Ćwiczenie

„Przesyłanie danych i zarządzanie danymi”

Tematy ćwiczenia

- przesyłanie danych,
- praca z łańcuchami znaków

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 podrozdziały "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 jako "PN", grupę, numer ćwiczenia i nazwisko studenta.

Pliki ze sprawozdaniem są przekazywane do archiwum grupy.

Wskazówki

Wywołać edytor MASM32 Editor środowiska MASM32.

Ustawić edytor na swój katalog.

W środowisku MASM32 jest możliwość zmiany kolorów tekstu i tła. Aby zmienić kolory, należy wybrać punkt menu File/Settings/Edit Colors lub nacisnąć przycisk Ctrl+Shift+C.

a) Przesyłanie przez rejestry

Zadanie

W programie zdefiniować bajtową tablicę tekstową z tekstem - funkcją `...`[1].

Wyświetlić zawartość tablicy.

Ponieważ tekst - funkcja zawiera 7 znaków plus znak '\0' (koniec wierszu), zawartość tablicy przypisać do dwóch 4-bajtowych rejestrów ECX i EAX.

Przestawić miejscami zawartość ECX i EAX.

Przypisać zawartość rejestrów ECX i EAX do tablicy buforowej.

Wyświetlić zawartość tablicy buforowej.

Uwaga. Adresy tablic ustawiać w rejestrach za pomocą instrukcji "lea", na przykład;

```
lea EBX, tablica
lea EDX, bufor
```

Opracowanie zadania

<<tekst programu>>

Testowanie

<<zrzut ekranu MS DOS>>

---

[1] `patrz tabele wariantów`

b) Przesyłanie elementów tablic

Zadanie

Napisać program, w którym wykonać działania następujące.

Zdefiniować i wyświetlić bajtową tablicę tekstową z funkcją ...[2].

Zdefiniować buforową bajtową tablicę tekstową.

Przypisywać elementy tablicy tekstowej do tablicy buforowej tak, żeby kolejność znaków odpowiadała *notacji polskiej*, na przykład: ++AB+CD zamiast A+B+C+D.

Wyświetlić zawartość tablicy buforowej.

Opracowanie zadania

<<tekst programu>>

Testowanie

<<zrzut ekranu MS DOS>>

Tabela wariantów

Np	Funkcja		Np	Funkcja		Np	Funkcja		Np	Funkcja
1	A + B + C + D		**9**	A + B * C + D		**17**	A + B + C – D		**25**	A + B * C - D
2	A - B + C + D		**10**	A - B * C + D		**18**	A - B + C - D		**26**	A - B * C - D
3	A * B + C + D		**11**	A * B * C + D		**19**	A * B + C - D		**27**	A * B * C – D
4	A / B + C + D		**12**	A / B * C + D		**20**	A / B + C - D		**28**	A / B * C - D
5	A + B - C + D		**13**	A + B / C + D		**21**	A + B - C - D		**29**	A + B / C - D
6	A - B - C + D		**14**	A - B / C + D		**22**	A - B - C - D		**30**	A - B / C - D
7	A * B - C + D		**15**	A * B / C + D		**23**	A * B - C - D		**31**	A * B / C - D
8	A / B - C + D		**16**	A / B / C + D		**24**	A / B - C - D		**32**	A / B / C - D

Wskazówki

Zamiana typu wyrażenia

Ponieważ dane różnych typów mogą mieć jednakowy rozmiar, to w sytuacjach niejednoznacznych należy typ danej określić jawnie.

Do określenia typu wyrażenia jest używany operator PTR.

Składnia operatora PTR:

```
_Typ PTR _Wyrażenie
```

Przykład, w którym operator PTR jest użyty do operacji na znakach w tablicy tekstowej:

```
.DATA
tabl DWORD „abc”,0
.CODE
    mov AL, BYTE PTR tabl[0]
    mov BYTE PTR tabl[2], AL
```

[2] z tabeli wariantów

Przesyłanie danych

Daną można przesyłać z rejestru do rejestru, z rejestru do komórki pamięci lub z komórki pamięci do rejestru. Nie jest możliwe bezpośrednie przesyłanie danej między dwoma komórkami pamięci.

Jedna instrukcja zawsze przesyła jedną daną, ale część instrukcji jest przystosowana do powtórzenia przesyłania.

Grupa Data Transfer

Wykonując rozkaz „`mov odbiorca, źródło`" grupy *Data Transfer* procesor może przesyłać daną z rejestru do rejestru, z komórki pamięci do rejestru, w tym do rejestra-akumulatora, i odwrotnie, oraz zapisuje do rejestru lub do komórki pamięci daną bezpośrednią.

Dana może być 8-, 16- lub 32 bitowa.

Nie istnieje możliwość przesyłania danych bezpośrednio między komórkami pamięci.

W przypadku przesyłania danej do rejestru segmentowego ma miejsce zakaz przerwań.

Instrukcje do przesyłania danej są połączone w grupę `Data Transfer` rozkazów procesorów `Intel` i przedstawione w tabele.

Tabela

Kodowanie rozkazów grupy `Data Transfer` (przesyłanie danych)

Zaznaczono: R – rejestr, P – pamięć, A - rejestr-akumulator, Rs – rejestr segmentowy

Instrukcja	Bajt 0	Bajt 1	Bajt 2
`mov R/P, R`	`1000100w`	`mod_reg_r/m`	
`mov R, R/P`	`1000101w`	`mod_reg_r/m`	
`mov R/P, Rs`	`10001100`	`mod_sr3_r/m`	
`mov Rs, R/P`	`10001110`	`mod_sr3_r/m`	
`mov A, P`	`1010000w`	`przes`	
`mov P, A`	`1010001w`	`przes`	
`mov R, dana`	`1011wreg`	`dana`	
`mov R/P,dana`	`1100011w`	`mod_000_r/m`	`dana`
`movsx R, R/P`	`00001111`	`1011111w`	`mod_reg_r/m`
`movzx R, R/P`	`00001111`	`1011011w`	`mod_reg_r/m`
`push R/P`	`11111111`	`mod_110_r/m`	
`push R`	`01010reg`		
`push Rs`	`000s2110`		
`push FS/GS`	`00001111`	`10_sr3_000`	
`push dana`	`011010s0`	`dana`	

pop R/P pop R pop Rs pop FS/GS	10001111 01011reg 000s2111 00001111	mod_000_r/m 10_sr3_001	
pusha pushad	01100000		
popa popad	01100001		
xchg R/P, R xchg R, R/P xchg R, A xchg A, R	1000011w 10010reg	mod_reg_r/m	
in A, Num in A, DX	1110010w 1110110w	Num	
out Num, A out DX, A	1110011w 1110111w	Num	
lea R, P	10001101	mod_reg_r/m	

Przesyłanie danej między rejestrami

Przykłady:

```
mov EAX,0
mov ECX,10
mov EBX, OFFSET zmienna
```

Zamianę miejscami zawartości rejestru i drugiego rejestru, lub rejestru i akumulatora, lub rejestru i komórki pamięci wykonuje rozkaz „`xchg`”.

W instrukcjach przesyłania danych można wskazywać rejestr segmentowy przed nazwą zmiennej. Na przykład przy kompilacji instrukcji:

```
mov EAX, ES:symb
```

asembler wykorzysta zawartość rejestru segmentowego ES.

Przesyłanie danej na stos / ze stosu

Przesyłanie danej na stos i ze stosu przy wykonaniu instrukcji grupy *Data Transfer* i grupy *Flag Control* jest równoważne przesyłaniu danej między rejestrem i komórką pamięci, ponieważ stos programowy to grupa komórek pamięci.

Oprócz przesyłania zawartości pojedynczego rejestru istnieje możliwość przesyłania grupowego, do 8 rejestrów w grupie.

Przesyłanie danych przez stos często jest używane do przesyłania danej między komórkami pamięci, na przykład:

```
.CODE
 push zm1 ; odkładanie na stos
 pop temp ;do zmiennej "temp" jest przypisana zmienna "zm1"
```

W procesie wykonania rozkazów „push” i „pop” procesor ładuje daną na stos lub zdejmuje daną ze stosu.

Wierzch stosu znajduje się pod adresem SS:ESP, a stos rośnie w stronę mniejszych adresów.

Wykonując rozkaz „push” procesor zmniejsza ESP o cztery (lub dwa w zależności od typu procesora) i zapisuje operand na wierzch stosu.

W przypadku odkładania na stos bajta ma miejsce rozszerzenie znaku.

Rozkaz „pop” powoduje odczyt danej z wierzchu stosu spod adresu SS:ESP, a następnie zwiększenie ESP o cztery (lub dwa w zależności od typu procesora). Operacja „pop” nie jest możliwa, jeżeli miejscem przeznaczenia danej jest rejestr segmentowy CS.

Rozkazy „pusha/pushad” i „popa/popad” są bardzo przydatne na początku i w końcu podprogramu, ponieważ ładują na stos i zdejmują ze stosu grupę rejestrów ogólnego przeznaczenia. Kolejność odkładania na stos zawartości rejestrów:

dla pusha: AX, CX, DX, BX, SP (przed odkładaniem), BP, SI, DI;

dla pushad: EAX, ECX, EDX, EBX, ESP (przed odkładaniem), EBP, ESI, EDI.

Oczywiście, że kolejność zdejmowania ze stosu zawartości rejestrów jest odwrotna:

dla popa: DI, SI, BP, SP, BX, DX, CX, AX;

dla popad: EDI, ESI, EBP, ESP, EBX, EDX, ECX, EAX.

Przesyłanie danej do portu / z portu

Do wprowadzenia danej z portu jest używany rozkaz „in”, a do wyprowadzenia danej do portu - rozkaz „out”.

Jeżeli numer portu znajduje się w granicach 0 – 255, to stosuje się rozkaz z bezpośrednim adresowaniem operandu.

W przypadku, gdy numer portu jest większy niż 255, stosuje się rejestr DX i adresowanie pośrednie rejestrowe.

Maksymalnie możliwy numer portu w komputerach PC jest równy 1023.

Pobieranie danej z portu i ładowanie danej do portu przy wykonaniu instrukcji in i out posiada ograniczenie: przesyłanie jest możliwe tylko między rejestrem – akumulatorem a portem.

Zmiana formatu danej w procesie przesyłania

Przesyłanie danej można połączyć ze zmianą formatu, a mianowicie z rozszerzeniem znaku (instrukcja movsx) lub z zerowaniem starszych bitów (instrukcja movzx).

Rozkaz „movsx” w procesie przesyłania danej rozszerza znak z bajta na słowo (podwójne słowo) lub ze słowa na podwójne słowo.

Podobny rozkaz „movzx” rozszerza daną zerami w stronę starszego bajta lub słowa.

Przesyłanie adresów i przesunięć segmentowych

Do operacji z adresami i przesunięciami segmentowymi służą specjalne instrukcje lea, lds, les, lfs, lgs, lss.

Wykonując rozkaz „lea” procesor przesyła do miejsca przeznaczenia przesunięcie adresu komórki pamięci.

Kodowanie rozkazów

Procesor 32-bitowy Intel ma skomplikowane reguły kodowania rozkazów, ponieważ procesor może operować 8-, 16- lub 32-bitowymi danymi, 16- lub 32-bitowymi adresami i wykonywać rozkazy 16- lub 32-bitowe.

Rozkazy 32-bitowych procesorów Intel zajmują od 1 do 12 bajtów. Średni rozmiar rozkazu 3,2 bajta. W pierwszych 1 lub 2 bajtach znajdują się kod rozkazu, informacja o typie danej, informacja o istnieniu następnych bajtów, w których są umieszczane stałe i adres względny (przesunięcie). Jednobajtowe rozkazy nie potrzebują operandów.

Rozkazy mają strukturę:

`Pierwszy_bajt_kodu_operacji [Drugi_bajt_kodu_operacji] [Byte_"mod r/m"] [Byte_"sib"] [Bajty_przesunięcia] [Bajty_danej]`

Niekonieczne elementy struktury są pokazane w nawisach kwadratowych. Pola "`mod r/m`" i "`sib`" definiują tryb adresacji i rejestry. Każde z pól `Bajty_przesunięcia` i `Bajty_danej` może być pustym lub mieć rozmiar 1, 2 lub 4 bajty.

Struktury pól rozkazów przedstawia tab. 1.

Tabela 1.

Struktury pól rozkazów 32-bitowego procesora Intel

Pole rozkazu	Struktura bajta
Pierwszy_bajt_kodu_operacji Drugi_bajt_kodu_operacji	B B B B B B B B
Pierwszy_bajt_kodu_operacji Drugi_bajt_kodu_operacji	B B B B B B B w
Pierwszy_bajt_kodu_operacji	B B B B B B s B
Pierwszy_bajt_kodu_operacji	B B B B B B s w
Pierwszy_bajt_kodu_operacji	B B B B B B d w
Pierwszy_bajt_kodu_operacji	B B B B B reg
Pierwszy_bajt_kodu_operacji	B B B B w reg
Pierwszy_bajt_kodu_operacji	B B B s2 B B B
Drugi_bajt_kodu_operacji	B B sr3 B B B
Byte_"mod r/m"	mod B B B r/m
Byte_"mod r/m"	mod reg r/m
Byte_"mod r/m"	mod sr3 r/m
Byte_"sib"	ss ind base
Byte_"eee"	1 1 eee reg
Bajty_przesunięcia	przes8
Bajty_przesunięcia	przes16_0 przes16_1
Bajty_przesunięcia	przes32_0 przes32_1 przes32_2 przes32_3
Bajty_danej	dana8
Bajty_danej	dana16_0 dana16_1
Bajty_danej	dana32_0 dana32_1 dana32_2 dana32_3
Bajty_danej	numer portu

W tej tabeli i dalej oznaczono:

`B` – bit z wartością 0 lub 1,

d – pole kierunku (d = 0 - pole „reg” jest interpretowane jako źródło, a pole „mod r/m” lub „mod ss ind base” – jako miejsce przeznaczenia, d = 1 - pole „reg” jest interpretowane jako miejsce przeznaczenia, a pole „mod r/m” lub „mod ss ind base” – jako źródło),

dana – bajty rozkazu zawierające daną,

eee – 3-bitowe pole z indeksem rejestru sterowania (Control Register), debugowania (Debug Register) lub testowania (Test Register),

mod – 2-bitowe pole z informacją o strukturze rozkazu i trybie adresacji,

przes – bajty rozkazu zawierające adres względny (przesunięcie),

r/m – 3-bitowe pole z informacją o miejsce operandu,

reg – 3-bitowe pole z indeksem rejestru ogólnego przeznaczenia,

s2 – 2-bitowe pole z indeksem rejestru segmentowego,

sr3 – 3-bitowe pole z indeksem rejestru segmentowego,

s - znacznik rozszerzenia znakowego dla danej,

w – jednobitowy znacznik rozmiaru danej (w = 0 – jeden bajt, w = 1 – „pełny” rozmiar, tj. 16 lub 32 bitów w zależności od wartości pola reg).

Wartość pola reg wskazuje na rejestr w zależności od pola „w” w sposób przedstawiony w tab. 2.

Tabela 2.

Kodowanie rejestru ogólnego przeznaczenia w zależności od pól reg i w

Pole reg	Rejestr			
	Operacja 16-bitowa		Operacja 32-bitowa	
	w=0	w=1 lub brak pola w	w=0	w=1 lub brak pola w
000	AL	AX	AL	EAX
001	CL	CX	CL	ECX
010	DL	DX	DL	EDX
011	BL	BX	BL	EBX
100	AH	SP	AH	ESP
101	CH	BP	CH	EBP
110	DH	SI	DH	ESI
111	BH	DI	BH	EDI

Pola s2 i sr3 zawierają informacji o indeksie rejestru segmentowego (tab. 3).

Tabela 3.

Kodowanie rejestrów segmentowych w polach s2 i sr3

Rejestr segmentowy											
Pole s2				Pole sr3							
00	01	10	11	000	001	010	011	100	101	110	111
ES	CS	SS	DS	ES	CS	SS	DS	FS	GS	-	-

Pola 2-bitowe `mod` i 3-bitowe `r/m` używane są do kodowania informacji o strukturze rozkazu i trybie adresacji (tab. 4a i 4b).

Tabela 4a.

Tryb adresacji w przypadku istnienia bajta „`mod r/m`" i braku bajta „`sib`"

Pole mod r/m	Adres efektywny lub rejestr							
	Adresacja 16-bitowa				Adresacja 32-bitowa			
	Operacja 16-bitowa		Operacja 32-bitowa		Operacja 16-bitowa		Operacja 32-bitowa	
	w=0	w=1	w=0	w=1	w=0	w=1	w=0	w=1
00 000	DS:[BX+SI]				DS:[EAX]			
00 001	DS:[BX+DI]				DS:[ECX]			
00 010	SS:[BX+SI]				DS:[EDX]			
00 011	SS:[BX+DI]				DS:[EBX]			
00 100	DS:[SI]				według bajta sib			
00 101	DS:[DI]				DS:przes32			
00 110	DS:przes16				DS:[ESI]			
00 111	DS:[BX]				DS:[EDI]			
01 000	DS:[BX+SI+przes8]				DS:[EAX+przes8]			
01 001	DS:[BX+DI+przes8]				DS:[ECX+przes8]			
01 010	SS:[BX+SI+przes8]				DS:[EDX+przes8]			
01 011	SS:[BX+DI+przes8]				DS:[EBX+przes8]			
01 100	DS:[SI+przes8]				według bajta sib			
01 101	DS:[DI+przes8]				SS:[EBP+przes8]			
01 110	SS:[BP+przes8]				DS:[ESI+przes8]			
01 111	DS:[BX+przes8]				DS:[EDI+przes8]			
10 000	DS:[BX+SI+przes16]				DS:[EAX+przes32]			
10 001	DS:[BX+DI+przes16]				DS:[ECX+przes32]			
10 010	SS:[BX+SI+przes16]				DS:[EDX+przes32]			
10 011	SS:[BX+DI+przes16]				DS:[EBX+przes32]			
10 100	DS:[SI+przes16]				według bajta sib			
10 101	DS:[DI+przes16]				SS:[EBP+przes32]			
10 110	SS:[BP+przes16]				DS:[ESI+przes32]			
10 111	DS:[BX+przes16]				DS:[EDI+przes32]			
11 000	AL	AX	AL	EAX	AL	AX	AL	EAX
11 001	CL	CX	CL	ECX	CL	CX	CL	ECX
11 010	DL	DX	DL	EDX	DL	DX	DL	EDX
11 011	BL	BX	BL	EBX	BL	BX	BL	EBX
11 100	AH	SP	AH	ESP	AH	SP	AH	ESP
11 101	CH	BP	CH	EBP	CH	BP	CH	EBP
11 110	DH	SI	DH	ESI	DH	SI	DH	ESI
11 111	BH	DI	BH	EDI	BH	DI	BH	EDI

Tabela 4b.

Tryb adresacji w przypadku istnienia bajtów „`mod r/m`" i „`sib`" (adresacja tylko 32-bitowa)

Pole		Adres efektywny
mod	base	
00	000	DS:[EAX+ind*ss]
00	001	DS:[ECX+ind*ss]
00	010	DS:[EDX+ind*ss]
00	011	DS:[EBX+ind*ss]
00	100	SS:[ESP+ind*ss]
00	101	DS:[przes32+ind*ss]
00	110	DS:[ESI+ind*ss]
00	111	DS:[EDI+ind*ss]
01	000	DS:[EAX+ind*ss+przes8]
01	001	DS:[ECX+ind*ss+przes8]
01	010	DS:[EDX+ind*ss+przes8]
01	011	DS:[EBX+ind*ss+przes8]
01	100	SS:[ESP+ind*ss+przes8]
01	101	DS:[przes32+ind*ss+przes8]
01	110	DS:[ESI+ind*ss+przes8]
01	111	DS:[EDI+ind*ss+przes8]
10	000	DS:[EAX+ind*ss+przes32]
10	001	DS:[ECX+ind*ss+przes32]
10	010	DS:[EDX+ind*ss+przes32]
10	011	DS:[EBX+ind*ss+przes32]
10	100	SS:[ESP+ind*ss+przes32]
10	101	DS:[przes32+ind*ss+przes32]
10	110	DS:[ESI+ind*ss+przes32]
10	111	DS:[EDI+ind*ss+przes32]

Pola ss i ind wskazują na wartość mnożnika indeksacji i na rejestr indeksowy (tab. 5).

Tabela 5.

Kodowanie mnożnika indeksacji i rejestru indeksowego w polach ss i ind

Mnożnik				Rejestr indeksowy							
Pole ss				Pole ind							
00	01	10	11	000	001	010	011	100	101	110	111
1	2	4	8	EAX	ECX	EDX	EBX	-	przes32	ESI	EDI

Pole eee jest zastosowane do kodowania rejestru sterowania (Control Register), debugowania (Debug Register) lub testowania (Test Register) (tab. 6).

Tabela 6.

Kodowanie rejestru sterowania (Control Register), debugowania (Debug Register) lub testowania (Test Register) w polu eee

Kod w polu eee	Control Register	Debug Register	Test Register
000	CR0	DR0	-
001	-	DR1	-
010	CR2	DR2	-
011	CR3	DR3	-
100	CR4	-	-
101	-	-	-
110	-	DR6	TR6
111	-	DR7	TR7

Na krok rozmieszczenia rozkazów w pamięci (2 lub 4 bajty) wskazuje wartość jednego z bitów znacznika dostępu deskryptora segmentu CS. Ten znacznik definiuje też domyślny rozmiar operandów i adresu efektywnego.

Rozkazy procesora Intel

W dokumentacji firmy Intel rozkazy 32- bitowych procesorów są podzielone na grupy:

- Data Transfer – przesyłanie danych,
- Segment Control – sterowanie segmentami,
- Flag Control – sterowanie znacznikami,
- Arithmetic – operacje arytmetyczne,
- Logic – operacje logiczne (bitowe),
- Shift/Rotate – przesuwanie bitowe,
- String Manipulation – operacje z wierszami (tablicami),
- Bit Manipulation – operacje bitowe,
- Control Transfer – przejście sterowane,
- Conditional Jumps – skoki warunkowe,
- Conditional Byte Set – warunkowe ustawienie bajtu,
- Interrupt Instructions – rozkazy przerwań,
- Processor Control – sterowanie procesorem,

- `Prefix Bytes` (bajty prefiksu),
- `Protection Control` (sterowanie ochroną),
- `High Level Language Support` – potrzymanie języka wysokiego poziomu,
- `Operating System Support` – potrzymanie systemu operacyjnego,
- `Processor Extension Instruction` (instrukcje koprocesora),
- `MMX Unit Instructions` (instrukcje jednostki MMX).

Rozpatrzmy rozkazy procesorów Intel według grup. 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.

Grupa `Data Transfer` (przesyłanie danych)

Wykonując rozkaz „`mov odbiorca, źródło`” (tab. 7) procesor może przesłać daną z rejestru do rejestru, z komórki pamięci do rejestru, w tym do akumulatora, i odwrotnie, oraz może zapisać do rejestru lub do komórki pamięci daną bezpośrednią. Dana może być 8-, 16- lub 32 bitowa. Nie istnieje możliwość przesyłania danych bezpośrednio między komórkami pamięci. W przypadku przesyłania danej do rejestru segmentowego ma miejsce zakaz przerwań.

Rozkaz „`movsx`” w procesie przesyłania danej rozszerza znak z bajta na słowo (podwójne słowo) lub ze słowa na podwójne słowo.

Podobny rozkaz „`movzx`” rozszerza daną zerami w stronę starszego bajta lub słowa.

W procesie wykonania rozkazów „`push`” i „`pop`” procesor ładuje daną na stos lub zdejmuje daną ze stosu. Wierzch stosu znajduje się pod adresem SS:ESP, a stos rośnie w stronę mniejszych adresów.

Wykonując rozkaz „`push`” procesor zmniejsza ESP o dwa lub cztery w zależności od typu procesora i zapisuje operand na wierzch stosu. W przypadku odkładania na stos bajta ma miejsce rozszerzenie znaku.

Rozkaz „`pop`” powoduje odczyt danej z wierzchu stosu z pod adresu SS:ESP, a następnie zwiększenie ESP o dwa lub cztery w zależności od typu procesora. Operacja „`pop`” nie jest możliwa, jeżeli miejscem przeznaczenia danej służy rejestr segmentowy CS.

Rozkazy „`pusha/pushad`” i „`popa/popad`” bardzo przydatne na początku i w końcu podprogramu, ponieważ ładują na stos i zdejmują ze stosu grupę rejestrów ogólnego przeznaczenia. Kolejność odkładania na stos zawartości rejestrów:

dla `pusha`: AX, CX, DX, BX, SP (przed odkładaniem), BP, SI, DI;
dla `pushad`: EAX, ECX, EDX, EBX, ESP (przed odkładaniem), EBP, ESI, EDI.

Tabela 7

Kodowanie rozkazów grupy `Data Transfer` (przesyłanie danych)

Mnemonik i operandy	Bajt 0	Bajt 1	Bajt 2

mov R/P, R
mov R, R/P
mov R/P, Rs
mov Rs, R/P
mov A, P
mov P, A
mov R, dana
mov R/P,dana
	1000100w
	1000101w
	10001100
	10001110
	1010000w
	1010001w
	1011wreg
	1100011w
		mod_reg_r/m
		mod_reg_r/m
		mod_sr3_r/m
		mod_sr3_r/m
		przes
		przes
		dana
		mod_000_r/m
			
			dana

movsx R, R/P	00001111	1011111w	mod_reg_r/m
movzx R, R/P	00001111	1011011w	mod_reg_r/m
push R/P push R push Rs push FS/GS push dana	11111111 01010reg 000s2110 00001111 011010s0	mod_110_r/m 10_sr3_000 dana	
pop R/P pop R pop Rs pop FS/GS	10001111 01011reg 000s2111 00001111	mod_000_r/m 10_sr3_001	
pusha; pushad	01100000		
popa; popad	01100001		
xchg R/P, R; xchg R, R/P xchg R, A; xchg A, R	1000011w 10010reg	mod_reg_r/m	
in A, Num in A, DX	1110010w 1110110w	Num	
out Num, A out DX, A	1110011w 1110111w	Num	
lea R, P	10001101	mod_reg_r/m	

Jasne, że kolejność zdejmowania ze stosu zawartości rejestrów jest odwrotna:

dla `popa`: DI, SI, BP, SP, BX, DX, CX, AX;

dla `popad`: EDI, ESI, EBP, ESP, EBX, EDX, ECX, EAX.

Zamianę miejscami zawartości rejestru i drugiego rejestru, lub rejestru i akumulatora, lub rejestru i komórki pamięci wykonuje rozkaz „`xchg`”.

Do wprowadzenia danej z portu służy rozkaz „`in`”, a do wyprowadzenia danej do portu - rozkaz „`out`”. Jeżeli numer portu znajduje się w granicach 0 – 255, to stosuje się rozkaz z bezpośrednim adresowaniem operandu. W przypadku, gdy numer portu jest większy niż 255, stosuje się rejestr DX i adresowanie pośrednie rejestrowe. Maksymalnie możliwy numer portu w komputerach PC jest równy 1023.

Wykonując rozkaz „`lea`” procesor przesyła do miejsca przeznaczenia przesunięcie adresu komórki pamięci.

Notacja polska

Notacja polska (*notacja Łukasiewicza*) przedstawiona Janem Łukasiewiczem w 1920 roku to sposob zapisu wyrażeń w kolejności: najpierw operator, potem operandy.

Na przykład, zamiast tradycyjnego wyrażenia (A * B + C / D) zapisujemy:

+*AB/CD, co odpowiada kolejności obliczeń w wyrażeniu ((A * B) + (C / D))

Notacja polska ma zastosowanie w kompilatorach, gdzie w procesie analizy wyrażenie jest zapisywane na stos w notacji polskiej.