkowa – wariantowość programu źródłowego

- Polecenia sterujące asemlacją (wykonywane w czasie asemlacji)
 - *IF-ELSE-ENDIF (IFE, IFDEF, IFNDEF...)*
 - *Jeden program – wiele wariantów → generacja innego kodu w zależności od „otoczenia”*
 - *np. pliku źródłowym zdefiniowano parametr=typ procesora*

Asemblacja warunkowa – wariantowość programu źródłowego

```
Processor      =      80386
                :      ;Set to 8086 for 8086-only code
                :
                if      Processor eq 80386


---


                shl     ax, 4


---


                else    ;Must be 8086 processor


---


                mov     cl, 4
                shl     ax, cl


---


                endif
```

Asemblacja warunkowa – wariantowość programu źródłowego

Mechanizm generacji wariantów programu →
dyrektywa może służyć jako metoda implementacji
wariantowego programu źródłowego

Wada rozwiązania: mechanizm statyczny –
sekwencja wykonania bloków odbywa się w
ustalonym porządku; ustalenie na etapie generacji
programu źródłowego

Konstrukcje warunkowe

(bloki programu wykonywane warunkowo)

Flagi statusu – przypomnienie

Flaga **Zero** ustawiana, gdy wynik wynosi 0 (**ZF** – x86, **Z** - AVR, ARM)

Flaga **Carry** ustawiana, gdy wynik (liczba nieujemna) jest zbyt duży (zbyt mały), aby umieścić w lokalizacji docelowej (**CF** - x86, **C** - AVR, ARM)

Flaga **Negative** jest kopią najstarszego bitu wyniku (**N** – AVR, ARM)

Flaga **Sign** jest kopią najstarszego bitu wyniku (**SF** – x86)

Flaga **Sign** ustawiana, gdy wynik (porównania) jest ujemny $S=N \oplus V$ (AVR)

Flaga **Overflow** ustawiana, gdy wynik w postaci liczby ze znakiem jest niewłaściwy (**OF** – x86, **V** – AVR, ARM)

Flagi statusu – ARM

Flagi GE – kodują relację „większe lub równe zero” dla części słów

GE[3]	GE[2]	GE[1]	GE[0]
A op B greater than or equal to C	A op B greater than or equal to C	A op B greater than or equal to C	A op B greater than or equal to C
$[31:16] + [31:16] \geq 0$	$[31:16] + [31:16] \geq 0$	$[15:0] + [15:0] \geq 0$	$[15:0] + [15:0] \geq 0$
$[31:16] - [31:16] \geq 0$	$[31:16] - [31:16] \geq 0$	$[15:0] - [15:0] \geq 0$	$[15:0] - [15:0] \geq 0$
$[31:24] + [31:24] \geq 0$	$[23:16] + [23:16] \geq 0$	$[15:8] + [15:8] \geq 0$	$[7:0] + [7:0] \geq 0$
$[31:24] - [31:24] \geq 0$	$[23:16] - [23:16] \geq 0$	$[15:8] - [15:8] \geq 0$	$[7:0] - [7:0] \geq 0$
$[31:16] + [31:16] \geq 2^{16}$	$[31:16] + [31:16] \geq 2^{16}$	$[15:0] + [15:0] \geq 2^{16}$	$[15:0] + [15:0] \geq 2^{16}$
$[31:16] - [31:16] \geq 0$	$[31:16] - [31:16] \geq 0$	$[15:0] - [15:0] \geq 0$	$[15:0] - [15:0] \geq 0$

Skoki warunkowe

Warunki odnoszą się do. . .

- flag
- równości (argumentów)
- wyników porównań (bez znaku)
- wyników porównań (ze znakiem)

Skoki warunkowe (od flag, x86)

Mnemonic	Description	Flags
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0
JP	Jump if parity (even)	PF = 1
JNP	Jump if not parity (odd)	PF = 0

Skoki warunkowe (od flag, x86)

JE=JZ
JNE=JNZ

Mnemonic	Description
JE	Jump if equal (<i>leftOp = rightOp</i>)
JNE	Jump if not equal (<i>leftOp \neq rightOp</i>)

Skoki warunkowe (od flag, AVR)

	BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC
	BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC
	BREQ	k	Branch if Equal	if (Z = 1) then PC
	BRNE	k	Branch if Not Equal	if (Z = 0) then PC
	BRCS	k	Branch if Carry Set	if (C = 1) then PC
	BRCC	k	Branch if Carry Cleared	if (C = 0) then PC

Skoki warunkowe (od flag, AVR)

BRMI	k	Branch if Minus	if (N = 1) then PC
BRPL	k	Branch if Plus	if (N = 0) then PC
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC

Skoki warunkowe (od bitu, AVR)

SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b) = 0) PC
SBRs	Rr, b	Skip if Bit in Register Set	if (Rr(b) = 1) PC
SBIC	A, b	Skip if Bit in I/O Register Cleared	if (I/O(A,b) = 0) PC
SBIS	A, b	Skip if Bit in I/O Register Set	If (I/O(A,b) = 1) PC

Skoki warunkowe (równość, x86)

Mnemonic	Description
JCXZ	Jump if CX = 0
JECXZ	Jump if ECX = 0

rejestr licznikowy

Skoki warunkowe (równość, AVR)

CPSE	Rd,Rr	Compare, skip if Equal	if (Rd = Rr) PC
------	-------	---------------------------	--------------------

Warunki rozgałęzień (x86)

- Zadanie: skocz do etykiety, jeżeli liczba całk. parzysta.
- Rozwiązanie: wyzeruj wszystkie bity poza najmłodszym ($x \text{ AND } 1$). Gdy rezultat 0 - liczba była parzysta.

```
mov ax,wordVal
and ax,1           ; zeruj bity poza najmłodszym
jz  EvenValue      ; skocz, gdy ustawiona flaga zera
```

- Zadanie: Skocz do etykiety, gdy zawartość AL \neq 0.
- Rozwiązanie: wykonaj AL OR AL ($0 \Leftrightarrow \text{AL}=0$). Użyj JNZ (jump if not zero).

Uwaga! $x \cup x = x$

```
or  al,al
jnz IsNotZero      ; skocz, gdy nie zero
```

Warunki rozgałęzień (x86)

Zastosowanie instrukcji **TEST**

Nie modyfikuje operandów; określa stan flagi „Zero”, „Znak”, „Parity”.
Wartość flagi „Zero” jak dla instrukcji AND

Przykład 1: skocz, gdy AL_0 lub AL_1 ustawione.

```
test al,00000011b  
jnz  ValueFound
```

Przykład 2: skocz, gdy ani AL_0 ani AL_1 ustawione.

```
test al,00000011b  
jz   ValueNotFound
```

Warunki rozgałęzień (x86)

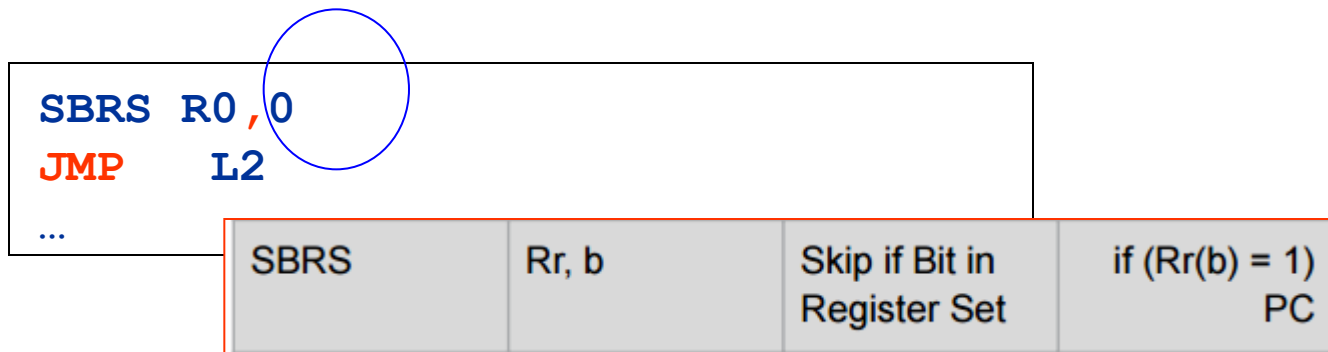
- Skocz, jeżeli podwójne słowo „adresowane” przez EDI jest parzyste

```
test DWORD PTR [edi], 1  
jz    L2
```

Mnemonic	Description	Flags
JZ	Jump if zero	ZF = 1

Warunki rozgałęzień (AVR)

- Skocz, jeżeli bajt w rejestrze jest parzysty



Warunki rozgałęzień (x86)

Warunki od wartości bitu:

Użyj **BT** do przekopiowania bitu do CF

Składnia: **BT** *symbol, n*

Użyj skoku od **CF**

r16, r32, lub imm8

r/m16 lub r/m32

Przykład: skocz, jeżeli $AX_9=1$:

```
bt AX,9           ; CF = bit 9  
jc L1             ; skocz, gdy CF=1
```

Warunki rozgałęzień (x86)

- W tym samym celu można użyć instrukcji TEST
- Skocz, jeżeli podwójne słowo „adresowane” przez EDI jest parzyste

```
test DWORD PTR [edi], 1  
jz    L2
```

Mnemonic	Description	Flags
JZ	Jump if zero	ZF = 1

Warunki rozgałęzień

Zastosowanie instrukcji **CMP**

Porównuje operand docelowy ze źródłowym

Wartości flag jak dla odejmowania operandu źródłowego od operandu docelowego; operandy niezmienione.

Składnia: **CMP** *arg docelowy, arg źródłowy*

docelowy \equiv źródłowy	ZF=1
docelowy > źródłowy	ZF=0 \cap CF=0
	SF=OF
docelowy < źródłowy	CF=1
	SF= \neg OF

Warunki rozgałęzień

Zastosowanie instrukcji **CMP**

Oprnd1	minus	Oprnd2	S	O
-----		-----	-	-
0FFFF (-1)	-	0FFFE (-2)	0	0
8000	-	00001	0	1
FFFE (-2)	-	0FFFF (-1)	1	0
07FFF (32767)	-	0FFFF (-1)	1	1

Operacje arytmetyczne – flagi

DODAWANIE LICZB W ZAPISIE STAŁOPRZECINKOWYM

- MSB=znak
- zapis liczby ujemnej=znak-uzup. do 2

Dodawanie wykonuje się jako dodawanie pozycyjne z przeniesieniem. Przeniesienie z pozycji MSB pomija się.

$$OVF = C_{MSB,MSB+1} \oplus C_{MSB-1,MSB}$$

Przykład obliczenia A+B

C	Znak	- uzup. do 2
+ 3	0	11
+ 2	0	10
+ 1	0	01
+ 0	0	00
- 0		-
- 1	1	11
- 2	1	10
- 3	1	01

A	B	→	+2			-2				
↓			0	1	0	1	1	0		
	+1		0	0	1	A	0	0	1	A
0	0	1	+0	1	0	B	+1	1	0	B
			0	1	1	A+B	1	1	1	A+B
	-1		1	1	1	A	1	1	1	A
1	1	1	+0	1	0	B	+1	1	0	B
			0	0	1	A+B	1	0	1	A+B

Operacje arytmetyczne – flagi

ODEJMOWANIE LICZB W ZAPISIE STAŁOPRZECINKOWYM

- MSB=znak
- zapis liczby ujemnej=znak-uzup. do 2

Odejmowanie wykonuje się jako odejmowanie pozycyjne z pożyczką. W razie potrzeby wykonuje się pożyczkę z pozycji o jeden wyższej od MSB.

$$OVF = P_{MSB+1,MSB} \oplus P_{MSB,MSB-1}$$

Przykład obliczenia A-B

C	Znak	- uzup. do 2
+ 3	0	11
+ 2	0	10
+ 1	0	01
+ 0	0	00
- 0		-
- 1	1	11
- 2	1	10
- 3	1	01

A ↓	B	→	+2				-2			
			0	1	0		1	1	0	
<div>+1</div> <div>001</div>			0	0	1	A	0	0	1	A
			-0	1	0	B	-1	1	0	B
			1	1	1	A-B	0	1	1	A-B
<div>-1</div> <div>111</div>			1	1	1	A	1	1	1	A
			-0	1	0	B	-1	1	0	B
			1	0	1	A-B	0	0	1	A-B

Skoki warunkowe (porównania bez znaku, x86)

Mnemonic	Description	
JA	Jump if above (if $leftOp > rightOp$)	CF=0 i ZF=0
JNBE	Jump if not below or equal (same as JA)	CF=0 i ZF=0
JAE	Jump if above or equal (if $leftOp \geq rightOp$)	CF=0
JNB	Jump if not below (same as JAE)	CF=0
JB	Jump if below (if $leftOp < rightOp$)	CF=1
JNAE	Jump if not above or equal (same as JB)	CF=1
JBE	Jump if below or equal (if $leftOp \leq rightOp$)	CF=1 lub ZF=1
JNA	Jump if not above (same as JBE)	CF=1 lub ZF=1

Skoki warunkowe (porównania ze znakiem, x86)

Mnemonic		Description	
JG		Jump if greater (if <i>leftOp</i> > <i>rightOp</i>)	SF=OF i ZF=0
JNLE		Jump if not less than or equal (same as JG)	
JGE		Jump if greater than or equal (if <i>leftOp</i> >= <i>rightOp</i>)	SF=OF
JNL		Jump if not less (same as JGE)	
JL		Jump if less (if <i>leftOp</i> < <i>rightOp</i>)	SF≠OF
JNGE		Jump if not greater than or equal (same as JL)	
JLE		Jump if less than or equal (if <i>leftOp</i> <= <i>rightOp</i>)	SF≠OF lub ZF=1
JNG		Jump if not greater (same as JLE)	

Skoki warunkowe (porównania, AVR)

BRSH	k	Branch if Same or Higher	if (C = 0) then PC
BRLO	k	Branch if Lower	if (C = 1) then PC
BRGE	k	Branch if Greater or Equal, Signed	if (N \oplus V = 0) then PC
BRLT	k	Branch if Less Than, Signed	if (N \oplus V = 1) then PC

Skoki warunkowe (porównania, AVR)

Brak instrukcji rozgałęzienia (ze znakiem i bez znaku):

👉 $A \leq B (\Leftrightarrow B \geq A)$

👉 $A > B (\Leftrightarrow B < A)$

Należy wykonać porównanie odwrotnie uporządkowanych argumentów i użyć odpowiedniego rozgałęzienia

Warunki rozgałęzień (x86)

- Zadanie: skocz do etykiety, jeżeli EAX>EBX (**bez znaku**)
- Rozwiązanie: użyj **CMP**, następnie **JA**

```
cmp eax,ebx  
ja  Larger
```

Mnemonic	Description
JA	Jump if above (if <i>leftOp</i> > <i>rightOp</i>)

- Zadanie: skocz do etykiety, jeżeli EAX>EBX (**ze znakiem**)
- Rozwiązanie: użyj **CMP**, następnie **JG**

```
cmp eax,ebx  
jg  Greater
```

Mnemonic	Description
JG	Jump if greater (if <i>leftOp</i> > <i>rightOp</i>)

Warunki rozgałęzień (x86)

- Skocz, jeżeli $EAX \leq Val1$ (bez znaku)

```
cmp eax,Val1
```

```
jbe L1
```

```
; mniejsze lub równe
```

Mnemonic	Description
JBE	Jump if below or equal (if <i>leftOp</i> <= <i>rightOp</i>)

- Skocz, jeżeli $EAX \leq Val1$ (ze znakiem)

```
cmp eax,Val1
```

```
jle L1
```

```
; mniejsze lub równe
```

Mnemonic	Description
JLE	Jump if less than or equal (if <i>leftOp</i> <= <i>rightOp</i>)

Warunki rozgałęzień (x86)

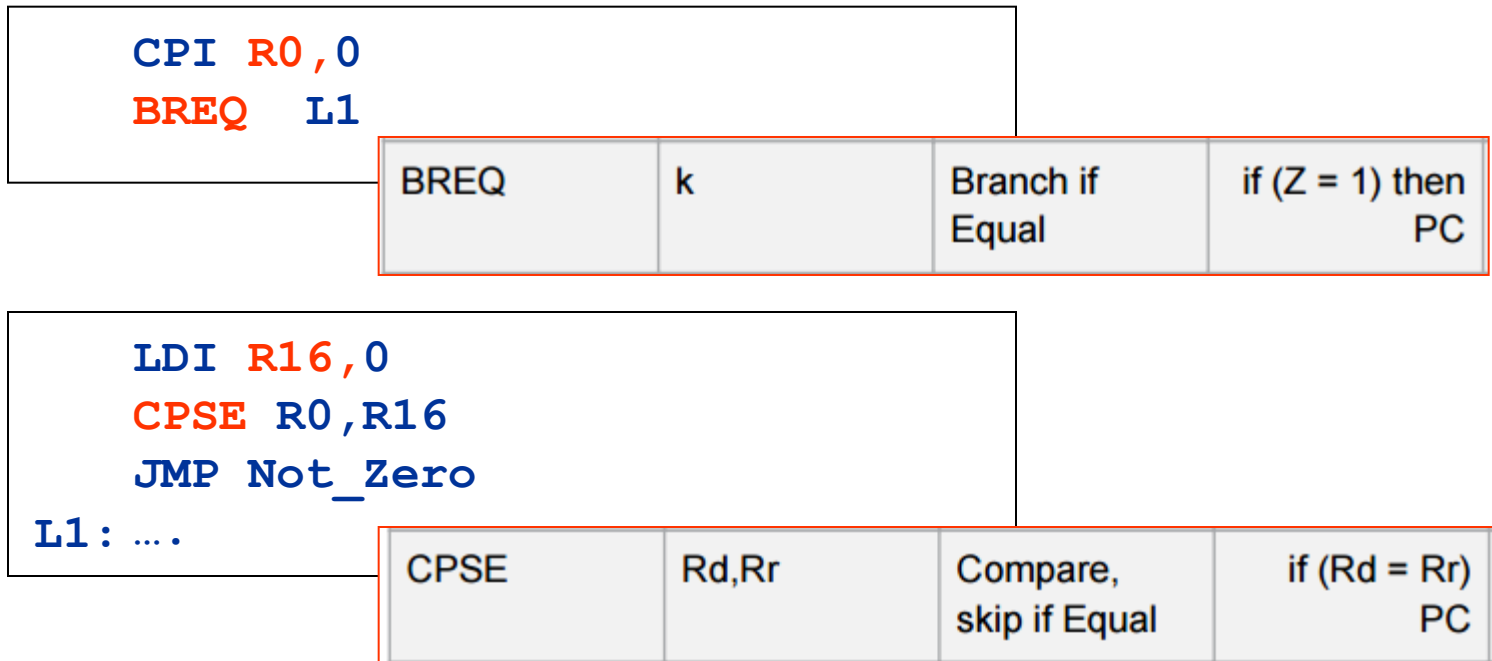
- Skocz, jeżeli słowo „adresowane” przez ESI równa się 0

```
cmp WORD PTR [esi],0  
je L1
```

Mnemonic	Description
JE	Jump if equal (<i>leftOp = rightOp</i>)

Warunki rozgałęzień (AVR)

- Skocz, jeżeli bajt w rejestrze równa się 0



Warunki rozgałęzień - ARM

- w każdej instrukcji 4-bitowy kod, określający warunki jej wykonania
- eliminacja wielu rozgałęzień programu,
- przyspieszenie wykonanie programu dzięki zmniejszeniu objętości kodu

CMP	R0, #5	;if (a==5)
MOVEQ	R0, #10	do ...
BLEQ	L2	;wyjdź z R0=10
... •		

Warunki rozgałęzień - ARM

- warunki w rejestrze CPSR

Mnemonik	Rozwinięcie	Warunek	Stan flag
EQ	equal	równy	Z=1
NE	not equal	nie równy	Z=0
CS	carry set (unsigned higher or same)	ustawiona flaga przeniesienia; większy lub równy (liczby bez znaku)	C=1
CC	carry clear (unsigned lower)	wyzerowana flaga przeniesienia; mniejszy (liczby bez znaku)	C=0
MI	negative (minus)	ujemny	N=1
PL	positive or zero (plus)	dodatni lub zerowy	N=0
VS	overflow set	ustawiona flaga przepełn.	V=1
VC	overflow clear	wyzerowana flaga przepełn.	V=0

Warunki rozgałęzień - ARM

- warunki w rejestrze CPSR

Mnemonik	Rozwinięcie	Warunek	Stan flag
HI	unsigned higher	większy (liczby bez znaku)	$C=1$ and $Z=0$
LS	unsigned lower or same	mniej lub równy (liczby bez znaku)	$C=0$ or $Z=1$
GE	greater or equal	większy lub równy	$N=V$
LT	less then	mniej	$N \neq V$
GT	greater then	większy	$Z=0$ and $(N=V)$
LE	less then or equal	mniej lub równy	$Z=1$ or $(N \neq V)$
AL	always	zawsze (mnemonik można pominąć)	bez znaczenia

Warunki rozgałęzień - ARM

- W instrukcjach arytmetycznych i logicznych bit wskazujący na to, czy dana instrukcja może zmienić zawartość rejestru stanu procesora (CPSR)

```

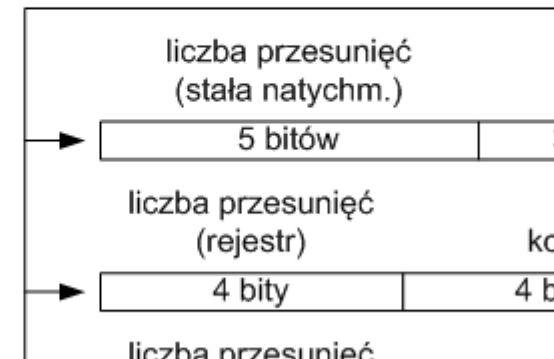
01.    mov    r4, #10
02.    loop_label:
03.    bl     do_something
04.    subs   r4, r4, #1
05.    bne    loop_label
    
```

- Użycie warunku do rozgałęzienia

warunek	kod grupy rozkazów	kod rozkazu	uaktualnienie CPSR	rejestr docelowy	pierwszy operand
			1 bit	4 bity	4 bity

```

01.    mov    r4, #10
02.    loop_label:
03.    bl     do_something
04.    sub     r4, r4, #1
05.    cmp     r4, #0
06.    bne    loop_label
    
```



Organizacja pętli – x86

Składnia:

LOOP *etykieta*

Działanie:

$ECX \leftarrow ECX - 1$

$ECX > 0$, skok do *etykiety*

Organizacja pętli –x86

Składnia:

LOOPE *etykieta* (\leftrightarrow **LOOPZ** *etykieta*)

Działanie:

$ECX \leftarrow ECX - 1$

$ECX > 0$ i $ZF=1$, skok do *etykiety*

Składnia:

LOOPNE *etykieta* (\leftrightarrow **LOOPNZ** *etykieta*)

Działanie:

$ECX \leftarrow ECX - 1$

$ECX > 0$ i $ZF=0$, skok do *etykiety*

Organizacja pętli –x86

LOOPE *etykieta*

Zastosowanie - należy powtarzać pętlę w czasie, gdy jedna wartość=drugiej, lecz nie przekroczyć maksymalnej liczby iteracji

Przykład: poszukiwanie pierwszego niezerowego elementu tablicy, w granicach tablicy:

	mov cx, 16	;max 16 elementów
	mov bx, -1	;indeks w tablicy
SearchLp:	inc bx	;przesuń indeks
	cmp Array[bx], 0	;czy zero?
	loope SearchLp	;powtarzaj (zero, <16)
	je AllZero	;gdy wszystkie= zero

Organizacja pętli –x86

LOOPNE *etykieta*

Zastosowanie, gdy należy powtarzać pętlę maksymalną liczbę razy oczekując na spełnienie pewnego warunku

Przykład: oczekiwanie, aż urządzenie gotowe:

	mov dx, 379h	;adres portu
	mov cx, 0	;pętla max 65536 razy
WaitNotBusy:	in al, dx	;czytaj port
	test al, 80h	;zajęty?
	loopne WaitNotBusy	;gdy zajęty i <65536x
	jne TimedOut	;skok, gdy „time out”(CX=0, ZF=0)

Organizacja pętli – AVR

BRNE *etykieta*

Implementacja pętli typu for:

	ldi r16, 10	;inic. licznika pętli
StartLoop:	;kod pętli
	dec r16	;dekrementuj licznik
	brne StartLoop	;zamknij pętlę
EndLoop:	

Struktury warunkowe

- Implementacja struktur warunkowych w ASM
 - Blokowe polecenia IF
 - Pętle WHILE
 - Przełącznik CASE

Blokowe polecenia IF

Odpowiedniki poleceń języków wysokiego poziomu
(x86):

```
if( op1 == op2 )  
    X = 1;  
else  
    X = 2;
```

```
mov  eax,op1  
cmp  eax,op2  
jne  L1  
mov  X,1  
jmp  L2  
L1:  mov  X,2  
L2:
```

Blokowe polecenia IF - ARM

ARM - konstrukcja if ... else:

```
CMP      R0, #5           ;if (R0<=5)
MOVLE    R0, #0           do .. R0=0
MOVGT    R0, #1           ;else R0=1
... .
```

ARM - konstrukcja if a||b do... :

```
CMP      R0, #'A'         ;if (R0=='A' OR R0== 'B')
CMPNE    R0, #'B'         ..
MOVEQ    R1, #1           ;do ... R1=1
... .
```


Złożone wyrażenia warunkowe

Wyrażenie warunkowe z iloczynem (AND)

Przykład: w poniższym przykładzie HLL pomija wyznaczanie drugiego wyrażenia, jeżeli pierwsze fałszywe.

```
if (a1 > b1) AND (b1 > c1)  
    x = 1;
```



Złożone wyrażenia warunkowe – x86

```
if (a1 > b1) AND (b1 > c1)
    X = 1;
```

Jedna z możliwych implementacji [IF (a1 > b1)] AND [IF (b1 > c1)]

```
{ cmp al,b1                ; pierwsze wyrażenie...
  ja  L1
  jmp next
L1: { cmp bl,cl              ; drugie wyrażenie...
    ja  L2
    jmp next
L2:                ; obydwa prawdziwe
    mov X,1        ; ustaw X na 1
next:
```

Złożone wyrażenia warunkowe – x86

```
if (a1 > b1) AND (b1 > c1)  
    X = 1;
```

Ale....poniższa implementacja używa prawie 30% kodu mniej: NOT {[IF NOT (a1 > b1)] OR [IF NOT (b1 > c1)]}

<code>cmp al,b1</code>	<code>; pierwsze wyrażenie...</code>
<code>jbe next</code>	<code>; kończ, gdy fałsz</code>
<code>cmp bl,c1</code>	<code>; drugie wyrażenie...</code>
<code>jbe next</code>	<code>; kończ, gdy fałsz</code>
<code>mov X,1</code>	<code>; obydwa prawdziwe</code>
<code>next:</code>	

Złożone wyrażenia warunkowe – x86

Wyrażenie warunkowe z sumą logiczną (OR)

Przykład: w poniższym przykładzie HLL pomija wyznaczanie drugiego wyrażenia, jeżeli pierwsze prawdziwe.

```
if (a1 > b1) OR (b1 > c1)  
    x = 1;
```



Złożone wyrażenia warunkowe – x86

```
if (a1 > b1) OR (b1 > c1)
    X = 1;
```

Można zastosować poprzednią koncepcję dla skrócenia programu:

```
    cmp al,b1                ; AL > BL?
    ja  L1                   ; tak
    cmp bl,cl                ; nie: BL > CL?
    jbe next                 ; nie: omiń
L1: mov X,1                   ; ustaw X na 1
next:
```

Pętla WHILE – x86

Pętla **WHILE** jest w rzeczywistości poleceniem IF po którym następuje treść pętli zakończona skokiem bezwarunkowym na początek pętli.

Przykład:

```
while( eax < ebx)
    eax = eax + 1;
```

Możliwa implementacja:

<code>top: cmp eax, ebx</code>	<code>; sprawdź warunek pętli</code>
<code> jae next</code>	<code>; fałsz? Opuść pętłę</code>
<code> inc eax</code>	<code>; program pętli</code>
<code> jmp top</code>	<code>; powtarzaj pętłę</code>
<code>next:</code>	

Pętla WHILE – AVR

Pętla **WHILE** jest w rzeczywistości poleceniem IF po którym następuje treść pętli zakończona skokiem bezwarunkowym na początek pętli.

Przykład: oczekiwanie na gotowość urządzenia (pamięci EEPROM w AVR)

Implementacja:

```
top: sbic EECR,EEPE      ; sprawdź czy gotowy
    breq EE_READY        ; tak, gotowy (program pętli)
    rjmp top            ; powtarzaj pętlę
EE_READY:
```

Przełącznik CASE – x86

Przełącznik CASE o następującej składni:

CASE zmienna in

"wzorzec1") polecenie1 ;;

"wzorzec2") polecenie2 ;;

"wzorzec3") polecenie3 ;;

*) polecenie_domyślne

esac

... Porównuje zmienną z wzorcami i po napotkaniu pierwszej zgodności wykonuje odpowiadające polecenie a następnie kończy CASE



Przełącznik – implementacja x86

Krok 1: utwórz tablicę zawierającą wyróżniki i przesunięcia do procedur:

```
.data
CaseTable BYTE 'A'                ; wartość sprawdzana
          DWORD Process_A          ; adres procedury
          EntrySize = ($ - CaseTable)
          BYTE 'B'
          DWORD Process_B
          BYTE 'C'
          DWORD Process_C
          BYTE 'D'
          DWORD Process_D
```

wyróżnik

adres

```
NumberOfEntries = ($ - CaseTable) / EntrySize
```

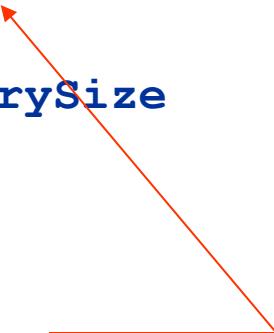
Przełącznik - implementacja x86

Krok 2: Użyj pętli do przeszukiwania tablicy. Wywołaj procedurę, której przesunięcie wpisane w znalezionej pozycji:

```
mov ebx,OFFSET CaseTable    ; EBX wskazuje tablicę
mov ecx,NumberOfEntries     ; licznik pętli

L1: cmp al,[ebx]             ; znaleziono?
    jne L2                  ; nie: kontynuuj
    call NEAR PTR [ebx + 1]  ; tak: wywołaj procedurę
    jmp L3                  ; po zakończeniu opuść pętlę
L2: add ebx,EntrySize        ; przesun na nast.pozycję
    loop L1                 ; powtarzaj, aż ECX = 0

L3:
```



wymagane dla
wskaźników do
procedury

Struktury warunkowe

- Implementacja struktur warunkowych w ASM
 - Blokowe polecenia IF
 - Pętle WHILE
 - Przełącznik CASE
- Wsparcie makroasemblera (MASM86)
 - Polecenia blokowe IF
 - Pętle WHILE
 - Pętle REPEAT

Dyrektywy makroasemblera

- Dyrektywy **.IF**, **.ELSE**, **.ELSEIF**, **.ENDIF** mogą być użyte do utworzenia blokowych poleceń IF.
- Przykłady:

```
.IF eax > ebx  
    mov edx,1  
.ELSE  
    mov edx,2  
.ENDIF
```

```
.IF eax > ebx && eax > ecx  
    mov edx,1  
.ELSE  
    mov edx,2  
.ENDIF
```

- Makroasembler (MASM) utworzy „ukryty” kod źródłowy składający się z instrukcji, etykiet, CMP i skoków warunkowych.

Używane operatory logiczne i porównań


Operator	Description
<i>expr1</i> == <i>expr2</i>	Returns true when <i>expression1</i> is equal to <i>expr2</i> .
<i>expr1</i> != <i>expr2</i>	Returns true when <i>expr1</i> is not equal to <i>expr2</i> .
<i>expr1</i> > <i>expr2</i>	Returns true when <i>expr1</i> is greater than <i>expr2</i> .
<i>expr1</i> >= <i>expr2</i>	Returns true when <i>expr1</i> is greater than or equal to <i>expr2</i> .
<i>expr1</i> < <i>expr2</i>	Returns true when <i>expr1</i> is less than <i>expr2</i> .
<i>expr1</i> <= <i>expr2</i>	Returns true when <i>expr1</i> is less than or equal to <i>expr2</i> .
! <i>expr</i>	Returns true when <i>expr</i> is false.
<i>expr1</i> && <i>expr2</i>	Performs logical AND between <i>expr1</i> and <i>expr2</i> .
<i>expr1</i> <i>expr2</i>	Performs logical OR between <i>expr1</i> and <i>expr2</i> .
<i>expr1</i> & <i>expr2</i>	Performs bitwise AND between <i>expr1</i> and <i>expr2</i> .
CARRY?	Returns true if the Carry flag is set.
OVERFLOW?	Returns true if the Overflow flag is set.
PARITY?	Returns true if the Parity flag is set.
SIGN?	Returns true if the Sign flag is set.
ZERO?	Returns true if the Zero flag is set.

Ekwiwalentny kod ASM

```
.data
val1    DWORD 5
result  DWORD ?

.code
mov eax,6
.IF eax > val1
    mov result,1
.ENDIF
```

Odpowiadający kod ASM:



```
mov eax,6
cmp eax,val1
jbe @C0001
mov result,1
@C0001:
```


MASM automatycznie stosuje **JBE** (bez znaku), gdyż val1 jest liczbą bez znaku.

Ekwiwalentny kod ASM

```
.data
val1      SDWORD 5
result SDWORD ?

.code
mov eax,6
.IF eax > val1
    mov result,1
.ENDIF
```

Odpowiadający kod ASM:




```
mov eax,6
cmp eax,val1
jle @C0001
mov result,1
@C0001:
```

MASM automatycznie stosuje **JLE** (ze znakiem), gdyż val1 jest liczbą ze znakiem.

Ekwiwalentny kod ASM

```
.data
result DWORD ?
.code
mov ebx,5
mov eax,6
.IF eax > ebx
    mov result,1
.ENDIF
```

Odpowiadający kod ASM:



```
mov ebx,5
mov eax,6
cmp eax,ebx
jbe @C0001
mov result,1
@C0001:
```

MASM automatycznie stosuje **JBE** (bez znaku), gdyż obydwa operandy są rejestrami

Ekwiwalentny kod ASM

```
.data
result SDWORD ?
.code
mov ebx,5
mov eax,6
.IF SDWORD PTR eax > ebx
    mov result,1
.ENDIF
```

Odpowiadający kod ASM:

```
mov ebx,5
mov eax,6
cmp eax,ebx
jle @C0001
    mov result,1
@C0001:
```

...chyba, że operand rejestrowy zostanie poprzedzony prefiksem w postaci **operatora SDWORD PTR**. Wówczas używany jest skok **JLE (ze znakiem)**.

Dyrektywa .REPEAT

Program pętli jest wykonywany **przed sprawdzeniem** warunku pętli, umieszczonego po dyrektywie .UNTIL.

Przykład:

```
; Display integers 1 - 10:  
  
mov eax,0  
.REPEAT  
    inc eax  
    call WriteDec  
    call Crlf  
.UNTIL eax == 10
```

Dyrektywa .REPEAT

Ekwiwalentny kod ASM

implementacja:

```
; Display integers 1 - 10:
```

```
@C0001:    mov eax,0  
            inc eax  
            call WriteDec  
            call Crlf  
            cmp eax,10      ; sprawdź warunek pętli  
            jb @C0001       ; fałsz? Opuść pętlę
```

Dyrektywa .WHILE

Warunek pętli sprawdzany jest **przed wykonaniem programu** pętli. Koniec pętli oznaczony jest dyrektywą .ENDW.

Przykład:

```
; Display integers 1 - 10:

mov eax,0
.WHILE eax < 10
    inc eax
    call WriteDec
    call Crlf
.ENDW
```

Dyrektywa .WHILE

Ekwiwalentny kod ASM

implementacja:

```
; Display integers 1 - 10:
```

```
mov eax,0
```

```
cmp eax,10 ; sprawdź warunek pętli
```

```
jae @C0001 ; fałsz? Opuść pętlę
```

```
inc eax
```

```
call WriteDec
```

```
call Crlf
```

```
@C0001:
```

Definicje symboli

Etykiety przy instrukcjach lub polach danych

Wartością etykiety jest adres przypisany linii programu
ASM

Stałe



symbol **EQU** **wartość**

< wartość > ::= <stała > | <inny symbol> | <wyrażenie>

Zwiększa czytelność i zrozumiałość programu

definiowanie uprzednie (niemożliwe odwołania w przód)

Definiowanie symboli

Przykład 1

```
MAXLEN      EQU    4096  
             MOV    EX,#MAXLEN
```

Przykład 2 (def. wielu rejestrów uniwersalnych)

```
BASE  EQU  R1  
COUNT EQU R2  
INDEX EQU  R3
```

Przykład 3

```
MAXLEN      EQU    BUFEND-BUFFER
```

Definiowanie zmiennych programu

Rezerwacja komórek pamięci na zmienne programu

- zmienne z wartością zainicjalizowaną
- zmienne bez zainicjalizowanej wartości

Np. dyrektywy: **DB, DW, DQ, DT**



Dyrektywa DB:

symbol **DB** wyrażenie, [wyrażenie],

Lista odpowiadająca zainicjalizowanym wartościom.
Zwykle „?” oznacza wartość dowolną

Definiowanie zmiennych programu

Przykład 1

etykieta	mnemonik	operand	komentarz
SUB	MOV	ACC,R1	program zaczyna się od procedury
		
	RET		koniec procedury
	DB	1,2,3	dane
STRT	CLR	R2	pierwsza instrukcja programu
		
	END	STRT	wskazuje początek programu

Przykład 2

etykieta	mnemonik	operand
ABC	DB	4 DUP (?)
ENTRY	MOV	ABC, EAX
	
	END	

Przykład 3

etykieta	mnemonik	operand	komentarz
N	EQU	5	
	CLR	R3	
ABC	DB	3	
	ADD	ACC,R2	
		
	END		

Błędnie!!! Zostanie
potraktowane jak kod instrukcji



Przypisanie adresu początkowego

Dyrektywa ORG (origin):

ORG wartość

< wartość > ::= < stała > | < inny symbol > | < wyrażenie >

Pośrednio przypisuje wartości symbolom

Ustawia wskaźnik lokalizacji kodu na wyspecyfikowaną wartość

definiowanie uprzednie (niemożliwe odwołania w przód)

LC	etykieta	mnemonik	operand
000000		MOV	R1,R2
....		...	
000150		ADD	ACC,R1
----		ORG	170
000170		LCALL	NONAME

Przypisanie bloku do pamięci, definiowanie tablic

Przykład

	SYMBOL	VALUE	FLAGS
STAB	6 bajtów	4 bajty	2 bajty
(100 pozycji)			
	:	:	:
	:	:	:
	:	:	:

•Przy użyciu ORG

STAB	DB	1190 DUP (?)
	ORG	STAB
SYMBOL	DB	6 DUP (?)
VALUE	DW	?
FLAGS	DB	2 DUP (?)
	ORG	STAB+1190

Przy użyciu EQU

STAB	DB	1190 DUP (?)
SYMBOL	EQU	STAB
VALUE	EQU	STAB+6
FLAG	EQU	STAB+10

Złożone struktury zmiennych programu



Dyrektywa RECORD:

symbol RECORD n_pola:długość [wyrażenie], [n_pola:długość
[wyrażenie]]

Definiuje strukturę o danej nazwie.
Nie dokonuje alokacji

etykieta	mnemonik	operand	komentarz
DATE	RECORD	YR:16, MO:8, DY:8	definicja struktury „DATE”
		
EARN	DATE	6 DUP (2006,1,1)	Alokacja 6-ciu rekordów z wartościami początkowymi

Złożone struktury zmiennych programu



Dyrektywa STRUCT:

symbol STRUCT [alignment]

deklaracje pól

symbol ENDS

Definiuje strukturę o danej nazwie. Nie dokonuje alokacji

Ustala „raster” lokalizacji w pamięci

etykieta	mnemonik	operand	komentarz
DIM	STRUCT		pocz. definicji struktury „DIM”
LNGHT	DW	?	długość
HGHT	DW	?	wysokość
WDTH	DW	?	szerokość
DIM	ENDS		koniec definicji

Symbole lokalne

dyrektywa QUAL:

QUAL symbol

ASM poprzedza każdy kolejny symbol przedrostkiem
/<symbol>/

etykieta	mnemonik	operand	komentarz
	QUAL	PROC1	symbole lokalne w PROC1
		
ABC		Pełna nazwa /PROC1/ABC
	1,2,3	Dane
	QUAL	PROC2	symbole lokalne w PROC2
		
ABC		Pełna nazwa /PROC2/ABC
	RET		
		
	JMP	PROC2/ABC	

Nie występuje
w MASM 86