Ćwiczenie

„Sterowanie przebiegiem wykonania programu”

Tematy ćwiczenia

- porównania i skoki warunkowe
- pętle

Sprawozdanie

Na każdym ćwiczeniu sporządza się sprawozdanie na bazie materiałów ćwiczenia. Bazowa zawartość sprawozdania musi być przygotowana w domu przed ćwiczeniem (sprawozdanie do ćwiczenia pierwszego jest przygotowywane w czasie ćwiczenia). W czasie ćwiczenia do sprawozdania są dodawane wyniki testowania.

Treść sprawozdania:

strona tytułowa,

spis treści sporządzony za pomocą *Word'*a,

dla każdego punktu rozdziały "Zadanie ", "Opracowanie zadania" (rozdział z tekstem programu i komentarzami), "Testowanie" (rozdział z opisem danych wejściowych i wynikami testowania, w tym zrzuty aktywnego okna).

Nazwa (bez polskich liter, żeby można było archiwizować) pliku ze sprawozdaniem musi zawierać nazwę przedmiotu, numer ćwiczenia i nazwisko studenta, na przykład "PN_...".

Pliki ze sprawozdaniem są przekazywane do archiwum grupy.

Pętle

Zadanie

Porównać 10 obiektów według swojego zadania (patrz tabele wariantów).

Wprowadzać dane dla każdego obiektu w cyklu powtórzeń.

Porównać obiekty każdy z każdym według swojej funkcji porównania**.**

Wynik porównania pary obiektów zapisywać do tabeli kwadratowej jako znak = (równe), znak > (większy), znak < (mniejszy).

Wyniki porównań obiektów pokazać w postaci tabeli, na przykład:

```
  1 2 3 4
1 = > > <
2 < = = <
3 < = = >
4 > > < =
```

Stosować dyrektywę INVOKE dla wywołania podprogramów, dyrektywę PROTO dla opisu podprogramów oraz dyrektywę LOCAL dla lokalnych zmiennych podprogramów.

Opracowanie zadania

<<tekst programu>>

Testowanie

<<zrzut ekranu MS DOS>>

Tabela wariantów

Np	Obiekt	Funkcja porównania		Np	Obiekty	Funkcja porównania
1	Liczba	Równość		**17**	Liczba	Mniejszy
2	Odcinek	Równość		**18**	Odcinek	Mniejszy
3	Kwadrat	Równość		**19**	Kwadrat	Mniejszy
4	Romb	Równość		**20**	Romb	Mniejszy
5	Prostokąt	Równość		**21**	Prostokąt	Mniejszy
6	Trójkąt	Równość		**22**	Trójkąt	Mniejszy
7	Imię	Równość		**23**	Imię	Mniejszy
8	Nazwisko	Równość		**24**	Nazwisko	Mniejszy
9	Liczba	Nierówność		**25**	Liczba	Większy
10	Odcinek	Nierówność		**26**	Odcinek	Większy
11	Kwadrat	Nierówność		**27**	Kwadrat	Większy
12	Romb	Nierówność		**28**	Romb	Większy
13	Prostokąt	Nierówność		**29**	Prostokąt	Większy
14	Trójkąt	Nierówność		**30**	Trójkąt	Większy
15	Imię	Nierówność		**31**	Imię	Większy
16	Nazwisko	Nierówność		**32**	Nazwisko	Większy

Pole S trójkąta z bokami a, b, c można obliczyć według wzoru Herona:

```
p = (a + b + c)/2;
S = sqrt(p*(p - a)*(p - b)*(p - c))
```

Ponieważ trójkąty tylko są porównywane, to można uniknąć pierwiastkowania przez porównywanie wartości `(p*(p - a)*(p - b)*(p - c))` lub

```
((a + b + c)*(-a + b + c)*(a - b + c)*(a + b - c))
```

Wskazówki

Pętle

Do realizacji pętli można stosować instrukcje `LOOP, LOOPE (LOOPZ), LOOPNE (LOOPNZ)`:

```
 loop powt ;skok do etykiety "powt", jeśli zawartość ECX>0

 loope powt ;skok do etykiety "powt", jeśli zawartość ECX>0 i znacznik ZF
jest ustawiony

 loopne powt ;skok do etykiety "powt", jeśli zawartość ECX>0 i znacznik ZF
nie jest ustawiony
```

Instrukcje pętli `LOOP, LOOPE (LOOPZ)` i `LOOPNE (LOOPNZ)` są instrukcjami skoku bliskiego (ang. *short jump*), co oznacza, że etykieta musi znajdować się na odległości od -128 do +127 bajtów. Programista może nie obliczać dlugość skoku (a zrobić to bardzo trudno, ponieważ rozkazy zajmują od 1 do 12 bajtów), bo kompilator sam zamieni instrukcję `LOOP` na instrukcję długiego skoku (instrukcję `JMP`).

Instrukcja pętli `LOOPE (LOOPZ)` jest często stosowana do wyszukiwania w tablicy pierwszego elementu nierównego wzorcowi, na przykład:

```
.DATA
 tabl DW 100 dup (0)
.CODE
 ...
 mov ECX, 100   ;100 - rozmiar tablicy
 lea EBX, tabl ;adres tablicy „tabl”
 sub EBX,2 ;cofnięcie w tablicy typu DW o jeden element
powt:
 add EBX,2 ;przesunięcie w tablicy typu DW o jeden element
 cmp WORD PTR [EBX],1 ;porównanie ze wzorcem (liczbą 1)
 loope powt ;skok, jeśli ECX > 0 i ZF == 1
 ... ;wyjście z pętli, jeśli element tablicy nie równa się 1
```

Do odnalezienia elementu równego wzorcowi jest zwykle stosowana instrukcja pętli `LOOPNE (LOOPNZ)`, na przykład:

```
.DATA
 tabl DW 100 dup (0)
.CODE
 ...
 mov ECX,100          ;100 - rozmiar tablicy
 lea EBX,tabl   ;adres tablicy „tabl”
 sub EBX,2  ;cofnięcie w tablicy typu DW o jeden element
powt:
 add EBX,2 ;przesunięcie w tablicy typu DW o jeden element
 cmp WORD PTR [BX], 1;porównanie ze wzorcem (liczbą 1)
 loopne     powt      ;skok,loopne powt ;skok, jeśli CX > 0 i ZF == 0
 ... ;wyjście z pętli, jeśli element tablicy równa się 1
```

W przypadku dużej odległości (tj. większej niż 127 bajtów) od miejsca skoku do początku cyklu należy stosować skok bezwarunkowy *jmp*.

Możliwa struktura cyklu powtórzeń ze sprawdzaniem warunku powtórzenia przed obliczeniami (tj. struktura "while-do"):

```
;--- cykl powtórzeń
  mov   N,0 ;zerowanie zmiennej N cyklu (w sekcji danych: N DD 0)
etC:       ; etykieta początku cyklu (etykieta miejsca powrotu)
  cmp   N, 100 ; porównanie z wartością graniczną, na przykład 100
  jb    etD ;skok, jeśli aktualna wartość N jest mniejsza niż maksymalna
  jmp   etK ; skok poza granicę cyklu
etD:       ; etykieta początku obliczeń wewnątrz cyklu
... obliczenia wewnątrz cyklu
;--- przejście na powtórzenie
  inc   N       ; zwiększenie zmiennej cyklu
  jmp   etC     ; skok do etykiety miejsca powrotu
;--- koniec cyklu
etK:            ; etykieta poza granicą cyklu
```

Zamiast etykiety "etD" można zastosować etykietę anonimową:

```
  jb    @F  ;skok, jeśli aktualna wartość N jest mniejsza niż maksymalna
  jmp   etK ; skok poza granicę cyklu
@@:        ; etykieta początku obliczeń wewnątrz cyklu
... obliczenia wewnątrz cyklu
```

Inna struktura cyklu powtórzeń ze sprawdzaniem warunku powtórzenia na końcu obliczeń (tj. struktura "do-while"):

```
;--- cykl powtórzeń
  mov   N,0 ;zerowanie zmiennej N cyklu (w sekcji danych: N DD 0)
etC:       ; etykieta początku cyklu (etykieta miejsca powrotu)
           ; początek obliczeń wewnątrz cyklu
... obliczenia wewnątrz cyklu
;--- przejście na powtórzenie
  inc   N       ; zwiększenie zmiennej cyklu
  cmp   N, 100 ; porównanie z wartością graniczną, na przykład 100
  je etK  ;skok poza granicę cyklu (jeśli wartość N jest równa maksymalnej)
  jmp   etC     ; skok do etykiety początku cyklu (miejsca powrotu)
;--- koniec cyklu
etK:            ; etykieta poza granicą cyklu
```

Zamiast etykiety "etK" można zastosować etykietę anonimową:

```
;--- przejście na powtórzenie
  inc   N       ; zwiększenie zmiennej cyklu
  cmp   N, 100 ; porównanie z wartością graniczną, na przykład 100
  je @F   ;skok poza granicę cyklu,
            ;jeśli aktualna wartość N jest równa maksymalnej
  jmp   etC     ; skok do etykiety początku cyklu (miejsca powrotu)
;--- koniec cyklu
@@:             ; etykieta poza granicą cyklu
```

Funkcja wsprintf

Do formatowania tekstu przy wyprowadzeniu często jest stosowana funkcja API Win32 wsprintf w dwóch modyfikacjach:

wsprintA dla wierszy ze znakami ASCII i
wsprintfW dla wierszy ze znakami Unicode.

Zaletą funkcji wsprintf jest zmienna liczba argumentów. Funkcja oblicza ilość argumentów na podstawie ilości specyfikatorów formatu w ciągu znaków formatującym.

Argumenty są przekazywane do funkcji wsprintf według reguł języka C. Dlatego prototyp funkcji musi wyglądać następująco:

```
wsprintfA PROTO C :VARARG
```

Po wywołaniu funkcji wsprintf należy przesunąć wskaźnik stosu ESP o ilość bajtów odłożonych na stos przed wywołaniem funkcji.

Funkcja wsprintf ma argumenty:

- wskaźnik na bufor do umieszczenia tekstu wynikowego,
- wskaźnik na wiersz z formatującym ciągiem znaków oraz zakończony zerem,
- argumenty formatowania.

Funkcja wsprintf zwraca w rejestrze EAX ilość znaków tekstu wynikowego.

Przykład wywołania funkcji wsprintf:

```
_DATA SEGMENT
  zapr DB 0Dh,0Ah,"Proszę wprowadzić parametr %ld obiektu %ld [Enter]: ",0
  ALIGN         4
  Npar  DD    ? ; indeks parametru
  Nob   DD    ? ; indeks obiektu
  temp     DB   128 dup(?)
  rznak    DD   ? ; ilość znaków po formatowaniu
_DATA ENDS
_TEXT SEGMENT

;--- zaproszenie
     push  Nob  ; indeks obiektu
     push  Npar ; indeks parametru
     push  OFFSET zapr
      push  OFFSET temp
      call  wsprintfA  ; zwraca ilość znaków w buforze
      add   ESP, 16          ; czyszczenie stosu
      mov   rznak, EAX ; zapamiętywanie ilości znaków
```

lub (z dyrektywą INVOKE)

```
;--- zaproszenie
      INVOKE wsprintfA,OFFSET temp,OFFSET zapr,Npar,Nob ;zwraca ilość znaków
      mov   rznak, EAX ; zapamiętywanie ilości znaków
```

Operacje na wierszach

Do operacji nad wierszami (ciągami znaków ASCII zakończonych zerem) można użyć funkcji **lstrcmpA, lstrcatA, lstrlenA.** Wierszy są przekazywane przez adresy pierwszych znaków.

Definicje funkcji znajdują się w pliku *kernel32.inc*.

Funkcja **lstrcmpA** porównuje dwa wierszy *lp1* i *lp2*:

```
lstrcmpA PROTO STDCALL lp1:DWORD,lp2:DWORD
```

Funkcja zwraca

-1 jeśli lp1 < lp2 (przy równej długości są porównywane kody znaków);

0 jeśli lp1 = lp2 (wierszy są równe po długości, oraz równe też kody znaków);

+1 jeśli lp1 > lp2 (przy równej długości są porównywane kody znaków).

Funkcja **lstrcatA** dołącza wiersz *lp2* do wierszu *lp1*:

```
lstrcatA PROTO STDCALL lp1:DWORD,lp2:DWORD
```

Funkcja **lstrlenA** oblicza i zwraca długość wierszu *lp*:

```
lstrlenA PROTO STDCALL lp:DWORD
```

Rozkazy procesorów Intel

Kodowanie rozkazów procesorów Intel jest przedstawione w tabelach przytoczonych niżej, gdzie oznaczono:

`A` – rejestr-akumulator EAX (AX, AH, AL),
`EA` – adres efektywny,
`Num` – numer portu,
`param8/16/32` – parametr 8-, 16- lub 32-bitowy,
`przes8/16/32` – przesunięcie 8-, 16- lub 32-bitowe,
`P` – pamięć,
`R` – rejestr,
`r8/16/32` – rejestr 8-, 16- lub 32-bitowy,
`R/P` – rejestr lub pamięć,
`Rs` – rejestr segmentowy.

Rozkaz „cmp” jest opisany w grupie rozkazów arytmetycznych.

Grupa `String Manipulation` (operacje z wierszami, tablicami)

Rozkazy „`cmps`” i „`scas`” tej grupy są przystosowane do wielokrotnego porównania dwóch argumentów.

Tabela

Kodowanie rozkazów grupy `String Manipulation` (operacje z wierszami, tablicami)

Mnemonik i operandy	Bajt 0	Bajt 1
`cmps P1, P2`	`1010011w`	
`scas P`	`1010111w`	
`repe cmps P1, P2`	`11110011`	`1010011w`
`repe scas P`	`11110011`	`1010111w`
`repne cmps P1, P2`	`11110010`	`1010011w`
`repne scas P`	`11110010`	`1010111w`
`ins P1, DX`	`0110110w`	
`outs P1, DX`	`0110111w`	
`lods P`	`1010110w`	
`stos P`	`1010101w`	
`movs P1, P2`	`1010010w`	
`rep ins P1, DX`	`11110010`	`0110110w`
`rep outs P1, DX`	`11110010`	`0110111w`
`rep lods P`	`11110010`	`1010110w`
`rep stos P`	`11110010`	`1010101w`
`rep movs P1, P2`	`11110010`	`1010010w`
`xlat P`	`11010111`	

Wykonując rozkaz „cmps” procesor porównuje dwa operandy przez odejmowanie, w wyniku którego ma miejsce ustawienie znaczników: OF, SF, ZF, AF, PF, CF, - ale wynik odejmowania nie jest zapisywany.

Operand pierwszy jest pobierany z komórki pamięci z pod adresu ES:EDI, a operand drugi - z komórki pamięci z pod adresu DS:ESI.

W końcu operacji zawartość EDI i ESI zwiększą się lub zmniejszą się w zależności od znacznika kierunku DF w rejestrze znaczników (DF=0 - zwiększenie, DF=1 - zmniejszenie) o 1 dla operacji bajtowej, 2 dla operacji nad słowem, lub 4 dla operacji nad podwójnym słowem.

Przy wykonaniu rozkazu „scas” procesor też porównuje operandy, przy tym procesor operuje z domyślnym drugim operandem z rejestru-akumulatora.

Operand pierwszy jest pobierany z komórki pamięci z pod adresu ES:EDI.

Wynik odejmowania też nie jest zapisywany, ma miejsce tylko ustawienie znaczników: OF, SF, ZF, AF, PF, CF. W końcu operacji rejestr EDI zwiększa się lub zmniejsza się w zależności od znacznika kierunku DF w rejestrze znaczników (DF=0 - zwiększenie, DF=1 - zmniejszenie) o 1 dla operacji bajtowej, 2 dla operacji nad słowem, lub 4 dla operacji nad podwójnym słowem.

Rozkazy „repe cmps”, „repne cmps”, „repe scas” i „repne scas” zawierają przedrostki „repe” i „repne”.

Przedrostek „repe” (nazwa od ang. *repeat equal*) powoduje powtórzenie operacji porównania „cmps” lub „scas”. Ilość powtórzeń musi być zapisana do rejestru CX. Po każdemu porównaniu zawartość rejestru CX jest zmniejszana o 1. Cykl będzie zakończony przy zerowej zawartości rejestru CX. Drugim warunkiem zakończenia cyklu jest nierówność operandów, tj. zerowa wartość znacznika ZF.

Stosując rozkazy „repe cmps” i „repe scas” można porównywać elementy dwóch tablic lub element tablicy i zawartość rejestru-akumulatora. Ilość porównań jest ograniczona początkowej zawartością rejestru CX, i porównania mogą być zakończone wcześniej na pierwszych nierównych elementach dwóch tablic (dla „repe cmps”) lub elementu tablicy i zawartości rejestru-akumulatora (dla „repe scas”).

Podobny przedrostek „repne” (nazwa od ang. *repeat not equal*) też powoduje powtórzenie operacji porównania „cmps” lub „scas” w ilości równej początkowej zawartości rejestru CX. Dodatkowy warunek zakończenia porównań – niezerowa wartość znacznika ZF. Innymi słowy porównania będą zakończone na pierwszych równych elementach dwóch tablic (dla „repne cmps”) lub elementu tablicy i zawartości rejestru-akumulatora (dla „repne scas”).

Do wprowadzenia elementu tablicy z portu i wyprowadzenia elementu tablicy do portu służą rozkazy „ins” i „outs” odpowiednio. Element tablicy jest pobierany z komórki lub zapisywany do komórki pamięci z pod adresu ES:EDI, a numer portu musi być zapisany do rejestru DX. W końcu operacji procesor zmienia zawartość rejestru EDI w zależności od znacznika kierunku DF: zwiększa przy DF równym 0 lub zmniejsza przy DF równym 1.Wielkość zmiany zawartości rejestru indeksowego EDI zależy od rozmiaru danej i może być równa 1, 2 lub 4.

Wykonując rozkazy „lods” i „stos” procesor przesyła daną między rejestrem-akumulatorem i komórką pamięci. Adres komórki pamięci znajduje się w rejestrach:

- DS:ESI dla rozkazu „lods”,
- ES:EDI dla rozkazy „stos”.

Zawartość rejestru indeksowego ESI lub EDI jest zwiększana, jeśli znacznik kierunku DF ma wartość zerową, i zmniejszana, jeśli znacznik kierunku DF ma wartość niezerową. Zwiększenie lub zmniejszenie zawartości rejestru EDI jest równe 1, 2 lub 4 w zależności od rozmiaru danej.

Rozkaz „movs” przesyła daną 8-, 16- lub 32-bitową między dwoma komórkami pamięci, adresy których muszą być zapisane do rejestrów DS:ESI (źródło danej) i ES:EDI (miejsce przeznaczenia).

Po zakończeniu przesyłania zawartość obydwóch rejestrów indeksowych ESI i EDI zmienia się w kierunku, który zależy od wartości znacznika kierunku DF: zwiększa się, jeśli znacznik kierunku DF ma wartość zerową, i zmniejsza się, jeśli znacznik kierunku DF ma wartość niezerową.

Przedrostek „rep” (nazwa od ang. *repeat*) w rozkazach „rep ins”, „rep outs”, „rep lods”, „rep stos” i „rep movs” powoduje powtórzenie operacji. Procesor powtarza wykonanie rozkazu „rep ins”, „rep outs”, „rep lods”, „rep stos” lub „rep movs” dopóki zawartość rejestru CX jest niezerowa.

Po każdemu wykonaniu rozkazu zawartość rejestru CX jest zmniejszana o 1.

Rozkaz „xlat” (nazwa na pewnie od ang. *exchange low accumulator table*) wykonuje zamianę zawartości rejestru AL na bajt z komórki z pod adresu [DS:EBX+AL]. Jak widać poprzednia zawartość rejestru AL stosuje się jako indeks bajtowy w tablicy z pod adresu DS:EBX.

Grupa `Control Transfer` (przejście sterowane)

Do grupy `Control Transfer` (przejście sterowane) należą rozkazy związane z wywołaniem podprogramów:

rozkaz „`call`" wywołujący podprogram i

rozkaz „`ret`" do powrotu z podprogramu.

Procesor wykonując rozkaz „`call`" ładuje na stos stan licznika rozkazów. W przypadku tzw. *dalekiego wywołania podprogramu* (formaty „`call wskaz16/32`" i „`call wskazR/P`") ma miejsce dodatkowe ładowanie na stos zawartości rejestru segmentowego CS.

Wykonując rozkaz „`ret`" lub „`retf`" procesor bierze ze stosu stan licznika rozkazów, i tym samym wraca do miejsca wywołania podprogramu. Dalekiemu wywołaniu podprogramu musi odpowiadać daleki rozkaz „`retf`", przy wykonaniu którego ze stosu jest zdejmowany segmentowy adres, który jest zapisywany do rejestru CS. Formaty „`ret param`" i „`retf param`" rozkazu „`ret`" zawierają parametr równy ilości bajtów stosu, które były potrzebny do przekazywania argumentów. Procesor koryguje wskaźnik stosu na ilość bajtów równą parametrowi.

Rozkaz „`jmp`" jest wykonywany jako skok do wskazanego w operandzie adresu komórki pamięci. W formatach „`call wskaz16/32`" i „`call wskazR/P`" adres skoku zawiera nową zawartość rejestru segmentowego CS, tj. wykonuje się *daleki skok*.

Skok wywołany przez rozkaz „`jmp`" do pierwszej instrukcji podprogramu jest interpretowany jako wywołanie podprogramu. Można przed rozkazem „`jmp`" załadować na stos adres powrotu i tym samym imitować rozkaz „`call`".

Tabela

Kodowanie rozkazów grupy `Control Transfer` (przejście sterowane)

Mnemonik i operandy	Bajt 0	Bajt 1

`jmp przes8`
`jmp przes16/32`
`jmp wskaz16/32`
`jmp przesR/P`
`jmp wskazR/P`
	`11101011`
	`11101001`
	`11101010`
	`11111111`
	`11111111`
		`przes8`
		`przes16/32`
		`wskaz16/32`
		`mod_100_r/m`
		`mod_101_r/m`

`call przes16/32`
`call wskaz16/32`
`call przesR/P`
`call wskazR/P`
	`11101000`
	`10011010`
	`11111111`
	`11111111`
		`przes16/32`
		`wskaz16/32`
		`mod_010_r/m`
		`mod_011_r/m`

`ret`
`ret param`
`retf`
`retf param`
	`11000011`
	`11000010`
	`11001011`
	`11001010`
		
		`param16`
		
		`param16`

Grupa `Conditional Jumps` (skoki warunkowe)

Liczna grupa skoków warunkowych zawiera warunkowe skoki w granicach segmentu.
Mnemoniki rozkazów to skróty słów języka angielskiego:
jo – *jump if overflow* (skok, jeśli nadmiar),
jno – *jump if not overflow* (skok, jeśli brak nadmiaru),
jb – *jump if below* (skok, jeśli „mniejszy"),
jnae – *jump if not above and not equal* (skok, jeśli „niewiększy" i „nierówny"),
jnb – *jump if not below* (skok, jeśli „niemniejszy"),
jae – *jump if above or equal* (skok, jeśli „większy" lub „równy"),
je – *jump if equal* (skok, jeśli „równy"),
jz – *jump if zero* (skok, jeśli „zerowy"),
jne – *jump if not equal* (skok, jeśli „nierówny"),
jnz – *jump if not zero* (skok, jeśli „niezerowy"),
jbe – *jump if below or equal* (skok, jeśli „mniejszy" lub „równy"),
jna – *jump if not above* (skok, jeśli „niewiększy"),
jnbe – *jump if not below and not equal* (skok, jeśli „niemniejszy" i „nierówny"),
ja – *jump if above* (skok, jeśli „większy"),
js – *jump if signed* (skok, jeśli „ujemny"),
jns – *jump if not signed* (skok, jeśli „nieujemny"),
jp – *jump if parity* (skok, jeśli parzystość),
jpe – *jump if parity even* (skok, jeśli „parzyste"),
jnp – *jump if not parity* (skok, jeśli brak parzystości),
jpo – *jump if parity odd* (skok, jeśli „nieparzyste"),
jl – *jump if less* (skok, jeśli „mniejszy (ze znakiem)"),
jnge – *jump if not greater and not equal* (skok, jeśli „niewiększy (ze znakiem)" i „nierówny (ze znakiem)"),
jnl – *jump if not less* (skok, jeśli „niemniejszy (ze znakiem)"),
jge – *jump if greater or equal* (skok, jeśli „większy (ze znakiem)" lub „równy (ze znakiem)"),
jle – *jump if less or equal* (skok, jeśli „mniejszy (ze znakiem)" lub „równy (ze znakiem)"),
jng – *jump if not greater* (skok, jeśli „niewiększy (ze znakiem)"),
jnle – *jump if not less and not equal* (skok, jeśli „niemniejszy (ze znakiem)" i „nierówny (ze znakiem)"),
jg – *jump if greater* (skok, jeśli „większy (ze znakiem)"),
jcxz – *jump if CX zero* (skok, jeśli „CX zerowy"),
jecxz – *jump if ECX zero* (skok, jeśli „ECX zerowy").

Do grupy są dołączone rozkazy przydatne do skonstruowania pętli programowej: `loop`, `loopz (loope)` i `loopnz (loopne)`.

Operandem tych rozkazów jest przesuniecie w kodzie do etykiety – adresu pierwszej instrukcji pętli. Przesuniecie może być w zakresie od -128 do +127.

Wykonując każdy z tych rozkazów procesor zmniejsza o 1 zawartość rejestru CX, sprawdza, czy zawartość jest równa 0 i przechodzi do etykiety, jeśli zawartość rejestru CX niezerowa.

Wykonując rozkaz `loopz (loope)` procesor dodatkowo sprawdza stan znacznika ZF i przechodzi do etykiety, gdy znacznik ZF ma wartość 1. Rozkaz `loopnz (loopne)` różni się od rozkazu `loopz (loope)` tylko tym, że procesor przechodzi do etykiety, gdy znacznik ZF ma wartość 0.

Tabela

Kodowanie rozkazów grupy `Conditional Jumps` (skoki warunkowe)

Mnemonik i operandy	Bajt 0	Bajt 1	Od bajta 2
jo przes8 jo przes16/32	01110000 00001111	przes8 10000000	 przes16/32
jno przes8 jno przes16/32	01110001 00001111	przes8 10000001	 przes16/32
jb (jnae) przes8 jb (jnae) przes16/32	01110010 00001111	przes8 10000010	 przes16/32
jnb (jae) przes8 jnb (jae) przes16/32	01110011 00001111	przes8 10000011	 przes16/32
je (jz) przes8 je (jz) przes16/32	01110100 00001111	przes8 10000100	 przes16/32
jne (jnz) przes8 jne (jnz) przes16/32	01110101 00001111	przes8 10000101	 przes16/32
jbe (jna) przes8 jbe (jna) przes16/32	01110110 00001111	przes8 10000110	 przes16/32
jnbe (ja) przes8 jnbe (ja) przes16/32	01110111 00001111	przes8 10000111	 przes16/32
js przes8 js przes16/32	01111000 00001111	przes8 10001000	 przes16/32
jns przes8 jns przes16/32	01111001 00001111	przes8 10001001	 przes16/32
jp (jpe) przes8 jp (jpe) przes16/32	01111010 00001111	przes8 10001010	 przes16/32
jnp (jpo) przes8 jnp (jpo) przes16/32	01111011 00001111	przes8 10001011	 przes16/32
jl (jnge) przes8 jl (jnge) przes16/32	01111100 00001111	przes8 10001100	 przes16/32
jnl (jge) przes8 jnl (jge) przes16/32	01111101 00001111	przes8 10001101	 przes16/32
jle (jng) przes8 jle (jng) przes16/32	01111110 00001111	przes8 10001110	 przes16/32
jnle (jg) przes8 jnle (jg) przes16/32	01111111 00001111	przes8 10001111	 przes16/32
jcxz (jecxz) przes8	11100011	przes8	
loop przes8	11100010	przes8	
loopz (loope) przes8	11100001	przes8	
loopnz (loopne) przes8	11100000	przes8	

Znaczniki wyników operacji

Programowanie na poziomie asemblera jest możliwe tylko przy dokładnym śledzeniu za wartościami znaczników rejestru EFLAGS.

W tabeli są pokazane zmiany znaczników rejestru EFLAGS po wykonaniu rozkazów.

Zastosowano następujące zaznaczenia:

„+” w przypadku ustawienia wartości według wyniku operacji,

„0” lub ‘1’ w przypadku ustawienia konkretnej wartości znacznika,

„?” w przypadku nieprzewidywalnej zmiany wartości znacznika.

Tabela.

Stan znaczników rejestru EFLAGS po wykonaniu rozkazu

Mnemonic	Znaczniki								
	OF	DF	IF	TF	SF	ZF	AF	PF	CF
Rozkazy do przesyłania danych									
in, out, lahf, lds, lea, les, lods, lodsb, lodsw, mov, movs, movsb, movsw, pop, push, pushf, stos,stosb,stosw, xchg, xlat, xlatb									
popf	+	+	+	+	+	+	+	+	+
sahf					+	+	+	+	+
Rozkazy do operacji arytmetycznych									
aaa	?				?	?	+	?	+
aas, cbw, cwd									
aad, aam	?				+	+	?	+	?
adc, add, cmp	+				+	+	+	+	+
daa, das	?				+	+	+	+	+
dec, inc	+				+	+	+	+	
div	?				?	?	+	?	+
idiv	?				?	?	?	?	?
imul	+				?	?	?	?	+
mul	+				?	?	?	?	+
neg, ssb, sub	+				+	+	+	+	+
Rozkazy do operacji bitowych									
and, test, xor	0				+	+	?	+	0
not									
or	0				+	+	?	?	0
rcl, rcr, rol, ror, sal, sar, shl, shr	+								+
Rozkazy do sterowania przebiegiem programu									
clc									0
cmc									+
cld		0							
cli			0						
cmps,cmpsb,cmpsw	+				+	+	+	+	+
int, into					0	0			
iret	+	+	+	+	+	+	+	+	+
call, ja, jnbe, jae, jnb, jb, jnae, jbe, jna, jc, jz, jcxz, je, jz, jg, jnle, jge, jnl, jl, jnge, jnc, jne, jnz, jng, jle, jno, jnp, jpo, jns, jo, jp, jpe, js, jmp, loop, loope, loopz, loopne, loopnz, ret, retf									
rep, repe, repz, repne, repnz						+			
scas,scasb,scasw	+				+	+	+	+	+
stc									1
std		1							
sti			1						
Rozkazy do sterowania procesorem									
esc, hlt, lock, wait									