Sterowanie pracą programu

Umożliwia podejmowanie decyzji w oparciu o określone warunki.

- **Skoki bezwarunkowe**

Podstawową instrukcją umożliwiającą przeniesienie sterowania do innego punktu programu oznaczonego etykietą jest **JMP** o następującej składni:

JMP etykieta

Ażeby zadeklarować etykietę w programie podajemy jej nazwę, a po niej dwukropek "**:**". Oto przykłady prawidłowych nazw etykiet:

```
label1:
label2:
a:
```

Etykieta może być zadeklarowana w osobnej linii lub przed dowolną instrukcją, np.:

```
x1:
mov  AX, 1

x2:  mov AX, 2
```

Przykłady wykorzystania instrukcji **JMP**:

```
org    100h
mov    ax, 5
mov    bx, 2

jmp    calc

back:  jmp  stop

calc:
add    ax, bx
jmp    back

stop:
ret
```

Instrukcja **JMP** pozwala przenieść sterowanie do przodu lub do tyłu, w dowolne miejsce wewnątrz segmentu.
Prześledzić wykonanie powyższego programu, sprawdzać sposób kodowania instrukcji skoku oraz zachowanie licznika rozkazów.

- **Skoki warunkowe bliskie**

Pozwalają przenieść sterowanie przy spełnieniu określonych warunków. Podzielone są na trzy grupy: pierwsza testuje pojedyncze znaczniki, druga testuje liczby ze znakiem, a trzecia – liczby bez znaku.

Instrukcje skoku warunkowego testujące pojedyncze znaczniki

Instrukcja	Opis	Warunek	Instrukcja przeciwna
JZ , JE	Jump if Zero (Equal).	ZF = 1	JNZ, JNE
JC , JB, JNAE	Jump if Carry (Below, Not Above Equal).	CF = 1	JNC, JNB, JAE
JS	Jump if Sign.	SF = 1	JNS
JO	Jump if Overflow.	OF = 1	JNO
JPE, JP	Jump if Parity Even.	PF = 1	JPO
JNZ , JNE	Jump if Not Zero (Not Equal).	ZF = 0	JZ, JE
JNC , JNB, JAE	Jump if Not Carry (Not Below, Above Equal).	CF = 0	JC, JB, JNAE
JNS	Jump if Not Sign.	SF = 0	JS
JNO	Jump if Not Overflow.	OF = 0	JO
JPO, JNP	Jump if Parity Odd (No Parity).	PF = 0	JPE, JP

Można zauważyć, że niektóre instrukcje o różnych nazwach działają tak samo, są one kodowane za pomocą tych samych kodów maszynowych. Skompilować i przeanalizować następujący fragment kodu.

```
org  100h
  jnc  et
  jnb  et
  jae  et

  mov  AX, 4
et:  mov  AX, 5
ret
```

Instrukcje te są dwubajtowe – pierwszy bajt zawiera kod rozkazu, a drugi określa zakres skoku od -128 do +127 bajtów).

Instrukcje skoku warunkowego dla liczb ze znakiem

Instrukcja	Opis	Warunek	Instrukcja przeciwna
JE , JZ	Jump if Equal (=). Jump if Zero.	ZF = 1	JNE, JNZ
JNE , JNZ	Jump if Not Equal (<>). Jump if Not Zero.	ZF = 0	JE, JZ
JG , JNLE	Jump if Greater (>). Jump if Not Less or Equal (not <=).	ZF = 0 and SF = OF	JNG, JLE
JL , JNGE	Jump if Less (<). Jump if Not Greater or Equal (not >=).	SF <> OF	JNL, JGE
JGE , JNL	Jump if Greater or Equal (>=). Jump if Not Less (not <).	SF = OF	JNGE, JL
JLE , JNG	Jump if Less or Equal (<=). Jump if Not Greater (not >).	ZF = 1 or SF <> OF	JNLE, JG

Instrukcje skoku warunkowego dla liczb bez znaku

Instrukcja	Opis	Warunek	Instrukcja przeciwna
JE , JZ	Jump if Equal (=). Jump if Zero.	ZF = 1	JNE, JNZ
JNE , JNZ	Jump if Not Equal (<>). Jump if Not Zero.	ZF = 0	JE, JZ
JA , JNBE	Jump if Above (>). Jump if Not Below or Equal (not <=).	CF = 0 and ZF = 0	JNA, JBE
JB , JNAE, JC	Jump if Below (<). Jump if Not Above or Equal (not >=). Jump if Carry.	CF = 1	JNB, JAE, JNC
JAE , JNB, JNC	Jump if Above or Equal (>=). Jump if Not Below (not <). Jump if Not Carry.	CF = 0	JNAE, JB
JBE , JNA	Jump if Below or Equal (<=). Jump if Not Above (not >).	CF = 1 or ZF = 1	JNBE, JA

Dla porównania wartości numerycznych używana jest instrukcja **CMP** która działa jak instrukcja **SUB** lecz nie zapisuje wyniku odejmowania tylko ustawia odpowiednio znaczniki.

Przykłady użycia instrukcji **CMP** i skoków warunkowych:

```
include "emu8086.inc"
org    100h

mov    AL, 25     ; ustaw AL na 25.
mov    BL, 10     ; ustaw BL na 10.

cmp    AL, BL     ; porównaj AL - BL.

je     equal      ; skocz jeśli AL = BL (ZF = 1).

putc   'n'        ; program w tym miejscu jeśli AL <> BL,
jmp    stop       ; wyświetla 'n' i skacze do stop.

equal:             ; jeśli program osiągnie to miejsce
putc   'y'        ; oznacza to, że AL = BL, więc wyświetlane jest  'y'.

stop:
ret              ; koniec programu.
```

Sprawdź działanie tego przykładu dla różnych wartości zapisanych do rejestrów **AL** i **BL**, otwórz okno podglądu znaczników i użyj trybu pracy krokowej. Można użyć klawisza skrótu **F5** do rekompilacji i przeładowania program w emulatorze

Organizacja pętli

Instrukcja **Loop et** jest przykładem instrukcji warunkowej, która zmniejsza zawartość rejestru CX o 1, sprawdza, czy zawartość tego rejestru jest różna od 0 i jeśli tak - wykonuje skok do miejsca oznaczonego etykietą **et**, w przeciwnym razie procesor przechodzi do następnej po **loop** instrukcji.

```
mov   CX, 10  //ustawienie licznika powtórzeń pętli
et:
   ciąg instrukcji do wykonania w pętli
loop   et
```

Instukcje pętli

Instrukcja	Operacja i warunek skoku	Instrukcja przeciwna
LOOP	decrease CX, jump to label if CX not zero.	DEC CX and JCXZ
LOOPE	decrease CX, jump to label if CX not zero and equal (ZF = 1).	LOOPNE
LOOPNE	decrease CX, jump to label if CX not zero and not equal (ZF = 0).	LOOPE
LOOPNZ	decrease CX, jump to label if CX not zero and ZF = 0.	LOOPZ
LOOPZ	decrease CX, jump to label if CX not zero and ZF = 1.	LOOPNZ
JCXZ	jump to label if CX is zero.	OR CX, CX and JNZ

W przeciwieństwie do instrukcji **JMP** instrukcje skoków warunkowych i pętli pozwalają przenieść sterowanie o **127** bajtów do przodu i **128** bajtów do tyłu (pamiętajmy, że instrukcje w kodzie maszynowym mają długość od 1 do 6 bajtów)

Możemy łatwo obejść to ograniczenie w następujący sposób:

- Bierzemy z tabeli przeciwny rozkaz skoku warunkowego i podajemy adres skoku do etykiety ***label_x***.
- Używamy instrukcji **JMP** do przeskoczenia do pożądanej lokalizacji.
- Definiujemy etykietę ***label_x:*** zaraz za instrukcją **JMP**.

Etykieta ***label_x:*** musi być unikalna w całym programie.

Przykład:

```
include "emu8086.inc"
org    100h

mov    AL, 5
mov    BL, 5

cmp    AL, BL     ; porównanie  AL - BL.


; je equal                ; rozkaz 2-bajtowy

jne    not_equal  ; skok, jeśli  AL <> BL (ZF = 0).

jmp    equal              ; skok w dowolne miejsce w segmencie

not_equal:

add    BL, AL
sub    AL, 10
xor    AL, BL

jmp  skip_data
db 256 dup(0)     ; 256 bytes

skip_data:

putc   'n'                ; jeśli  AL <> BL,
jmp    stop               ; wyświetlamy 'n'  i skok do etykiety stop.

equal:                    ; jeśli AL = BL
putc   'y'                ; wyświetlamy 'y'.

stop:
ret
```

Funkcje systemowe (BIOS, DOS) do obsługi urządzeń wejścia-wyjścia

Wyświetlanie znaku na ekranie

```
mov    DL, ‘A’
mov    AH, 2
int    21h
```

Czytanie znaku z klawiatury do AL.

```
mov    AH, 1            // lub  mov  AH, 7 gdy czytamy bez echa
int    21h
```

Czytanie napisu z klawiatury

```
mov   DX, offset bufor
mov   AH, 0Ah
int    21h
…
bufor  DB  10, ?, 10 dup (‘-‘)
```

gdzie 10 oznacza wielkość bufora na napis, zaś w miejscu ? będzie umieszczona liczba znaków przeczytanych. Aby wyświetlić odczytany napis używamy funkcji 9 i adresu DS:DX+2

Wyświetlenie napisu na ekranie

(napis w tablicy msg musi być zakończony znakiem $)

```
mov   DX, offset msg          // lub    lea  DX, msg
mov   AH, 9
int   21 h
…..
msg   DB  “Witaj $”
```

Przykład wczytania napisu i wyświetlenie go na ekranie

```
org 100h
      mov  dx, offset buffer
      mov  ah, 0Ah
      int  21h          ; wczytanie napisu do bufora

      xor  bx, bx       ; zerowanie rejestru bx
      mov  bl, buffer[1]
      mov  buffer[bx+2],  '$'   ; dodanie znaku ‘$’ na końcu napisu
      mov  dx, offset buffer + 2
      mov  ah, 9
      int  21h          ; wyświetlenie napisu na ekranie
ret
      buffer  db  10, ?,  10 dup('-')
```

Oczekiwanie na wciśnięcie klawisza – pobranie znaku bez echa

```
mov  AH, 0
int  16h
```

Po wykonaniu

AH = kod przeglądania BIOS

AL = kod znaku ASCII

Sprawdzenie czy w buforze klawiatury jest znak (bez pobierania znaku)

```
mov   AH, 1
int   16h
```

Po wykonaniu

ZF = 1 jeśli klawisz nie był wciśnięty
ZF = 0 jeśli klawisz nie wciśnięty
AH = kod przeglądania BIOS
AL = kod znaku ASCII

Dalsze informacje na temat przerwań systemowych w pomocy pod tytułem Interrupts

Zadania

1. Napisać program, który porównuje liczby zapisane w AX oraz BX i zeruje rejestr o mniejszej zawartości (zakładamy, że liczby są różne).

```
org 100h                ; Uwaga: wyświetlać okienko rejestru znaczników flags
   mov AX, 5
   mov BX, 3
   cmp AX, BX       ; porównanie AX i BX
   jl et1           ; przeskocz zerowanie BX jeśli AX<BX
   mov BX,0         ; wyzeruj BX
   jmp et2
et1: mov AX, 0      ; w przeciwnym razie wyzeruj AX

et2:
   mov AX, 4
   mov BX, 6
   cmp AX, BX
   jl et3
   mov BX,0
   jmp et4
et3: mov AX, 0

et4:
   mov AX, -10      ; działa również na liczbach ujemnych w kodzie U2
   mov BX, 3
   cmp AX, BX
   jl et5
   mov BX,0
   jmp et6
et5: mov AX, 0

et6:

   ret
```

2. Napisać program, który sprawdza, czy liczba zapisana w AX jest dodatnia – jeśli tak to pozostawia ją bez zmiany, a jeśli nie - to zeruje rejestr AX.

```
org 100h          ; Uwaga: wyświetlać okienko rejestru znaczników flags
   mov AX, 5   ; jeśli dodatnia to pozostawia bez zmiany
   cmp AX, 0
   jge et1
   mov AX, 0
et1:
   mov AX, -3  ; jeśli ujemna to zeruje
   cmp AX, 0
   jge et2
   mov AX, 0
et2:
   mov AX, 0   ; jeśli zero to pozostawia bez zmiany
   cmp AX, 0
   jge et3
   mov AX, 0
et3:

   ret
```

3. Napisać program, który zawartość zmiennych X i Y ustawia tak, by w zmiennej X była wartość większa (liczby są różne).

```
org 100h     ; Uwaga: wyświetlać okienko zmiennych vars
   mov X, 5
   mov Y, 3
   mov AL, Y   ; nie można porównywać bezpośrednio dwóch komórek pamięci X i Y
   cmp AL, X
   jl dalej1      ; jeśli Y<X to pomijamy zamianę
   xchg AL, X
   xchg AL, Y
dalej1:
   mov X, 6
   mov Y, 10
   mov AL, Y
   cmp AL, X
   jl dalej2
   xchg AL, X
   xchg AL, Y
dalej2:
   ret
   X DB 0  ; zmienne deklarujemy poza obszarem instrukcji (za instrukcja ret)
   Y DB 0
```

4. Zadeklarowana jest tablica o 8 elementach typu bajt. Policzyć ile elementów jest ujemnych i wynik umieścić w zmiennej **licznik**.

```
org 100h    ; Uwaga: wyświetlać zawartość zmiennych w okienku vars
   mov licznik, 0
   mov SI, 0   ; odwołanie do elementu tablice za pomocą indeksu
   mov CX, 8   ; licznik powtórzeń pętli loop, obserwować CX i SI w czasie wykonywania pętli
et: cmp Tablica[SI], 0
   jge dalej   ; jeśli liczba jest dodatnia lub 0 - nie liczymy jej
   inc licznik
dalej:
   inc SI      ; zwiększanie rejestru indeksowego - przejście do kolejnego elementu tablicy
   loop et     ; licznik powtórzeń pętli CX zmniejszany o 1, jeśli CX > 0 to powrót do et:

   ret

licznik DB 0
Tablica DB 9, -3, -5, 2, 4, -2, 7, 1    ; tablica 8-elementowa, można zmieniać jej zawartość
```

5. Zadeklarowana jest tablica o 12 elementach typu słowo 16-bitowe. Policzyć ile elementów jest zerowych i wynik umieścić w zmiennej **zero**.

```
org 100h    ; Uwaga: wyświetlać zawartość zmiennych w okienku vars
   mov zero, 0
   mov SI, 0   ; odwołanie do elementu tablice za pomocą indeksu
   mov CX, 12   ; licznik powtórzeń pętli loop, obserwować CX i SI w czasie wykonywania pętli
et: mov BX, Tab[SI]
   cmp w.Tab[SI], 0
   jne dalej   ; jeśli liczba jest różna od 0 - nie liczymy jej
   inc zero
dalej:
   inc SI      ; zwiększanie rejestru indeksowego - przejście do kolejnego elementu tablicy
   inc SI      ; dwa razy inc bo tablica typu word
   loop et     ; licznik powtórzeń pętli CX zmniejszany o 1, jeśli CX > 0 to powrót do et:

   ret

zero DB 0
Tab DW 9, 0, -3, -5, 0,  2, 4, 0, 0, -2, 7, 1    ; tablica 12-elementowa, można zmieniać jej zawartość
```

6. Zadeklarować stałą K o wartości 7. Napisać program, który wpisuje tą wartość do wszystkich 8-bitowych rejestrów procesora 8086 oraz do 10-elementowej tablicy T.
7. Zadeklarować tablicę o wielkości 10 elementów i wypełnić ją w programie kolejnymi potęgami liczby 2: 1, 2, 4, 8....
8. Zadeklarować tablicę o wielkości 16 elementów i wypełnić ją w programie kolejnymi wielokrotnościami liczby 20: 20, 40, 60...
9. Zadeklarować tablicę i wypełnić ją określoną wartością stałej **wzor EQU 5.**

10. Zadeklarować tablicę 10-elementową i wypełnić ją kolejnymi wartościami parzystymi.
11. Zadeklarować tablicę Tab1 DB 10 dup (?) oraz Tab2 zawierającą napis ‘As ma kota’. Skopiować zawartość tablicy Tab2 do Tab1.
12. Utworzyć dwie tablice zawierające napisy ‘HALO’ i ‘ALFA’. Korzystając z rozkazu XCHG zamienić napisy miejscami.
13. Wczytać do tablicy Bufor 10 znaków z klawiatury.
14. Wczytać z klawiatury liczbę podaną w postaci szesnastkowej i zapisać ją w zmiennej **liczba**.
15. Wyświetlić na ekranie znak wprowadzony z klawiatury zadaną liczbę razy.
16. Wczytać z klawiatury swoje imię i wyświetlić na ekranie 3 razy.
17. Wyświetlić na ekranie znaki o kodach ASCII z przedziału od 32 do 255 po 16 znaków w wierszu.
18. Wczytać z klawiatury napis złożony z małych liter i wyświetlić dużymi literami (sprawdzić w tablicy kodów ASCII przesunięcie pomiędzy małymi i dużymi literami).
19. Wyświetlić na ekranie zadany obszar pamięci w postaci:
    wartość szesnastkowa – znak ASCII – wartość dziesiętna.
    Adres początkowy tablicy i jej wielkość podać w zmiennych **adres** i **liczba**.