ԵՐԵՎԱՆԻ ՊԵՏԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ

Ա. Հ. Առաքելյան, Շ. Ա. Բաղդասարյան,
Կ. Գ. Գալստյան, Վ. Ա. Զաքարյան, Գ. Ա. Մարտիրոսյան,
Զ. Ա. Նազարյան, Գ. Ռ. Սարդարյան

ԻՆԹԵԼ x86 ԱՍԵՄԲԼԵՐ

(իրական ռեժիմ)
ՈՒՍՈՒՄՆԱՄԵԹՈԴԱԿԱՆ ՁԵՌՆԱՐԿ

ԵՐԵՎԱՆ
ԵՊՀ ՀՐԱՏԱՐԱԿՉՈՒԹՅՈՒՆ
2016

ՀՏԴ 004 (07)
ԳՄԴ 32.81 ց7
Ի 510

Հրատարակության է երաշխավորել
ԵՊՀ ինֆորմատիկայի և կիրառական մաթեմատիկայի
ֆակուլտետի գիտական խորհուրդը

Ա. Հ. Առաքելյան, Շ. Ա. Բաղդասարյան, Կ. Գ. Գալստյան,
Վ. Ա. Զաքարյան, Գ. Ա. Մարտիրոսյան, Ջ. Ա. Նազարյան,
Գ. Ռ. Սարդարյան

Ի 510 **ԻՆԹԵԼ x86 ԱՍԵՄԲԼԵՐ** (իրական ռեժիմ): Ուսումնամեթոդական ձեռնարկ: -Եր., ԵՊՀ հրատ., 2016, 268 էջ:

Ձեռնարկն անդրադառնում է Ինթել x86 ասեմբլերի (իրական ռեժիմ) հիմնական հրամաններին, տվյալների ներկայացմանը, ասեմբլերով ծրագրավորման հիմնարար գաղափարներին: Պարունակում է մեծ թվով օրինակներ: Ինքնուրույն աշխատանքի նպատակով յուրաքանչյուր բաժին համալրված է խնդիրներով և վարժություններով:

Նախատեսված է «ԷՀՄ ճարտարապետություն և ասեմբլեր լեզու» առարկայի ուսումնասիրության համար:

ՀՏԴ 004 (07)
ԳՄԴ 32.81 ց7

ISBN 978-5-8084-2148-6

© ԵՊՀ հրատ., 2016
© Հեղ. խումբ., 2016

ԲՈՎԱՆԴԱԿՈՒԹՅՈՒՆ

ՆԵՐԱԾՈՒԹՅՈՒՆ

Ձեռնարկում ներկայացված է ԵՊՀ ինֆորմատիկայի և կիրառական մաթեմատիկայի ֆակուլտետի «ԷՀՄ ճարտարապետություն և ասեմբլեր լեզու-1» դասընթացը:

Դասընթացի նպատակն է ներկայացնել ԷՀՄ-ի կառուցվածքը, տվյալների ներկայացումը, հրամանների համակարգը, ընդհատումների համակարգը Ինթել x86 (Intel x86) պրոցեսորների համար, ինչպես նաև ասեմբլեր ցածր մակարդակի լեզուն:

Ներկայացված են ասեմբլեր ծրագրերի օրինակներ Ինթել x86 պրոցեսորների համար DOS միջավայրում: Ձեռնարկը համալրված է տարբեր թեմաներին վերաբերող խնդիրներով, վարժություններով:

ԷՀՄ ՃԱՐՏԱՐԱՊԵՏՈՒԹՅՈՒՆ

«ԷՀՄ ճարտարապետություն» (Computer Architecture) տերմինը առաջին անգամ կիրառվել է IBM ֆիրմայի աշխատակիցների կողմից 1959 թվականին:

Ներկայումս տարբեր էլեկտրոնային բառարաններում այն մեկնաբանվում է տարբեր կերպ:

Այսպիսով՝ ԷՀՄ ճարտարապետություն տերմինը չունի մեկ սահմանում:

Ներկայացնենք մի քանի սահմանումներ:

Սահմանում 1. ԷՀՄ ճարտարապետությունը սահմանվում է որպես.

- ԷՀՄ-ի կառուցվածքային սխեման,
- ԷՀՄ-ի կառուցվածքային սխեմայի տարրերին դիմելու եղանակները,
- ԷՀՄ-ի միջերեսի (ինտերֆեյս) կազմակերպումը և կարգաթիվը (լայնություն),
- ռեգիստրների հավաքածուն և նրանց դիմելու եղանակը,
- հիշողության կազմակերպումը և հասցեավորման եղանակները,
- ԷՀՄ-ի տվյալների ֆորմատները և ներկայացման եղանակները,
- ԷՀՄ-ի մեքենայական հրամանների հավաքածուն,
- մեքենայական հրամանների ֆորմատները,
- ընդհատումները:

Սահմանում 2. ԷՀՄ ճարտարապետություն ասելով՝ կհասկանանք.

1. մեքենայի ֆունկցիոնալ բլոկներն իրենց կապերով,
2. տվյալների ներկայացումը մեքենայում (ֆորմատները),
3. լեզուն (հրամանների համակարգը),
4. ընդհատումների համակարգը:

ԷՀՄ-ի ֆունկցիոնալ բլոկներն են.

- պրոցեսորը կամ կենտրոնական պրոցեսորը (CPU/Central Processing Unit),
- հիմնական հիշասարքը կամ ֆիզիկական հիշողությունը (Physical Memory/Main Memory/*RAM*),
- ներածող-արտածող սարքերը (Input/Output Devices),
- արտաքին հիշասարքերը (External Memory):

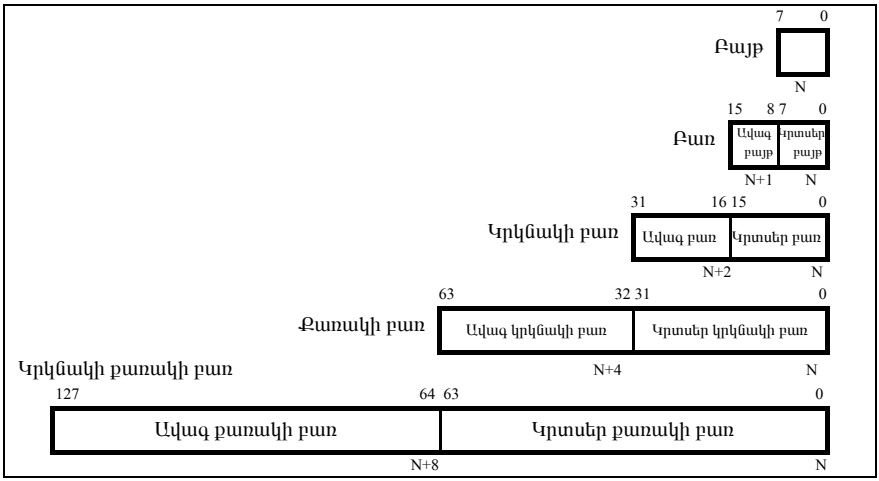
Ֆունկցիոնալ բլոկները միմյանց հետ կարող են կապվել 2 եղանակով՝

- մեկ ընդհանուր կապուղով (bus, шина),
- շատ կապուղիներով:

Սահմանում 2.-ի 2-4 կետերը կղիտարկենք Ինթել (Intel) պրոցեսորների ընտանիքի օրինակով:

ԻՆԹԵԼ x86 ՊՐՈՑԵՍՈՐԻ ՏՎՅԱԼՆԵՐԻ ՏԻՊԵՐԸ

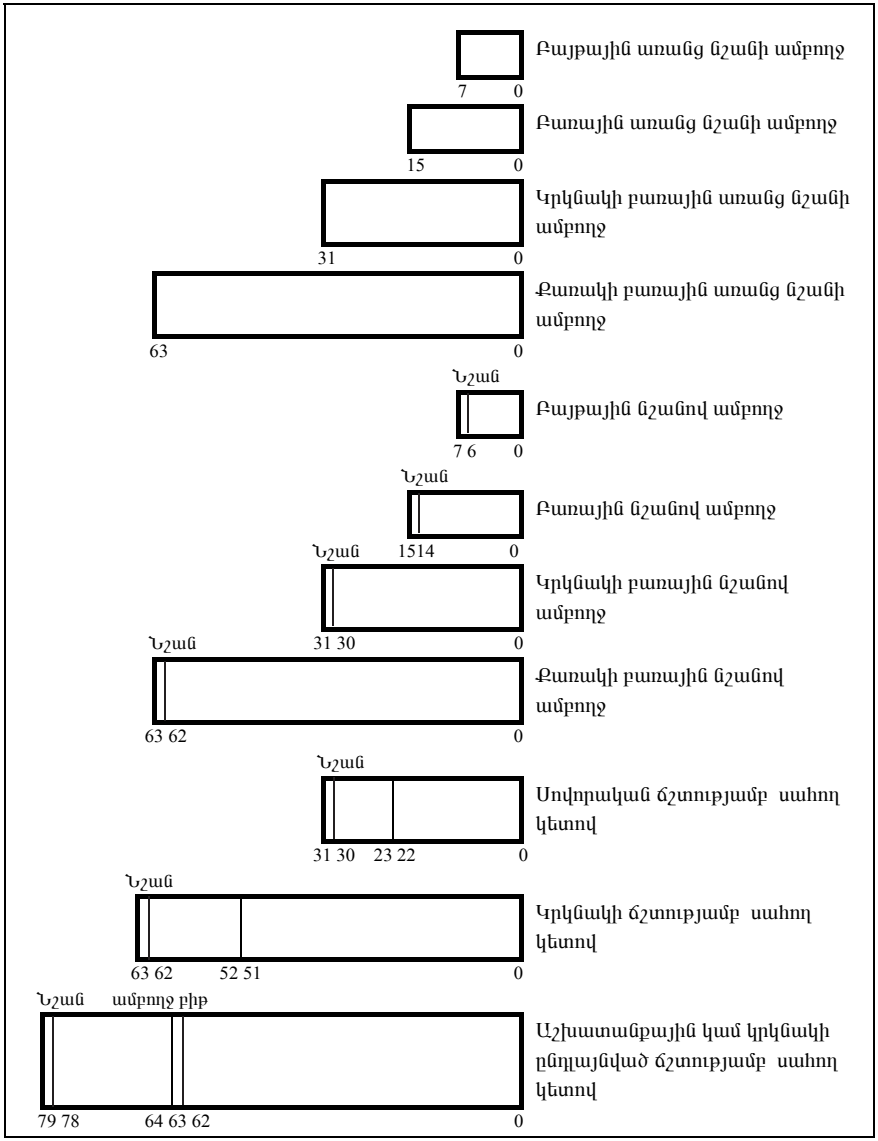
IA-32 ճարտարապետության հիմնական տվյալների տիպերն են բայթը /Byte (8 բիթ), բառը /Word (16 բիթ), կրկնակի բառը /Doubleword (32 բիթ), քառակի բառը /Quadword (64 բիթ) և կրկնակի քառակի բառը /Double quadword (128 բիթ):



Նկար 1. Հիմնական տվյալների տիպեր

Թվային տվյալների տիպեր
(Numeric Data Types)

Չնայած բայթը, բառը, կրկնակի/քառակի բառը IA-32 ճարտարապետության հիմնական տիպերն են, սակայն որոշ հրամաններ պահանջում են տվյալների տիպերի լրացուցիչ մեկնաբանություն և թույլ են տալիս գործողություններ կատարել թվային տիպերի հետ (նշանով և առանց նշանի թվեր, սահող կետով թվեր):



Նկար 2. Թվային տվյալների տիպեր

Ամբողջ թվեր (Integers)

IA-32 սահմանում է երկու տիպի ամբողջ թվեր՝ նշանով և առանց նշանի: Առանց նշանի ամբողջ թվերը սովորական երկուական թվեր են՝ 0-ից մինչև առավելագույն դրական թիվը, որը կարող է ներկայացվել օպերանդում: Նշանով ամբողջ թվերը երկուական թվերի լրացուցիչ կոդն են և կարող են օգտագործվել և՛ դրական, և՛ բացասական ամբողջ թվերի ներկայացման համար:

Որոշ հրամաններ (ADD, SUB հրամաններ) գործողություններ են կատարում և՛ առանց նշանի, և՛ նշանով ամբողջ թվերի հետ, իսկ այլ հրամաններ (IMUL, MUL, IDIV, DIV) գործողություններ են կատարում միայն մի տիպի թվերի հետ:

Սահող կետով տվյալների տիպեր (Floating-Point Data Types)

Իրական թվերը ներկայացվում են IEEE-754 ստանդարտով (տե՛ս հավելվածը) :

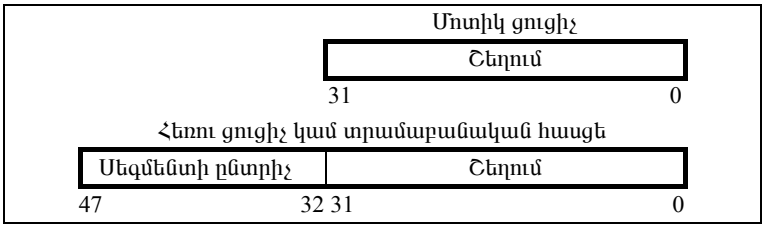
Տվյալի տիպը (Data Type)	Երկարություն (Length)	Ճշտություն (Precision Bits)	Մոտավոր նորմալիզացված միջակայք (Approximate Normalized Range)	
			Երկուական (Binary)	Տասական (Decimal)
Սովորական ճշտություն (Single Precision)	32	24	$2^{-126} - 2^{127}$	1.18×10^{-38} - 3.40×10^{38}
Կրկնակի ճշտություն (Double Precision)	64	53	$2^{-1022} - 2^{1023}$	2.23×10^{-308} - 1.79×10^{308}
Կրկնակի ընդլայնված ճշտություն (Double Extended Precision)	80	64	$2^{-16382} - 2^{16383}$	3.37×10^{-4932} – 1.18×10^{4932}

Նկար 3. Սահող կետով տվյալների տիպերի երկարությունը, ճշտությունը և միջակայքը

Ցուցչային տիպ (Pointer Data Type)

Ցուցիչները հիշողության հասցեներ են: IA-32 ճարտարապետությունը սահմանում է երկու տիպի ցուցիչ՝ մոտիկ (near) և հեռու (far):

Մոտիկ ցուցիչը սեգմենտի ներսում 32-բիթանոց (16-բիթանոց հասցեավորման դեպքում՝ 16-բիթանոց) շեղումն է (offset): Այն անվանում են նաև արդյունավետ կամ էֆեկտիվ հասցե (effective address): Այս ցուցիչն օգտագործվում է հիշողության հարթ և սեգմենտային մոդելներում հիշողությանը դիմելիս, երբ սեգմենտի արժեքը որոշվում է ոչ բացահայտ: Հեռու ցուցիչը տրամաբանական հասցե է՝ կազմված սեգմենտի 16-բիթանոց ընտրիչից (selector) և 32-բիթանոց (16-բիթանոց հասցեավորման դեպքում՝ 16-բիթանոց) շեղումից: Այն օգտագործվում է հիշողության սեգմենտային մոդելում, երբ սեգմենտի արժեքը տրվում է բացահայտ:



Նկար 4. Ցուցչային տիպեր

Բիթային դաշտ (Bit Field Data Type)

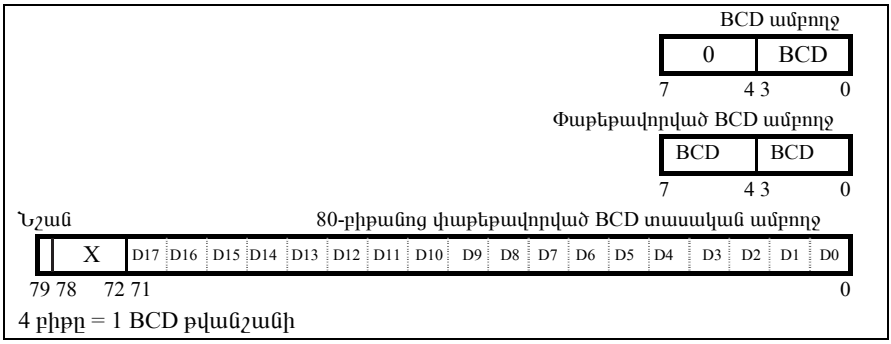
Բիթային դաշտը բիթերի հաջորդականություն է: Այն կարող է գտնվել հիշողության բայթի ցանկացած դիրքից սկսած և ունենալ առավելագույնը 32 բիթ երկարություն, ամենաշատը՝ 4 բայթում:

Տողեր (String Data Type)

Տողերը բիթերի, բայթերի, բառերի կամ կրկնակի բառերի հաջորդականություն են: Բիթային տողը կարող է սկսվել ցանկացած բայթի ցանկացած դիրքից և պարունակել մինչև 2^{32} - 1 բիթ:

Բայթային տողը կարող է կազմված լինել բայթերից, բառերից կամ կրկնակի բառերից և ունենալ մինչև 4ԳԲ երկարություն:

BCD փաթեթավորված և ոչ փաթեթավորված թվեր



Նկար 5. Երկուական կոդավորված տասական թվեր /BCD (Binary coded Decimal)

Երկուական կոդավորված տասական թվեր (BCD թվեր)

Լինում են երկու տեսակ՝ փաթեթավորված և ոչ փաթեթավորված: Ոչ փաթեթավորված տասական թիվն առանց նշանի տասական թվի (0-9 միջակայք) երկուական 4-բիթանոց ներկայացումն է մեկ բայթում: Փաթեթավորված տասական թիվը մեկ բայթում 2 հատ տասական թվի (0-9 միջակայք) երկուական 4-բիթանոց ներկայացումն է:

Տվյալների ներկայացումը ասեմբլեր ծրագրում

Տվյալներն ասեմբլեր ծրագրում ներկայացվում են տվյալների հրամանագրերի (դիրեկտիվներ/directives) միջոցով, որոնց տեսքն է.

Անուն D* <սկզբնական արժեքների ցուցակ>

Որտեղ՝

- անվան դաշտը պարտադիր չէ,
- *-ը կարող է ընդունել B (byte), W (word), D (doubleword), Q (quadword), T(ten byte), F (far) արժեքներ՝ համապատասխանաբար բայթ, բառ (2 բայթ), կրկնակի բառ (4 բայթ), քառակի բառ (8 բայթ), 10 բայթ և 6 բայթ սահմանելու համար,
- **<սկզբնական արժեքների ցուցակ>**-ը ներկայացնում է սկզբնական արժեքներ՝ անջատված ստորակետերով:
- Սկզբնական արժեքը ներկայացնում է թվային (ամբողջ կամ իրական) կամ սիմվոլային արժեք: Թվային ամբողջ արժեքը կարող է տրվել 10-ական, 16-ական (h/H վերջավորությամբ), 8-ական (q/Q վերջավորությամբ), 2-ական (b/B վերջավորությամբ) հիմքով: Սկզբնական արժեքը պետք է պատկանի սահմանված տիպի թույլատրելի արժեքների բազմությանը, այլապես թարգմանիչն այն կդիտարկի որպես սխալ: Սիմվոլային տողը պարփակվում է չակերտների՝ “ ” կամ ապաթարցների՝‘ ’ մեջ:
- Կարելի է սկզբնական արժեքի վոխարեն օգտագործել՝
  - ✓ ? նշանը, որը նշանակում է, որ սկզբնական արժեքը որոշված չէ,
  - ✓ <քանակ> DUP (<սկզբնական արժեքների ցուցակ>) գործողությունը, որը նշանակում է <սկզբնական արժեքների ցուցակ>-ը՝ <քանակ> անգամ կրկնած:

Օրինակ՝

```
A DB ?                  ; սահմանված է A բայթային փոփոխա-
                        ; կանը, որի սկզբնական արժեքը որոշ-
                        ; ված չէ,
B DB 5                  ; սահմանված է B բայթային փոփոխա-
                        ; կանը՝ 5 սկզբնական արժեքով,
C DW 0F123h             ; սահմանված է C բառային փոփոխա-
                        ; կան՝ F123h 16-ական արժեքով,
D DB “Hello”            ; սահմանված է D սիմվոլային զանգ-
                        ; ված՝ “Hello” արժեքով,
E DD 13.4               ; սահմանված է E կրկնակի բառային
                        ; փոփոխականը՝ 13.4 իրական թվային
                        ; արժեքով,
F DW 45, 101B, ?, 10H   ; սահմանված է F բառային զանգված՝
                        ; բաղկացած 4 տարրից՝ 45, 5 (101B),
                        ; որոշված չէ (?), 16 (10H) արժեքներով,
G DD 10 DUP (0)         ; սահմանված է 10 կրկնակի բառերից
                        ; կազմված G զանգված՝ սկզբնարժեքա-
                        ; վորված զրոյով:
```

Խնդիրներ և վարժություններ

1. Յուրաքանչյուր հրամանագրի համար գրել համարժեք այլ ներկայացում, որտեղ փոփոխականի սկզբնարժեքը տրված կլինի 16-ական տեսքով.

ա) A DB 17 բ) B DB 15 գ) V DB 255

դ) G DB 150 ե) D DB −17 զ) E DB −1

է) J DB −106 ը) Z DB −128 թ) I DW 17

ժ) K DW −17 ի) L DW −1 լ) M DW 256

2. Յուրաքանչյուր հրամանագրի համար գրել 2 համարժեք այլ ներկայացում, որտեղ փոփոխականի արժեքը տրված կլինի 10-ական տեսքով, մի դեպքում՝ առանց նշանի, իսկ մյուս դեպքում՝ նշանով.

ա) A DB 0Ah բ) B DB 0A5h գ) V DB 7Fh
դ) G DB 80h ե) D DB 101b զ) E DW 0FFFEh
է) J DW 7Fh ը) Z DW 80h

3. Տրված է հետևյալ նկարագրությունը.

A DW 1020h
B DD 10203040h

16-ական տեսքով նշել հետևյալ հասցեներով բայթերի արժեքները՝ A, A+1, B, B+1, B+2 և B+3:

4. Յուրաքանչյուր հրամանագրի համար գրել 2 համարժեք այլ ներկայացում, որտեղ փոփոխականի արժեքը տրված կլինի երկուական կողավորված տասական թվի (BCD) տեսքով, մի դեպքում՝ փաթեթավորված, իսկ մյուս դեպքում՝ ոչ փաթեթավորված.

ա) A DB 12 բ) B DB 15h գ) C DB 01011011b
դ) D DW 0A4h ե) E DW 130 զ) F DW 7892
է) G DD 725 ը) H DD 123456 թ) I DD 1A5h

5. Նկարագրել 10 բայթից բաղկացած A զանգված՝ հետևյալ սկզբնական արժեքներով.
   ա) տարրերը չունեն սկզբնական արժեք,
   բ) տարրերն ունեն 0 սկզբնական արժեք,
   գ) առաջին 5 տարրերը 1-եր են, մնացած 5 տարրերը՝ 0-ներ,
   դ) տարրերն առաջին 10 բնական զույգ թվերն են (2, 4, 6....),
   ե) առաջին 3 տարրերը պարզ թվեր են՝ 7, 11, 13, հաջորդ 4-ը սկզբնական արժեք չունեն, մնացած տարրերը 0-ներ են:

6. Տրված է հետևյալ նկարագրությունը.

   A DB 12h, 25h, 60, 48

   B DB 5, -7, 10, 32

   C DB 120, 85, 79, 46h

   Նշել A, B և C զանգվածների տարրերի արժեքները 16-ական համակարգով հետևյալ գործողություններից հետո.

   A[2] = 10, A[6] = 25, A[9] = 4, B[1] = 12, B[7] = 44h,
   C[2] = 10, C[-2] = 20, C[-5] = 33h:

7. Տրված է հետևյալ նկարագրությունը.

   A DB 250, 12, 64h, 89

   B DW 3552h, -27, -79, 3215

   C DD 256, 486915h, 791241h, -732

   Նշել A, B և C զանգվածների տարրերի արժեքները 16-ական համակարգով հետևյալ գործողություններից հետո.

   A[4] = 10, A[1] = 25, A[12] = 29h, B[-1] = 65h, B[6] = 47,
   B[8] = 5, C[1] = 0Ah, C[4] = 57h, C[-5] = 42h:

ԻՆԹԵԼ x86 ՊՐՈՑԵՍՈՐԻ ՌԵԳԻՍՏՐՆԵՐԻ ՀԱՎԱՔԱԾՈՒ

Ռեգիստրը պրոցեսորի փոքրածավալ հիշողություն է, որն ապահովում է տվյալներին արագ դիմելու հնարավորություն:

Ինթել x86 միկրոպրոցեսորի (կամ պարզապես պրոցեսորի[1]) ծրագրային մոդելը (ծրագրավորողին հասանելի) պարունակում է 32 ռեգիստր: Այն կարելի է բաժանել 2 խմբի.

1. 16 հատ օգտագործողի ռեգիստրներ,
2. 16 հատ համակարգային ռեգիստրներ:

Դիտարկենք օգտագործողի ռեգիստրները, որոնց մեծ մասն ունեն ֆունկցիոնալ նշանակություն: 8086-ում կային 14 հատ 2-բայթանոց ռեգիստրներ: 80186-ում ավելացան ևս 2 ռեգիստր՝ FS և GS: 80386-ում ռեգիստրներից 10-ը դարձել են 4-բայթանոց, և դրանց անվանումը սկսվում է 'E' տառով:

Օգտագործողի ռեգիստրներն իրենց հերթին կարելի է բաժանել 3 ենթախմբի.

1. 8 հատ 32-բիթանոց ընդհանուր օգտագործման ռեգիստրներ (General-Purpose registers), որոնք օգտագործվում են տվյալներ և հասցեներ պահելու համար: Դրանք են՝ EAX/AX/AH/AL, ECX/CX/CH/CL, EDX/DX/DH/DL, EBX/BX/BH/BL, ESP/SP, EBP/BP, ESI/SI, EDI/DI:

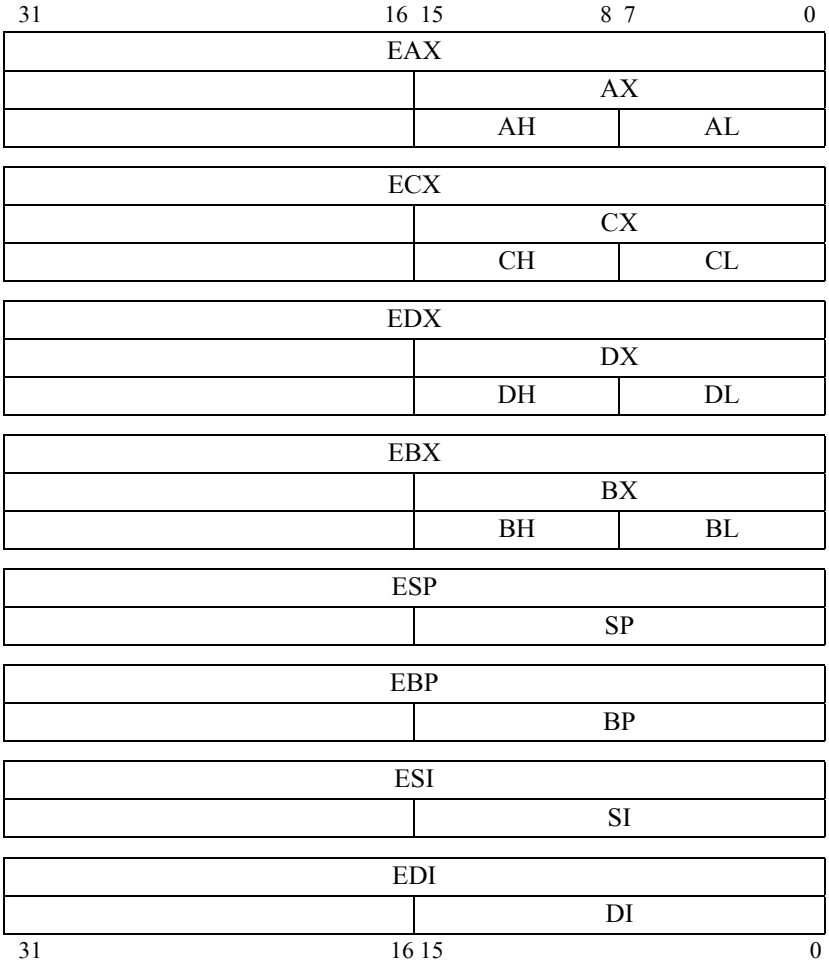
2. 6 հատ 16-բիթանոց սեգմենտային ռեգիստրներ՝ CS, DS, SS, ES, FS, GS:

3. 32-բիթանոց դրոշների ռեգիստրը՝ EFLAGS (կրտսեր 16 բիթը՝ FLAGS) և հրամանի ցուցիչը՝ EIP (կրտսեր 16 բիթը՝ IP):

Այժմ ավելի մանրամասն դիտարկենք նշված ռեգիստրները:

[1] Մեկ միկրոսխեմայի կամ մի քանի միկրոսխեմաների խմբի տեսքով իրականացված պրոցեսորը կոչվում է միկրոպրոցեսոր: Ներկայումս միկրոպրոցեսոր չհանդիսացող պրոցեսորների շատ քիչ քանակի պատճառով միկրոպրոցեսոր և պրոցեսոր տերմինները գործնականում համարժեք են:

Ընդհանուր օգտագործման ռեգիստրներ



Նկար 1. Ընդհանուր օգտագործման ռեգիստրներ

Ընդհանուր օգտագործման 32-բիթանոց ռեգիստրները հնարավորություն են տալիս դիմելու իրենց կրտսեր մասերին (օրինակ՝ EAX-ի կրտսեր մասերն են AX-ը, AH-ը, AL-ը): Ավագ 16 բիթերին ուղղակիորեն հնարավոր չէ դիմել:

Քանի որ այդ ռեգիստրները ֆիզիկապես գտնվում են ԹՏՍ-ի (թվաբանական-տրամաբանական սարքի) մեջ, դրանց նաև անվանում են ԹՏՍ-ի ռեգիստրներ: Բացի ընդհանուր օգտագործումից՝ առանձին ռեգիստրներ հանդես են գալիս նաև որոշակի ֆունկցիոնալ նշանակությամբ:

- EAX/AX/AH/AL (Accumulator register) – կուտակիչ: Օգտագործվում է միջանկյալ տվյալներ պահելու համար: Որոշ հրամաններում օգտագործվում է ոչ բացահայտ:
- EBX/BX/BH/BL (Base register) – բազային ռեգիստր: Օգտագործվում է հիշողության մեջ գտնվող օբյեկտի բազային հասցեն պահելու համար: Որոշ հրամաններում օգտագործվում է ոչ բացահայտ:
- ECX/CX/CH/CL (Count register) – հաշվիչի ռեգիստր: Օգտագործվում է հրամանի կամ հրամանների խմբի կրկնության համար: Հաճախ օգտագործվում է ոչ բացահայտ (օրինակ՝ LOOP հրամանում):
- EDX/DX/DH/DL (Data register) – տվյալների ռեգիստր: Հիշվում են միջանկյալ տվյալներ: Որոշ հրամաններում օգտագործվում է ոչ բացահայտ:
- ESI/SI (Source Index register) – աղբյուրի ինդեքսային ռեգիստր: Ոչ բացահայտ օգտագործվում է տողային հրամաններում՝ պարունակելով ընթացիկ 32, 16, 8-բիթանոց տարրի հասցեն:
- EDI/DI (Destination Index register) – ընդունիչի ինդեքսային ռեգիստր: Նման է ESI/SI-ին:
- ESP/SP (Stack Pointer register) – պահունակի ցուցիչի ռեգիստր: Պարունակում է պահունակ սեգմենտի մեջ պահունակի գագաթի ցուցիչը (հասցեն):
- EBP/BP (Base Pointer register) – պահունակի կադրի բազային ցուցիչի ռեգիստր: Օգտագործվում է պահունակի ներսի տվյալներին դիմելու համար:

Որոշ հրամանների համար ամրագրված ռեգիստների օգտագործումը հնարավորություն է տալիս ավելի կոմպակտ ներկայացնելու դրանց մեքենայական կողերը:

Սեգմենտային ռեգիստրներ

Սեգմենտային ռեգիստրներն են CS, SS, DS, ES, FS, GS-ը: Այս ռեգիստրների ներկայությունը պայմանավորված է Ինթել միկրոպրոցեսորի օպերատիվ հիշողության կազմակերպման և օգտագործման հանգամանքով: Միկրոպրոցեսորն ապարատային միջոցներով ապահովում է ծրագրի կառուցվածքային կազմակերպումը՝ բաղկացած սեգմենտներից: Հիշողության նման կազմակերպումն անվանում են հիշողության կազմակերպման սեգմենտային մոդել:

Որևէ կոնկրետ պահին ծրագրին հասանելի սեգմենտները նշելու համար օգտագործվում են սեգմենտային ռեգիստրներ: Միկրոպրոցեսորն օգտագործում է հետևյալ երեք տիպի սեգմենտները:

1. **Կոդ (հրամանների) սեգմենտ,** որը պարունակում է ծրագրի հրամանները: Այս սեգմենտին դիմելու համար օգտագործվում է CS (code segment) ռեգիստրը:

2. **Տվյալների սեգմենտ**, որը պարունակում է ծրագրի կողմից մշակվող տվյալները: Այս սեգմենտին դիմելու համար օգտագործվում է DS (data segment) ռեգիստրը:

Ենթադրվում է, որ ոչ բացահայտ մեքենայական հրամանների մշակած տվյալները գտնվում են այն տվյալների սեգմենտում, որի հասցեն գտնվում է DS (Data Segment) սեգմենտային ռեգիստրում: Եթե ծրագրին մեկ տվյալների սեգմենտը չի բավարարում, այն կարող է օգտագործել ևս երեք լրացուցիչ տվյալների սեգմենտներ: Լրացուցիչ տվյալների սեգմենտի հասցեն բացահայտ կերպով (բացի տողային հրամաններում օգտագործվող ES սեգմենտային ռեգիստրից) նշվում է հրամանի մեջ սեգմենտային ռեգիստրի վերաբեռնման

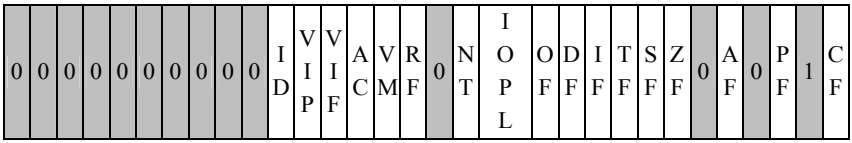
նախդիրի միջոցով: Լրացուցիչ տվյալների սեգմենտների հասցեները պահվում են ES, GS, FS սեգմենտային ռեգիստրներում:

3. **Պահունակ սեգմենտ**, որը հիշողության տիրույթ է և նույնպես օգտագործվում է տվյալների պահպանման համար: Պահունակի հետ աշխատանքը միկրոպրոցեսորն իրականացնում է հետևյալ սկզբունքով. այս տիրույթում գրանցված վերջին տարրն ընտրվում է առաջինը (LIFO – "Last Input, First Output" սկզբունք): Այս սեգմենտին դիմելու համար օգտագործվում է SS (stack segment) սեգմենտային ռեգիստրը:

Ղեկավարման և վիճակի դրոշներ

Այժմ մանրամասն դիտարկենք EFLAGS (80386+) դրոշների 32-բիթանոց ռեգիստրը, որի առանձին բիթերն ունեն որոշակի ֆունկցիոնալ նշանակություն, և կոչվում են դրոշներ: Այս ռեգիստրի կրտսեր մասը՝ 16-բիթանոց FLAGS ռեգիստրը, i8086-ի համար է:

EFLAGS դրոշների ռեգիստրն ունի հետևյալ տեսքը.



Նկար 2. EFLAGS դրոշների ռեգիստր

Չի օգտագործվում

ID (ID Flag), VIP (Virtual Interrupt Pending), VIF (Virtual Interrupt Flag), AC (Alignment Check), VM (Virtual-8086 Mode), RF (Resume Flag), NT (Nested Task), IOPL (I/O Privilege Level) դրոշները կոչվում են **համակարգային դրոշներ** և ներմուծվել են 80386-ից սկսած:

Իրական ռեժիմում օգտագործվում են միայն CF, PF, AF, ZF, SZ, IF, TF, DF և OF դրոշները՝ i8086-ի դրոշները: Ավելի մանրամասն դիտարկենք այդ դրոշները:

CF (բիթ 0)՝ **Carry Flag – փոխանցման դրոշ**: Եթե գումարման ժամանակ ավագ բիթից 1 է փոխանցվում դուրս, կամ հանման ժամանակ դրսից ավագ բիթը պարտք է վերցնում, ապա CF-ը ընդունում է 1 արժեք, հակառակ դեպքում՝ 0: Այս դրոշն ունի գերհագեցման իմաստ առանց նշանի թվերի համար:

PF (բիթ 2)՝ **Parity Flag – զույգության դրոշ**: Եթե գործողության արդյունքի կրտսեր բայթի մեջ 1-երի քանակը զույգ է, ապա PF = 1, հակառակ դեպքում՝ 0:

AF (բիթ 4)՝ **Auxiliary Carry Flag – լրացուցիչ փոխանցման դրոշ**: Եթե գումարման ժամանակ 3-ից 4-րդ բիթ փոխանցում է կատարվում կամ հանման ժամանակ 4-ից 3-րդ բիթ պարտք է վերցվում, ապա AF = 1, հակառակ դեպքում՝ 0: Այս դրոշն օգտագործվում է 10-ական թվաբանության հրամաններում:

ZF (բիթ 6)՝ **Zero Flag – զրոյի դրոշ**: Եթե գործողության արդյունքը 0 է, ապա ZF = 1, հակառակ դեպքում՝ 0:

SF (բիթ 7)՝ **Sign Flag – նշանի դրոշ**: Այս բիթն իրենից ներկայացնում է գործողության արդյունքի ավագ բիթի արժեքը: Եթե գործողության արդյունքը բացասական է՝ SF = 1, հակառակ դեպքում՝ SF = 0:

TF (բիթ 8)՝ **Trace Flag – կանգառի դրոշ**: Եթե TF = 1, ապա պրոցեսորն աշխատում է քայլային ռեժիմում, այսինքն՝ ծրագրի յուրաքանչյուր հրամանի կատարումից հետո պրոցեսորի աշխատանքը կանգնում է, և իրականացվում է int 1 հրամանը, հակառակ դեպքում հերթով կատարվում են ծրագրի բոլոր հրամանները մինչև ծրագրի ավարտը: Այս դրոշն օգտագործվում է ճշգրտման ծրագրերի աշխատանքի ժամանակ (օր.՝ debug.exe):

IF (բիթ 9)՝ **Interrupt Enable Flag – ընդհատումը թույլատրող դրոշ**: Եթե IF = 0, ապա պրոցեսորն անտեսում է արտաքին սարքերից եկած ընդհատումները:

DF (բիթ 10)՝ **Direction Flag - ուղղության դրոշ**: Այս դրոշն օգտագործվում է տողային հրամանների հետ աշխատելիս: Եթե DF = 0, ապա տողային հրամաններն իրականացվում են հասցեների աճման կարգով, իսկ եթե DF = 1՝ հասցեների նվազման կարգով:

OF (բիթ 11)՝ **Overflow flag - գերհագեցման դրոշ:** Եթե նշանով թվերի գործողության (գումարում/հանում) արդյունքը չի պատկանում արդյունքի համար նախատեսված թույլատրելի արժեքների բազմությանը, ապա OF = 1, հակառակ դեպքում՝OF = 0:

DF դրոշը կոչվում է **ղեկավարող դրոշ**, IF և TF դրոշները կոչվում են **համակարգային դրոշներ**, իսկ CF, PF, AF, ZF, SF, OF դրոշները՝ **վիճակի** կամ **թվաբանական դրոշներ**:

Օրինակ

դիտարկենք 1-բայթանոց 255 և 1 թվերի գումարման արդյունքում թվաբանական դրոշների ընդունած արժեքները:

255 = 11111111b
1 = 00000001b
1111 1111
\+
0000 0001
1 0000 0000

CF = 1, քանի որ տեղի է ունեցել փոխանցում:

AF = 1**,** քանի որ տեղի է ունեցել 3-ից 4-րդ բիթ փոխանցում**:**

SF = 0, քանի որ նշանի բիթը 0 է:

ZF = 1, քանի որ գործողության արդյունքը զրո է:

PF = 1, քանի որ գումարման արդյունքում ստացված մեկերի քանակը զույգ է (0):

OF = 0, քանի որ գերհագեցում տեղի չի ունեցել (որպես նշանով թվեր՝ -1-ին գումարվել է 1 և արդյունքում ստացվել է 0):

ՀԻՇՈՂՈՒԹՅԱՆ ԿԱԶՄԱԿԵՐՊՈՒՄԸ ԻՐԱԿԱՆ ՌԵԺԻՄՈՒՄ

IA-32 պրոցեսորներն աշխատում են իրական (real), պաշտպանված (protected) և համակարգը ղեկավարող ռեժիմով (System Management Mode-SMM):

Իրական ռեժիմը համակարգի գործարկման ռեժիմն է, որտեղ պրոցեսորն աշխատում է որպես արագագործ 8086:

Իրական ռեժիմը բնութագրվում է 1ՄԲ հասցեային տարածությամբ, հիշողության ցանկացած տիրույթի, մուտք/ելքի հասցեների և արտաքին սարքերի հետ ուղղակի աշխատելու անսահմանափակ հնարավորություններով:

Իրական ռեժիմը չի ապահովում հիշողության պաշտպանություն, բազմախնդրայնություն կամ կողի արտոնության մակարդակներ:

Ամենաներքին մակարդակով հիշողությունը կարելի է դիտարկել որպես բիթերի (bit/binary digit) զանգված: Սովորաբար հիշողությունը կազմակերպում են որպես բջիջների (կամ բայթերի) հաջորդականություն: Բջիջը մեկ կամ մեկից ավելի կարգավորված բիթերի համախումբ է, որն ունի հասցե: 8 հաջորդական բիթերին անվանում են բայթ: Երբ բջջի պարունակությունը 8 բիթ է՝ 1 բայթ, ապա ասում են, որ օգտագործվում է բայթային հասցեավորում: Պատմականորեն բայթի չափը կախված է եղել ապարատուրայից, և գոյություն չի ունեցել ստանդարտ, որը կպարտադրեր բայթի չափը: Դե ֆակտո 8 բիթը դարձավ ստանդարտ, քանի որ այն երկուսի աստիճան է:

Հասցեների փոփոխման միջակայքը կախված է հասցեի կապուղու լայնությունից: i486-ի և Pentium-ի համար այն գտնվում է 0-ից 2^{32}-1 միջակայքում (4ԳԲ): Հիշողության ղեկավարումը կատարվում է ապարատային միջոցներով, որը նշանակում է, որ ծրագիրը չի կարող ձևափոխել ֆիզիկական հասցեն հասցեի կապուղու միջոցով:

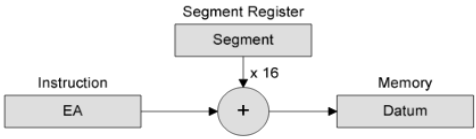
Դիտարկենք հիշողության կազմակերպումն իրական ռեժիմում:

Ինթել պրոցեսորներն իրական ռեժիմում օգտագործում են հիշողության կազմակերպման սեգմենտային եղանակը: Սեգմենտային հասցեավորման դեպքում հիշողությունը բաժանվում է տիրույթների (իրական ռեժիմում՝ առավելագույն չափը 64ԿԲ)՝ սեգմենտների, և յուրաքանչյուր բայթի հասցե ձևավորվում է երկու բաղադրիչներով՝ հիմք:շեղում (տրամաբանական հասցե), որտեղ հիմքը (base) ցույց է տալիս սեգմենտի սկզբի հասցեն, իսկ շեղումը (offset)՝ սեգմենտի սկզբի նկատմամբ տվյալ բայթի շեղումը:

Իրական ռեժիմում հիմքի և շեղման արժեքները 16-բիթանոց են (պաշտպանված ռեժիմում սեգմենտի սկզբի հասցեն որոշվում է ընտրիչի (selector) միջոցով, իսկ շեղումը 32-բիթանի է): 16-բիթանոց ծրագրում (իրական ռեժիմ) հիմքի արժեքը տրվում է սեգմենտային ռեգիստրով, իսկ շեղումը (արդյունավետ հասցե-effective address) կարող է տրվել ուղղակիորեն կամ անուղղակի արդյունավետ հասցեի բաղադրիչների միջոցով: Հիմքի արժեքը կարող է ծրագրում բացահայտ տրված չլինել:

Իրական ռեժիմում պրոցեսորը ծրագրում հիմք:շեղում տեսքով տրված տրամաբանական հասցեն ձևափոխում է 20-բիթանոց ֆիզիկական հասցեի հետևյալ եղանակով.

$$\textit{Ֆիզիկական\_հասցե}_{20} = \textit{հիմք} * 16 + \textit{շեղում}$$



Նկար 1. Տրամաբանական հասցեից ֆիզիկական հասցեի ստացումը իրական ռեժիմում

Իրական ռեժիմում ֆիզիկական հասցեն 20-բիթանոց է, հասցեային տարածությունը հասնում է մինչև 2^{20} բայթ կամ 1,048,576 բայթ (1ՄԲ), հասցեային տարածությունը տրվում է 00000h-ից 0FFFFFh միջակայքում:

Օրինակ՝

7522H:F139H տրամաբանական հասցեին իրական ռեժիմում կհամապատասխանի հետևյալ ֆիզիկական հասցեն.

$$75220H + F139H = 84359H$$

Իրական ռեժիմում մեծածավալ տվյալների հետ աշխատելը հանգեցնում է խնդիրների, քանի որ շեղումը չի կարող գերազանցել 16 բիթը, և սեգմենտի չափը սահմանափակվում է 64ԿԲ-ով:

ՀԱՍՑԵԱՎՈՐՄԱՆ ԵՂԱՆԱԿՆԵՐԸ

IA-32 օպերանդները կարող են գտնվել.

- ռեգիստրում,
- հիշողության մեջ,
- անմիջապես հրամանում (անմիջական օպերանդ),
- մուտքի/ելքի կայանում:

Գործողության արդյունքը կարող է գրանցվել.

- ռեգիստրում,
- հիշողության մեջ,
- մուտք/ելքի կայանում:

Հասցեավորման եղանակ (Addressing mode) ասելով՝ կհասկանանք օպերանդի տրման եղանակը:

IA-32 կան հասցեավորման հետևյալ եղանակները:

Ռեգիստրային հասցեավորում

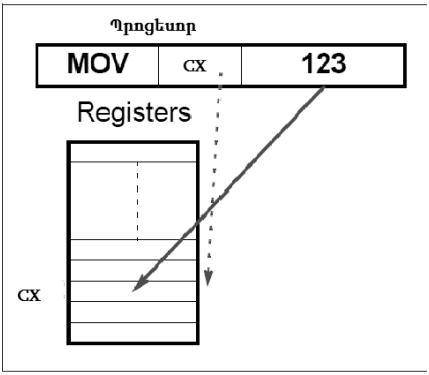
Այս հասցեավորման դեպքում օպերանդները ռեգիստրներ են:

```
MOV AX, BX          ; ռեգիստրային հասցեավորում
```

Անմիջական հասցեավորում

Անմիջական հասցեավորում օգտագործվում է երկրորդ օպերանդի համար: Այն ներկայացնում է անմիջական արժեք, որը գրանցվում է մեքենայական հրամանում:

```
MOV CX, 123
```



Եթե օպերանդը գտնվում է հիշողության մեջ, ապա այն կարող է տրվել **ուղղակի կամ ոչ ուղղակի**: Արդյունավետ հասցեի (Effective Address/EA) կամ շեղման (Offset) ձևավորմանը մասնակցում են հետևյալ բաղադրիչները.

- բազա (base),
- ինդեքս (Index),
- մասշտաբ (Scale),
- տեղաշարժ (Displacement):

Հիշեցնենք, որ հիշողության սեգմենտային կազմակերպման ժամանակ հասցեն տրվում է սեգմենտ/ընտրիչ (selector) և շեղում (կամ արդյունավետ հասցե) զույգի միջոցով:

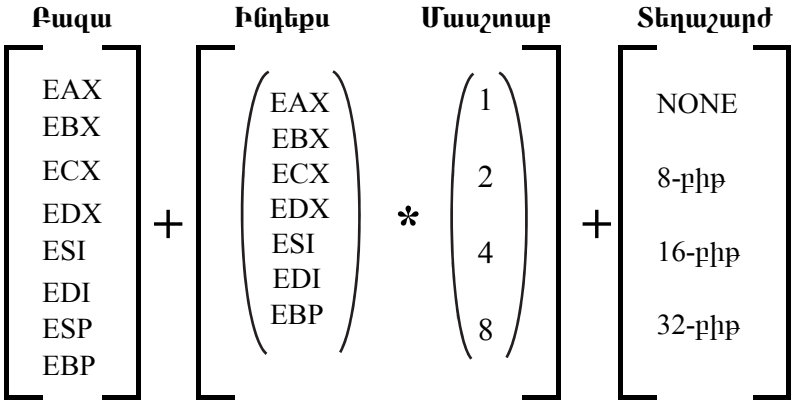
16-բիթանոց հասցեավորման դեպքում արդյունավետ հասցեն = բազա + ինդեքս + տեղաշարժ.

$$\underset{\text{Բազա}}{\begin{bmatrix} BX \\ BP \end{bmatrix}} + \underset{\text{Ինդեքս}}{\begin{bmatrix} SI \\ DI \end{bmatrix}} + \underset{\text{Տեղաշարժ}}{\begin{bmatrix} None \\ 8\,բիթ \\ 16\,բիթ \end{bmatrix}}$$

Նկար 1. Շեղման (կամ արդյունավետ հասցեի) հաշվարկ 16-բիթանոց հասցեավորման դեպքում

Այստեղ որպես բազա կարող են հանդես գալ միայն BX կամ BP ռեգիստրները, իսկ որպես ինդեքս՝ SI կամ DI-ն:

32-բիթանոց հասցեավորման դեպքում որպես բազա կարող են հանդես գալ EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP ռեգիստրները, իսկ որպես ինդեքս՝ EAX, EBX, ECX, EDX, ESI, EDI, EBP ռեգիստրները:

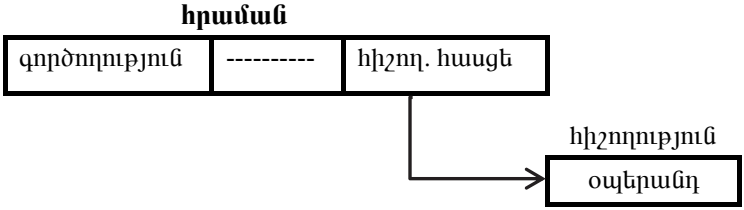


Նկար 2. Շեղման (կամ արդյունավետ հասցեի) հաշվարկ 32-բիթանոց հասցեավորման դեպքում

Հիմնականում լռությամբ սեգմենտային ռեգիստր է համարվում DS-ը, բայց այն դեպքում, երբ որպես բազա օգտագործվում են BP/EBP կամ ESP ռեգիստրները, որպես սեգմենտային ռեգիստր լռությամբ վերցվում է SS-ը:

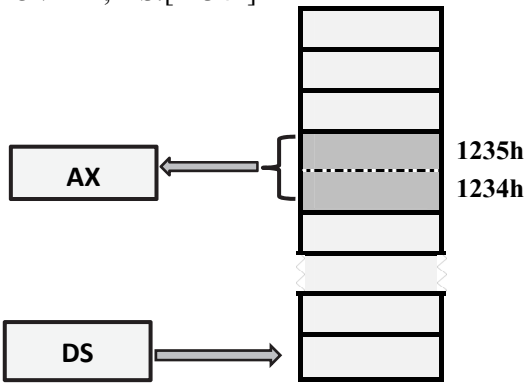
Ուղղակի հասցեավորում

Արդյունավետ հասցեի ձևավորմանը մասնակցում է միայն տեղաշարժը (displacement), որը գրանցվում է մեքենայական հրամանում:



Օրինակ

MOV AX, DS:[1234h]

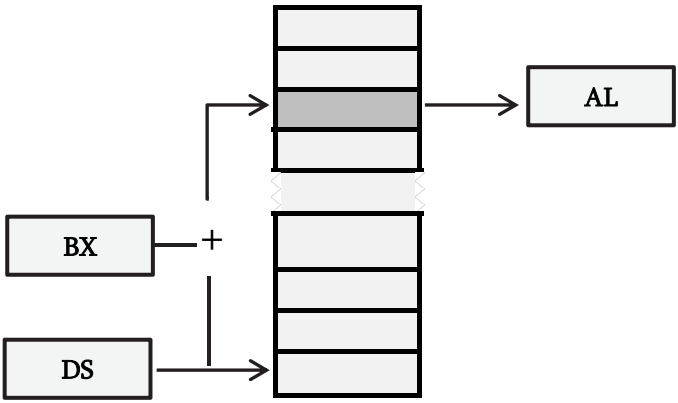


Անուղղակի ռեգիստրային հասցեավորում

Այս հասցեավորման դեպքում արդյունավետ հասցեի ձևավորմանը մասնակցում է միայն բազան կամ ինդեքսը:

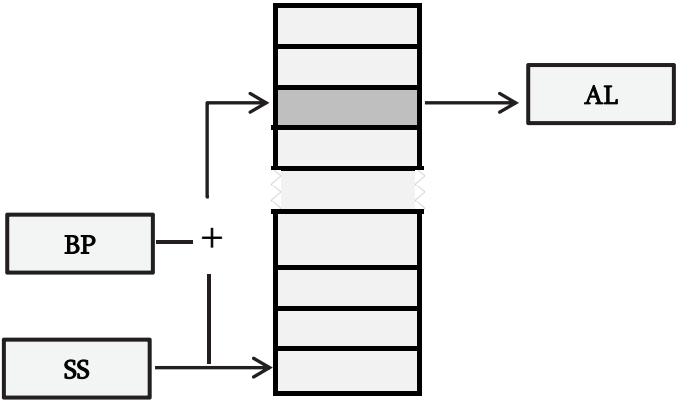
Օրինակ

MOV AL, [BX]



Օրինակ

MOV AL, [BP]



Օրինակ

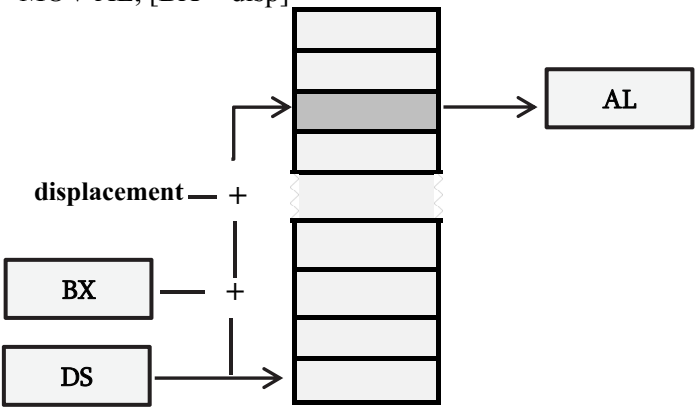
```
MOV   EAX, [ECX]     ; EAX = DWORD PTR DS:[ECX]
MOV   AX,   [SI]     ; AX  = WORD PTR DS:[SI]
```

«Բազա + տեղաշարժ» անուղղակի հասցեավորում

Այս հասցեավորման դեպքում արդյունավետ հասցեի ձևավորմանը մասնակցում են բազան կամ ինդեքսը և տեղաշարժը (displacement):

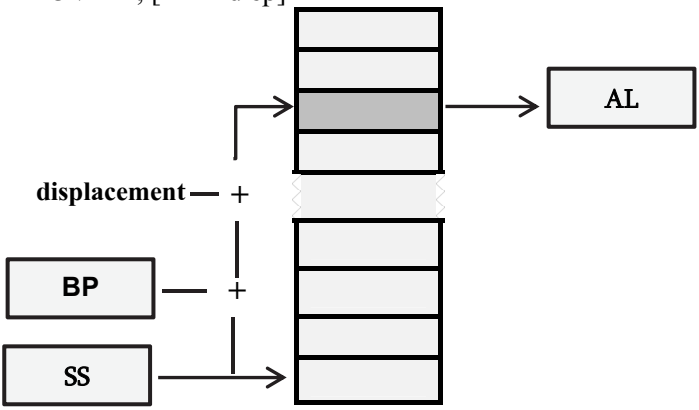
Օրինակ

MOV AL, [BX + disp]



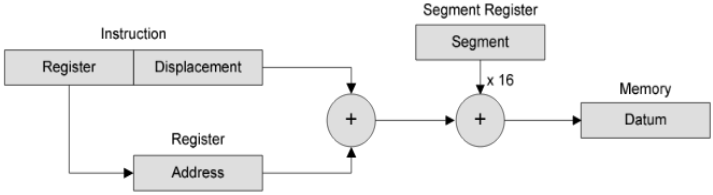
Օրինակ

MOV AL, [BP + disp]



Օրինակ

MOV AL, [BX] + 4
MOV EAX, [EDX + 100]



Նկար 3. 16-բիթանոց հասցեավորման դեպքում «Բազա + տեղաշարժ» արդյունավետ հասցեով ֆիզիկական հասցեի չհավորումը

Այս հասցեավորումն օգտագործվում է.

- զանգվածի հետ աշխատելիս, որտեղ տեղաշարժը ցույց է տալիս զանգվածի սկիզբը, իսկ բազայի միջոցով որպես ինդեքս ընտրվում է զանգվածի տարրը (օգտագործվում է, երբ տարրի չափը 2, 4, 8 չէ),
- գրառման դաշտին դիմելու համար, որտեղ բազան ցույց է տալիս գրառման սկիզբը, իսկ տեղաշարժը՝ դաշտի սկիզբը:

«(Ինդեքս * մասշտաբ) + տեղաշարժ» անուղղակի հասցեավորում

Այս հասցեավորման դեպքում արդյունավետ հասցեից մասնակցում են կա՛մ ինդեքսային ռեգիստրը, կա՛մ ինդեքսային ռեգիստրն ու տեղաշարժը:

Մասշտաբն օգտագործվում է 386-երից սկսած:

Օրինակ

```
MOV EAX, A[EBX * 4]       ; որտեղ A  DD 6, 89,…
```

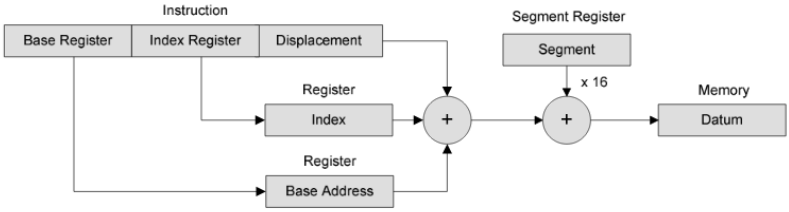
Այս հասցեավորման եղանակն առաջարկում է արդյունավետ եղանակ՝ ստատիկ զանգվածի տարրերի հետ աշխատելու համար, երբ տարրի չափը 2, 4 կամ 8 բայթ է: Տեղաշարժը տեղայնացնում է զանգվածի սկիզբը, իսկ ինդեքսային ռեգիստրը ցույց է տալիս զանգվածի տարրի ինդեքսը:

«Բազա + ինդեքս + տեղաշարժ» անուղղակի հասցեավորում

Օրինակ

```
MOV     DX, [BX + DI]
MOV     DX, [DI][BX]
MOV     DX, [BX][DI]
MOV     DX, A[BX][SI]
```

Առաջին 3 օրինակներում տեղաշարժը բացակայում է:



Նկար 4. 16-բիթանոց հասցեավորման դեպքում «Բազա + ինդեքս + տեղաշարժ» արդյունավետ հասցեով ֆիզիկական հասցեի ձևավորումը

Երկու ռեգիստրների միաժամանակյա օգտագործումը հնարավորություն է տալիս աշխատելու կա՛մ երկչափ զանգվածի (տեղաշարժը զանգվածի սկզբի հասցեն է), կա՛մ գրառումների զանգվածի (տեղաշարժը գրառման դաշտի շեղումն է) հետ:

«Բազա + ինդեքս * մասշտաբ + տեղաշարժ» (բազաինդեքսային հասցեավորում մասշտաբով) անուղղակի հասցեավորում

Օրինակ

```
MOV   AL, 20[EBX * 2 + ECX]       ; գրությաև
MOV   AL, [20 + EBX * 2 + ECX]    ; երեք ձևերը
MOV   AL, [EBX * 2 + ECX + 20]    ; համարժեք են:
```

Հասցեավորման բոլոր բաղադրիչների օգտագործումն արդյունավետ է երկչափ զանգվածի տարրերին դիմելու համար, երբ տարրի չափը 2, 4 կամ 8 բայթ է:

ԱՍԵՄԲԼԵՐ ԼԵԶՈՒ

Ասեմբլեր լեզուն ցածր մակարդակի լեզու է ԷՀՄ միկրոպրոցեսորների, միկրովերահսկիչ սարքերի և ծրագրավորվող այլ սարքերի համար: Այն մեքենայական հրամանի սիմվոլիկ ներկայացումն է և հատկորոշվում է ԷՀՄ ճարտարապետությամբ:

Ասեմբլեր անվանում են նաև ասեմբլերի թարգմանչին:

Բանավեճը՝ ասեմբլերի կիրառության օգտակարության վերաբերյալ, համեմատած բարձր մակարդակի լեզուների հետ, ներկայումս ևս շարունակվում է: Ասեմբլեր լեզուն նախընտրում են օգտագործել հետևյալ իրավիճակներում.

- Ծրագիրն աշխատում է անմիջապես սարքերի հետ (օրինակ՝ դրայվերները, ընդհատումը մշակող ծրագրերը):
- Անհրաժեշտ է օգտագործել պրոցեսորի հրամաններ, որոնք հասանելի չեն բարձր մակարդակի լեզուների թարգմանիչներին (compiler): Օրինակ՝ պտույտի հրամանները, որոնք օգտագործվում են կողավորման ծրագրերում:
- Օպտիմիզացումն անհրաժեշտություն է (օրինակ՝ խաղերի ծրագրերում):
- Գոյություն չունի բարձր մակարդակի լեզու նոր կամ հատուկ պրոցեսորների համար:
- Անհրաժեշտ է փոփոխել գոյություն ունեցող երկուական կոդը, երբ նրա սկզբնաղբյուր բարձր մակարդակի լեզվով գրված ծրագիրը չկա:
- Անհրաժեշտ է գրել կոմպիլյատոր (compiler), որը գեներացնելու է ասեմբլեր կոդ (նման աշխատանքի համար պահանջվում է ասեմբլերի լավ իմացություն):

x86 ասեմբլեր լեզուն ունի երկու հիմնական շարահյուսություն՝ Ինթել շարահյուսություն և AT&T շարահյուսություն: Ինթել շարահյուսությունը օգտագործվում է MS-DOS և Windows ՕՀ համար, իսկ AT&T շարահյուսությունը՝ Unix ՕՀ համար: Արտաքին տեսքով այս երկու շարահյուսությունները տարբերվում են միմյանցից (օրինակ՝

ըստ Ինթել շարահյուսության գրված ‘MOV AX,1’ հրամանը համարժեք է ըստ AT&T շարահյուսության գրված ‘MOVL $1, %EAX’): Մենք կծանոթանանք Ինթել շարահյուսությանը:

Ասեմբլեր լեզուն մեծատառ-փոքրատառերի նկատմամբ ոչ զգայուն (case-insensitive) լեզու է:

Ասեմբլեր ծրագրի ընդհանուր կառուցվածքը

Այս բաժնում կդիտարկենք IA-32 ճարտարապետության ասեմբլեր լեզվով գրված ծրագրի ընդհանուր կառուցվածքը: Կուսումնասիրենք նաև այդ ծրագրերի թարգմանությունը, կապերի խմբագրումը և իրականացումը:

Ասեմբլեր ծրագիրը կազմված է նախադասություններից (statement), ընդ որում՝ յուրաքանչյուր տողում գրվում է միայն մեկ նախադասություն: Կառուցվածքային առումով ասեմբլեր ծրագիրը սեգմենտների համախմբություն է՝ կազմված նախադասություններից: Յուրաքանչյուր սեգմենտ ունի հետևյալ երեք տիպերից մեկը՝ կոդ (հրամաններ), տվյալներ կամ պահունակ: Ասեմբլեր լեզվի նախադասությունները լինում են չորս տեսակի՝ հրամաններ, հրամանագրեր, մակրոհրամաններ և մեկնաբանություններ:

- **Հրամաններ** – ասեմբլեր լեզվի հրամանը մեքենայական հրամանի սիմվոլիկ ներկայացումն է: Ծրագրի թարգմանության ընթացքում ասեմբլերի հրամանները փոխարինվում են համապատասխան մեքենայական հրամաններով:
- **Հրամանագրեր** – հրամանագրերը հրահանգներ են թարգմանչի, խմբագրիչի և բեռնիչի համար: Դրանք չեն թարգմանվում մեքենայական հրամանների:
- **Մակրոհրամաններ** – մակրոհրամանները որոշակի ձևով սահմանված նախադասություններ են, որոնք թարգմանության առաջին փուլում փոխարինվում են ուրիշ նախադասություններով:

- **Մեկնաբանություններ** – մեկնաբանությունները կարող են պարունակել կամայական սիմվոլներ: Դրանք անտեսվում են թարգմանչի կողմից: Մեկնաբանությունը նաև կարելի է դիտարկել որպես նախադասության մաս:

Ասեմբլեր հրամանի կառուցվածքը հետևյալն է՝

[նշիչ:] հրամանի_հուշանուն [օպերանդ(ներ)] [; մեկնաբանություն]

Քառակուսի փակագծեր պարունակող դաշտերը պարտադիր չեն:

- **Նշիչը** անուն է՝ բաղկացած տառերից, թվանշաններից և հատուկ սիմվոլներից՝ ? . @ _ $ % (կոմպիլյատորից կախված): Անունը չի կարող սկսվել թվանշանով (եթե օգտագործված է կետ, այն չպետք է լինի առաջին սիմվոլը): Նշիչն օգտագործվում է ծրագրի հիշողության հասցեին սիմվոլիկ անուն տալու համար, որպեսզի հետագայում հնարավոր լինի ըստ անվան դիմում կատարել կողի նշված հատվածին:
- **Հրամանի_հուշանունը** նկարագրում է գործողությունը և ներկայացնում է գործողության կողի սիմվոլիկ անվանումը (օրինակ՝ ADD, MOV):
- **Օպերանդները** գործողությանը մասնակցող անդամներն են: Ասեմբլեր հրամանը կարող է ունենալ 0-3 օպերանդ: 2 օպերանդ ունեցող հրամանների համար սահմանված են որոշ սահմանափակումներ: Դրանք են.
  - օպերանդները միաժամանակ չեն կարող լինել հիշողության հասցե (բացի տողային հրամաններից),
  - առաջին օպերանդը չի կարող լինել անմիջական արժեք (բացի OUT հրամանից),
  - օպերանդները պետք է ունենան միևնույն չափը:
- **Մեկնաբանությունը** կամայական սիմվոլների համախմբություն է, որն անտեսվում է ասեմբլեր թարգմանչի կողմից: Մեկնաբանությունը սկսվում է ‘;’ սիմվոլով (մակրոներում կարող է լինել ‘;;') և շարունակվում է մինչև տողի վերջը կամ

սկսվում է COMMENT <սիմվոլ> գրառումով և շարունակվում է մինչև որևէ տողում այդ <սիմվոլ>-ին հանդիպելը: Ծրագրավորման լավ ոճը ենթադրում է մեկնաբանությունների առկայություն ծրագրում:

```
STEK  SEGMENT para stack ‘stack’
        DB      256  dup(?)
STEK  ENDS
DATA SEGMENT para public ‘data’
        CHAR DB 'A'
DATA ENDS
COD    SEGMENT para public ‘code’
MAIN  PROC FAR
ASSUME CS:COD, DS:DATA, SS:STEK
        PUSH  DS
        XOR    AX, AX
        PUSH  AX
 ; ds-ի սկզբնարժեքավորում
 ; տվյալների սեգմենտով
        MOV  AX, DATA
        MOV  DS, AX
; dl-ում գտնվող սիմվոլի(‘A’)
; արտածում էկրանին
        MOV   AH, 2
        MOV   DL, CHAR
        INT     21H
; վերադարձ DOS
        RET
MAIN ENDP
COD   ENDS
END   MAIN ; ծրագրի ավարտ / մուտքի կետ
```

```
; սեգմենտների նկարագրում
պարզեցված
հրամանագրերով

MODEL small
.STACK 256
.DATA
        CHAR DB 'A'
.CODE
MAIN:
; ds-ի սկզբնարժեքավորում տվյալների
; սեգմենտով
        MOV AX, @data
        MOV DS, AX
 ; dl-ում գտնվող սիմվոլի (‘A’)
 ; արտածում էկրանին
        MOV AH, 2
        MOV DL, CHAR
        INT    21H
 ; վերադարձ DOS
        MOV AH, 4CH
        INT    21H
END MAIN ; ծրագրի ավարտ /
                ;մուտքի կետ
```

Նկար 1. Ասեմբլեր ծրագրի օրինակ

Նշված ծրագիրն էկրանին է արտածում CHAR փոփոխականում գտնվող սիմվոլը: Ներկայացված է նաև ծրագրի սեգմենտների նկարագրման պարզեցված հրամանագրերով տարբերակը:

Ծրագիրը կազմված է պահունակի (stek սեգմենտ կամ .STACK տողը), տվյալների (data սեգմենտ կամ .DATA և նրան հաջորդող տողը) և կոդ (cod սեգմենտ կամ .CODE) սեգմենտներից:

Օպերանդը կարող է տրվել արտահայտության տեսքով:

Ասեմբլեր **արտահայտությունը** օպերանդների և գործողությունների համախմբություն է: Գործողությունները լինում են թվաբանական, տրամաբանական, համեմատության և այլն: Օպերանդը կա՛մ հաստատուն է, կա՛մ սահմանված անուն է, կա՛մ՝ արդեն սահմանված արտահայտություն: Արտահայտության արժեքը հաշվարկվում է թարգմանության ժամանակ:

Օրինակ՝ MOV AL, 90 MOD 7

MOV AX, 1234h AND 4321h

Գործողություն	**Շարահյուսություն**	**Նկարագրություն**
+	+ <արտ>	
-	- <արտ>	Ունար մինուս
+	<արտ> + <արտ>	Գումարում
-	<արտ> - <արտ>	Հանում
*	<արտ> * <արտ>	Բազմապատկում
/	<արտ> / <արտ>	Ամբողջ բաժանում
MOD	<արտ> MOD <արտ>	Մնացորդ

Նկար 2. Թվաբանական գործողությունների աղյուսակ

Գործողություն	**Շարահյուսություն**	**Նկարագրություն**
SHR	<արտ> SHR <հաշվիչ>	Բիթային աջ տեղաշարժ հաշվիչի չափով
SHL	<արտ> SHR <հաշվիչ>	Բիթային ձախ տեղաշարժ հաշվիչի չափով
NOT	NOT <արտ>	Տրամաբանական ժխտում
AND	<արտ> AND <արտ>	Տրամաբանական and
OR	<արտ> OR <արտ>	Տրամաբանական or
XOR	<արտ> XOR <արտ>	Տրամաբանական xor

Նկար 3. Տրամաբանական գործողությունների աղյուսակ

Գործողություն	Շարահյուսություն	Նկարագրություն
EQ	<արտ> EQ <արտ>	Ճիշտ է (0FFh), եթե հավասար են (equal), սխալ՝ հակ. դեպքում
NE	<արտ> NE <արտ>	Ճիշտ է (0FFh), եթե հավասար չեն (not equal), սխալ՝ հակ. դեպքում
LT	<արտ> LT <արտ>	Ճիշտ է (0FFh), եթե փոքր է (less), սխալ՝ հակ. դեպքում
LE	<արտ> LE <արտ>	Ճիշտ է (0FFh), եթե փոքր է կամ հավասար (less or equal), սխալ՝ հակ. դեպքում
GT	<արտ> GT <արտ>	Ճիշտ է (0FFh), եթե մեծ է (greater), սխալ՝ հակ. դեպքում
GE	<արտ> GE <արտ>	Ճիշտ է (0FFh), եթե մեծ է կամ հավասար (greater or equal), սխալ՝ հակ. դեպքում

Նկար 4. Համեմատության գործողությունների աղյուսակ

Գործողություն	Շարահյուսություն	Նկարագրություն
SEG	SEG հասցե	Վերադարձնում է հասցեի սեգմենտ բաղադրիչը
OFFSET	OFFSET հասցե	Վերադարձնում է հասցեի շեղում բաղադրիչը
THIS	THIS	Ստեղծում է օպերանդ, որի հասցեն հաշվիչի ընթացիկ արժեքն է
PTR	<տիպ> PTR	Ստեղծում է արգումենտ, որի հասցեն տրված տիպի տրված արտահայտության արժեքն է

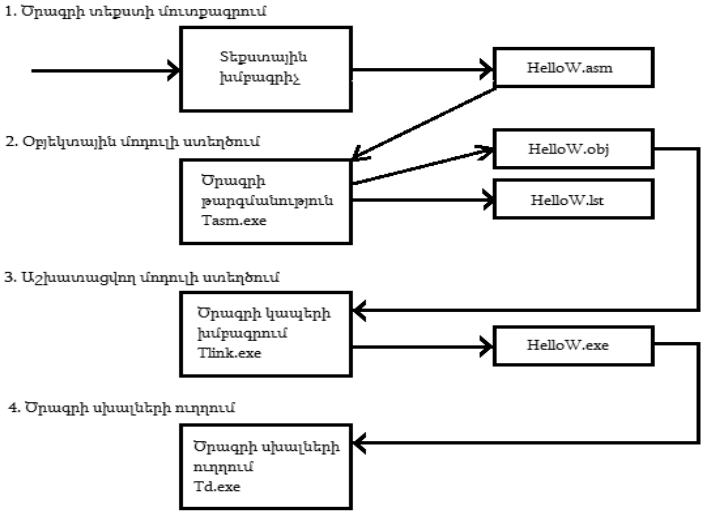
Նկար 5. Հասցեավորման գործողությունների աղյուսակ

Ասեմբլեր ծրագրի թարգմանությունը

Այս բաժնում կուսումնասիրենք ասեմբլեր լեզվով գրված ծրագրի ստեղծման պրոցեսը: Կնկարագրենք ինչպես ասեմբլեր լեզվով գրված ծրագիրը թարգմանել մեքենայական հրամանների, աշխատեցնել այն և ուղղել հայտնաբերված սխալները:

Մենք կդիտարկենք ասեմբլեր ծրագրի ստեղծումը MS-DOS օպերացիոն համակարգում: Ներկայիս օպերացիոն համակարգերում հնարավորություն կա MS-DOS օպերացիոն համակարգը աշխատեցնելու էմուլյատորների միջոցով: Մենք կօգտագործենք DOSBox էմուլյատորը: Ասեմբլեր ծրագրի թարգմանության, կապերի ստեղծման և սխալների ուղղման համար կօգտագործենք TASM (Turbo Assembler) փաթեթը:

Այժմ, օգտագործելով DOSBox էմուլյատորը և TASM փաթեթի որոշ ծրագրեր, գրենք ասեմբլեր ծրագիր և այն աշխատեցնենք: Ծրագրի ստեղծման պրոցեսը բաժանվում է չորս քայլի (Նկար 1):



Նկար 1. Ասեմբլեր ծրագրի ստեղծման պրոցեսի սխեման

Որպես օրինակ՝ դիտարկենք ծրագիր, որը էկրան է դուրս բերում “Hello World!” տողը:

Որևէ տեքստային խմբագրիչով (օրինակ notepad.exe) գրենք ասեմբլեր ծրագրի տեքստ, որը կիրառություն ստեղծելուց հետո աշխատեցնելիս արտածում է “Hello World!” տողը:

Նշված ծրագրի տեքստը կլինի հետևյալը.

```
.MODEL SMALL
.STACK      100h

.DATA
    message    DB     'Hello World!', 13, 10, '$'

.CODE
  START:
        MOV   AX, @DATA              ; տվյալների սեգմենտի հասցեն
                                     ; հիշվում է DS սեգմենտային
                                     ; ռեգիստրում
        MOV   DS, AX
        MOV   DX, OFFSET message     ; message տողի շեղումը հիշվում
                                     ; է DX ռեգիստրում
        MOV   AH, 09h                ; message տողը էկրան է
                                     ; արտածվում
        INT   21h
        MOV   AX, 4C00h              ; աշխատանքի/ ծրագրի ավարտը
        INT   21h
END  START
```

Այս ֆայլն անվանենք HelloW.asm:

2-րդ քայլը կատարելու համար անհրաժեշտ է օպերացիոն համակարգի ֆայլային համակարգում ունենալ մի թղթապանակ, որում կլինեն HelloW.asm ֆայլը, TASM փաթեթի Tasm.exe, Tlink.exe, Td.exe, Rtm.exe ծրագրերը, և այդ թղթապանակը հասանելի կլինի DOSBox էմուլյատորին: Ենթադրենք, որ նշված ծրագրերը տեղադրված են D:\Asm\Work թղթապանակում: DOSBox-ին այդ թղթապանակի հասանելիության համար անհրաժեշտ է այն կցել DOS-ի ֆայլային համակարգին: Դա արվում է հետևյալ հրամանով.

mount D D:\Asm\Work

Այս հրամանի աշխատանքից հետո DOS-ի ֆայլային համակարգի D մասը ցույց կտա D:\Asm\Work թղթապանակի վրա: Այժմ D-ն դարձնենք ընթացիկ աշխատանքային թղթապանակ ‘d:’ հրամանի միջոցով:

Ինչպես նշված է Նկար 1-ում, ծրագրի տեքստից թարգմանություն անելու համար անհրաժեշտ է աշխատեցնել Tasm.exe ծրագիրը: Tasm.exe-ն թարգմանիչ է, որը ասեմբլեր լեզվով գրված տեքստը թարգմանում է մեքենայական հրամանների: Թարգմանության ընթացքում ստեղծվում են օբյեկտային մոդուլի և ցանկության դեպքում նաև լիսթինգի ֆայլերը: Օբյեկտային մոդուլը պարունակում է մեքենայական հրամանները և լրացուցիչ տվյալներ Tlink.exe ու Td.exe ծրագրերի համար: Լիսթինգի ֆայլը պարունակում է համաբակավված մուտքային ասեմբլեր ծրագրի տեքստը՝ համալրված այլ տվյալներով (մեքենայական հրամաններ, սիմվոլային աղյուսակ, խմբերի և սեգմենտների մասին տեղեկատվություն): Եթե թարգմանության ընթացքում սխալներ են հայտնաբերվում, ապա լիսթինգի ֆայլում դրանց մասին նշվում է: Այժմ կատարենք 2-րդ քայլը HelloW.asm ասեմբլեր ծրագրի տեքստի համար:

Tasm.exe /zi /l HelloW.asm

Այս հրամանի արդյունքում էկրանին հայտնվում է tasm.exe-ի աշխատանքը նկարագրող տողերի հաջորդականություն: Եթե ծրագիրը սխալներ է պարունակում, ապա թարգմանիչն էկրանին դուրս է բերում “Fatal error”, “Error” կամ “Warning” բառերով սկսվող տողեր: Վերը նշված հրամանում /l պարամետրը նշանակում է, որ tasm.exe-ն պետք է ստեղծի նաև լիսթինգի ֆայլը: HelloW.asm ֆայլի թարգմանության արդյունքում ստացված լիսթինգի ֆայլն ունի հետևյալ տեսքը.

HelloW.asm

```
1    0000                    .model small
2    0000                    .stack 100h
3
4    0000                    .data
5    0000 48 65 6C 6C 6F 20  57+ message db 'Hello World!',
 13, 10, '$'
6        6F 72 6C 64 21 0D   0A+
7        24
8
9    000F                .code
10   0000                start:
11   0000 B8 0000s       mov ax, @data ; Store address of data segment in DS
segment register
12   0003 8E D8          mov ds, ax
13   0005 BA 0000r       mov dx, offset message ; Store offset of message into
dx register
14   0008 B4 09          mov ah, 09h ; Code for displaying the
message
15   000A CD 21          int 21h
16   000C B8 4C00        mov ax, 4c00h ; End of executable
17   000F CD 21          int 21h
18                       end start
```

Symbol Table

Symbol Name	Type	Value
??DATE	Text	"11/07/15"
??FILENAME	Text	"HelloW "
??TIME	Text	"19:57:22"
??VERSION	Number	040A
@32BIT	Text	0
@CODE	Text	_TEXT
@CODESIZE	Text	0
@CPU	Text	0101H
@CURSEG	Text	_TEXT
@DATA	Text	DGROUP
@DATASIZE	Text	0

```
@FILENAME                       Text    HELLOW
@INTERFACE                      Text    000H
@MODEL                  Text    2
@STACK                  Text    DGROUP
@WORDSIZE                       Text    2
MESSAGE                 Byte    DGROUP:0000
START                           Near    _TEXT:0000

Groups & Segments       Bit Size Align  Combine Class

DGROUP                  Group
 STACK                  16  0100 Para       Stack   STACK
 _DATA                  16  000F Word       Public  DATA
_TEXT                           16  000C Word       Public  CODE
```

Լիսթինգի ֆայլը բաղկացած է հետևյալ մասերից.

- էջի վերնագիր,
- ֆայլի վերնագիր,
- սկզբնական կողին համապատասխանող տողեր,
- հաղորդագրություն՝ սխալների մասին,
- սիմվոլների կամ խաչածև հղումների աղյուսակ:

Լիսթինգի ֆայլում սկզբնական կողին համապատասխանող տողերն ունեն հետևյալ կառուցվածքը.

տողի_համար I շեղում մնքևնայական_կոդ ասեմբլեր_հրաման

Տողի_համար – լիսթինգի ֆայլում տողի համարն օգտագործվում է սխալները տեղայնացնելու համար:

I - սյունը տվյալ օրինակում բացակայում է: Այն նշում է ներառվող ֆայլի (include ֆայլ) կամ մակրոյի ներդրվածությունը: Առաջին ներդրվող ֆայլը ստանում է 1 մակարդակ: Եթե այդ ներդրվող ֆայլը ներառում է մեկ այլ ֆայլ, ապա այն կստանա 2 մակարդակ և այդպես շարունակ: Մակարդակների նշման նույն մեխանիզմը կիրառվում է մակրոների ներդրվածությունը ցուցադրելու համար:

Շեղում – կոդ սեգմենտի սկզբի նկատմամբ հերթական հրամանի բայթերով արտահայտված շեղումը: Այս շեղումը նաև անվանում են հասցեի հաշվիչ:

մեքենայական_կոդ – նույն տողի ասեմբլեր հրամանի մեքենայական ներկայացումն է:

Սիմվոլների աղյուսակը (symbol table) ծրագրի բոլոր սիմվոլների ցուցակն է այբբենական կարգով: Այն պարունակում է սիմվոլի անունը, տիպը, արժեքը:

Վերոհիշյալ լիսթինգի օրինակը սխալներ չի պարունակում:

Լիսթինգի ֆայլն օգտագործվում է ծրագրում սխալների հայտնաբերման համար: Սխալներն ուղղելուց հետո tasm.exe-ի աշխատանքի արդյունք հանդիսացող օբյեկտային մոդուլը (HelloW.obj ֆայլը) պարունակում է մեքենայական հրամաններ: Սակայն այն դեռ պատրաստ չէ աշխատանքի համար: Անհրաժեշտ է կատարել Նկար 1-ում պատկերված 3-րդ քայլը: Կապերի խմբագրիչը մուտքում ստանում է մեկ կամ մի քանի օբյեկտային մոդուլ և դրանք միավորում մեկ կիրառության մեջ: Tlink.exe ծրագրի մուտքային պարամետրերի բազմությունը տեսնելու համար բավական է այն կանչել առանց որևէ պարամետրի:

Մեր օրինակի համար Tlink.exe-ն կանչենք հետևյալ պարամետրերով՝

```
Tlink.exe /v HelloW.obj
```

Արդյունքում ստացվում է HelloW.exe կիրառությունը:

Եթե HelloW.exe կիրառության աշխատանքի ընթացքում հայտնաբերվում են սխալներ, ապա դրանց պատճառը կարելի է գտնել որևէ debugger-ով: Մենք կդիտարկենք Turbo Debugger-ի (Td.exe) աշխատանքը: Td.exe ծրագիրը հնարավորություն է տալիս քայլ առ քայլ կատարելու տրված ծրագիրը և ամեն քայլից հետո տեսնելու ու փոփոխելու օգտագործողի ռեգիստրները, կիրառությանը հասանելի հիշողությունը և մեքենայական կոդը: Այդպես հնարավոր է դառնում

գտնել ծրագրում սխալի տեղը և պատճառը: Սխալն ուղղելուց հետո անհրաժեշտ է կրկին անցնել վերը նշված 1-ից 3-րդ քայլերով ու ստեղծել նոր կիրառություն: Td.exe-ի միջոցով կիրառությունը աշխատեցնելու համար անհրաժեշտ են հետևյալ 3 պայմանները.

1. Ծրագրի տեքստում պետք է հայտարարված լինի նշիչ ծրագրի առաջին հրամանի համար: Այդ նշիչի անունը պետք է նշված լինի ծրագրի տեքստի վերջում որպես END հրամանագրի արգումենտ:

2. Ծրագրի թարգմանության համար ցանկալի է տրված լինի /zi պարամետրը: Սա թարգմանչին թույլ է տալիս պահել սիմվոլային անունների և կոդ սեգմենտում դրանց շեղումների կապը: Հարկ է նշել, որ եթե /zi պարամետրը տրված չլինի, ապա Td.exe-ն սիմվոլային անունների փոխարեն ցույց կտա դրանց շեղումները:

3. Կապերի խմբագրման համար պետք է տրված լինի /v պարամետրը: Սա թույլ է տալիս պահպանել թարգմանության ժամանակ ստեղծված տվյալները:

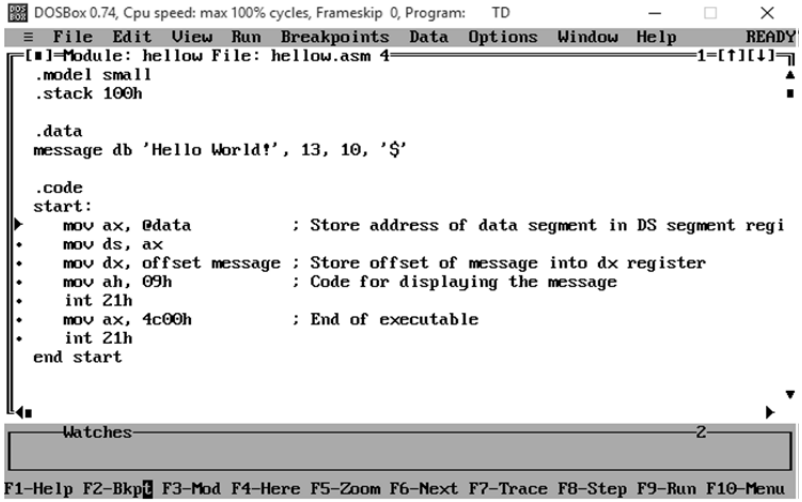
Նշված քայլերը կատարելուց հետո HelloW.exe-ն կարելի է աշխատեցնել Td.exe-ի միջոցով հետևյալ կերպ.

Td.exe HelloW.exe

Այս հրամանը կբացի Td.exe ծրագրի պատուհանը (Նկար 2):

Start նշիչի տակ գտնվող եռանկյունը ցույց է տալիս հաջորդ հրամանը, որը պետք է կատարվի:

F8 սեղմելու դեպքում կկատարվի այդ հրամանը, որից հետո ղեկավարումը կրկին կտրվի Td.exe-ին, որպեսզի հնարավոր լինի տեսնել համակարգի վիճակն այդ քայլից հետո:



Նկար 2. TurboDebugger ծրագրի պատուհանը

Համակարգի վիճակը տեսնելու համար կան մի քանի պատուհաններ, որոնք կարելի է բացել View ընտրացուցակից: Դիտարկենք դրանցից մի քանիսը:

1. **CPU** պատուհանում ցույց է տրվում ծրագրի մեքենայական կոդը:
2. **Registers** պատուհանում ցույց են տրվում ընդհանուր օգտագործման ռեգիստրների արժեքները:
3. **Flags** պատուհանում ցույց են տրվում դրոշների արժեքները:
4. **Stack** պատուհանում ցույց է տրվում պահունակ սեգմենտի պարունակությունը:
5. **Dump** պատուհանն արտապատկերում է ձախ կողմում նշված հասցեների պարունակությունը:

Td.exe-ի աշխատանքը դադարեցնելու համար կարելի է սեղմել Alt+X:

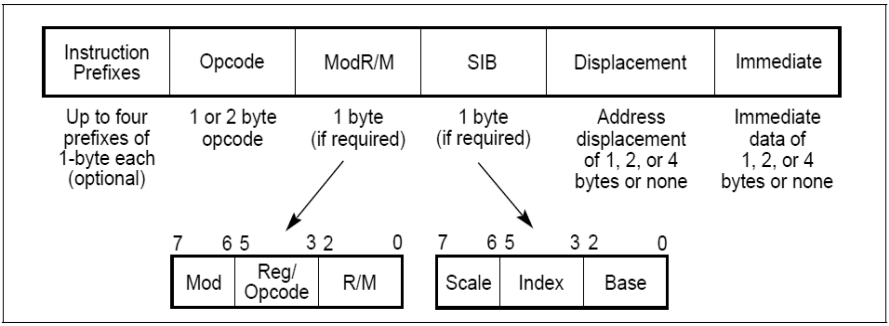
ՄԵՔԵՆԱՅԱԿԱՆ ՀՐԱՄԱՆԻ ՖՈՐՄԱՏԸ

Հրամանը թարգմանվելուց կարող է ստացվել 1-ից մինչև 15 բայթ:

Մեքենայական հրամանը կազմված է հետևյալ դաշտերից.

- հրամանի նախդիր/Instruction Prefix (մինչև 4 նախդիր, յուրաքանչյուրը՝ 1 բայթ),
- գործողության կոդ/Opcode (1 կամ 2 բայթ),
- ModR/M բայթ,
- SIB բայթ (Scale-Index-Base),
- տեղաշարժ/Displacement (0,1, 2 կամ 4 բայթ),
- անմիջական օպերանդ/Immediate (0,1, 2 կամ 4 բայթ):

Նշված դաշտերից պարտադիր է միայն գործողության կոդի դաշտը:



Նկար 1. IA-32 մեքենայական հրամանի ֆորմատը

Հրամանի նախդիր – օգտագործվում է հրամանի մեջ եղած որոշ դաշտեր փոխելու համար: Նախդիրները բաժանվում են 4 խմբի.

1. սեգմենտի վերաբեռնում (Segment override prefixes),
2. օպերանդի չափի վերաբեռնում (66H – Operand-size override prefix),
3. հասցեի չափի վերաբեռնում (67H – Address-size override prefix),

4. կապուղու (computer bus, шина) փակման և կրկնման նախդիրներ (F0H-Lock prefix and repeat prefixe):

Սեգմենտային ռեգիստրի վերաբեռնում կատարվում է, եթե հրամանում բացահայտ նշված է սեգմենտային ռեգիստրը:

Օրինակ

MOV AX, GS:T ; ds սեգմենտային ռեգիստրի փոխարեն կվերցնի GS-ը

Նշված հրամանում կատարված է լռությամբ գործող DS սեգմենտային ռեգիստրի վերաբեռնում GS սեգմենտային ռեգիստրով: Թարգմանված մեքենայական կոդը կպարունակի GS սեգմենտային ռեգիստրի կոդը որպես 1) տեսակի նախդիր:

Եթե սեգմենտը 16-բիթանի է, ապա լռությամբ օպերանդի չափը և հասցեավորման եղանակը 16-բիթանի են: 32-բիթանոց սեգմենտի դեպքում լռությամբ օպերանդի չափը և հասցեավորման եղանակը 32-բիթանի են:

Լռությամբ օպերանդի կամ հասցեավորման չափը փոխելու համար օգտագործվում են **օպերանդի չափի վերաբեռնման և հասցեավորման չափի վերաբեռնման նախդիրները:**

Կրկնության նախդիր – հրամանի կրկնման նախդիրն օգտագործվում է տողերի մշակման հրամաններում, որոնց օգնությամբ հրամանը կարող է կրկնվել մեկից ավելի անգամ:

Կապուղու փակում – այս նախդիրն օգտագործվում է միայն որոշակի հրամաններից առաջ: Այլ հրամանների մեջ օգտագործելը կհանգեցնի սխալի:

Գործողության կոդ – գործողության կոդը զբաղեցնում է մեկ կամ 2 բայթ: Այն կարող է ունենալ 3-բիթանոց ընդլայնում, որը գրանցվում է ModR/M բայթում:

Mod R/M բայթ – mod R/m-ը բաժանվում է երեք մասի:

7 6	5 4 3	2 1 0
Mod	Reg/OpCode	R/M

Mod-ը և R/M-ը միասին որոշում են հասցեավորման եղանակները (32 հնարավոր արժեք՝ 8 ռեգիստր և 24 հասցեավորման եղանակ):

- *reg/opcode* դաշտը որոշում է կա՛մ ռեգիստրի կողը, կա՛մ գործողության կողի ընդլայնումը:
- *r/m* դաշտը կարող է որոշել ռեգիստր կամ միավորված mod դաշտի հետ հասցեավորման եղանակը:

16-բիթանոց հասցեավորման դեպքում mod դաշտի նշանակությունը հետևյալն է.

00 – օգտագործված է հիշողության հասցեավորում՝ առանց տեղաշարժի (հրամանի մեջ տեղաշարժ դաշտը բացակայում է),

01 – օգտագործված է հիշողության հասցեավորում 1-բայթանի տեղաշարժով (հրամանի մեջ տեղաշարժը 1 բայթ է),

10 – օգտագործված է հիշողության հասցեավորում երկու բայթանի տեղաշարժով,

11 – ռեգիստրային կամ անմիջական հասցեավորում:

reg	AL AX EAX	CL CX ECX	DL DX EDX	BL BX EBX	AH SP ESP	CH BP EBP	DH SI ESI	BH DI EDI
2-ական կողը	000	001	010	011	100	101	110	111

Աղյուսակ 1. Ռեգիստրների կոդերի աղյուսակ

R/M	Mod = 00	Mod = 01	Mod = 10
000	[BX + SI]	[BX + SI] + d8	[BX + SI] + d16
001	[BX + DI]	[BX + DI] + d8	[BX + DI] + d16
010	[BP + SI]	[BP + SI] + d8	[BP + SI] + d16
011	[BP + DI]	[BP + DI] + d8	[BP + DI] + d16
100	[SI]	[SI] + d8	[SI] + d16
101	[DI]	[DI] + d8	[DI] + d16
110	Ուղղակի հասցեավորում	[BP] + d8	[BP] + d16
111	[BX]	[BX] + d8	[BX] + d16

Աղյուսակ 2. Արդյունավետ հասցեի հաշվարկ

SIB դաշտ – այս դաշտն օգտագործվում է միայն 32-բիթանոց հասցեավորման դեպքում և կազմված է **մասշտաբ** (2 բիթ), **ինդեքս** (3 բիթ) և **բազա** (3 բիթ) դաշտերից:

Մասշտաբ դաշտը որոշում է մասշտաբային գործակիցը: Այն ընդունում է 4 հնարավոր արժեքներ, որոնց նշանակությունը հետևյալն է.

00 – մասշտաբը 1 է,

01 – մասշտաբը 2 է,

10 – մասշտաբը 4 է,

11 – մասշտաբը 8 է:

Ինդեքս և բազա - համապատասխանաբար որոշում են ինդեքսային և բազային ռեգիստրները:

Խնդիրների լուծման օրինակներ

Դիտարկենք 16-բիթանոց հասցեավորման մի քանի օրինակներ:

Խնդիր

Որոշել հետևյալ հրամանների մեքենայական կոդը՝ օգտվելով գումարման գործողության (ADD) կոդային աղյուսակից:

Կոդային աղյուսակում գործողության կողին հաջորդող նշաններին նշանակությունը հետևյալն է.

“/r” – մեքենայական հրամանի մեջ գործողության կողի դաշտին հետևում է ModR/M բայթը,

Ib – անմիջական 1-բայթանի օպերանդ,

Iw – անմիջական 2-բայթանի օպերանդ,

Id – անմիջական 4-բայթանի օպերանդ,

/n – գործողության կողի n ընդլայնում:

Գործողության կոդ	Հրաման	Նկարագրություն
04 ib	ADD AL, imm8	AL = AL + imm8
05 iw	ADD AX, imm16	AX = AX + imm16
05 id	ADD EAX, imm32	EAX = EAX + imm32
80 / 0 ib	ADD r/m8, imm8	r/m8 = r/m8 + imm8
81 / 0 iw	ADD r/m16, imm16	r/m16 = r/m16 + imm16
81 / 0 id	ADD r/m32, imm32	r/m32 = r/m32 + imm32
83 / 0 ib	ADD r/m16, imm8	r/m16 = r/m16 + imm8 (imm8-ը նշանով թիվ է)
83 / 0 ib	ADD r/m32, imm8	r/m32 = r/m32 + imm8 (imm8-ը նշանով թիվ է)
00 / r	ADD r/m8, r8	r/m8 = r/m8 + r8
01 / r	ADD r/m16, r16	r/m16 = r/m16 + r16
01 / r	ADD r/m32, r32	r/m32 = r/m32 + r32
02 / r	ADD r8, r/m8	r8 = r8 + r/m8
03 / r	ADD r16, r/m16	R16 = r16 + r/m16
03 / r	ADD r32, r/m32	R32 = r32 + r/m32

Աղյուսակ 3. ADD հրաման

1. ADD DI, DX

Լուծում

Նշված հրամանն ունի

ADD r16, r16

տեսքը: Հետևաբար, ըստ աղյուսակի, այն ունի 01/r կամ 03/r գործողության կողը: Առաջին դեպքում ընդունիչ օպերանդը r/m16-ն է, իսկ երկրորդ դեպքում r/m16-ը աղբյուր օպերանդ է: Ընտրենք ևվազագույն կողը՝ 01:

Քանի որ 01 գործողության կողի դաշտին հետևելու է ModR/M բայթը, կողավորենք այն: Mod դաշտը կունենա 11 արժեք, քանի որ տվյալ դեպքում օգտագործված է ռեգիստրային հասցեավորում: DI ռեգիստրի կողը 111 է, իսկ DX–ինը՝ 010 (տե՛ս Աղյուսակ 1-ը):

Քանի որ DI-ն ընդունիչ օպերանդն է, այն կգրանցվի R/M դաշտում, իսկ DX-ի կոդը կգրանցվի Reg/OpCode դաշտում: Այսպիսով՝ կունենանք.

Mod	DX	DI
11	010	111

Պատասխան

ADD DX, DI հրամանի մեքենայական կոդը 16-ական համակարգով կլինի 01 D7 h, որտեղ՝

01 – գործողության կոդ (OpCode) է,

D7 – ModR/M բայթն է,

2. ADD CX, [BX + 7]

Լուծում

Տրված հրամանն ունի հետևյալ շարահյուսությունը.

ADD r16, m16

Հետևաբար, ըստ աղյուսակի, այն ունի 03 գործողության կոդ (համապատասխանում է աղյուսակի 03/r r16, r/m16 տողին), և տվյալ հրամանի մեքենայական կոդը բաղկացած է գործողության կոդի (03), ModR/M բայթի և 8-բիթանոց (7) տեղաշարժի դաշտերից:

Ստանանք ModR/M բայթի արժեքը: Տվյալ դեպքում Mod դաշտի արժեքը կլինի 01, քանի որ [BX + 7] հասցեավորման մեջ մասնակցում է 8-բիթանոց տեղաշարժ: CX ռեգիստրի կոդը, ըստ աղյուսակի, 001 է, իսկ [BX] + d8 հասցեավորման եղանակինը՝ 111:

Mod	CX	[BX]
01	001	111

Պատասխան

ADD CX, [BX + 7] հրամանի մեքենայական կոդը 16-ական համակարգով՝ 03 8F 07h , որտեղ՝

03 – գործողության կոդ (Opcode),

8F – ModR/M,

07 – տեղաշարժ (Displacement):

3. ADD BX, 5

Լուծում

ADD BX, 5 հրամանն ունի հետևյալ շարահյուսությունը՝

ADD r16, imm8,

որին համապատասխանում է 83 /0 ib կոդը:

Տվյալ դեպքում գործողության կոդն ունի ընդլայնում՝ 0: BX ռեգիստրի կոդը 011 է: ib նշանակում է մեքենայական կոդում մասնակցելու է անմիջական օպերանդ (5): Mod-ը կունենա 11 արժեք, քանի որ տվյալ դեպքում օգտագործված է անմիջական հասցեավորում.

Mod	Opcode	BX
11	000	011

Պատասխան

ADD BX, 5 հրամանի մեքանայական կոդը 16-ական համակարգով ստացվեց՝ 83 C3 05h:

83 – գործողության կոդ (Opcode),

C3 – ModR/M,

05 – անմիջական օպերանդ (Immidiate):

4. ADD DX, [BX + SI]

Լուծում

ADD DX, [BX + SI] ⇔ ADD r16, m16 ⇔ 03/r r16, r/m16

=> 03 10h

Պատասխան

03 10h

5. ADD AX, [SI]

Լուծում

ADD AX, [SI] => 00000011 00000100 = 03 04 h

Պատասխան
03 04h

6. ADD [BX][DI] + 1234h, AX

Լուծում
ADD [BX][DI] + 1234h, AX => 00000001 10000001 __ __ h
=> 01 81 34 12h

Պատասխան
01 81 34 12h

Խնդիրներ և վարժություններ

1. Ստանալ հետևյալ ասեմբլեր հրամանների մեքենայական կոդը.
   1) ADD CX, 7
   2) ADD BX, [SI + 300h]
   3) ADD [BX], CX
   4) MOV DX, [BX + SI + 10]
   5) SUB CX, AX
   6) SUB DX, 200
   7) SUB DX, [SI + 100h]

ԻՆԹԵԼ x86 ՊՐՈՑԵՍՈՐԻ ՀՐԱՄԱՆՆԵՐԻ ՀԱՄԱԿԱՐԳԸ: ՈՒՆԻՎԵՐՍԱԼ ՀՐԱՄԱՆՆԵՐ

IA-32 հրամանները բաժանվում են հետևյալ հիմնական խմբերի.

- ընդհանուր օգտագործման,
- x87 FPU,
- Ինթել MMX տեխնոլոգիայի,
- SSE ընդլայնումներ,
- SSE2 ընդլայնումներ,
- համակարգային:

Կուսումնասիրենք միայն ընդհանուր օգտագործման հրամանները:

Ընդհանուր օգտագործման հրամաններ

Այս հրամանները կատարում են տվյալների տեղափոխության, թվաբանական, տրամաբանական և տողային գործողություններ, որոնք ծրագրավորողներն օգտագործում են կիրառություններ և համակարգային ծրագրային ապահովում գրելու համար: Այս խմբի հրամանները գործողություններ են կատարում հիշողության, ընդհանուր օգտագործման ռեգիստրների և դրոշների հետ: Այս խումբը բաղկացած է հետևյալ ֆունկցիոնալ ենթախմբերից.

1. տվյալների տեղափոխության հրամաններ,
2. երկուական թվաբանության հրամաններ,
3. տասական թվաբանության հրամաններ,
4. տրամաբանական հրամաններ,
5. տեղաշարժի և պտույտի հրամաններ,
6. բիթերի և բայթերի հետ աշխատող հրամաններ,
7. դրոշների հետ աշխատող հրամաններ,
8. տողային հրամաններ,

9. ղեկավարությունը փոխանցող հրամաններ,
10. այլ հրամաններ:

Տվյալների տեղափոխության հրամաններ

Տվյալների տեղափոխության հրամանները տեղափոխում են տվյալները հիշողության հասցեների, ընդհանուր օգտագործման ռեգիստրների և սեգմենտային ռեգիստրների միջև: Այս խմբի հրամանները նաև իրականացնում են հատուկ գործողություններ, ինչպիսիք են՝ պայմանական տեղաշարժը, պահունակի կամ կայանի (port, պորտ) հետ աշխատանքը և տվյալների ձևափոխությունը: Տեղափոխության կարող է ենթարկվել բայթը, բառը կամ կրկնակի բառը: Օպերանդները պետք է լինեն նույն չափի: **Այս խմբի հրամանները չեն ազդում դրոշների վրա:**

MOV Տեղափոխում / Move data

CMOVxx Պայմանական տեղաշարժ, եթե xx պայմանը տեղի ունի /Conditional move if xx

XCHG Փոխանակում / Exchange

BSWAP Բայթային փոխանակում / Byte swap

CBW/CWD Բայթի/բառի ձևափոխում բառի/կրկնակի բառի /Convert byte/word to doubleword

CDQ Կրկնակի բառի ձևափոխում քառակի բառի /Convert doubleword to quadword

CWDE Բառի ձևափոխում կրկնակի բառի EAX ռեգիստրում / Convert word to doubleword in EAX register

MOVSX Տեղափոխում և նշանի ընդլայնում / Move and sign extend

MOVZX Տեղափոխում և զրոյով ընդլայնում/Move and zero extend

LEA Արդյունավետ հասցեի բեռնում / Load effective address

XLAT Թարգմանություն / Table look-up translation

PUSH Գրել պահունակում / Push onto stack

POP Կարդալ պահունակից / Pop off of stack

PUSHA/PUSHAD Ընդհանուր օգտագործման ռեգիստրների արժեքները ուղարկել պահունակ/Push general-purpose registers onto stack

POPA/POPAD Ընդհանուր օգտագործման ռեգիստրները վերարժեքավորել պահունակից/Pop general-purpose registers from stack

IN Կարդալ կայանից (port, պորտ) /Read from a port

OUT Գրել կայանում (port, պորտ) /Write to a port

MOV – Տեղափոխում

Շարահյուսություն: **MOV ընդունիչ, աղբյուր**

Նկարագրություն: ընդունիչ ← աղբյուր

Ընդունիչը կարող է լինել ռեգիստր (reg8/reg16/reg32), հիշողության հասցե (mem8/mem16/mem32) կամ սեգմենտային ռեգիստր (segreg), իսկ աղբյուրը՝ ռեգիստր (reg8/reg16/reg32), հիշողության հասցե (mem8/mem16/mem32), սեգմենտային ռեգիստր (segreg) կամ անմիջական արժեք (im8/im16/im32):

Այս հրամանը կատարում է տվյալների տեղափոխություն ընդհանուր օգտագործման ռեգիստրների կամ ընդհանուր օգտագործման և սեգմենտային ռեգիստրների միջև, հիշողության և ընդհանուր օգտագործման կամ սեգմենտային ռեգիստրի միջև կամ տեղափոխում է անմիջական արժեքը ընդհանուր օգտագործման ռեգիստրի կամ հիշողության մեջ:

MOV հրամանը տեղափոխում է աղբյուրով որոշվող արժեքի պատճենը ընդունիչ: Գործողության իրականացումից հետո ընդունիչ օպերանդի հին արժեքը կորչում է:

- MOV հրամանի բոլոր տարբերակներում երկու օպերանդները պետք է լինեն նույն տիպի (չափի) (բացառություն են կազմում MOV segreg,r/m32; MOV r/m32,segreg հրամանները):

- Շարահյուսությունը չի թույլատրում անմիջական արժեքը կամ սեգմենտային ռեգիստրը տեղափոխել սեգմենտային ռեգիստր:
- MOV հրամանը չի կարող CS ռեգիստրը օգտագործել որպես ընդունիչ:
- 386+ պրոցեսորների համար MOV հրամանը կարելի է օգտագործել նաև ղեկավարող (CR0/2/3/4) և ճշգրտման (DR0/1/2/3/6/7 ռեգիստրների հետ աշխատելու համար
  MOV reg32, CR0/2/3/4
  MOV reg32, DR0/1/2/3/6/7
  MOV CR0/2/3/4, reg32
  MOV DR0/1/2/3/6/7, reg32:
- CR4 ռեգիստրը հասանելի է միայն Pentium և բարձր պրոցեսորների համար:

Ինքել x86 և x86-64 պրոցեսորներն օգտագործում են little-endian ֆորմատը (այն հայտնի է նաև որպես “Ինքել համաձայնեցում”/Intel convention"), որն ապահովում է կրտսեր բայթերը հիշողության փոքր հասցեում տեղափոխելու մեխանիզմը (ի տարբերություն big-endian ֆորմատի, որտեղ կրտսեր բայթը գրանցվում է մեծ հասցեում): big-endian ֆորմատը նաև հայտնի է որպես "Motorola convention":

Օրինակ

DS = ACEDH, SI = 0724H

MOV AX, [SI]

AH	AL
55	02

Ֆիզիկական հասցե =
ACEDH * 10H + 0724H =
= AD5F4H, AD5F5H

072AH	**2EH**	AD5FAH
0729H	**AAH**	AD5F9H
0728H	**18H**	AD5F8H
0727H	**A3H**	AD5F7H
0726H	**7EH**	AD5F6H
0725H	**55H**	AD5F5H
0724H	**02H**	AD5F4H
0723H	**72H**	AD5F3H

Օրինակ

```
MOV AX, 7       ; անմիջական արժեքը գրել ընդհանուր օգտա-
                ; գործման ռեգիստրի մեջ,
MOV mem, 7      ; անմիջական արժեքը գրել 1 բայթում՝ հիշողու-
                ; թյան mem հասցեից սկսած,
MOV memw, 7     ; անմիջական արժեքը գրել 1 բառում՝ հիշողու-
                ; թյան memw հասցեից սկսած,
MOV memw, DS    ; սեգմենտային ռեգիստրի արժեքը գրել հիշողու-
                ; թյան memw հասցեում,
MOV memw, AX    ; ընդհանուր օգտագործման ռեգիստրի արժեքը
                ; գրել հիշողության memw հասցեում,
MOV AX, memw    ; ընդհանուր օգտագործման ռեգիստրի մեջ գրել
                ; 2 բայթը՝ հիշողության memw հասցեից սկսած,
MOV AX, BX      ; ընդհանուր օգտագործման ռեգիստրի արժեքը
                ; գրել ընդհանուր օգտագործման ռեգիստրի մեջ,
MOV DS, AX      ; ընդհանուր օգտագործման ռեգիստրի արժեքը
                ; գրել սեգմենտային ռեգիստրի մեջ,
MOV CX, ES      ; սեգմենտային ռեգիստրի արժեքը գրել ընդհա-
                ; նուր օգտագործման ռեգիստրի մեջ:
```

Որտեղ՝

```
MEMW    DW    ?
MEM     DB    ?
```

CMOVcc – Պայմանական տեղաշարժ/Conditional Move (P6+)

Շարահյուսություն:	**CMOVcc ընդունիչ, աղբյուր**
Նկարագրություն:	ընդունիչ ← աղբյուր, եթե cc պայմանը տեղի ունի

Ընդունիչը կարող է լինել ռեգիստր (reg16/reg32), իսկ աղբյուրը՝ ռեգիստր (reg16/reg32) կամ հիշողության հասցե (mem16/mem32):

CMOVcc հրամանը տեղափոխում է աղբյուրի արժեքի պատճենը ընդունիչ օպերանդ, եթե տրված պայմանը տեղի ունի, հակառակ դեպքում ոչինչ չի անում:

Յուրաքանչյուր հրաման ունի իրեն համարժեք մեկ այլ անվանում: Համարժեք (նույն մեքենայական կոդն ունեցող) հրամանները բաժանված են ‘/’ սիմվոլով:

Հրամանի հուշանուն	**Պայմանի ստուգում՝ ըստ դրոշների**	**Հրամանի լրիվ անվանումը**
CMOVA / CMOVNBE	CF = 0 & ZF = 0	Տեղափոխել, եթե բարձր է (Move if above) / Տեղափոխել, եթե ցածր կամ հավասար չէ (Move if not below or equal)
CMOVAE / CMOVNB / CMOVNC	CF = 0	Տեղափոխել, եթե բարձր է կամ հավասար (Move if above or equal) / Տեղափոխել, եթե ցածր չէ (Move if not below) / Տեղափոխել, եթե փոխանցում չկա (Move if not carry)
CMOVB / CMOVNAE / CMOVC	CF = 1	Տեղափոխել, եթե ցածր է (Move if below) / Տեղափոխել, եթե բարձր կամ հավասար չէ (Move if not above or equal) / Տեղափոխել, եթե փոխանցում կա (Move if carry)
CMOVBE / CMOVNA	CF = 1 | ZF = 1	Տեղափոխել, եթե ցածր կամ հավասար է (Move if below or equal) / Տեղափոխել, եթե բարձր չէ (Move if not above)
CMOVE / CMOVZ	ZF = 1	Տեղափոխել, եթե հավասար է (Move if equal) / Տեղափոխել, եթե զրո է (Move if zero)
CMOVG / CMOVNLE	ZF = 0 & SF = OF	Տեղափոխել, եթե մեծ է (Move if greater) / Տեղափոխել, եթե փոքր կամ հավասար չէ (Move if not less or equal)
CMOVGE /	SF = OF	Տեղափոխել, եթե մեծ կամ հավասար է (Move if greater or equal)

CMOVNL		/ Տեղափոխել, եթե փոքր չէ (Move if not less)
CMOVL / CMOVNGE	SF <> OF	Տեղափոխել, եթե փոքր է (Move if less) /Տեղափոխել, եթե մեծ կամ հավասար չէ (Move if not greater or equal)
CMOVLE / CMOVNG	ZF =1 | SF <> OF	Տեղափոխել, եթե փոքր կամ հավասար է (Move if less or equal) / Տեղափոխել, եթե մեծ չէ (Move if not greater)
CMOVNE / CMOVNZ	ZF = 0	Տեղափոխել, եթե հավասար չէ (Move if not equal) / Տեղափոխել, եթե զրո չէ (Move if not zero)
CMOVNO	OF = 0	Տեղափոխել, եթե գերհագեցում չկա (Move if not overflow)
CMOVO	OF = 1	Տեղափոխել, եթե գերհագեցում կա (Move if overflow)
CMOVNP / CMOVPO	PF = 0	Տեղափոխել, եթե զույգություն չկա (Move if not parity) / Տեղափոխել, եթե կենտ է (Move if parity odd)
CMOVP / CMOVPE	PF = 1	Տեղափոխել, եթե զույգություն կա (Move if parity) / Տեղափոխել, եթե զույգ է (Move if parity even)
CMOVNS	SF = 0	Տեղափոխել, եթե նշան չկա. ոչ բացասական է (Move if not sign)
CMOVS	SF = 1	Տեղափոխել, եթե նշան կա. բացասական է (Move if sign)

Նկար 1. CMOVcc հրամանի իրականացման պայմանները

XCHG –Փոխանակում

Շարահյուսություն: **XCHG ընդունիչ, աղբյուր**

Նկարագրություն: ընդունիչ ↔ աղբյուր

Ընդունիչը և աղբյուրը կարող են լինել ռեգիստր (reg8/reg16/reg32) կամ հիշողության հասցե (mem8/mem16/ mem32):

Այս հրամանով ընդունիչ և աղբյուր օպերանդների արժեքները փոխանակվում են:

Օրինակ

```
XCHG   AX, BX        ; փոխանակվում են AX-ի և BX-ի ար-
                     ; ժեքները,
XCHG   mem16, AX     ; փոխանակվում են հիշողության
                     ; բառի՝ mem16-ի և  AX-ի արժեքները,
XCHG   DL, mem8      ; փոխանակվում են DL-ի և հիշողու-
                     ; թյան բայթի՝ mem8-ի  արժեքները,
XCHG   AL, CL        ; փոխանակվում են AH-ի և CL-ի ար-
                     ; ժեքները:
```

BSWAP – բայթային փոխանակում (486+)

Շարահյուսություն: **BSWAP reg32**

Նկարագրություն: 32-բիթանոց ռեգիստրի բայթերը դասավորում է հակառակ հերթականությամբ

BSWAP հրամանը 32-բիթանոց ռեգիստրի 0-7 բիթերը փոխանակում է 24-31 բիթերի հետ, իսկ 8-15 բիթերը՝ 16-23 բիթերի հետ: Այս հրամանի 16-բիթանոց համարժեքը չկա:

Նշված հրամանը կարելի է օգտագործել “**little endian”** ֆորմատից “**big endian”** ֆորմատ և, հակառակը, անցումներ կատարելու համար:

Օրինակ

```
MOV    EAX, 01020304h  ; EAX = 01020304h
BSWAP  EAX             ; EAX = 04030201h
```

CBW, CWD - Բայթի/բառի ձևափոխում բառի/կրկնակի բառի /Convert byte/word to doubleword

Շարահյուսություն: **CBW/ CWD**

Նկարագրություն: AX ← CBW(AL) / (DX:AX) ← CWD(AX)

CBW և CWD-ն առանց օպերանդի հրամաններ են: Լռությամբ օպերանդը AL (CBW-ի դեպքում) կամ AX (CWD-ի դեպքում) ռեգիստրներն են: Այս հրամանները տարածում են են AL/AX-ի նշանի բիթը AH/DX ռեգիստրում: Արդյունքում AL/AX-ում գրված թիվը գըրվում է AX/(DX:AX) ռեգիստրներում կրկնակի չափով:

Օրինակ

```
MOV  AL, 24      ; AL = 24 = 00011000b = 18h
CBW              ; AX = 0000 0000 0001 1000b = 0018h = 24
MOV  AL, -5      ; AL = 11111011b = 0FBh
CBW              ; AX = 11111111 11111011b = 0FFFBh = -5
MOV  AX, 1059    ; AX = 1059 = 0000 0100 0010 0011b = 0423h
CWD              ; DX:AX = 0000 0423h = 1059
MOV  AX, -1059   ; AX = -1059 = 1111 1011 1101 1101b =
                 ; = 0FBDDh
CWD              ; DX:AX = 0FFFF FBDDh = -1059
```

CDQ – Կրկնակի բառի ձևափոխում քառակի բառի / Convert doubleword to quadword (386+)

Շարահյուսություն: **CDQ**

Նկարագրություն: (EDX:EAX) ← CDQ(EAX)

CDQ-ն առանց օպերանդի հրաման է: Լռությամբ օպերանդը EAX ռեգիստրն է: CDQ հրամանը պատճենում է EAX-ի նշանի բիթը EDX ռեգիստրի բիթերում: Արդյունքում EDX:EAX ռեգիստրներում ստանում ենք մինչ այդ EAX-ում գրված թիվը՝ կրկնակի չափով և նշանը պահպանած:

Օրինակ

```
MOV      EAX, 12153    ; EAX  = 12153 = 0000 2F79h
CDQ                    ; EDX:EAX = 0000 0000 0000 2F79h
MOV      EAX, -12153   ; EAX = -12153 = 0FFFF D087h
CDQ                    ; EDX:EAX = 0FFFF FFFF FFFF C087h
```

CWDE – Բառի ձևափոխում կրկնակի բառի EAX ռեգիստրում /
Convert word to doubleword in EAX register (386+)

Շարահյուսություն: **CWDE**
Նկարագրություն: EAX ← CWDE(AX)

CWDE-ն առանց օպերանդի հրաման է: Լռությամբ օպերանդը AX ռեգիստրն է: CWDE հրամանը պատճենում է AX-ի նշանի բիթը EAX ռեգիստրի ավագ 16 բիթերում: Արդյունքում AX-ում գրված թիվը գրվում է EAX ռեգիստրում՝ նշանը պահպանած:

Օրինակ

```
MOV     AX, 12153       ; AX = 12153 = 2F79h
CWDE                    ; EAX = 0000 2F79h = 12153
MOV     AX, -12153      ; AX = -12153 = 0D087h
CWDE                    ; EAX = 0FFFF D087h = -12153
```

MOVSX - Տեղափոխում և նշանի ընդլայնում/
Move and sign extend (386+)

Շարահյուսություն: **MOVSX ընդունիչ, աղբյուր**
Նկարագրություն: աղբյուր ← CBW(ընդունիչ) / CWD(ընդունիչ)

Ընդունիչը կարող է լինել 16/32-բիթանոց ռեգիստր (reg16/reg32), իսկ աղբյուրը՝ 8/16-բիթանոց ռեգիստր (reg8/reg16) կամ հիշողություն հասցե (mem8/mem16): MOVSX հրամանը ընդլայնում է աղբյուրի արժեքը կրկնակի չափով՝ լրացնելով ավագ 8/16 բիթերը աղբյուրի նշանի բիթի արժեքով (CBW/CWD-ի նման): Ստացված թիվը գրանցվում է ընդունիչի մեջ:

Օրինակ

```
MOV     CH, 25          ; CH = 25 = 19h
MOVSX   BX, CH          ; BX = 0019h = 25
MOV     BL, -9          ; BL = 0F7h
MOVSX   DI, BL          ; DI = 0FFF7h = -9
MOV     AX, 1070        ; AX = 1070 = 042Eh
MOVSX   ESI, AX         ; ESI = 0000 042Eh = 1070
MOV     AX, -1070       ; AX = -1070 = 0FBD2h
MOVSX   EBX, AX         ; EBX = 0FFFF FBD2h = -1070
```

MOVZX - Տեղափոխում և զրոյով ընդլայնում /
Move and zero extend (386+)

Շարահյուսություն: **MOVZX ընդունիչ, աղբյուր**
Նկարագրություն: ընդունիչ ← MOVZX(աղբյուր)

Ընդունիչը կարող է լինել 16/32-բիթանոց ռեգիստր (reg16/reg32), իսկ աղբյուրը՝ 8/16-բիթանոց ռեգիստր (reg8/reg16) կամ հիշողություն հասցե (mem8/mem16): MOVZX հրամանը աղբյուրի արժեքը ընդլայնում է կրկնակի չափով՝ լրացնելով ավագ 8/16 բիթերը զրոներով: Ստացված թիվը գրանցվում է ընդունիչի մեջ:

Օրինակ

```
MOV     CH, 25          ; CH = 25 = 19h
MOVZX   BX, CH          ; BX = 0019h = 25
MOV     BL, 248         ; BL = 248 = 0F8h
MOVZX   DI, BL          ; DI = 00F8h = 248
MOV     AX, 1070        ; AX = 1070 = 042Eh
MOVZX   ESI, AX         ; ESI = 0000 042Eh = 1070
MOV     AX, 35420       ; AX = 35420 = 8A5Ch
MOVZX   EBX, AX         ; EBX = 0000 8A5Ch = 35420
```

LEA - Արդյունավետ հասցեի բեռնում / Load effective address

Շարահյուսություն: **LEA ընդունիչ, աղբյուր**
Նկարագրություն: ընդունիչ ← արդյունավետ հասցե (աղբյուր)

Ընդունիչը կարող է լինել 16/32-բիթանոց ռեգիստր (reg16/reg32), իսկ աղբյուրը՝ հիշողություն հասցե: LEA հրամանը հաշվում է աղբյուրի 16 կամ 32-բիթանոց արդյունավետ հասցեն (սեգմենտի բիթայնությունից կախված) և գրանցում ընդունիչի մեջ:

Եթե աղբյուրի հասցեն 16-բիթանոց է, իսկ ընդունիչը՝ 32-բիթանոց ռեգիստր, ապա ընդունիչի ավագ 16 բիթերը լրացվում են 0-ներով: Եթե աղբյուրի հասցեն 32-բիթանոց է, իսկ ընդունիչը՝ 16-բիթանոց, ապա ընդունիչի մեջ գրվում է աղբյուրի արդյունավետ հասցեի կրտսեր 16 բիթը, իսկ ավագ 16 բիթի արժեքը կորչում է:

Օրինակ

```
LEA     DX, a           ; DX ← a փոփոխականի հասցեն՝ 0
LEA     AX, b[2]        ; AX ← b[2] = [b + 2] = 5-ի հասցեն՝ 3
MOV     SI, 1
LEA     BX, C[SI + 2]   ; BX ← C[SI + 2] = C[3] = [C + 3] = 40-ի
                        ; հասցեն՝12
```

որտեղ.

```
.DATA
    A       DB      10
    B       DW      4, 5, -10, 7
    C       DB      2, 7, 15, 40
```

XLAT/ XLATB – Թարգմանություն / Table look-up translation

Շարահյուսություն: **XLAT աղբյուր / XLATB**

Նկարագրություն: AL ← DS: (BX / EBX + AL)

Աղբյուրը հիշողության հասցե է (mem8): XLAT/XLATB հրամանը DS:BX/EBX հասցեով որոշվող աղյուսակի AL-րդ տարրի 1-բայթանոց արժեքը գրանցում է AL ռեգիստրում:

XLATB հրամանը XLAT-ի „կարճ գրության" ձևն է: Աղբյուրը (աղյուսակի անունը) նշվում է միայն իբրև մեկնաբանություն, այն կարելի է օգտագործել DS սեգմենտային ռեգիստրը վերաբեռնելու համար:

Օրինակ 1

AL ռեգիստրում գրանցված է [0, 15] միջակայքի թիվ: Ստանալ AL-ի 16-ական համարժեք թվանշանը:

```
    MOV     AL, 15
    LEA     BX, table
    XLATB                   ; AL←'F'
```

Որտեղ՝

```
table   DB      '0123456789ABCDEF'
```

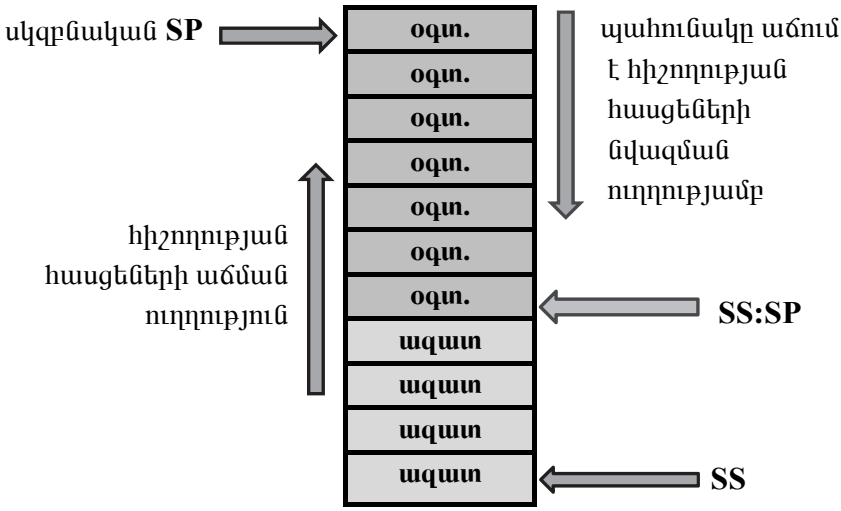
Օրինակ 2

```
    MOV     AL, 5
```

```
LEA     BX, table[2]    ; BX = table[2] = [table + 2] = 'C'-ի
                        ; հասցեն՝ 2
XLATB                   ; AL = [BX + AL] = 'H'
որտեղ.
  table  DB     'ABCDEFGH'
```

Պահունակի հետ ուղղակիորեն աշխատող հրամաններ (PUSH/POP/PUSHA/PUSHAD/ POPA/ POPAD)

Պահունակը հիշողության տիրույթ է, որն օգտագործվում է որպես տվյալների ժամանակավոր պահեստ: Պահունակի գագաթի հասցեն պահվում է SS:SP/ESP–ում: Պահունակի հետ աշխատանքն իրականացվում է (LIFO system- Last In First Out) սկզբունքով, այսինքն՝ պահունակից տարրերը հեռացվում են համալրմանը հակառակ հերթականությամբ: Սկզբնապես պահունակը տվյալ չի պարունակում: Տվյալները համալրվում և հեռացվում են պահունակից համապատասխանաբար PUSH/POP հրամանների միջոցով:



Նկար 2. Պահունակի իրականացումը հիշողության մեջ

Պահունակը ոչ բացահայտ կերպով օգտագործվում է ևան որոշ հրամանների (CALL, RET, INT, IRET) իրականացման ժամանակ:

PUSH – Գրել պահունակում /Push onto stack

Շարահյուսություն: **PUSH աղբյուր**

Նկարագրություն: պահունակ ← աղբյուր

Աղբյուրը կարող է լինել ռեգիստր (reg16/reg32), հիշողության հասցե (mem16/mem32) կամ սեգմենտային ռեգիստր, 286+ պրոցեսորների համար՝ ևան անմիջական արժեք (imm8/imm16, imm32 [386+]):

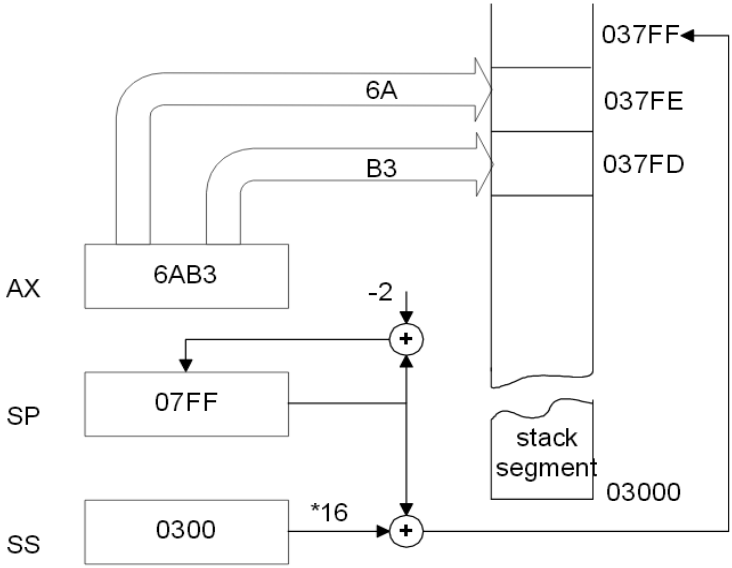
- PUSH հրամանը պակասեցնում է պահունակի ցուցիչի (SP/ESP) արժեքը 2-ով կամ 4-ով և գրանցում աղբյուրի արժեքը [SS:SP] կամ [SS:ESP] հասցեում՝ կախված սեգմենտի բիթայնությունից՝ 16 կամ 32,
- օպերանդի տիպը որոշում է պահունակի ցուցիչի նվազման չափը՝ 2 թե 4,
- 8086-ի PUSH SP հրամանի իրականացումը տարբերվում է 8086+ պրոցեսներում տվյալ հրամանի իրականացումից: 8086-ի դեպքում պահունակում գրանցվում է SP-ի նոր արժեքը, իսկ մյուս պրոցեսորների համար՝ SP-ի հին արժեքը:

Օրինակ

"PUSH AX" հրամանը կատարում է հետևյալը.

PUSH AX ==> SUB SP, 2 ; MOV [SS:SP], AX

Նկար 3-ում պատկերված է PUSH AX հրամանի իրականացումը.



Նկար 3. PUSH AX գործողության իրականացումը

POP – Կարդալ պահունակից / POP off of stack

Շարահյուսություն: **POP ընդունիչ**

Նկարագրություն: պահունակ → ընդունիչ

Ընդունիչը կարող է լինել ռեգիստր (reg16/reg32), հիշողության հասցե (mem16/mem32) կամ սեգմենտային ռեգիստր:

Այս հրամանը PUSH հրամանին հակառակ գործողությունն է:

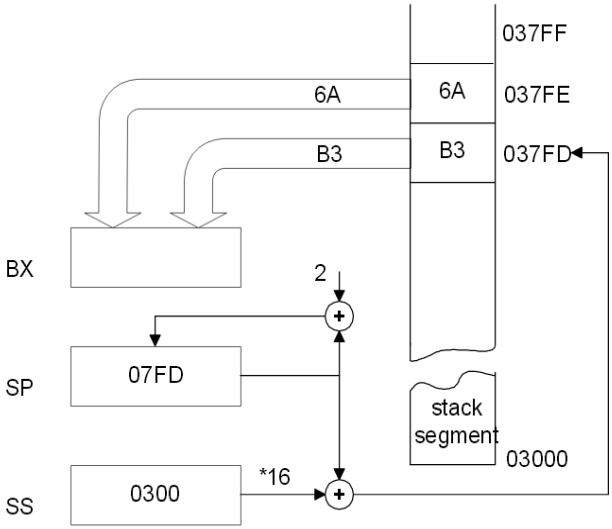
- POP հրամանը կարդում է տվյալը պահունակից և տեղադրում այն ընդունիչում,
- POP հրամանից հետո SP-ի արժեքն ավելանում է 2-ով կամ 4-ով՝ կախված օպերանդի չափից,
- CS ռեգիստրը չի կարող օգտագործվել որպես ընդունիչ:

Օրինակ

"POP BX" հրամանը համարժեք է հետևյալին.

POP BX ==> MOV BX, [SS:SP] ; ADD SP, 2

Նկար 4-ում պատկերված է "POP BX" հրամանի իրականացումը.



Նկար 4. POP BX գործողության իրականացումը

PUSHA/PUSHAD – Ընդհանուր օգտագործման ռեգիստրների արժեքները ուղարկել պահունակ (186+/386+)

Շարահյուսություն: **PUSHA/PUSHAD**

Նկարագրություն: պահունակ ← AX, CX, DX, BX, SP, BP, SI, DI /
պահունակ ←EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI

- PUSHA հրամանը AX, CX, DX, BX, SP, BP, SI և DI ընդհանուր օգտագործման ռեգիստրների արժեքները նշված հերթականությամբ ուղարկում է պահունակ: Պահունակի ցուցիչի արժեքը նվազում է 16-ով:
- PUSHAD հրամանը EAX, ECX, EDX, EBX, ESP, EBP, ESI և EDI ընդհանուր օգտագործման ռեգիստրների արժեքները

ձշված հերթականությամբ ուղարկում է պահունակ: Պահունակի ցուցիչի արժեքը ևվազում է 32-ով:

- Երկու դեպքում էլ SP և ESP պահունակում պահվում են իրենց նախնական արժեքով (նախքան PUSHA-ն):
- Նշենք, որ ռեգիստրների հերթականությունը համապատասխանում է նրանց կոդերի աճմանը

POPA/POPAD – Ընդհանուր օգտագործման ռեգիստրների վերարժեքավորել պահունակից (186+/386+)

Շարահյուսություն: **POPA/POPAD**

Նկարագրություն: պահունակ→ DI, SI, BP, ?, BX, DX, CX, AX /
պահունակ→ EDI, ESI, EBP, ?, EBX, EDX, ECX, EAX

- POPA հրամանը հաջորդաբար պահունակից հանում է 8 բառ՝ զետեղելով դրանք DI, SI, BP ռեգիստրներում, ոչ մի տեղ (SP-ին համապատասխանողը), BX, DX, CX և AX ռեգիստրներում համապատասխանաբար: Պահունակի ցուցիչի արժեքն աճում է 16-ով: Այն PUSHA-ի հակառակ գործողությունն է:
- POPAD հրամանը հաջորդաբար պահունակից հանում է 8 կրկնակի բառ՝ զետեղելով դրանք EDI, ESI, EBP, ոչ մի տեղ (ESP-ին համապատասխանողը), EBX, EDX, ECX, EAX ռեգիստրներում՝ համապատասխանաբար: Պահունակի ցուցիչի արժեքն աճում է 32-ով: POPAD-ն PUSHAD-ի հակառակ գործողությունն է:

Կայանի (port, պորտ) հետ աշխատող հրամաններ (IN/OUT)

IN - Կարդալ կայանից (port, պորտ) /Read from a port

Շարահյուսություն: **IN ընդունիչ, աղբյուր**

Նկարագրություն: ընդունիչ ← [աղբյուր]

Ընդունիչը AL/AX/EAX(386+) ռեգիստրն է, իսկ աղբյուրը կարող է լինել անմիջական արժեք (imm8) կամ DX ռեգիստրը: Աղբյուրը որոշում է կայանի (port, պորտ) համարը (թիվ [0, 255] միջակայքից կամ DX - ի արժեքը), որտեղից ինֆորմացիան գրվում է կուտակիչի մեջ:

Օրինակ

```
IN      AL, 0F8H        ; կարդում է 1 բայթ 0F8h համարի կա-
                        ; յանից և գրում AL-ի մեջ,
IN      AX, 0F9H        ; կարդում է 1 բառ 0F9h համարի կա-
                        ; յանից և գրում AX-ի մեջ,
MOV  DX, 0C800H
IN      AL, DX          ; կարդում է 1 բայթ DX-ով որոշվող կա-
                        ; յանից (0C800H համարով) և գրում
                        ; AL-ի մեջ,
IN      EAX, DX         ; կարդում է 1 կրկնակի բառ DX-ով
                        ; որոշվող կայանից (0C800H համա-
                        ; րով) և գրում EAX-ի մեջ:
```

OUT - Գրել կայանում (port, պորտ) /Write to a port

Շարահյուսություն: **OUT ընդունիչ, աղբյուր**

Նկարագրություն: [ընդունիչ] ← աղբյուր

Ընդունիչն անմիջական արժեք (imm8) է կամ DX ռեգիստրը, իսկ աղբյուրը AL/AX/EAX(386+) ռեգիստրն է: Ընդունիչը որոշում է կայանի (port, պորտ) համարը (թիվ [0, 255] միջակայքից կամ DX-ի արժեքը), որտեղ գրվում է կուտակիչի արժեքը:

Օրինակ

```
OUT   64H, AL         ; գրում է AL-ի արժեքը 64h համարի
                      ; կայանի մեջ,
OUT   60H, AX         ; գրում է AX-ի արժեքը 60h համարի
                      ; կայանի մեջ,
MOV  DX, 0D1H
OUT   DX, AL          ; գրում է AL-ի արժեքը DX-ով որոշ-
                      ; վող կայանի (00D1H համարով) մեջ,
OUT  DX, EAX          ; գրում է EAX-ի արժեքը DX-ով որոշ-
                      ; վող կայանի (00D1H համարով) մեջ:
```

Խնդիրներ և վարժություններ

1. Տրված է՝

```
A    DW    ?, ?
B    DW    ?, ?
X    DD    ?
Y    DD    ?
```

   A փոփոխականին վերագրել B փոփոխականի արժեքը (2 բառ), իսկ X փոփոխականին՝ Y-ի արժեքը:

2. Տրված է հետևյալ նկարագրությունը.

```
X    DB    ?
Y    DB    ?
```

   Փոխանակել X և Y փոփոխականների արժեքները:

3. Տրված ‘Z DW ?’ նկարագրության համար փոխանակել Z բառի բայթերը:

4. Տրված է հետևյալ նկարագրությունը.

```
A    DW    −73
B    DW    ?
```

   Հետևյալ հրամանների կատարումից հետո նշել A և B փոփոխականների արժեքները.

```
MOV     AX, A
MOV     BYTE  PTR  B, AH
MOV     BYTE  PTR  B + 1, AL
```

5. Տրված է հետևյալ նկարագրությունը.

```
B    DB    ?
W    DW    ?
```

   Նշել, թե հետևյալ հրամաններից որոնք են պարունակում շարահյուսական սխալ.

   1) MOV BP, SP 2) MOV AX, BL 3) MOV DX, '*'
   4) MOV CH, 500 5) MOV BX, B 6) MOV SI, W + 1
   7) MOV W, SI + 1 8) XCHG W, DS 9) MOV B, 80h

10) MOV B, W - B 11) MOV B, -130 12) MOV W, W + 2

13) XCHG AH, AL 14) XCHG W, ECX 15) XCHG B, B + 1

16) MOV B, BYTE PTR CX

Երկուական թվաբանության հրամաններ

Երկուական թվաբանության հրամանները կատարում են հիշողության և/կամ ընդհանուր օգտագործման ռեգիստրներում տեղադրված բայթ, բառ, կրկնակի բառ ամբողջ թվերի հետ երկուական թվաբանության հիմնական գործողությունները.

ADD	ամբողջ գումարում/Integer add,
ADC	փոխանցումով գումարում /Add with carry,
SUB	հանում/Subtract,
SBB	պարտքով հանում /Subtract with borrow,
IMUL	նշանով բազմապատկում/Signed multiply,
MUL	առանց նշանի բազմապատկում/Unsigned multiply,
IDIV	նշանով բաժանում/Signed divide,
DIV	առանց նշանի բաժանում/Unsigned divide,
INC	ավելացում /Increment,
DEC	նվազեցում /Decrement,
NEG	նշանափոխում/Negate,
CMP	համեմատում/Compare,
CMPXCHG	համեմատում և փոխանակում/Compare and exchange:

Թվաբանական հրամաններն ազդում են վիճակի դրոշների վրա: Յուրաքանչյուր հրամանի համար նշված են ազդեցության ենթարկվող դրոշների ցուցակը: Բոլոր հրամաններում շարահյուսությունը պահանջում է օպերանդների չափերի համապատասխանություն: Երկու օպերանդները միաժամանակ չեն կարող լինել հիշողության հասցեներ: Գործողության արդյունքը գրանցվում է առաջին օպե-

րանդում (բացի CMP և CMPXCHG հրամաններից), որը հրամանի նկարագրության մեջ նշված է որպես «ընդունիչ», իսկ երկրորդ օպերանդը նշված է որպես «աղբյուր»: Առաջին օպերանդը՝ ընդունիչը, չի կարող լինել անմիջական արժեք:

ADD – Գումարում

Շարահյուսություն:	**ADD ընդունիչ, աղբյուր**
Նկարագրություն:	ընդունիչ ← ընդունիչ + աղբյուր
Դրոշներ:	AF CF OF PF SF ZF

Ընդունիչը կարող է լինել ռեգիստր (reg8/reg16/reg32) կամ հիշողության հասցե (mem8/mem16/mem32), իսկ աղբյուրը՝ ռեգիստր (reg8/reg16/reg32), հիշողության հասցե (mem8/mem16/mem32) կամ անմիջական օպերանդ (im8/im16/im32): Հրամանն ազդում է բոլոր վիճակի դրոշների վրա:

Հրամանի օրինակներ՝

```
ADD   EBP, EAX
ADD   BYTE PTR [DI], 3
ADD   BX, [EAX + 2 * ECX]
```

Օրինակ

```
MOV   AL, 43          ; 43 = 00101011B
ADD   AL, 94          ; 94 = 01011110B
      00101011
      01011110
AL  = 10001001
```

Արդյունքում՝ CF = 0, OF = 1, SF = 1, AF = 1, PF = 0:

ADC – Փոխանցումով գումարում (Add With Carry)

Շարահյուսություն:	**ADC ընդունիչ, աղբյուր**
Նկարագրություն:	ընդունիչ ← ընդունիչ + աղբյուր + CF
Դրոշներ:	AF CF OF PF SF ZF

ADC-ն կարելի է օգտագործել երկար թվեր գումարելու համար: Ընդունիչը կարող է լինել ռեգիստր (reg8/reg16/reg32) կամ հիշողության հասցե (mem8/mem16/mem32), իսկ աղբյուրը՝ ռեգիստր (reg8/reg16/reg32), հիշողության հասցե (mem8/mem16/mem32) կամ անմիջական օպերանդ (im8/im16/im32):

Օրինակ

Ենթադրենք ցանկանում ենք կատարել 32-բիթանոց 2 թվերի գումարում 8086 – 80286 պրոցեսորների համար:

(SI, DX, CX) ← (DX, CX) + (BX, AX)

Գումարումը կկատարենք հետևյալ կերպ.

```
XOR   SI, SI
ADD   CX, AX
ADC   DX, BX
ADC   SI, 0
```

INC – Ավելացում/Increment

Շարահյուսություն:	**INC ընդունիչ**
Նկարագրություն:	ընդունիչ ← ընդունիչ + 1
Դրոշներ:	AF OF PF SF ZF

Ընդունիչը կարող է լինել ռեգիստր (reg8/reg16/reg32) կամ հիշողության հասցե (mem8/mem16/mem32):

Գործողությունը **չի ազդում փոխանցման դրոշի վրա** և կատարվում է ավելի արագ, քան **ADD ընդունիչ, 1** հրամանը:

Օրինակ

```
MOV   AL, 255
INC   AL        ; AL–ի արժեքը այժմ 0 է, CF-ը չի փոխվում:
```

SUB – Հանում/Subtract

Շարահյուսություն:	**SUB ընդունիչ, աղբյուր**
Նկարագրություն:	ընդունիչ ← ընդունիչ - աղբյուր
Դրոշներ:	AF CF OF PF SF ZF

Ընդունիչը կարող է լինել ռեգիստր (reg8/reg16/reg32) կամ հիշողության հասցե (mem8/mem16/mem32), իսկ աղբյուրը՝ ռեգիստր (reg8/reg16/reg32), հիշողության հասցե (mem8/mem16/mem32) կամ անմիջական օպերանդ (im8/im16/im32):

SBB – Պարտքով հանում /Subtract with Borrow/Carry

Շարահյուսություն:	**SUB ընդունիչ, աղբյուր**
Նկարագրություն:	ընդունիչ ← ընդունիչ - աղբյուր - CF
Դրոշներ:	AF CF OF PF SF ZF

Ընդունիչը կարող է լինել ռեգիստր (reg8/reg16/reg32) կամ հիշողության հասցե (mem8/mem16/mem32), իսկ աղբյուրը՝ ռեգիստր (reg8/reg16/reg32), հիշողության հասցե (mem8/mem16/mem32) կամ անմիջական օպերանդ (im8/im16/im32):

Օրինակ

Կատարել 32-բիթանոց թվերի հանում 8086 պրոցեսորի համար՝

(DX, CX) ← (DX, CX) – (BX, AX)

Հանումը կարող է կատարվել հետևյալ եղանակով.

```
SUB   CX, AX
SBB   DX, BX
```

DEC –նվազեցում/Decrement

Շարահյուսություն:	**DEC ընդունիչ**
Նկարագրություն:	ընդունիչ ← ընդունիչ - 1
Դրոշներ:	AF OF PF SF ZF

Ընդունիչը կարող է լինել ռեգիստր (reg8/reg16/reg32) կամ հիշողության հասցե (mem8/mem16/mem32): Գործողությունը **չի ազդում փոխանցման դրոշի վրա** և կատարվում է ավելի արագ, քան **SUB ընդունիչ, 1** հրամանը:

NEG –Նշանափոխություն/Two's Complement Negation

Շարահյուսություն:	**NEG ընդունիչ**
Նկարագրություն:	ընդունիչ ← 0 - ընդունիչ
Դրոշներ:	AF CF OF PF SF ZF

Ընդունիչը կարող է լինել ռեգիստր (reg8/reg16/reg32) կամ հիշողության հասցե (mem8/mem16/mem32), իսկ աղբյուրը՝ ռեգիստր (reg8/reg16/reg32), հիշողության հասցե (mem8/mem16/mem32) կամ անմիջական օպերանդ (im8/im16/im32): Այս հրամանն իմաստալի է միայն նշանով օպերանդների համար: Փոխանցման դրոշը միշտ ընդունում է 1 արժեք՝ բացառությամբ այն դեպքի, երբ ընդունիչը 0 է:

Օրինակ 1

հաշվել EAX-ի բացարձակ արժեքը.

```
    CMP   EAX, 0
    JGE   DONE
    NEG   EAX
DONE:
```

Նշում

հետևյալ հրամանների կատարումից հետո

```
    MOV   AL, -128      ; սա նշանով 8-բիթանոց ամենափոքր
                        ; արժեքն է,
    NEG   AL            ; գերհագեցման դրոշը՝ OF, դառնում է
                        ; 1, բայց AL-ի արժեքը չի փոխվում:
```

Օրինակ 2

հաշվել EAX–ի բացարձակ արժեքը.

```
L1: NEG   EAX
    JS    L1
```

Նշում

EAX = -2^{31} արժեքի դեպքում կստացվի անվերջ կրկնություն:

CMP –Համեմատում /Compare

Շարահյուսություն:	**CMP ընդունիչ, աղբյուր**
Նկարագրություն:	ընդունիչը համեմատում է աղբյուրի հետ
Դրոշներ:	AF CF OF PF SF ZF

Ընդունիչը կարող է լինել ռեգիստր (reg8/reg16/reg32) կամ հիշողության հասցե (mem8/mem16/mem32), իսկ աղբյուրը՝ ռեգիստր (reg8/reg16/reg32), հիշողության հասցե (mem8/mem16/mem32) կամ անմիջական օպերանդ (im8/im16/im32): Այս հրամանը համեմատում է ընդունիչի արժեքն աղբյուրի արժեքի հետ՝ առանց ընդունիչի արժեքը փոխելու: Համեմատման համար ընդունիչի արժեքից հանվում է աղբյուրի արժեքը, բայց արդյունքը ոչ մի տեղ չի պահվում: Հանման գործողության արդյունքում թարմացվում են դրոշները: Հրամանը կարող է օգտագործվել պայմանական անցման հրամաններից առաջ:

Օրինակ

```
MOV   DL, 27
CMP   DL, 17
JG    NEXT
```

CMPXCHG –Համեմատում և փոխանակում / Compare and exchange (486+)

Շարահյուսություն:	**CMPXCHG ընդունիչ, աղբյուր**
Դրոշներ:	AF CF OF PF SF ZF

Ընդունիչը կարող է լինել ռեգիստր (reg8/reg16/reg32) կամ հիշողության հասցե (mem8/mem16/mem32), իսկ աղբյուրը՝ միայն ռեգիստր (reg8/reg16/reg32): Այս հրամանը համեմատում է AL/AX/EAX ռեգիստրի (կախված օպերանդների չափից) արժեքը ընդունիչի արժեքի հետ: Եթե հավասար են, ապա աղբյուրի արժեքը պատճենվում է ընդունիչի մեջ և ZF դրոշն ընդունում է 1 արժեք: Հակառակ դեպքում ընդունիչի արժեքը պատճենվում է AL/AX/EAX ռեգիստրում, և ZF

դրոշն ընդունում է 0 արժեք: AF,CF,OF,PF,SF դրոշների արժեքները որոշվում են ըստ համեմատման գործողության (CMP) արդյունքի:

Օրինակ 1

```
MOV       DL, 27
MOV       AL, 15
MOV       CL, 3
CMPXCHG   DL, CL        ; AL = DL = 27, ZF = 0
```

Օրինակ 2

```
MOV       BX, 32
MOV       AX, 32
MOV       CX, 10
CMPXCHG   BX, CX        ; BX = CX = 10, ZF = 1
```

Բազմապատկման հրամաններ (MUL և IMUL)

Նշանով և առանց նշանի բազմապատկումը հանգեցնում է տարբեր արդյունքների:

Օրինակ

Դիտարկենք 10000000B * 11111111B:

Եթե մեկնաբանենք այս թվերը որպես առանց նշանի թվեր՝ արտադրյալը կլինի.

$$128 * 255 = 32640 = 0111\ 1111\ 1000\ 0000B:$$

Որպես նշանով թվեր մեկնաբանելու դեպքում արտադրյալը կլինի.

$$(-128) * (-1) = 128 = 0000\ 0000\ 1000\ 0000B.$$

Քանի որ նշանով և առանց նշանի բազմապատկումները հանգեցնում են տարբեր արդյունքների, կան երկու բազմապատկման հրամաններ՝ MUL և IMUL:

MUL –առանց նշանի բազմապատկում

Շարահյուսություն: **MUL աղբյուր**

Դրոշներ: AF = ? CF OF PF = ? SF = ? ZF = ?

IMUL –նշանով բազմապատկում

Շարահյուսություն: **IMUL աղբյուր**

Դրոշներ: AF = ? CF OF PF = ? SF = ? ZF = ?

Օպերանդներ	**Հրամանի շարահյուսությունը**	**Նկարագրություն**
8-բիթանոց	**MUL աղբյուր8** **IMUL աղբյուր8**	AX ← AL * աղբյուր8
16-բիթանոց	**MUL աղբյուր16** **IMUL աղբյուր16**	DX:AX←AX * աղբյուր16
32-բիթանոց	**MUL աղբյուր32** **IMUL աղբյուր32**	EDX:EAX←EAX * աղբյուր32

Որտեղ աղբյուր8/16/32-ը կա՛մ 8/16/32-բիթանոց ռեգիստր է, կա՛մ 8/16/32-բիթանի հիշողության օպերանդ:

IMUL հրամանի 3 (186+) կամ 2 (386+) օպերանդանոց տարբերակները

IMUL -նշանով բազմապատկում (386+)

Շարահյուսություն: **IMUL ընդունիչ, աղբյուր**

Դրոշներ: AF = ? CF OF PF = ? SF = ? ZF = ?

IMUL –առանց նշանի բազմապատկում (186+)

Շարահյուսություն: **IMUL ընդունիչ, աղբյուր1, աղբյուր2**

Դրոշներ: AF = ? CF OF PF = ? SF = ? ZF = ?

Օպերանդներ	Հրամանի շարահյուսությունը	Նկարագրություն
2 օպերանդ	IMUL reg16, reg16/mem16 IMUL reg32, reg32/mem32	ընդունիչ ← ընդունիչ * աղբյուր
3 օպերանդ	IMUL reg16, reg16/mem16, im8 IMUL reg16, reg16/mem16, im16 IMUL reg32, reg32/mem32, im8 IMUL reg32, reg32/mem32, im32	ընդունիչ ←աղբյուր1 * աղբյուր2

IMUL-ի այս տարբերակներում ռեգիստրային և հիշողության օպերանդները միշտ պետք է լինեն նույն չափի: Ռեգիստրները ընդհանուր օգտագործման ռեգիստրներ են:

Օրինակ

```
IMUL    ECX          ; EDX:EAX ← EAX * ECX
IMUL    BX, CX, 7    ; BX ← CX * 7
```

Նշում

Դրական թվերի բազմապատկման դեպքում MUL և IMUL հրամանները տալիս են նույն արդյունքը:

MUL և IMUL հրամաններից հետո վիճակի դրոշներն անորոշ են՝ բացառությամբ փոխանցման (CF) և գերհագեցման (OF) դրոշների: MUL գործողության դեպքում CF և OF դրոշները դառնում են 0, եթե արդյունքի ավագ կեսը (AH/DX/EDX-ը 8/16/32-բիթանոց օպերանդների դեպքում համապատասխանաբար) զրո է, հակառակ դեպքում OF-ը և CF-ը 1 են: Որոշ պրոցեսորներում SF դրոշն ընդունում է նշանի բիթի արժեքը:

Երկու կամ երեք օպերանդներով IMUL-ի դեպքում CF-ը և OF-ը զրո են, եթե բազմապատկման արդյունքը տեղավորվում է ընդունիչում, հակառակ դեպքում այդ դրոշները 1 են:

Նշում

Ի տարբերություն ADD, SUB, NEG, SHL կամ SAL գործողությունների գերհագեցման՝ MUL և IMUL գործողությունների համար OF-ի 1 լինելը չի տեղեկացնում հաշվումներում սխալի մասին:

Օրինակ

Ծրագրի հետևյալ հատվածներում որոշել բազմապատկման արդյունքը և պարզել՝ գերհագեցում տեղի կունենա, թե ոչ:

```
1. MOV     AL, 05H
   MOV     BL, 10H
   MUL     BL          ; AX = AL * BL = 0050H, CF = 0,
                       ; OF= 0, գերհագեցում չկա
2. .DATA
   VAR1    DW   2000H
   VAR2    DW   0010H
   . . .
   MOV     AX, VAR1
   MUL     VAR2        ; DX:AX = AX * VAR2 = 0002 0000H,
                       ; CF= 1, OF=1,
3. MOV     AL, 1
   MOV     BL, -1
   IMUL    BL          ; AX = -1 = 11111111 11111111B,
                       ; CF = 0, OF = 0, գերհագեցում չկա,
                       ; 11111111B 8-բիթանոց արդյունքը
                       ; նշանով ընդլայնվել է AH ռեգիստրում,
4. MOV     AX, 10
   MOV     CX, -48
   IMUL    CX          ; DX:AX = -480 = FFFF FE20H,
                       ; CF = 0, OF = 0, գերհագեցում չկա,
                       ; FE20H 16-բիթանոց արդյունքը նշա-
                       ; նով ընդլայնվել է DX ռեգիստրում,
```

Օրինակ

Եենթադրելով, որ W և Y-ը բառային նշանով փոփոխականներ են, հաշվել W = 5 * W - 12 * Y արտահայտության արժեքը՝ գերհագեցման դեպքում անցում կատարելով OVERFLOW նշիչին.

```
        MOV     AX, 5
        IMUL    W               ; DX:AX = 5 * W
        JO      OVERFLOW
        MOV     W, AX           ; W = 5 * W
        MOV     AX, 12
        IMUL    Y               ; DX:AX = 12 * Y
        JO      OVERFLOW
        SUB     W, AX           ; W = 5 * W - 12 * Y
        JO      OVERFLOW
        . . .
        JMP     DONE
OVERFLOW:
        . . .
DONE:
```

Բաժանման հրամաններ (DIV և IDIV)

Բաժանման հրամանները նույնպես երկուսն են՝ DIV և IDIV: DIV հրամանն օգտագործվում է առանց նշանի ամբողջ բաժանման, իսկ IDIV հրամանը՝ նշանով ամբողջ բաժանման դեպքում:

DIV –առանց նշանի բաժանում

Շարահյուսություն:	**DIV աղբյուր**
Դրոշներ:	AF = ? CF = ? OF = ? PF = ? SF = ? ZF = ?

IDIV –նշանով բաժանում

Շարահյուսություն:	**IDIV աղբյուր**
Դրոշներ:	AF = ? CF = ? OF = ? PF = ? SF = ? ZF = ?

Ամբողջ բաժանման արդյունքում ստացվում է քանորդ և մնացորդ:

Օպերանդներ	Նկարագրություն	Քանորդ	Մնացորդ
8-բիթանոց	$\frac{\text{AX}}{\text{աղբյուր8}}$	AL	AH
16-բիթանոց	$\frac{\text{DX: AX}}{\text{աղբյուր16}}$	AX	DX
32-բիթանոց	$\frac{\text{EDX: EAX}}{\text{աղբյուր32}}$	EAX	EDX

Օպերանդ8/16/32-ը 8/16/32-բիթանոց ռեգիստր է կամ հիշողության հասցե: Բաժանման հրամանների կատարման արդյունքում SF, PF, CF, OF, AF և ZF դրոշների արժեքներն անորոշ են:

Նշում

- Եթե բաժանելին և բաժանարարը դրական են, DIV և IDIV տալիս են նույն արդյունքը:
- Նշանով բաժանման դեպքում մնացորդը (եթե զրո չէ) ունի նույն նշանը, ինչ բաժանելին:
- Եթե բաժանման ժամանակ բաժանարարը զրո է կամ քանորդը մեծ է և չի տեղավորվում քանորդի համար նախատեսված ռեգիստրում, ապա տեղի է ունենում **բաժանման գերհագեցում/Division overflow**: Պրոցեսորն ավտոմատ կանչում է 00H ընդհատումը (զրոյի վրա բաժանում (#DE (divide error/by zero)), և ծրագիրն ավարտվում է:

AH/DX/EDX ռեգիստրի արժեքը պետք է զրոյացվի մինչև 8/16/32-բիթանոց առանց նշանի բաժանումը.

1. DIV հրամանի օգտագործման դեպքում AH ռեգիստրը, բնականաբար, պետք է զրոյացվի, եթե բաժանելին բայթ է:

Օրինակ

```
MOV     AL, BYTE_VAR
MOV     AH, 0       ; AH-ը պետք է զրոյացվի
MOV     BL, 5
DIV     BL
```

Օրինակ

```
MOV     AX, 20      ; AH-ի զրոյացում պետք չէ
MOV     CL, 4
DIV     CL
```

Օրինակ

կարդալ ASCII թվանշան, ձևափոխել այն թվային արժեքի, ապա բաժանել 3-ի.

```
MOV     AH, 01H
INT     21H
SUB     AL, 30H
MOV     AH, 0       ; AH-ը զրոյացնել
MOV     BL, 3
DIV     BL
```

Նշում

AH-ը 0-ով սկզբնարժեքավորելու համար կարելի է օգտագործել MOVZX (Move with Zero Extension) հրամանը.

```
MOVZX   AX, BYTE_VAR
MOV     BL, 5
DIV     BL
```

Օրինակ

DX ռեգիստրի սկզբնարժեքավորումը մինչ IDIV հրամանի օգտագործումը.

```
MOV     AX, -12     ; AX =  FFF4H
CWD                 ; DX:AX =  FFFF FFF4H
```

```
MOV     BX, 7
IDIV    BX          ; AX = -1 = FFFFH,
                    ; DX = -5 = FFFBH
```

Օրինակ

EDX ռեգիստրի սկզբնարժեքավորումը մինչ DIV հրամանի օգտագործումը.

```
.DATA
   DIVIDEND   DWORD    8003FFBCH
   DIVISOR    DWORD    10000000H
   . . .
   MOV        EDX,  0
   MOV        EAX, DIVIDEND
   DIV        DIVISOR
   . . .
```

Օրինակ

EDX ռեգիստրի սկզբնարժեքավորումը մինչ IDIV հրամանի օգտագործումը.

```
   MOV        EAX, 8FBA2FECh
   CDQ
   MOV        EBX, 2FFFFFFFh
   IDIV       EBX
```

Օրինակ

գերհագեցում բաժանման ժամանակ կամ զրոյի վրա բաժանում (Division overflow).

```
   MOV        AX, 0200H
   MOV        BL, 02H
   DIV        BL
```

Հրամանների հաջորդականությունը կհանգեցնի բաժանման գերհագեցման, քանի որ BL ≤ AH: Քանորդն այս դեպքում 100H է, այսինքն՝ 256, որն ավելին է, քան 255-ը (առավելագույն առանց նշանի արժեքը):

Եթե AH-ը զրոյացվի նախքան բաժանումը, ապա գերհագեցում տեղի կունենա, միայն եթե բաժանարարը լինի զրո:

8-բիթանոց առանց նշանի բաժանում-գերհագեցումից խուսափելու համար կարելի է կատարել հետևյալ ստուգումները.

```
                CMP     Divisor8, AH
                JBE     DIV_OVERFLOW
                DIV     Divisor8
                . . .
                JMP     DONE
DIV_OVERFLOW:
                . . .
DONE:
```

8-բիթանոց նշանով բաժանում-գերհագեցումից խուսափելու համար կարելի է կատարել հետևյալ ստուգումները.

```
                MOV     AL, Dividend
                CBW
                CMP     Divisor8, 0
                JE      IDIV_OVERFLOW
                CMP     Divisor8, -1
                JNE     L1
                CMP     AL, -128
                JNE     L1
                JMP     IDIV_OVERFLOW
        L1:     IDIV    Divisor8
                . . .
                JMP     DONE
IDIV_OVERFLOW:
                . . .
DONE:
```

Խնդիրների լուծման օրինակներ

1. Հաշվել հետևյալ արտահայտության արժեքը.

$$X = \frac{A * 2 + B * C}{D - 3};$$

Բոլոր փոփոխականները 16-բիթանոց նշանով թվեր են: Ենթադրել, որ գերհագեցում չի առաջանում:

```
.MODEL SMALL
.DATA
        X       DW      ?
        A       DW      5
        B       DW      -10
        C       DW      4
        D       DW      3
.CODE
        MOV             AX, @DATA
        MOV             DS, AX
        MOV             AX, 2
        IMUL            A               ; DX:AX = A * 2
        MOV             BX, DX
        MOV             CX, AX          ; BX:CX = A * 2
        MOV             AX, B
        IMUL            C               ; DX:AX = B * C
        ADD             AX, CX
        ADC             DX, BX          ; DX:AX = A * 2  +  B * 2
        MOV             CX, D
        SUB             CX, 3           ; CX = D – 3
        IDIV             CX             ; AX = (A * 2 + B * C) / (D – 3),
                                        ; DX-ում մնացորդը
        MOV             X, AX
        MOV             AH, 4Ch
        INT             21h
END
```

2. Բառերի զանգվածի տարրերի գումարում.

```
.MODEL SMALL
.DATA
        Array   DW      1234h, 5678h, 9ABCh, 0DEF0h, 0AA11h
        Count   DW       0005h
        Result  DW      0000h, 0000h
.CODE
         MOV            AX, @DATA
         MOV            DS, AX
         MOV            DX, 00h
         MOV            CX, Count
         LEA            SI, Array
         MOV            BX, [SI]
NEXT:    INC            SI
         INC            SI
         ADD            BX, [SI]
         ADC            DX, 0
         LOOP           NEXT
         MOV            Result, BX
         MOV            Result + 2, DX
         MOV            AH, 4Ch
         INT            21h
END
```

Խնդիրներ և վարժություններ

1. Նշել AL ռեգիստրի որևէ սկզբնական արժեք, որի դեպքում ADD AL, 2 հրամանի կատարումից հետո դրոշները կունենան հետևյալ արժեքները.

   ա) CF = 1, OF = 0, SF = 0; բ) CF = 0, OF = 0, SF = 1:

2. Որոշել AL ռեգիստրի (առանց նշանի տասական թվի տեսքով) և CF, ZF դրոշների արժեքները հետևյալ հրամանների կատարումից հետո.

ա) MOV AL, 100
ADD AL, 100

բ) MOV AL, 100
ADD AL, 156

գ) MOV AL, 100
SUB AL, 90

դ) MOV AL, 100
SUB AL, 190

3. Որոշել BH ռեգիստրի (նշանով տասական թվի տեսքով) և OF, SF դրոշների արժեքները հետևյալ հրամանների կատարումից հետո.

ա) MOV BH, 80
ADD BH, 40

բ) MOV BH, 90
ADD BH, 40

գ) MOV BH, -80
ADD BH, -40

դ) MOV BH, -80
ADD BH, -50

ե) MOV BH, 80
ADD BH, -40

զ) MOV BH, -80
ADD BH, 40

է) MOV BH,80
SUB BH,100

ը) MOV BH, -80
SUB BH, 50

թ) MOV BH, 125
ADD BH, 10

4. Որոշել CL ռեգիստրի (ինչպես նշանով, այնպես էլ առանց նշանի տասական թվի տեսքով) և CF,OF, SF,ZF դրոշների արժեքները հետևյալ հրամանների կատարումից հետո.

ա) MOV CL, 128
ADD CL, 128

բ) MOV CL, -10
ADD CL, -40

գ) MOV CL, 246
ADD CL, 216

դ) MOV CL, 40
SUB CL, 100

5. Որոշել AH և AL ռեգիստրների (առանց նշանի տասական թվի տեսքով) և CF, ZF դրոշների արժեքները հետևյալ հրամանների կատարումից հետո.

ա)
```
MOV AH, 0
MOV  AL, 160
ADD  AL, 60
ADC  AH, 3
```

բ)
```
MOV AH, 0
MOV AL, 160
ADD  AL, 160
ADC  AH, 3
```

գ)
```
MOV AH, 255
MOV  AL, 255
ADD  AL, 1
ADC  AH, 0
```

դ)
```
MOV AH, 20
MOV AL, 10
SUB  AL, 16
SBB  AH, 0
```

6. Նշել բոլոր օգտագործված ռեգիստրների և վիճակի դրոշների արժեքները հետևյալ հրամանների կատարումից հետո.

ա)
```
MOV AL, 0
MOV  DX, 1283
ADD  AL, DL
NEG  DL
MUL  DH
SBB   DL,AL
```

բ)
```
SUB   DL, DL
MOV CL, 3
MOV AX, 155
DIV   CL
ADC  DL, AL
DEC  AH
```

գ)
```
MOV  DH, -33
MOV  AL, DH
IMUL DH
ADC  DH, AH
INC    DH
SUB   AL, DH
```

դ)
```
MOV  AX, 6
MOV  BX, -4
IMUL BL
SBB   AX, BX
ADD  BX, AX
NEG   BH
```

ե)
```
MOV AX, 275
MOV CX, 7
IMUL CL
ADC  AH, CL
SUB  CX, AX
INC   CX
```

զ)
```
MOV BX, 260
MOV AX, -21
IDIV  BL
NEG  AH
SBB   AL, 5
ADD   AH, BH
```

է)
```
MOV  AX, 210
MOV  CX, 10
MUL  CX
ADC   DX, CX
SUB   CL, AL
DEC   AH
```

ը)
```
MOV AX, 1060
MOV BX, -2
IDIV  BL
SBB   BL, AH
INC   AL
NEG   BL
```

թ)
```
MOV AX, 1235h
MOV  BX, 720h
MUL  BH
ADC  AH, BL
SUB   BX, AX
DEC   AX
```

7. Որոշել ADD AX, BX և SUB AX, BX հրամանների կատարման արդյունքը, եթե EAX-ը և EBX-ը համապատասխանաբար պարունակում են.
   ա) 7FFAF15BH, EF012AF1H
   բ) 0102000BH, FFFFEEFFH
   գ) 0000FFFFH, AB001111H
8. Որոշել MUL BH և IMUL BH հրամանների կատարման արդյունքը, եթե AL և BH ռեգիստրները համապատասխանաբար պարունակում են.
   ա) 01H, 0FFH
   բ) 0F3H, 04H
   գ) 0FFH, 0FFH
   դ) 10H, 2AH
9. Որոշել DIV CH և IDIV CH հրամանների կատարման արդյունքը, եթե AX և CH ռեգիստրները համապատասխանաբար պարունակում են.
   ա) 0FFFFH, 0FFH
   բ) 8000H, 7FH
   գ) 0375H, 0A0H
   դ) 0600H, 05H
10. A, B, C, X, Y, Z-ը ձշանով թվեր են: Իրականացնել հետևյալ վերագրումները.
    ա) A = ABS (X) ,
    բ) B = MAX (X, Y),
    գ) C = MIN (X + 5, 3 * Y, 10 - Z)
11. Տրված է N թիվը: Հաշվել 1-ից N թվերի գումարը:
12. Տրված է՝

```
K       DW      ?       ; K ≥ 0 :
```

DL ռեգիստրում ստանալ.
   ա) K թվի տասական ներկայացման իմաստալից թվանշանների քանակը,

բ) K թվի տասական ներկայացման մեջ 5-ից մեծ թվանշանների գումարը,

գ) K թվի տասական ներկայացման մեջ ամենամեծ թվանշանը:

13. Տրված է՝

K DW ? ; K > 0:

DL ռեգիստրում գրանցել K թվի տասական ներկայացման բարձր կարգի թվանշանը:

14. Հաշվել հետևյալ արտահայտությունների արժեքները՝ ենթադրելով, որ արտահայտության մեջ օգտագործված փոփոխականները նշանով/առանց նշանի բայթեր/բառեր են և նրանց արժեքները որոշված են տվյալների սեգմենտում.

Z = (25 * a – 32 * b) / (14 * c + 29)

R = (24 * c – 55 * d - 1) / (13 * a + 17)

T = -(39 * c + 14 * d) / (29 * a - 1)

N = 31 + (46 * h – 4 * j) / 82 * c

N = (45 * c – 22 * d) / (13 * a + 1)

D = (33 * f + 17 * h) / 4 * k – 25

N = -(87 * s + 31 * d) / (56 * a - 3)

Q = (18 * k + 15 * v + 1) / (42 * p - 23)

$X = (32 * a^3 – 3* b^5) / (b^3 – 4 * a^2)$

$Y = (4 * a^5 – 3* a * b^3) / (5 * b^2 – 9 * a)$

$Z = (7 * b * a^2 – 3 * a * b^3) / (8 * a * b^2 – a^3)$

$W = (12 * a^6 – 5 * b^3) / (2 * b – 8 * a^2)$

$M = (2 * c * a^3 – 19 * b^3) / (b^4 – 6 * c + a^2)$

$P = (9 * c * a^3 – 4 * a * b^3) / (c * b^3 + b * a^2)$

$F = (12 * a^4 – 7 * b^2)/(b^4 – 3 * a^2)$

$G = (b * a^2 – 25 * a * c * b^3) / (9 * b^3 – c * a^2)$

$S = (2 * c * b * a^3 – 13 * a * c * b^3) / (3 * b^3 – 9 * c * a^2)$

$T = (7 * a^7 – 3 * b^2) / (b^2 – 10 * a^2)$

15. Գրել ծրագրի հատված, որը համարժեք կլինի C++ լեզվով գրված հետևյալ ծրագրի հատվածին (անհրաժեշտ փոփոխականները հայտարարել տվյալների սեգմենտում).

ա)
```
int i, n, s = 1;
n = 50;  i = 10;
s += i * i - n;
```

բ)
```
int s, n;
s = 0; n = 4325;
s -= n % 8;
n /= 10;
```

գ)
```
int k = 11, n;
n = 2157;
n /= 10;
k += n % 15;
```

Տասական թվաբանության հրամաններ

Այս խմբի հրամաններն օգտագործվում են տասական թվերի հետ աշխատելու համար: Նրանք կատարում են ճշգրտումներ (շրջկումներ) երկուական թվաբանության հրամաններից **հետո** (բացառություն է կազմում բաժանում գործողությունը): Հրամանները չունեն բացահայտ օպերանդ: Ոչ բացահայտ օպերանդ է դիտարկվում AL(AX) ռեգիստրը: Գումարում և հանում գործողությունների ճշգրտումները նախատեսված են 10-ական և՛ փաթեթավորված, և՛ ոչ փաթեթավորված թվերի համար, իսկ բաժանումը և բազմապատկումը՝ միայն ոչ փաթեթավորված թվերի համար:

DAA Գումարումից հետո տասական ճշգրտում /
Decimal adjust after addition

DAS Հանումից հետո տասական ճշգրտում /
Decimal adjust after subtraction

AAA Գումարումից հետո ASCII ճշգրտում /
ASCII adjust after addition

AAS Հանումից հետո ASCII ճշգրտում /
ASCII adjust after subtraction

AAM Բազմապատկումից հետո ASCII ճշգրտում /
ASCII adjust after multiplication

AAD Բաժանումից առաջ ASCII ճշգրտում /
ASCII adjust before division

AAA – Գումարումից հետո տասական ճշգրտում ոչ փաթեթավորված տասական թվերի համար / ASCII Adjust after Addition

Շարահյուսություն: **AAA**

Նկարագրություն: Եթե ((AL & 0FH) > 9) || (AF = 1), ապա

{

AL ← (AL + 6);

AH ← AH + 1;

AF ← 1;

CF ← 1;

}

Հակառակ դեպքում՝

{

AF ← 0;

CF ← 0;

}

AL ← AL & 0FH;

Դրոշներ: AF CF OF = ? PF = ? SF = ? ZF = ?

AF և CF դրոշների արժեքները որոշվում են ըստ նկարագրության: OF, SF, ZF, PF դրոշների արժեքներն անորոշ են:

Օրինակ

```
MOV   AH, 0
MOV   AL, ‘5’
MOV   BL, ‘7’
ADD   AL, BL        ; AL = 6Ch, AF = 0
AAA                 ; AL = 02h, CF = 1, AF = 1, AH = 1
```

AAS – Հանումից հետո տասական ճշգրտում ոչ փաթեթավորված տասական թվերի համար /ASCII Adjust AL after Subtraction

Շարահյուսությունը: **AAS**

Նկարագրությունը: Եթե ((AL & 0FH) > 9) || (AF = 1), ապա

{

AL ← (AL - 6);

AH ← AH - 1;

AF ← 1;

CF ← 1;
}
Հակառակ դեպքում
{
AF ← 0;
CF ← 0;
}
AL ← AL & 0FH;

Դրոշներ: AF CF OF = ? PF = ? SF = ? ZF = ?

AF և CF դրոշների արժեքները որոշվում են ըստ նկարագրության: OF, SF, ZF, PF դրոշների արժեքներն անորոշ են:

Օրինակ

```
SUB   AL, BL
AAS
```

Նախքան SUB հրամանի կատարումը	SUB հրամանից հետո	AAS հրամանից հետո
AX = 00 08h BL = 03	AX = 00 05h AF = 0	AX = 00 05h AF = 0, CF = 0
AX = 00 03h BL = 07	AX = 00 FCh AF = 1	AX = FF 06h AF = 1, CF = 1
AX = 05 02h BL = 09	AX = 05 F9h AF = 1	AX = 04 03h AF = 1, CF = 1

DAA– Գումարումից հետո տասական ճշգրտում փաթեթավորած տասական թվերի համար / Decimal Adjust AL after Addition

Շարահյուսություն: **DAA**

Նկարագրություն: Եթե (((AL AND 0FH) > 9) կամ AF = 1), ապա
{
AL ← AL + 6;
CF ← CF կամ փոխանցում վերջին գումարումից՝
AL + 6 - ից հետո (CF OR carry from AL ← AL + 6)
AF ← 1;
}
Հակառակ դեպքում՝
AF ← 0;

Եթե ((AL AND F0H) > 90H) կամ CF = 1), ապա
{
AL ← AL + 60H;
CF ← 1;
}
Հակառակ դեպքում՝
CF ← 0;

Դրոշներ: AF CF OF = ? PF SF ZF

AF և CF դրոշների արժեքները որոշվում են ըստ նկարագրության: SF, ZF, և PF դրոշներն արժեքներ են ընդունում ըստ արդյունքի: OF դրոշի արժեքն անորոշ է:

Օրինակ

```
ADD     AL, BL
DAA
```

Նախքան ADD հրամանի կատարումը	ADD հրամանից հետո	DAA հրամանից հետո
AL = 34h BL = 25h	AL = 59h AF = 0, CF = 0	AL = 59h AF =0, CF = 0
AL = 37h BL = 25h	AL= 5Ch AF= 0, CF = 0	AL = 62h AF =1, CF = 0
AL = 93h BL = 25h	AL = B8h AF = 0, CF = 0	AL = 18h AF = 0, CF = 1
AL = 28h BL = 39h	AL = 61h AF = 1, CF = 0	AL = 67h AF = 1, CF = 0
AL = 79h BL = 99h	AL = 12h AF = 1, CF = 1	AL = 78h AF = 1, CF = 1

Օրինակ

```
1.MOV     AL, 53h
  MOV     CL, 29h
  ADD     AL, CL        ; AL ← AL + CL
                        ; AL ← 53h + 29h = 7Ch
  DAA                   ; AL ← 7Ch + 06 ( քանի որ 0Ch > 9 )
                        ; AL ← 82h
```

```
2. MOV   AL, 73h
   MOV   CL, 29h
   ADD   AL, CL        ; AL ← AL + CL
                       ; AL ← 73h + 29h = 9Ch
   DAA                 ; AL ← 02 և CF = 1
```

```
  AL = 73
     +
  CL = 29
       9C
        +
       6
      A2
       +
      60
```

AL = 02 և CF = 1

DAS – Հանումից հետո տասական ճշգրտում փաթեթավորված տասական թվերի համար - Decimal Adjust AL after Subtraction

Շարահյուսություն: **DAS**

Նկարագրություն: Եթե (AL AND 0FH) > 9 || AF ← 1), ապա
{
AL ← AL − 6;
CF ← CF կամ պարտք վերջին հանումից՝ AL-6-ից հետո
(CF OR borrow from AL ← AL − 6)
AF ← 1;
}
Հակառակ դեպքում՝
AF ← 0;
Եթե ((AL AND F0h) > 90h) կամ CF = 1), ապա
{
AL ← AL − 60H;
CF ← 1;
}

Հակառակ դեպքում՝
CF ← 0;

Դրոշներ: AF CF OF = ? PF SF ZF

AF և CF դրոշների արժեքները որոշվում են ըստ նկարագրության: SF, ZF, և PF դրոշներն արժեքներ են ընդունում ըստ արդյունքի: OF դրոշի արժեքն անորոշ է:

Օրինակ

```
SUB     AL, BL
DAS
```

Նախքան SUB հրամանի կատարումը	SUB հրամանից հետո	DAS հրամանից հետո
AL = 25h BL = 69h	AL = BCh AF = 1, CF = 1	AL = 56h AF = 1, CF = 1
AL = 37h BL = 25h	AL = 12h AF = 0, CF = 0	AL = 12h AF = 0, CF = 0
AL = 93h BL = 25h	AL = 6Eh AF = 1, CF = 0	AL = 68h AF =1, CF = 0
AL = 92h BL = 39h	AL = 59h AF = 1, CF = 0	AL = 53h AF =1, CF = 0
AL = 79h BL = 95h	AL = E4h AF = 0, CF = 1	AL = 84h AF =0, CF = 1

Օրինակ

AL ռեգիստրում գրված 16-ական թվանշանը ձևափոխել իրեն համապատասխան սիմվոլի ASCII կոդի.

```
CMP     AL, 10
SBB     AL, 69h
DAS
```

AAD – Բաժանումից առաջ ճշգրտում տասական ոչ փաթեթավորված թվերի համար / ASCII Adjust AX before Division

Շարահյուսություն: AAD
Նկարագրություն: AL ← (AL + AH ∗ 10) AND FFH, AH ← 0
Դրոշներ: AF = ? CF = ? OF = ? PF SF ZF

SF, ZF և PF դրոշներն արժեք են ստանում ըստ AL ռեգիստրի արժեքի, AF, CF և OF դրոշներն անորոշ են:

Օրինակ

```
AAD
DIV     DH
```

Նախքան AAD հրամանի կատարումը	AAD հրամանից հետո	DIV հրամանից հետո
AX = 07 05h DH = 08h	AX = 00 4Bh DH = 08h	AX = 03 09h
AX = 06 02h DH = 04h	AX = 00 3Eh DH = 04h	AX = 02 0Fh
AX = 09 03h DH = 02h	AX = 00 5Dh DH = 02h	AX = 01 2Eh

AAM – Բազմապատումից հետո ճշգրտում տասական ոչ փաթեթավորված թվերի համար / ASCII Adjust AX after Multiply

Շարահյուսություն: **AAM**
Նկարագրություն: AH ← AL / 10; AL ← AL MOD 10;
Դրոշներ: AF = ? CF = ? OF = ? PF SF ZF

SF, ZF և PF դրոշներն արժեք են ստանում ըստ AL ռեգիստրի արժեքի: AF, CF և OF դրոշներն անորոշ են:

Օրինակ

```
MUL    BH
AAM
```

Նախքան MUL հրամանի կատարումը	MUL հրամանից հետո	AAM հրամանից հետո
AX = 00 09h BH = 09h	AX = 00 51h	AX = 08 01h
AX = 00 05h BH = 06h	AX = 00 1Eh	AX = 03 00h
AX = 00 06h DH = 07h	AX = 00 2Ah DH = 02h	AX = 04 02h

Խնդիրների լուծման օրինակներ

1. Գումարել տրված A և B 20 սիմվոլից բաղկացած 10-ական թվերը և արդյունքն արտածել 10- ական համակարգով.

```
.MODEL SMALL
.DATA
        N       EQU     20
        A       DB      “90822346771463283930”
        B       DB      “21839839390913214263”
        C       DB      21 dup(?)
.CODE
Main:   MOV             AX, @DATA
        MOV             DS, AX
        MOV             SI, N-1
        MOV             CX, N
        CLC
Sum:    MOV             AL, A[SI]       ; AL ← A թվի հերթական
                                        ; թվանշանը,
        ADC             AL, B[SI]       ; AL-ի կրտսեր 4 բիթում A և B
                                        ; թվերի հերթական թվանշան-
                                        ; ների տասական  գումարն է,
                                        ; CF = 1` եթե կա փոխանցում
                                        ; (այսինքն` 9 + 2 = 1 և CF = 1),
        AAA                             ; AL ← A և B թվերի հերթական
                                        ; թվանշանների տասական
                                        ; գումարը:
```

```
        MOV     C[SI + 1], AL
        DEC     SI
        LOOP    sum
        MOV     C, 0
        ADC     C, 0
; Արտածել գումարը
        MOV     CX, N + 1
        MOV     AH, 2
        MOV     SI, 0
Print:  MOV     DL, C[SI]
        ADD     DL, '0'
        INT     21h
        INC     SI
        LOOP    print
        MOV     AH, 4Ch
        INT     21h
END     main
```

2. Բազմապատկել N-նիշանոց X ոչ փաթեթավորված տասական թիվը միանիշ Y թվի հետ և արդյունքը գրանցել Z-ում:

```
    N   EQU   …                 ; N > 0
    X   DB    N DUP(?)
    Y   DB    ?                 ; 0-ից 9
    Z   DB    ?, N DUP (?)      ; առաջին բայթը՝ փոխանցման
                                ; համար
    …
        MOV     CX, N
        MOV     SI, N - 1       ; X-ի ինդեքսը (աջից ձախ)
        MOV     BH, 0
mlt:    MOV     AL, X[SI]
        MUL     Y
        AAM
```

```
        ADD     AL, BH      ; հաշվի առնել կրտսեր կարգից
                            ; եկած փոխանցումը
        AAA
        MOV     Z[SI + 1], AL
        MOV     BH, AH      ; հիշել փոխանցումը
        DEC     SI
        LOOP    mlt
        MOV     Z, BH       ; արտադրյալի ավագ թվանշանը
```

3. Բաժանել N-նիշանոց X ոչ փաթեթավորված տասական թիվը միանիշ Y թվի վրա՝ գրանցելով ստացված քանորդը Q-ում, իսկ մնացորդը՝ R-ում:

```
N   EQU   …                 ; N > 0
X   DB    N DUP(?)
Y   DB    ?                 ; 0-ից 9
Q   DB    N DUP (?)         ; X div Y
R   DB    ?                 ; X mod Y
    …
    MOV     CX, N
    MOV     SI, 0
    MOV     AH, 0
DV: MOV     AL, X[SI]
    AAD                     ; AX-ում՝ AH և AL թվանշան-
                            ; ներից ստացված թիվը
    MOV     Q[SI], AL
    INC     SI
    LOOP    DV
    MOV     R, AH           ; X-ը Y–ի վրա բաժանելուց
                            ; ստացված մնացորդը
```

Խնդիրներ և վարժություններ

1. Ի՞նչ արժեքներ կունենան AX ռեգիստրը և CF դրոշը հետևյալ հրամանների կատարումից հետո.

```
MOV   AX, 0039h
ADD   AL, 44h
DAA
```

2. Կազմել ծրագիր, որը ներածի տասական կետից հետո N = 20 տասական թվանշաններից կազմված տասական համակարգի թվի կոտորակային մասը, այն թարգմանի 8-ական համակարգի և արտածի ստացված թիվը:
3. Կազմել ծրագիր, որը ներածի 7 թվանշաններից կազմված 10-ական համակարգի թիվ, թարգմանի այն 8-ական համակարգի և արտածի:
4. Ներածել 20 սիմվոլից բաղկացած երկու տասական թիվ, կատարել հանում գործողություն և արդյունքն արտածել:
5. Կազմել ծրագիր, որը հաշվում է հետևյալ արտահայտության արժեքը տասական տեսքով.

   Y = (3 * A + 5 * B) / (2 * C - 7), որտեղ A = 5, B = 3, C = 6:
6. Կազմել ծրագիր, որը ներածում է տասական համակարգի ութանիշ ամբողջ թիվ, թարգմանում է այն վեցական համակարգի և արտածում:

Տրամաբանական հրամաններ

Տրամաբանական հրամանները բայթերի/բառերի/կրկնակի բառերի համար կատարում են հիմնական AND, OR, XOR, և NOT տրամաբանական գործողությունները.

AND կատարել բիթ առ բիթ տրամաբանական “և” (&),

OR կատարել բիթ առ բիթ տրամաբանական “կամ” (|),

XOR կատարել բիթ առ բիթ տրամաբանական “բացառող կամ” (^),

NOT կատարել բիթ առ բիթ տրամաբանական “ժխտում” (!),
TEST կատարել բիթ առ բիթ տրամաբանական ստուգում (&):

Տրամաբանական հրամաններով կարելի է առաջին օպերանդի՝ ընդունիչի որոշակի բիթեր դարձնել որոշակի արժեքներ: Օպերանդներն այս հրամաններում նույնպես պետք է բավարարեն ընդհանուր սահմանափակումներին (ինչպես MOV, ADD հրամաններում): Գործողության արդյունքը պահվում է ընդունիչում (բացառությամբ TEST հրամանի): Հրամաններն ազդում են վիճակի դրոշների վրա:

Գործողությունները կատարվում են բիթ առ բիթ՝ ըստ Աղյուսակ 1-ի և բավարարում են Աղյուսակ 2-ում բերված հատկությունները:

X	Y	X AND Y	X OR Y	X XOR Y	NOT X
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Աղյուսակ 1. Տրամաբանական գործողությունների ճշտության աղյուսակ

*	AND	OR	XOR
X * 0	0	X	X
X * 1	X	1	NOT X
X * X	X	X	0
X * NOT X	0	1	1

Աղյուսակ 2. Տրամաբանական գործողությունների հատկությունները

AND – Տրամաբանական բազմապատկում (և)

Շարահյուսություն: **AND ընդունիչ, աղբյուր**
Նկարագրություն: ընդունիչ ← ընդունիչ AND աղբյուր
Դրոշներ: AF = ? CF = 0 OF = 0 PF SF ZF

Ընդունիչը կարող է լինել ռեգիստր (reg8/reg16/reg32) կամ հիշողության հասցե (mem8/mem16/mem32), իսկ աղբյուրը՝ ռեգիստր (reg8/reg16/reg32), հիշողության հասցե (mem8/mem16/mem32) կամ

անմիջական արժեք (im8/im16/im32): Դրոշները՝ CF = OF = 0, AF-ը անորոշ է, իսկ PF, SF, ZF-ը՝ ըստ նշանակության:

Օրինակ

```
MOV     AL, 43          ; 00101011B
AND     AL, 94          ; 01011110B
        00101011B
        01011110B
  AL =  00001010B
```

Արդյունքում՝ CF = 0, OF = 0, SF = 0, ZF = 0, PF = 1 AF = ?

AND հրամանը կարելի է օգտագործել անհրաժեշտ բիթերը զրոյացնելու համար:

Հրամանը կարելի է օգտագործել նաև անհրաժեշտ բիթերը զատելու համար:

Օրինակ

```
AND     AL, 11
```

				X		X	X

11 = 00001011b

0	0	0	0	X	0	X	X

OR – Տրամաբանական գումարում (կամ)

Շարահյուսություն:	**OR ընդունիչ, աղբյուր**
Նկարագրություն:	ընդունիչ ← ընդունիչ OR աղբյուր
Դրոշներ:	AF = ? CF = 0 OF = 0 PF SF ZF

Ընդունիչը կարող է լինել ռեգիստր (reg8/reg16/reg32) կամ հիշողության հասցե (mem8/mem16/mem32), իսկ աղբյուրը՝ ռեգիստր (reg8/reg16/reg32), հիշողության հասցե (mem8/mem16/mem32) կամ

անմիջական արժեք(im8/im16/im32): Դրոշները՝ CF = OF = 0, AF-ը անորոշ է, իսկ PF, SF, ZF-ը՝ ըստ նշանակության:

Օրինակ

```
MOV     AL, 43          ; AL = 00101011B
OR      AL, 94          ; AL = 01011110B, CF = 0, OF = 0,
                        ; SF = 0, ZF = 0, PF = 0 AF = ?
        00101011B
        01011110B
AL  =   01111111B
```

Հրամանը կարելի է օգտագործել անհրաժեշտ բիթերը 1 դարձնելու համար:

Օրինակ

AL ռեգիստրի 0, 1, 3 համարի բիթերը դարձնել 1:

```
OR      AL, 11          ; կամ  OR  AL, 00001011B
```

				X		X	X

11 = 00001011b

				1		1	1

XOR – Բացառող կամ (գումարում ըստ մոդուլ 2-ի)

Շարահյուսություն:	**XOR ընդունիչ, աղբյուր**
Նկարագրություն:	ընդունիչ ← ընդունիչ XOR աղբյուր
Դրոշներ:	AF = ? CF = 0 OF = 0 PF SF ZF

Ընդունիչը կարող է լինել ռեգիստր (reg8/reg16/reg32) կամ հիշողության հասցե (mem8/mem16/mem32), իսկ աղբյուրը՝ ռեգիստր (reg8/reg16/reg32), հիշողության հասցե (mem8/mem16/mem32) կամ անմիջական արժեք (im8/im16/im32): Դրոշները՝ CF = OF = 0, AF-ը՝ անորոշ է, իսկ PF, SF, ZF-ը՝ ըստ նշանակության:

Օրինակ

```
MOV   AL, 43          ; AL = 00101011B
XOR   AL, 94          ; AL = 01011110B
      00101011B
      01011110B
AL =  01110101B
```

Արդյունքում՝ CF = 0, OF = 0, SF = 0, ZF = 0, PF = 0, AF = ?

Հրամանը կարելի է օգտագործել օպերանդը զրոյացնելու կամ անհրաժեշտ բիթերի արժեքները հակադարձելու համար:

Օրինակ

Հակադարձել AL ռեգիստրի 0, 1, 3 համարի բիթերը:

```
XOR   AL, 00001011B
```

				X		X	X
11 = 00001011b							
				!X		!X	!X

Օրինակ

Զրոյացնել AX ռեգիստրը:

```
XOR     AX, AX      ; ավելի արագ է, քան 'MOV AX, 0'
                    ; հրամանը
```

Օրինակ

Փոխանակել AX և BX ռեգիստրների արժեքները առանց երրորդ փոփոխականի օգտագործման:

```
XOR   AX, BX
XOR   BX, AX
XOR   AX, BX        ; համարժեք է 'XCHG AX, BX'
```

TEST– Տրամաբանական ստուգում

Շարահյուսություն:	**TEST ընդունիչ, աղբյուր**
Նկարագրություն:	ընդունիչ AND աղբյուր
Դրոշներ:	AF = ? CF = 0 OF = 0 PF SF ZF

Ընդունիչը կարող է լինել ռեգիստր (reg8/reg16/reg32) կամ հիշողության հասցե (mem8/mem16/mem32), իսկ աղբյուրը՝ ռեգիստր (reg8/reg16/reg32), հիշողության հասցե (mem8/mem16/mem32) կամ անմիջական արժեք (im8/im16/im32): Դրոշները՝ CF = OF = 0, AF-ը անորոշ է, իսկ PF, SF, ZF-ը՝ ըստ նշանակության:

TEST-ը կատարում է նույն գործողությունը, ինչ AND հրամանը, միայն ընդունիչը չի փոխվում: Հրամանն ազդում է վիճակի դրոշների վրա: TEST հրամանն ավելի արագ է կատարվում, քան AND հրամանը:

Օրինակ

```
MOV     AL, 43          ; AL = 00101011B
TEST    AL, 94          ; CF = 0, OF = 0, SF = 0, ZF = 0,
                        ; PF = 1 AF = ?
JZ      L1
        43 = 00101011B
        94 = 01011110B
             00001010B
    AL = 43 = 00101011B
```

Չի անցնի L1 նշիչին, քանի որ արդյունքի 1 և 3 բիթերը 1 են, և հետևաբար ZF = 0:

Օրինակ

Անցում կատարել L1 նշիչին, եթե AL ռեգիստրի 0,1,3 բիթերից գոնե մեկը 1 է:

```
TEST    AL, 11
JNZ     L1
```

				X		X	X

11 = 00001011b

0	0	0	0	X	0	X	X

NOT – Տրամաբանական ժխտում (ոչ)

Շարահյուսություն:	**NOT ընդունիչ**
Նկարագրություն:	ընդունիչ ← NOT (ընդունիչ)
Դրոշներ:	**------**

Ընդունիչը կարող է լինել ռեգիստր (reg8/reg16/reg32) կամ հիշողության հասցե (mem8/mem16/mem32): NOT հրամանով ընդունիչի բոլոր բիթերի արժեքները ժխտվում են (0→1, 1→0): **Հրամանը դրոշների վրա չի ազդում:**

Օրինակ

```
MOV     AL, 43          ; AL = 00101011B
NOT     AL              ; AL = 11010100B
```

Խնդիրների լուծման օրինակներ

1. DL ռեգիստի 0, 1 բիթերը դարձնել 0-ներ, 6, 7 բիթերը դարձնել 1-ներ, իսկ մնացածները՝ հակադարձել:

```
.MODEL  TINY
.CODE
        ORG     100h
BEGIN:  MOV     DL, 0B7h        ; DL = 10110111b
        AND     DL, 11111100b   ; DL = 10110100b
        OR      DL, 11000000b   ; DL = 11110100b
        XOR     DL, 00111100b   ; DL = 11001000b
        MOV     AX, 4C00h
        INT     21h
END     BEGIN
```

2. Գրել ծրագրի հատված, որը կստուգի AL ռեգիստրի 0, 2, 3 բիթերում 0-ների, իսկ 1 և 5 բիթերում 1-երի առկայությունը:

Լուծում: Խնդիրը կարելի է լուծել տարբեր եղանակներով: Մի քանի բիթերի միաժամանակ 0 լինելը հարմար է ստուգել մեկ TEST հրամանով, սակայն միաժամանակ 1 լինելը մեկ հրամանով չի ստուգվում: Բացի դրանից՝ կան այլ բիթեր՝ 4, 6, 7, որոնց արժեքները նույնպես անհայտ են: Հետևաբար նախ հարկավոր է այդ այլ բիթերի արժեքները որոշակիացնել, ապա՝ ստուգվող բիթերը համեմատել կոնկրետ արժեքի հետ:

Լուծման 1-ին եղանակ

```
MOV   DL, AL          ; հիշենք, եթե դրա կարիքը հետագայում կա,
AND   AL, 00101111b   ; այլ բիթերը դարձնենք 0,
CMP   AL, 00100010b   ; ստուգենք անհրաժեշտ բիթերի արժեքները,
JZ    L1              ; կանցնի L1 նշիչին, եթե բավարարում է
                      ; խնդրի պայմանին:
```

…

Լուծման 2-րդ եղանակ

```
MOV   DL, AL          ; հիշենք, եթե դրա կարիքը հետագայում կա,
OR    AL, 11010000b   ; այլ բիթերը դարձնենք 1,
CMP   AL, 11110010b   ; ստուգենք անհրաժեշտ բիթերի արժեքները,
JZ    LL              ; կանցնի LL նշիչին, եթե բավարարում է
                      ; խնդրի պայմանին:
```

…

Լուծման 3-րդ եղանակ

```
MOV   DL, AL          ; հիշենք, եթե դրա կարիքը հետագայում կա,
AND   AL, 00101111b   ; այլ բիթերը դարձնենք 0,
XOR   AL, 00100010b   ; 1 - երի բիթերը շրջենք, որպեսզի
TEST  AL, 00101111b   ; մեկ TEST-ով հնարավոր լինի ստուգել
                      ; արդյո՞ք բոլորը միաժամանակ 0 են,
JZ    L1              ; կանցնի L1 նշիչին, եթե բավարարում է
                      ;  խնդրի պայմանին:
```

…

Խնդիրներ և վարժություններ

1. Նշել ռեգիստրների և վիճակի դրոշների արժեքները հետևյալ հրամանների կատարումից հետո՝

ա)
```
MOV  AX, 22
MOV  DH, 5
OR   AL, DH
XOR  AX, 40
AND  AH, 0E3H
ADC  DH, AH
```

բ)
```
XOR  CX, CX
MOV  BX, 258
NOT  BL
OR   CL, BH
XOR  CH, BL
AND  CX, 0D5H
```

գ)
```
MOV  DL, 54
MOV  DH, 25h
AND  DL, DH
OR   DX, 2C4Ah
XOR  DH, 3
```

դ)
```
MOV  AX, 267
MOV  DX, 340
OR   DL, AL
AND  DH, 25
NOT  AL
```

2. Տրված A և B ճշանով ամբողջ թվերի միաչափ բայթային զանգվածներից ստանալ C միաչափ զանգվածը, որտեղ C[i]-ն հավասար է.
   ա) A[i]-ի և B[i]-ի տրամաբանական արտադրյալին,
   բ) A[i]-ի և B[i]-ի տրամաբանական գումարին,
   գ) A[i]-ի և B[i]-ի ըստ մոդուլ 2-ի գումարին,
   դ) A[i]-ի ժխտմանը:
3. Ներածել 9 սիմվոլներից բաղկացած միաչափ զանգված: Հաշվել և արտածել էկրանին այն սիմվոլների քանակը, որոնց 2 և 4 համարի բիթերում 0 է, իսկ 1 և 5 համարի բիթերում՝ 1: Այդպիսի սիմվոլներում 3 և 5 համարի բիթերը դարձնել 0, 0 համարի բիթը դարձնել 1, իսկ 6 և 7 համարի բիթերի արժեքը՝ հակադարձել: Ստացված զանգվածն արտածել էկրանին:
4. Ներածել 15 սիմվոլներից բաղկացած միաչափ զանգված: Առանձնացնել ենթազանգված այն սիմվոլներից, որոնց 4 և 5 հա-

մարի բիթերը 1 են, իսկ 2 և 7 համարի բիթերը՝ 0: Ենթազանգվածում սիմվոլների 0 և 5 համարի բիթերը դարձնել 0, 3 և 7 համարի բիթերը հակադարձել, իսկ համար 6 բիթը՝ դարձնել 1: Ենթազանգվածն արտածել էկրանին:

Տեղաշարժի և պտույտի հրամաններ

Տեղաշարժի և պտույտի հրամանները տեղաշարժում (shift) կամ պտտում են (rotate) բայթային, բառային կամ կրկնակի բառային ընդունիչ օպերանդի բիթերը: Դրանք են՝

SAR թվաբանական աջ տեղաշարժ /Shift arithmetic right,
SHR տրամաբանական աջ տեղաշարժ /Shift logical right,
SAL/SHL թվաբանական/տրամաբանական ձախ տեղաշարժ /Shift arithmetic left/Shift logical left,
SHRD կրկնակի աջ տեղաշարժ /Shift right double,
SHLD կրկնակի ձախ տեղաշարժ /Shift left double,
ROR ցիկլիկ աջ տեղաշարժ /Rotate right,
ROL ցիկլիկ ձախ տեղաշարժ /Rotate left,
RCR ցիկլիկ աջ տեղաշարժ փոխանցման դրոշով /Rotate through carry right,
RCL ցիկլիկ ձախ տեղաշարժ փոխանցման դրոշով /Rotate through carry left,

Տեղաշարժի և պտույտի հրամանները բաժանվում են երեք խմբի.

- տեղաշարժի հրամաններ,
- ցիկլիկ տեղաշարժի կամ պտույտի հրամաններ,
- կրկնակի տեղաշարժի հրամաններ (386+):

Առաջին երկու խմբերի շարահյուսությունն է.

<հրամանի հուշանուն> ընդունիչ, հաշվիչ

որտեղ՝

- ընդունիչը ռեգիստր է (reg8/reg16/reg32) կամ հիշողության հասցե (mem8/mem16/mem32),
- հաշվիչը 1 է կամ CL ռեգիստրը՝ 8086 պրոցեսորի համար, իսկ 80186 և բարձր պրոցեսորների համար՝ 8-բիթանոց անմիջական արժեք (im8) կամ CL ռեգիստրը:

Հաշվիչը որոշում է տեղաշարժվող բիթերի քանակը: 8086-ի դեպքում, եթե հաշվիչը CL է, տեղաշարժը կատարվում է CL-ում գրված թվի չափով: 80286-ից սկսած՝ տեղաշարժը կատարվում է CL-ի կրտսեր 5 բիթերի չափով՝ 0-ից 31 արժեքներով (64-բիթանոց ճարտարապետությունում՝ CL-ի 6 բիթերի չափով՝ 0-ից 63 արժեքներով): Հաշվիչի արժեքը 0 լինելու դեպքում տեղաշարժ չի կատարվում, և դրոշները չեն փոխվում:

Տեղաշարժի հրամաններ

Տեղաշարժի հրամանները լինում են տրամաբանական և թվաբանական և կատարում են տեղաշարժ դեպի աջ կամ ձախ:

Տեղաշարժի հրամաններ

տրամաբանական տեղաշարժի հրամաններ (առանց նշանի օպերանդների համար)	թվաբանական տեղաշարժի հրամաններ (նշանով օպերանդների համար)
SHL (Shift logical Left)	SAL (Shift arithmetic Left)
SHR (Shift logical Right)	SAR (Shift arithmetic Right)

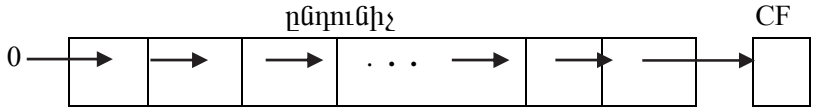
Տրամաբանական տեղաշարժի ժամանակ ազատված բիթերում գրվում են 0-ներ: Թվաբանական աջ տեղաշարժի ժամանակ ազատված բիթերում ընդունիչի նշանային բիթի արժեքն է կրկնվում, իսկ ձախ տեղաշարժի ժամանակ ազատված բիթերը լրացվում են 0–ներով:

Դրոշները ստանում են հետևյալ արժեքները.

- PF, SF և ZF-ը որոշվում են ըստ իրենց նշանակության,
- AF-ն անորոշ է,
- CF-ում ընդունիչից դուրս եկած վերջին բիթի արժեքն է,
- 1-ից ավելի տեղաշարժերի դեպքում OF-ի արժեքն անորոշ է, իսկ 1 դիրք տեղաշարժի ժամանակ OF-ը որոշվում է հետևյալ կերպ. SAL/SHL-ի դեպքում OF-ը հավասար է 1-ի, եթե տեղաշարժից հետո ընդունիչի ավագ բիթի արժեքը փոխվել է և 0՝ հակառակ դեպքում, SAR հրամանի դեպքում OF-ը հավասար է 0, իսկ SHR հրամանի դեպքում՝ OF-ը հավասար է տեղաշարժից առաջ ընդունիչի ավագ բիթի արժեքին:

SHR – Տրամաբանական աջ տեղաշարժ /Shift logical Right

Շարահյուսություն:	**SHR ընդունիչ, հաշվիչ**
Նկարագրություն:	ընդունիչ ← SHR(ընդունիչ)
Դրոշներ:	AF = ? CF OF PF SF ZF



Օրինակ

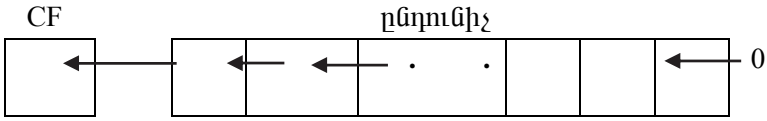
```
MOV     AL, 43          ; AL = 00101011B
SHR     AL, 1           ; AL = 00010101B
```

Դրոշները՝ CF = 1, OF = 0, SF = 0, ZF = 0, PF = 0, AF = ?

SHR հրամանը կարելի է օգտագործել (և արժի օգտագործել, քանի որ այն ավելի արագ կատարվող հրաման է) ընդունիչի արժեքը որպես առանց նշանի թիվ 2-ի հաշվիչ աստիճանի վրա բաժանելու համար: Օրինակում AL ռեգիստրում գրվել էր 43, հրամանից հետո AL-ի արժեքը հավասար է 21, այսինքն՝ 43/2: Եթե կրկին կատարվի SHR AL, 1 հրամանը, կամ միանգամից կատարվի 2 բիթ տեղաշարժ դեպի աջ (MOV CL, 2 և SHR AL, CL), ապա AL ռեգիստրի արժեքը կլինի 43/4 = 10:

SHL/SAL – Տրամաբանական / թվաբանական ձախ տեղաշարժ/
Shift logical / arithmetic Left

Շարահյուսություն:	**SHL ընդունիչ, հաշվիչ**
	SAL ընդունիչ, հաշվիչ
Նկարագրություն:	ընդունիչ ← SHL(ընդունիչ)
Դրոշներ:	AF = ? CF OF PF SF ZF



SHL և SAL հրամանները նույնն են (ունեն նույն գործողության կողը):

Օրինակ

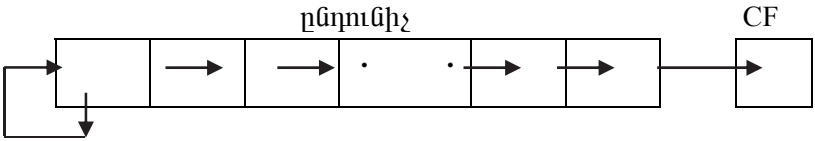
```
MOV     AL, 43          ; AL = 00101011B
SHL     AL, 1           ; AL = 01010110B
```

Դրոշները՝ CF = 0, OF = 0, SF = 0, ZF = 0, PF = 1, AF = ?
SHL AL, 1 հրամանը համարժեք է SAL AL, 1 հրամանին:

Հրամանը կարելի է օգտագործել (և արժի օգտագործել, քանի որ այն ավելի արագ կատարվող հրաման է) ընդունիչի արժեքը որպես առանց նշանի թիվ 2-ի հաշվիչ աստիճանով բազմապատկելու համար: Օրինակում AL ռեգիստրում գրվել էր 43, հրամանից հետո AL-ի արժեքը հավասար է 86, այսինքն՝ 43 * 2: Եթե մեկ անգամ ևս կատարվի SHL AL,1 հրամանը, կամ կատարվի միանգամից 2 բիթ տեղաշարժ դեպի ձախ (MOV CL, 2 և SHL AL, CL), ապա AL ռեգիստրի արժեքը կլինի 43 * 4 = 172:

SAR – Թվաբանական աջ տեղաշարժ / Shift arithmetic Right

Շարահյուսություն:	**SAR ընդունիչ, հաշվիչ**
Նկարագրություն:	ընդունիչ ← SAR (ընդունիչ)
Դրոշներ:	AF = ? CF OF PF SF ZF



Օրինակ

```
MOV     AL, 43          ; AL = 00101011B
SAR     AL, 1           ; AL = 00010101B
```

Դրոշները՝ CF = 1, OF = 0, SF = 0, ZF = 0, PF = 0, AF = ?

Հրամանը կարելի է օգտագործել (և արժի օգտագործել, քանի որ այն ավելի արագ կատարվող հրաման է) ընդունիչի արժեքը որպես նշանով թիվ 2-ի հաշվիչ աստիճանի վրա բաժանելու համար: Օրինակում AL ռեգիստրում գրվել էր 43, հրամանից հետո AL–ի արժեքը հավասար է 21, այսինքն՝ 43/2: Եթե մեկ անգամ ևս կատարվի SAR AL, 1 հրամանը, կամ կատարվի միանգամից 2 բիթ տեղաշարժ (MOV CL, 2 և SAR AL, CL), ապա AL ռեգիստրի արժեքը կլինի 43/4 = 10:

Օրինակ

```
MOV     AL, -43         ; AL = 11010101B
SAR     AL, 1           ; AL = 11101010B
```

Դրոշները՝ CF = 1, OF = 0, SF = 1, ZF = 0, PF = 0, AF = ?

Օրինակում AL ռեգիստրում գրվել էր -43, հրամանից հետո AL-ի արժեքը հավասար է -22, այսինքն՝ (-43)/2:

Ուշադրություն

Այս դեպքում բաժանման արդյունքը կլորացվում է դեպի մինուս անվերջություն, ոչ թե դեպի 0, ինչպես IDIV հրամանում էր: Եթե մեկ անգամ ևս կատարվի SAR AL, 1 հրամանը, կամ կատարվեր միանգամից 2 բիթ տեղաշարժ դեպի աջ (MOV CL, 2 և SAR AL, CL), ապա AL ռեգիստրի արժեքը կլինի -43/4 = -11:

Ցիկլիկ տեղաշարժի (պտույտի) հրամաններ

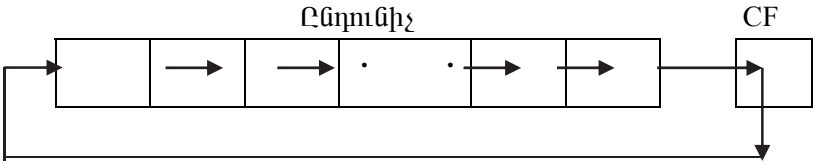
Այս խմբի հրամաններով ընդունիչի պարունակությունը տեղաշարժվում է աջ կամ ձախ, և ազատված բիթերը լրացվում են հակառակ կողմից դուրս եկած բիթերի արժեքներով՝ CF դրոշի արժեքը կամ պտույտում ներառելով կամ առանց CF դրոշի: Հրամանները չորսն են և բոլորն էլ սկսվում են R (rotate) տառով, երկրորդ տառը O կամ C է, եթե պտույտը առանց CF դրոշի արժեքի է՝O, եթե պտույտը CF դրոշի արժեքի հետ է՝ C, երրորդ տառը L կամ R է, եթե L է, ձախ տեղաշարժ է, եթե R է, աջ տեղաշարժ է:

Դրոշները ստանում են հետևյալ արժեքները.

- AF, PF, SF և ZF-ը չեն փոխվում,
- CF-ում ընդունիչից դուրս եկած վերջին բիթի արժեքն է,
- 1-ից ավելի տեղաշարժերի դեպքում OF-ի արժեքն աննորոշ է, իսկ 1 բիթ տեղաշարժի ժամանակ OF-ը հավասար է 1-ի, եթե տեղաշարժից հետո ընդունիչի ավագ բիթի արժեքը փոխվել է, և 0՝ հակառակ դեպքում:

RCR – Ցիկլիկ աջ տեղաշարժ փոխանցման դրոշով / Rotate through Carry Right

Շարահյուսություն: **RCR ընդունիչ, հաշվիչ**
Նկարագրություն: ընդունիչ ← RCR (ընդունիչ)
Դրոշներ: CF OF



Օրինակ

```
CLC             ; CF = 0
MOV     AL, 43  ; AL = 00101011B
RCR     AL, 1   ; AL = 00010101B, դրոշները՝ CF = 1, OF = 0
```

Օրինակ

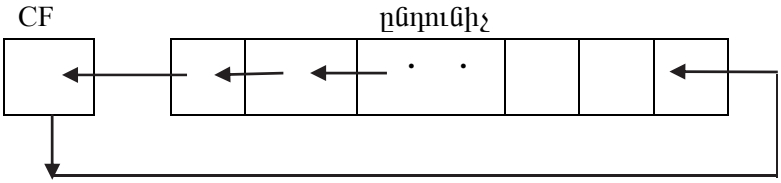
```
STC             ; CF = 1
MOV   AL, 43    ; AL = 00101011B
RCR   AL, 1     ; AL = 10010101B, դրոշները՝ CF = 1, OF = 1
```

RCL – Ցիկլիկ ձախ տեղաշարժ փոխանցման դրոշով / Rotate through Carry Left

Շարահյուսություն: **RCL ընդունիչ, հաշվիչ**

Նկարագրություն: ընդունիչ ← RCL(ընդունիչ)

Դրոշներ: CF OF



Օրինակ

```
CLC             ; CF = 0
MOV AL, 43      ; AL = 00101011B
RCL  AL, 1      ; AL = 01010110B, դրոշները՝ CF = 0, OF = 0
```

Օրինակ

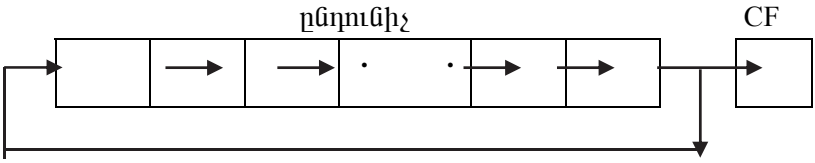
```
STC             ; CF = 1
MOV AL, 43      ; AL = 00101011B
RCL  AL, 1      ; AL = 01010111B, դրոշները՝ CF = 0, OF = 0
```

ROR – Ցիկլիկ աջ տեղաշարժ / Rotate right

Շարահյուսություն: **ROR ընդունիչ, հաշվիչ**

Նկարագրություն: ընդունիչ ←ROR(ընդունիչ)

Դրոշներ: CF OF



Օրինակ

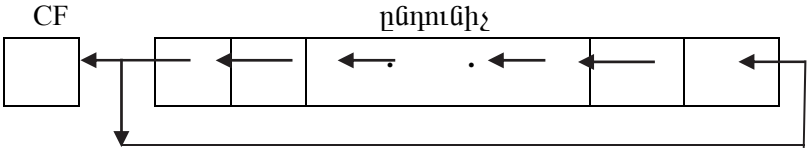
```
CLC             ; CF = 0
MOV     AL, 43  ; AL = 0010101**1**B
ROR     AL, 1   ; AL = **1**0010101B, դրոշները՝ CF = **1**, OF = 1
```

Օրինակ

```
STC             ; CF = 1
MOV     AL, 43  ; AL = 0010101**1**B
ROR     AL, 1   ; AL = **1**0010101B, դրոշները՝ CF = **1**, OF = 1
```

ROL – Ցիկլիկ ձախ տեղաշարժ / Rotate left

Շարահյուսություն: **ROL ընդունիչ, հաշվիչ**
Նկարագրություն: ընդունիչ ←ROL(ընդունիչ)
Դրոշներ: CF OF



Օրինակ

```
CLC             ; CF = 0
MOV     AL, 43  ; AL = **0**0101011B
ROL     AL, 1   ; AL = 0101011**0**B, դրոշները՝ CF = **0**, OF = 0
```

Օրինակ

```
STC             ; CF = 1
MOV     AL, 43  ; AL = **0**0101011B
ROL     AL, 1   ; AL = 0101011**0**B, դրոշները՝ CF = **0**, OF = 0
```

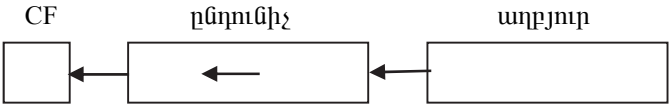
Կրկնակի տեղաշարժի հրամաններ (386+)

Հրամանները 2-ն են՝ SHRD, SHLD և ունեն 3 օպերանդ: Հրամանն իրականացնելիս ընդունիչի արժեքը տեղաշարժվում է աջ (SHRD) կամ ձախ (SHLD) հաշվիչի չափով, և ազատված բիթերը լրացվում են աղբյուրի ցածր կամ բարձր բիթերի պատճեններով համապատասխանաբար:

SHLD – Տրամաբանական կրկնակի ձախ տեղաշարժ (386+)/ Shift Left Double

Շարահյուսություն:	**SHLD ընդունիչ, աղբյուր,հաշվիչ**
Նկարագրություն:	ընդունիչ ← SHLD (ընդունիչ,աղբյուր)
Դրոշներ:	AF = ? CF OF PF SF ZF

Ընդունիչը կարող է լինել ռեգիստր (reg16/reg32) կամ հիշողության հասցե (mem16/mem32), աղբյուրը կարող է լինել ընդունիչի տիպին համապատասխան ռեգիստր (reg16/reg32), իսկ հաշվիչը՝ թիվ կամ CL ռեգիստրը: Դրոշները ստանում են հետևյալ արժեքները. AF-ը անորոշ է, CF-ում ընդունիչից դուրս եկած վերջին բիթի արժեքն է, OF-ի արժեքը 1-ից ավելի տեղաշարժերի դեպքում անորոշ է, իսկ 1 դիրք տեղաշարժի ժամանակ OF-ը հավասար է 1-ի, եթե տեղաշարժից հետո ընդունիչի նշանը փոխվել է, և 0՝ հակառակ դեպքում: PF, SF, ZF-ը որոշվում են ըստ արդյունքի:



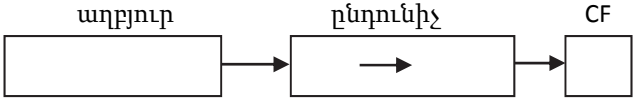
Օրինակ

```
MOV     AX, 43        ; AX = 0000 0000 00101011B
MOV     BX, 9300h     ; BX = 10010011 00000000B
SHLD    AX, BX, 2     ; AX = 00 0000 0010101110B
                      ; BX-ն անփոփոխ է, դրոշները՝ CF = 0,
                      ; OF = ?, SF = 0, ZF = 0, PF = 0, AF = ?
```

SHRD – Տրամաբանական կրկնակի աջ տեղաշարժ (386+) / Shift Right Double

Շարահյուսություն:	**SHRD ընդունիչ, աղբյուր, հաշվիչ**
Նկարագրություն:	ընդունիչ←SHRD(ընդունիչ, աղբյուր)
Դրոշներ:	AF = ? CF OF PF SF ZF

Ընդունիչը կարող է լինել ռեգիստր (reg16/reg32) կամ հիշողության հասցե (mem16/mem32), աղբյուրը կարող է լինել ռեգիստր (reg16/reg32), իսկ հաշվիչը՝ թիվ կամ CL ռեգիստր: Դրոշները ստանում են հետևյալ արժեքները. CF-ում ընդունիչից դուրս եկած վերջին բիթի արժեքն է, OF-ը 1 բիթ տեղաշարժի դեպքում ընդունում է 1 արժեք, եթե տեղի է ունեցել նշանի բիթի փոփոխություն, հակառակ դեպքում ընդունում է 0 արժեք, 1-ից ավելի բիթերի տեղաշարժի դեպքում OF դրոշն անորոշ է, AF-ը անորոշ է, PF, SF, ZF-ը որոշվում են ըստ արդյունքի արժեքի:



Օրինակ

```
MOV     AX, 43          ; AX = 0000 000000101011B
MOV     BX, 0093h       ; BX = 0000000010010011B
SHRD    AX, BX, 2       ; AX = 1100 0000 0000 1010B,
                        ; BX-ն անփոփոխ է, դրոշները՝ CF = 1,
                        ; OF = ?, SF = 1, ZF = 0, PF = 1, AF = ?
```

Խնդիրների լուծման օրինակներ

1. Հաշվել BL ռեգիստրի 2-ական ներկայացման մեջ 1-երի քանակը և արտածել էկրանին.

```
.MODEL      SMALL
.STACK      64
```

```
.CODE
BEGIN:    MOV     BL, 0D6h
          XOR     DL, DL
          MOV     CX, 8
CYCLE:    SHL     BL, 1
          ADC     DL, 0
          LOOP    CYCLE
          MOV     AH, 2
          ADD     DL, '0'          ; ⇔ OR DL,'0'
          INT     21h
          MOV     AH, 4Ch
          INT     21h
END  BEGIN
```

2. Շրջել AL ռեգիստրի 2-ական ներկայացումը.

```
. MODEL   SMALL
.STACK    64
.CODE
MAIN:     MOV     AL, 1101 0111B
          XOR     BL, BL
          MOV     CX, 8
L1:       SHL     AL, 1
          RCR     BL, 1
          LOOP    L1
          MOV     AL, BL
          MOV     AX, 4C00h
          INT     21h
END       MAIN
```

3. Գրել ծրագիր, որը BL ռեգիստրի արժեքը կարտածի էկրանին 2-ական համակարգով.

```
.MODEL    SMALL
.STACK    64
.CODE
```

```
BEGIN:  MOV     BL, 0D6h
        MOV     CX, 8
CYCLE:  XOR     DL, DL
        ROL     BL, 1
        ADC     DL, '0'
        MOV     AH, 2
        INT     21h
        LOOP    CYCLE
        MOV     AH, 4Ch
        INT     21h
END     BEGIN
```

Այս խնդիրը կարելի է լուծել տարբեր եղանակներով: Ընդհանրապես ամբողջ թիվը այլ համակարգերում ներկայացնելու համար այդ թիվը բաժանում են նոր համակարգի հիմքի վրա, սակայն, երբ համակարգերի հիմքերից մեկը մյուսի աստիճանն է, ապա դա կարելի է իրականացնել ավելի հեշտ՝ անջատելով աստիճանի ցուցիչի չափ բիթեր և այդ անջատված բիթերը առանձին-առանձին ներկայացնելով պահանջվող համակարգում:

Դիտարկենք խնդրի լուծման մեկ այլ տարբերակ: Ստորև բերված տարբերակը հարմար է նրանով, որ չնչին փոփոխություններով կարող է լինել նաև 4-ական, 8-ական և 16-ական համակարգերի համար լուծման օրինակ:

```
        . . .
        MOV   CL, 7       ; 4-ականի, 8-ականի համար՝ 6,
NEXT:   MOV   DL, BL      ; հիշենք և հետագայում
                          ; վերհիշենք,
        SHR   DL, CL      ; տեղաշարժ,
        AND   DL, 1b      ; թողնենք անհրաժեշտ բիթերը
                          ; /4-ականի համար՝ 11b,
                          ; 8-ականի, համար՝ 111b,
        ADD   DL, '0'     ; դարձնենք սիմվոլ,
        MOV   AH, 2       ; դուրս բերենք,
```

```
                INT     21H             ; էկրան,
                SUB     CL, 1           ; 4-ականի համար՝ 2, 8-ականի
                                        ; համար՝ 3
                JGE     NEXT
        . . .
```

4. SI ռեգիստրի պարունակությունը դիտարկել որպես առանց նշանի 2-ական թիվ: Շրջել և միաժամանակ արտածել այն:

```
.MODEL      SMALL
.STACK      64
.CODE
START:      MOV   SI, 6BD7h     ; SI = 0110 1011 1101 0111 b
            MOV   AH, 2
            XOR   BX, BX
            MOV   CX, 16
CYCLE:      MOV   DL, '0'
            SHL    BX, 1        ; BX = BX * 2
            SHR    SI, 1
            JNC    OUTB
            ADC   BX, 0
            INC    DL           ; DL = ‘1’
OUTB:       INT    21h
            LOOP  CYCLE         ; այժմ BX = 1110101111010110 b
                                ; և այդ էլ արտածվել է
            MOV    AX, 4C00h
            INT    21h
END         START
```

Նշում

Այս մեթոդով են լուծվում նաև N9-ը և N10 խնդիրները:

Խնդիրներ և վարժություններ

1. Երրորդ խնդիրն իրականացնել 16-ական համակարգի համար:
2. Օգտագործելով միայն տեղափոխման, տրամաբանական, տեղաշարժի, անցման հրամանները՝ իրականացնել 2 ռեգիստրների արժեքների գումարում (- ,* , /):
3. Որոշել AL ռեգիստրի և ZF դրոշի արժեքները հետևյալ հրամանների կատարումից հետո.

ա) MOV AL, 1010b NOT AL	բ) MOV AL, 0 NOT AL	գ) MOV AL, 1101b AND AL, 0111b
դ) MOV AL, 100b AND AL, 011b	ե) MOV AL, 100b TEST AL, 011b	զ) MOV AL, 0F0h OR AL, 0Fh
է) MOV AL, 0 XOR AL, 0FFh	ը) MOV AL, 101b XOR AL, AL	թ)MOV AL, 0A3h OR AL, 0D5h

4. DL ռեգիստրին վերագրել 1, եթե տեղի ունի հետևյալ պայմանը, և 0՝ հակառակ դեպքում.
   ա) AL ռեգիստրի մեջտեղի երկու բիթերի արժեքն է 10b,
   բ) BX ռեգիստրի երեք ձախ և երեք աջ բիթերն իրար հավասար են:
5. Հաշվել 20 հատ 1-բայթանոց առանց նշանի թվերի միաչափ զանգվածի այն տարրերի գումարը, որոնց վերջին երկու բիթերի արժեքն է 11b:
6. Որոշել BL ռեգիստրի և CF դրոշի արժեքները հետևյալ հրամանների կատարումից հետո.

ա) MOV BL, 101b SHL BL, 1	բ) MOV BL, 1110b MOV CL, 6 SHL BL, CL	գ) MOV BL, 110b SHR BL, 1

ղ) MOV BL, 110b
MOV CL, 3
SHR BL, CL

ե) MOV BL, 0F0h
ROL BL, 1

զ) MOV BL, 101b
MOV CL, 2
ROR BL, CL

է) MOV BX, 1122h
SHR BH, 1
RCL BL, 1

ը) MOV BX, 1122h
SHR BH, 1
RCR BL, 1

թ) MOV DL, 110b
MOV CL, 3
ROL BL, CL

7. X-ը և Y-ը առանց նշանի բառ-փովփոխականներ են: Չօգտագործելով բաժանման և բազմապատկման գործողություններ՝ կատարել հետևյալը.
   ա) Y = 2 * X – X/8 + X % 16,
   բ) Y = 19 * X:
8. Տրված է առանց նշանի X բառ-փովփոխականը: DL ռեգիստրին վերագրել 1, եթե X-ը զույգ թիվ է, իսկ հակառակ դեպքում՝ 0: Վերագրումը կատարել չորս եղանակով՝ DIV, AND, TEST, SHR հրամանների միջոցով և համեմատել այդ եղանակներն իրար հետ:
9. Շրջել և միաժամանակ արտածել BX ռեգիստրի պարունակությունը (16-ական ներկայացումը): Օրինակ, եթե BX = 0B5F4h, ապա պետք է որևէ ռեգիստրում ստանալ 4F5Bh և արտածել այն:
10. Շրջել և միաժամանակ արտածել BX ռեգիստրի պարունակությունը (4-ական ներկայացումը): Օրինակ, եթե BX = 5BA3h (այն 4-ական համակարգում կլինի 11232203), ապա որևէ ռեգիստրում պետք է ստանալ և ընթացքում արտածել 4-ական համակարգի 30223211 թիվը:
11. Գրել ծրագրի հատված, որը CL ռեգիստրի ավագ 3 բիթերը կդարձնի հավասար AL-ի ավագ 3 բիթերին, իսկ կրտսեր 5 բիթերը՝ հավասար BL-ի կրտսեր 5 բիթերին:
12. Ստուգել տրված ամբողջ թվի երկուական ներկայացման մեջ բոլոր բիթերի 1 լինելը:

13. Գտնել տրված N թվին հաջորդող այն առաջին թվի արժեքը, որի երկուական ներկայացման մեջ նույն քանակի 1-եր կան: (128 – 256, 127 – 191, 6-9, 12-17,…)
14. Հաշվել տրված 4-բայթանոց ամբողջ թվի երկուական ներկայացման մեջ ավագ 1-ին նախորդող 0-ների քանակը:
15. Գտնել տրված 4-բայթանոց ամբողջ թվի ավագ 1 բիթի համարը:
16. Հաշվել տրված թվի երկուական ներկայացման մեջ առկա 0-ների քանակը:
17. Ստանալ տրված N թվի p-րդ դիրքից հաշված k բիթերի արժեքը:
18. Տեղերով փոխել տրված 4-բայթանոց ամբողջ թվի i-րդ և j-րդ բիթերի արժեքները:
19. Որոշել՝ արդյոք տրված թիվը 2-ի աստիճան է: Առաջարկել լուծման տարբեր եղանակներ:
20. Իրականացնել 2 սիմվոլային տողերի (0 և 1 սիմվոլներից բաղկացած) գումարում և արդյունքն արտածել:
21. Որոշել՝ արդյոք տրված 2-բայթանոց ամբողջ թվի բիթային ներկայացումը սիմետրիկ (պոլինդրոմ) է, թե ոչ:
22. Որոշել, թե որ թիվն է բացակայում տրված N-1 չափի զանգվածում, եթե զանգվածի տարրերը պարունակում են [1, N] միջակայքի բոլոր թվերը՝ բացառությամբ փնտրվող թվի (օրինակ՝ 1, 5, 2, 7, 3, 4, 10, 9, 6 զանգվածում բացակայում է 8-ը):

Բիթային և բայթային հրամաններ

Բիթային հրամանները ստուգում, իսկ որոշ դեպքերում նաև փոխում են բառի կամ կրկնակի բառի որոշակի բիթերը:

Բայթային հրամաններն արժեք են տալիս բայթային օպերանդին՝ EFLAGS դրոշների ռեգիստրի պահանջվող դրոշների վիճակին համապատասխան:

BT Բիթային ստուգում/Bit test

BTS Բիթային ստուգում և 1-ի տեղադրում / Bit test and set

BTR Բիթային ստուգում և 0-ի տեղադրում / Bit test and reset
BTC Բիթային ստուգում և հակադարձում / Bit test and complement
BSF Բիթային որոնում առաջ / Bit scan forward
BSR Բիթային որոնում ետ / Bit scan reverse
SETcc Բայթի արժեքավորում՝ ըստ պայմանի / Set byte if on condition

BT – Բիթի ստուգում (Bit Test) (386+)

Շարահյուսություն:	**BT հիմք, շեղում**
Նկարագրություն:	CF ← Bit(հիմք, շեղում)
Դրոշներ:	AF = ? CF OF = ? PF = ? SF = ? ZF = ?

Հիմքը կարող է լինել ռեգիստր (reg16/reg32) կամ հիշողության հասցե (mem16/mem32), իսկ շեղումը՝ ռեգիստր (reg16/reg32) կամ անմիջական արժեք (imm8): Այս հրամանում հիմքը դիտարկվում է որպես բիթային տող: BT հրամանն ընտրում է 1 բիթ՝ հիմքի սկզբից նշված շեղումով, և այդ բիթի արժեքը գրում է CF դրոշի մեջ:

- Եթե շեղումը ռեգիստր է, ապա հիմքի և շեղման չափերը պետք է համընկնեն (այսինքն՝ լինեն 16-բիթանոց կամ 32-բիթանոց):
- Եթե հիմքը ռեգիստր է, ապա հիմքի սկիզբը ռեգիստրի 0-րդ բիթն է, իսկ որպես շեղում՝ վերցվում է շեղում օպերանդի արժեքի 16-ի կամ 32-ի (կախված հիմքի չափից՝ 16/32) վրա բաժանելուց ստացված մնացորդը: Այսպիսով՝ իրական շեղումը չի գերազանցում 15 կամ 31-ը:
- Եթե հիմքը հիշողության հասցե է, ապա հիմքի սկիզբն այդ հասցեում գտնվող բայթի 0-րդ բիթն է: Եթե շեղումը ռեգիստր է, ապա այն պարունակում է $[-2^{31}, 2^{31}-1]$ միջակայքի թիվ, իսկ եթե անմիջական արժեք է՝ [0, 31] միջակայքի թիվ (ավելի մեծ լինելու դեպքում որպես իրական շեղում վերցվում է 16-ի կամ 32-ի (կախված հիմքի չափից՝16/32) վրա բաժանելուց ստացված մնացորդը):

- Հրամանի արդյունքում CF-ը ստանում է ընտրված բիթի արժեքը, իսկ AF, OF, PF, SF և ZF դրոշների արժեքներն անորոշ են:

Օրինակ 1

```
MOV     EAX, 92456187h
BT      AX, 06          ; CF = 0
MOV     CX, 13
BT      AX, CX          ; CF = 1
BT      AX, 27          ; 27 mod 16 = 11 => չեղում = 11
                        ; => CF = 0
MOV     CX, 34
BT      AX, CX          ; 34 mod 16 = 2 => չեղում = 2
                        ; => CF = 1
BT      EAX, 10         ; CF = 0
MOV     ECX, 18
BT      EAX, ECX        ; CF = 1
BT      EAX, 39         ; 39 mod 32 = 7 => չեղում = 7
                        ; => CF = 1
MOV     ECX, 47
BT      EAX, ECX        ; 47 mod 32 = 15
                        ; => չեղում = 15 => CF = 0
```

Օրինակ 2

```
BT      A, 12           ; CF = 0
MOV     CX, 6
BT      A, CX           ; CF = 1
BT      B, 9            ; CF = 0
MOV     ECX, 22
BT      B, ECX          ; CF = 1
BT      A, 37           ; 37 mod 32 = 5 => չեղում = 5
                        ; => CF = 0
MOV     CX, 29
```

```
BT      A, CX           ; 29 mod 16 = 13 => շեղում = 13
                        ; => CF = 1
BT      B, 46           ; 46 mod 32 = 14 => շեղում = 14
                        ; => CF = 0
```

Որտեղ.

```
A       DW      0A245h
B       DD      0C2453467h
```

BTS – Բիթի ստուգում և 1-ի տեղադրում /Bit Test and Set (386+)

Շարահյուսություն:	**BTS հիմք, շեղում**
Նկարագրություն:	CF ← Bit(հիմք, շեղում) Bit(հիմք, շեղում) ← 1
Դրոշներ:	AF = ? CF OF = ? PF = ? SF = ? ZF = ?

BTR – Բիթի ստուգում և 0-ի տեղադրում /Bit Test and Reset (386+)

Շարահյուսություն:	**BTR հիմք, շեղում**
Նկարագրություն:	CF ← Bit(հիմք, շեղում) Bit(հիմք, շեղում) ← 0
Դրոշներ:	AF = ? CF OF = ? PF = ? SF = ? ZF = ?

BTC – Բիթի ստուգում և հակադարձում /Bit Test and Complement (386+)

Շարահյուսություն:	**BTC հիմք, շեղում**
Նկարագրություն:	CF ← Bit(հիմք, շեղում) NOT(Bit(հիմք, շեղում))
Դրոշներ:	AF = ? CF OF = ? PF = ? SF = ? ZF = ?

Այս 3 հրամանների շարահյուսությունը նույնն է, ինչ BT հրամանինը: Հրամանների կատարումը տարբերվում է նրանով, որ, բացի բիթի ստուգումից, յուրաքանչյուր հրաման կատարում է նաև լրացուցիչ գործողություն՝ ընտրված բիթը դարձնելով 1 (BTS), 0 (BTR) կամ հակադարձելով (BTC):

Նշված հրամանների գործողության արդյունքում CF-ը ստանում է ընտրված բիթի սկզբնական արժեքը, իսկ AF, OF, PF, SF և ZF դրոշների արժեքներն անորոշ են:

Օրինակ 3

```
MOV     AX, 6187h
BTR     AX, 7               ; CF = 1, AX = 6107h
MOV     CX, 12
BTS     AX, CX              ; CF = 0, AX = 7107h
BTC     AX, 27              ; 27 mod 16 = 11 => չեղում = 11
                            ;  => CF = 0, AX = 7907h
MOV     CX, 34
BTR     AX, CX              ; 34 mod 16 = 2 => չեղում = 2
                            ; => CF = 1, AX = 7903h
MOV     EBX, 92457903h
BTC     EBX, 17             ; CF = 0, EBX = 92477903h
MOV     ECX, 20
BTS     EBX, ECX            ; CF = 0, EBX = 92577903h
BTR     EBX, 48             ; 48 mod 32 = 16 => չեղում = 16
                            ; => CF = 1, EBX = 92567903h
MOV     CX, 40
BTC     EBX, ECX            ; 40 mod 32 = 8 => չեղում = 8
                            ; => CF = 1, EBX = 92567803h
BTC     A, 12               ; CF = 0, A = 0B245h
MOV     CX, 6
BTR     A, CX               ; CF = 1, A = 0B205h
BTS     B, 9                ; CF = 0, B = 0C2453667h
MOV     ECX, 22
BTC     B, ECX              ; CF = 1, B = 0C2053667h
BTS     A, 37               ; 37 mod 16 = 5 => չեղում = 5
                            ; => CF = 0, A = 0B225h
BTC     B, 45               ; 45 mod 32 = 13 => չեղում = 13
```

```
                                    ; => CF = 1, B = 0C2051667h
    MOV     ECX, 20
    BTS     B, ECX                  ; CF = 0, B = 0C2151667h
```

Որտեղ.

```
    A       DW      0A245h
    B       DD      0C2453467h
```

BSF – Բիթային որոնում առաջ / Bit Scan Forward (386+)

Շարահյուսություն:	**BSF ընդունիչ, աղբյուր**
Նկարագրություն:	ընդունիչ ← BSF(աղբյուր)
Դրոշներ:	AF = ? CF = ? OF = ? PF = ? SF = ? ZF

Ընդունիչը կարող է լինել ռեգիստր (reg16/reg32), իսկ աղբյուրը՝ ռեգիստր (reg16/reg32) կամ հիշողության հասցե (mem16/mem32): Օպերանդների չափերը պետք է համընկնեն: BSF հրամանը որոնում է աղբյուրի մեջ կրտսեր՝ 0 բիթից սկսած առաջին 1 արժեք ունեցող բիթը և նրա համարը (0-ական բիթից հաշված) գրում ընդունիչի մեջ: Եթե աղբյուրի արժեքը 0 է, ապա ընդունիչի արժեքն անորոշ է: AF, CF, OF, PF, SF դրոշների արժեքներն անորոշ են: ZF-ը 1 է, եթե աղբյուրի արժեքը 0 է, հակառակ դեպքում՝ ZF = 0:

BSR – Բիթային որոնում ետ / Bit Scan Reverse (386+)

Շարահյուսություն:	**BSR ընդունիչ, աղբյուր**
Նկարագրություն:	ընդունիչ ← BSR(աղբյուր)
Դրոշներ:	AF = ? CF = ? OF = ? PF = ? SF = ? ZF

Ընդունիչը կարող է լինել ռեգիստր (reg16/reg32), իսկ աղբյուրը՝ ռեգիստր (reg16/reg32) կամ հիշողության հասցե (mem16/mem32): Օպերանդների չափերը պետք է համընկնեն:

BSF հրամանը որոնում է աղբյուրի մեջ ավագ՝ 15 կամ 31 բիթից սկսած առաջին 1 արժեք ունեցող բիթը և նրա համարը (0-ական բիթից հաշված) գրում ընդունիչի մեջ: Եթե աղբյուրի արժեքը 0 է, ընդու-

նիչի արժեքն անորոշ է: AF, CF, OF, PF, SF դրոշների արժեքներն անորոշ են: ZF-ը 1 է, եթե աղբյուրի արժեքը 0 է, հակառակ դեպքում՝ ZF = 0:

Օրինակ

```
1.  MOV   CX, 1256h
    BSF   AX, CX          ; AX = 1
    BSR   DX, CX          ; DX = 12

2.  MOV   ECX, 49782463h
    BSF   EAX, ECX        ; EAX = 0
    BSR   EDX, ECX        ; EDX = 30

3.  BSF   AX, A           ; AX = 2
    BSR   DX, A           ; DX = 14

4.  BSF   EDI, B          ; EDI = 5
    BSR   ESI, B          ; ESI = 31
```

Որտեղ.

```
A   DW    6A54h
B   DD    0C2453460h
```

SETcc – Բայթի արժեքավորում՝ ըստ պայմանի / Set byte on condition (386+)

Շարահյուսություն:	**SETcc ընդունիչ**
Նկարագրություն:	ընդունիչ ←1, եթե cc պայմանը տեղի ունի, և 0՝ հակառակ դեպքում,
Դրոշներ:	----------

Ընդունիչը կարող է լինել ռեգիստր (reg8) կամ հիշողության հասցե (mem8):

Եթե cc պայմանը տեղի ունի, ընդունիչը ստանում է 1 արժեք, հակառակ դեպքում՝ 0: Պայմանը որոշվում է CF, OF, PF, SF, ZF դրոշներից մեկի կամ մի քանիսի արժեքներով:

Հրամանը չի ազդում դրոշների վրա:

Հրամանի հուշանուն	**Պայմանի ստուգում՝ ըստ դրոշների**	**Հրամանի լրիվ անվանումը**
SETA / SETNBE	CF = 0 & ZF = 0	Տեղադրել, եթե բարձր է (Set if above) / Տեղադրել, եթե ցածր կամ հավասար չէ (Set if not below or equal)
SETAE / SETNB / SETNC	CF = 0	Տեղադրել, եթե բարձր է կամ հավասար (Set if above or equal) / Տեղադրել, եթե ցածր չէ (Set if not below) / Տեղադրել, եթե փոխանցում չկա (Set if not carry)
SETB / SETNAE / SETC	CF = 1	Տեղադրել, եթե ցածր է (Set if below) / Տեղադրել, եթե բարձր կամ հավասար չէ (Set if not above or equal) / Տեղադրել, եթե փոխանցում կա (Set if carry)
SETBE / SETNA	CF = 1 | ZF = 1	Տեղադրել, եթե ցածր կամ հավասար է (Set if below or equal) / Տեղադրել, եթե բարձր չէ (Set if not above)
SETE / SETZ	ZF = 1	Տեղադրել, եթե հավասար է (Set if equal) / Տեղադրել, եթե զրո է (Set if zero)
SETG / SETNLE	ZF = 0 & SF = OF	Տեղադրել, եթե մեծ է (Set if greater) / Տեղադրել, եթե փոքր կամ հավասար չէ (Set if not less or equal)
SETGE / SETNL	SF = OF	Տեղադրել, եթե մեծ կամ հավասար է (Set if greater or equal) / Տեղադրել, եթե փոքր չէ (Set if not less)
SETL	SF <> OF	Տեղադրել, եթե փոքր է (Set if less)

/ SETNGE		/ Տեղադրել, եթե մեծ կամ հավասար չէ (Set if not greater or equal)
SETLE / SETNG	ZF = 1 | SF <> OF	Տեղադրել, եթե փոքր կամ հավասար է (Set if less or equal) / Տեղադրել, եթե մեծ չէ (Set if not greater)
SETNE / SETNZ	ZF = 0	Տեղադրել, եթե հավասար չէ (Set if not equal) / Տեղադրել, եթե զրո չէ (Set if not zero)
SETNO	OF = 0	Տեղադրել, եթե գերհագեցում չկա (Set if not overflow)
SETO	OF = 1	Տեղադրել, եթե գերհագեցում կա (Set if overflow)
SETNP / SETPO	PF = 0	Տեղադրել, եթե զույգություն չկա (Set if not parity) / Տեղադրել, եթե կենտ է (Set if parity odd)
SETP / SETPE	PF = 1	Տեղադրել, եթե զույգություն կա (Set if parity) / Տեղադրել, եթե զույգ է (Set if parity even)
SETNS	SF = 0	Տեղադրել, եթե նշան չկա և ոչ բացասական է (Set if not sign)
SETS	SF = 1	Տեղադրել, եթե նշան կա և բացասական է (Set if sign)

Նկար 1. SETcc հրամանի իրականացման պայմանները

Օրինակ 1

```
MOV     AL, 43      ; AL = 43
MOV     BL, 50      ; BL = 50
ADD     AL, BL      ; AL = 93, CF = 0 ZF = 0
SETA    DL          ; (CF = 0 & ZF = 0) => DL = 1
```

Օրինակ 2

```
MOV     AL, 100     ; AL = 100
MOV     BL, 62      ; BL = 62
```

```
SUB     BL, AL      ; BL = -38, SF = 1, ZF = 0, OF = 0
SETLE   DL          ; (ZF = 1 | SF <> OF) => DL = 1
```

Եթե SETcc հրամանն օգտագործում ենք CMP հրամանից հետո, պայմանը որոշվում է CMP հրամանի արդյունքով.

Օրինակ 1

```
MOV     AL, 43      ; AL = 43
MOV     BL, 50      ; BL = 50
CMP     AL, BL      ; CF = 1, ZF = 0, SF = 1, OF = 0, PF =1
SETBE   DL          ; AL ≤ BL => DL=1
                    ; (պայմանն է CF = 1| ZF = 1)
```

Օրինակ 2

```
MOV     AL, 100     ; AL = 100
MOV     BL, 62      ; BL = 62
CMP     AL, BL      ; CF = 0, ZF = 0, SF = 0, OF = 0, PF = 0
SETE    DL          ; AL <> BL => DL = 0
                    ; (պայմանն է ZF = 1)
```

Ղեկավարությունը փոխանցող հրամաններ

Ղեկավարությունը փոխանցող հրամաններն ապահովում են հետևյալ գործողությունների կատարումը.

- անցում՝ առանց պայմանի (JMP) կամ պայմանով (Jcc),
- ցիկլի կազմակերպում (LOOP, LOOPE, LOOPZ, LOOPNE, LOOPNZ),
- պրոցեդուրայի կանչ (CALL) կամ ընդհատում (INT),
- վերադարձ պրոցեդուրայից (RET) կամ ընդհատումից (IRET):

Անցումները լինում են 4 տեսակի:

1. Կարճ անցում (short jump) – անցման հրամանից մինչև անցման հասցե հեռավորությունը գտնվում է [-128,127] միջակայքում: Օպերանդը rel8 է (նշիչ): Փոխվում է IP/EIP-ի արժեքը:
2. Մոտիկ անցում (near jump) – անցման հրամանը և անցման հասցեն գտնվում են նույն սեգմենտում: Օպերանդը rel16/rel32/reg16/reg32/mem16/mem32 է: Փոխվում է IP/EIP-ի արժեքը:
3. Հեռու անցում (far jump) – անցման հրամանը և անցման հասցեն գտնվում են տարբեր սեգմենտներում: Օպերանդը ptr16:16 / ptr16:32 / mem16:16 / mem16:32 է: Փոխվում են CS-ի և IP/EIP-ի արժեքները:
4. Անցում՝ խնդիրների փոխարկումով (task switch) – բազմախնդրային ռեժիմում անցում է կատարում մի խնդրից մյուսը: Անցման այս տիպը օգտագործվում է միայն **պաշտպանված ռեժիմում**:

Կարճ և մոտիկ անցումների կատարման ժամանակ պրոցեսորը տվյալ սեգմենտի ներսում անցում է կատարում օպերանդով որոշվող հասցեին: Հասցեն տրվում է **բացարձակ** կամ **հարաբերական** շեղումով: Բացարձակ շեղումը հրամանային սեգմենտի սկզբից ունեցած շեղումն է, որը որոշվում է ոչ ուղղակի հասցեավորմամբ՝ ընդհանուր օգտագործման ռեգիստրի կամ հիշողության միջոցով (*r/m16* կամ *r/m32*): **Հարաբերական** շեղումը՝ rel8, rel16, կամ rel32, հրամանի ցուցիչի ռեգիստրի՝ EIP-ի ընթացիկ արժեքին համեմատ հարաբերական տեղաշարժն է: Հարաբերական շեղումն ասեմբլեր ծրագրում սովորաբար տրվում է որպես նշիչ, սակայն մեքենայական կոդում այն կոդավորվում է որպես նշանով 8,16 կամ 32-բիթանոց անմիջական արժեք՝ im8/im16/im32, որը գումարվում է EIP-ի արժեքին: Կարճ և մոտիկ անցումների ժամանակ CS ռեգիստրի արժեքը չի փոխվում:

JMP –Անցում/ Jump

Շարահյուսություն: **JMP օպերանդ**

Նկարագրություն: առանց պայմանի անցում

JMP հրամանն առանց պայմանի և անվերադարձ անցում է կատարում այն հրամանին, որի հասցեն տրված է օպերանդում: Հասցեն կարող է տրվել ուղղակի՝ ճշիչի, ոչ ուղղակի՝ ռեգիստրի (reg16/reg32) կամ հիշողության հասցեի (mem16/mem32) միջոցով:

JMP հրամանով հնարավոր է կատարել կարճ, մոտիկ, հեռու և խնդիրների փոխարկումով անցումներ:

JMP հրամանը կարճ, մոտիկ և հեռու անցումների դեպքում չի ազդում դրոշների վրա, իսկ խնդիրների փոխարկումով անցման ժամանակ դրոշները փոխվում են:

Օրինակ 1

```
MOV     DX, 1234H
JMP     DX
```

Այս հրամանն անցում կկատարի CS:1234H հասցեով: 1234H չի դիտարկվում որպես ճշանով հարաբերական շեղում:

Օրինակ 2

```
MOV     BX, 1234H
JMP     WORD PTR 2000H[BX]
```

Եթե DS:3234H-ի պարունակությունը 5678H է, ապա անցում կկատարվի CS:5678H հասցեին:

***Օրինակ* 3**

```
MOV     BX, 1234H
JMP     DWORD PTR 2000H[BX]
```

Եթե DS:3234H-ի պարունակությունը 5678H է, իսկ DS:3236H-ինը՝ 0ABCDH, ապա անցում կկատարվի 0ABCDH:5678H հասցեին: Այս հրամանը **հեռու, ոչ ուղղակի** անցման օրինակ է:

Jcc – Պայմանական անցում / Jump if Condition is Met

Շարահյուսություն: **Jcc նշիչ**

Նկարագրություն: անցում նշիչով որոշվող հրամանին, եթե cc պայմանը տեղի ունի

Դրոշներ: ----------

Jcc հրամանները կատարում են կարճ կամ մոտիկ անցում տրված նշիչով որոշվող հրամանին, եթե պայմանը տեղի ունի: Պայմանը որոշվում է կոնկրետ դրոշների արժեքներով կամ CX ռեգիստրի արժեքով:

Հաճախ Jcc հրամանն օգտագործում են CMP հրամանից անմիջապես հետո և այդ դեպքում պայմանը որոշվում է CMP հրամանի կատարումից առաջացած դրոշների արժեքներով:

Պայմանի հուշանվան մեջ մասնակցում են բարձր (above) և ցածր (below) տերմինները՝ առանց նշանի անցումների դեպքում և մեծ (greater) ու փոքր (less) տերմինները՝ նշանով անցումների դեպքում: Առանց նշանի անցումների դեպքում դիտարկվում է CF դրոշը, իսկ նշանով անցումների դեպքում՝ OF և SF դրոշները:

Յուրաքանչյուր հրաման ունի իրեն համարժեք մեկ այլ անվանում: Համարժեք (նույն մեքենայական կոդն ունեցող) հրամանները բաժանված են ‘/’ սիմվոլով:

Հրամանի հուշանուն	**Պայմանի ստուգում՝ ըստ դրոշների կամ CX ռեգիստրի**	**Հրամանի լրիվ անվանումը**
JA /JNBE	CF = 0 & ZF = 0	Անցում, եթե բարձր է (Jump if above) / Անցում, եթե ցածր կամ հավասար չէ (Jump if not below or equal)
JAE /JNB /JNC	CF = 0	Անցում, եթե բարձր է կամ հավասար (Jump if above or equal) / Անցում, եթե ցածր չէ (Jump if not below)/ Անցում, եթե փոխանցում չկա (Jump if not carry)
JB /JNAE /JC	CF = 1	Անցում, եթե ցածր է (Jump if below) / Անցում, եթե բարձր կամ հավասար չէ (Jump if not above or equal) / Անցում, եթե փոխանցում կա (Jump if carry)

JBE /JNA	CF = 1 | ZF = 1	Անցում, եթե ցածր կամ հավասար է (Jump if below or equal) / Անցում, եթե բարձր չէ (Jump if not above)
JE /JZ	ZF = 1	Անցում, եթե հավասար է (Jump if equal)/ Անցում, եթե զրո է (Jump if zero)
JG /JNLE	ZF = 0 & SF = OF	Անցում, եթե մեծ է (Jump if greater) / Անցում, եթե փոքր կամ հավասար չէ (Jump if not less or equal)
JGE /JNL	SF = OF	Անցում, եթե մեծ կամ հավասար է (Jump if greater or equal) / Անցում, եթե փոքր չէ (Jump if not less)
JL /JNGE	SF <> OF	Անցում, եթե փոքր է (Jump if less) / Անցում, եթե մեծ կամ հավասար չէ (Jump if not greater or equal)
JLE /JNG	ZF = 1 | SF <> OF	Անցում, եթե փոքր կամ հավասար է (Jump if less or equal) / Անցում, եթե մեծ չէ (Jump if not greater)
JNE /JNZ	ZF = 0	Անցում, եթե հավասար չէ (Jump if not equal) / Անցում, եթե զրո չէ (Jump if not zero)
JNO	OF = 0	Անցում, եթե գերհագեցում չկա (Jump if not overflow)
JO	OF = 1	Անցում, եթե գերհագեցում կա (Jump if overflow)
JNP /JPO	PF = 0	Անցում, եթե զույգություն չկա (Jump if not parity) / Անցում, եթե կենտ է (Jump if parity odd)
JP /JPE	PF = 1	Անցում, եթե զույգություն կա (Jump if parity) /Անցում, եթե զույգ է (Jump if parity even)
JNS	SF = 0	Անցում, եթե նշան չկա և ոչ բացասական է (Jump if not sign)
JS	SF = 1	Անցում, եթե նշան կա և բացասական է (Jump if sign)
JCXZ	CX = 0	Անցում, եթե CX = 0 (Jump if CX is zero)
JECXZ (386+)	ECX = 0	Անցում, եթե ECX = 0 (Jump if ECX is zero)

Նկար 1. Jcc հրամանի իրականացման պայմանները

Օրինակ 1

Հաշվել և BL-ում պահել տրված A և B նշանով թվերից մեծի արժեքը:

```
      . . .
      MOV     AL, A
      MOV     BL, B
      CMP     AL, BL
      JLE     next
      MOV     BL, AL
next: . . .
```

Օրինակ 2

Հաշվել և BL-ում պահել տրված A և B առանց նշանի թվերից փոքրի արժեքը:

```
      . . .
      MOV     AL, A
      MOV     BL, B
      CMP     AL, BL
      JBE     next
      MOV     BL, AL
next: . . .
```

Օրինակ 3

BL-ին վերագրել 1, եթե տրված A և B նշանով թվերի գումարը բացասական է, հակառակ դեպքում՝ BL-ին վերագրել 0:

```
      . . .
      MOV     AL, A
      XOR     BL, BL
      ADD     AL, B
      JNS     next
      INC     BL
next: . . .
```

LOOP / LOOPcc – Ցիկլ /LOOP according to ECX counter

Շարահյուսություն:	**LOOP /LOOPcc նշիչ**
Նկարագրություն:	ցիկլի կազմակերպում՝ ըստ CX/ECX ռեգիստրի և ZF դրոշի
Դրոշներ:	----------

Բոլոր ցիկլի հրամանները CX/ECX ռեգիստրի արժեքը (առանց նշանի ամբողջ թիվ) նվազեցնում են 1-ով, ապա կարճ անցում կատարում նշիչով որոշվող հրամանին, եթե հաշվիչը զրո չէ, և cc պայմանը տեղի ունի:

Հրամանի հուշանուն	Հրամանի աշխատանքի նկարագրություն
LOOP	1. CX = CX - 1 2. Անցում նշիչին, եթե CX ≠ 0
LOOPE/LOOPZ	1. CX = CX - 1 2. Անցում նշիչին, եթե CX ≠ 0 և ZF = 1
LOOPNE/LOOPNZ	1. CX = CX - 1 2. Անցում նշիչին, եթե CX ≠ 0 և ZF = 0

Նկար 2. LOOP /LOOPcc հրամանների իրականացման պայմանները

Ցիկլի հրամաններն օգտագործվում են ցիկլերի կազմակերպման համար, որտեղ CX/ECX ռեգիստրը հանդես է գալիս որպես ցիկլի հաշվիչ, ընդ որում՝ ցիկլի մարմնի մեջ մտնող հրամանները գոնե մեկ անգամ կատարվում են:

Ցիկլի հրաման օգտագործող ծրագրի կողմն ունի հետևյալ ընդհանուր տեսքը.

```
            . . .
            MOV CX, N
Loop_label: հրաման_1
            հրաման_2
            . . .
            հրաման_K
            LOOP/LOOPcc   loop_label
```

Հրաման_1, հրաման_2, ... հրաման_K հրամանների խումբը կազմում են ցիկլի մարմինը:

LOOP հրամանի դեպքում ցիկլի մարմնի մեջ մտնող հրամանները կկատարվեն N (N ≠ 0) անգամ, իսկ LOOPcc հրամանների դեպքում, քանի դեռ CX-ը 0 չէ, և պայմանը տեղի ունի: Պայմանը որոշվում է ZF դրոշով, որն արժեք է ստանում LOOPcc-ին նախորդող ZF-ի վրա ազդող հրամանով:

Օրինակ 3

Հաշվել և BX-ում պահել [a,b] միջակայքի զույգ թվերի գումարը:

```
            . . .
            MOV   CX, b
            SUB   CX, a
            INC   CX          ; միջակայքի տարրերի քանակը
            XOR   BX, BX
            MOV   DX, a       ; DX-ում պահում ենք միջակայքի
                              ; երբական թիվը
CYRCLE:     TEST  DX, 1
            JNZ   next        ; եթե զույգ չի, անցնել next նշիչին
            ADD   BX, DX      ; եթե զույգ է՝ գումարել BX-ին
 next:      INC   DX
            LOOP  CYRCLE
            . . .
```

CALL – Պրոցեդուրայի կանչ / Call procedure

Շարահյուսություն:	**CALL օպերանդ**
Նկարագրություն:	օպերանդով որոշվող պրոցեդուրայի կանչ
Դրոշներ:	**----------**

CALL հրամանով տեղի է ունենում օպերանդով որոշվող պրոցեդուրայի կանչ: Օպերանդն անմիջական արժեք է (նշիչ), ընդհանուր օգտագործման ռեգիստր՝ reg16/reg32, կամ հիշողության հասցե՝ mem16/mem32, և պարունակում է կանչվող պրոցեդուրայի հասցեն (այսինքն՝ պրոցեդուրայի առաջին հրամանի հասցեն): Եթե օպերանդը նշիչ է (պրոցեդուրայի անունը), ապա օպերանդի արժեքը նշիչի հարաբերական հասցեն է (այսինքն՝ շեղումը պրոցեդուրայի անվան և CALL հրամանի միջև), և պրոցեդուրայի սկզբի բացարձակ հասցեն կներկայացնի ընթացիկ IP/EIP + հարաբերական հասցե: Եթե օպերանդը ռեգիստր կամ հիշողության հասցե է, ապա այն պարունակում է պրոցեդուրայի սկզբի բացարձակ հասցեն (շեղումը սեգմենտի սկզբից):

CALL հրամանն ընթացիկ հրամանի՝ վերադարձի հասցեն, պահում է պահունակում և կատարում անցում օպերանդում նշված հասցեով: Իրական ռեժիմում կա պրոցեդուրայի 2 տիպի կանչ.

1. մոտիկ կանչ – պրոցեդուրայի սահմանումն ու կանչը գտնվում են նույն սեգմենտում,
2. հեռու կանչ – պրոցեդուրայի սահմանումն ու կանչը կարող են գտնվել տարբեր սեգմենտներում:

Մոտիկ կանչի դեպքում պրոցեսորը նախ IP/EIP-ի ընթացիկ արժեքը գրում է պահունակի մեջ, ապա անցում կատարում կանչվող պրոցեդուրային՝ IP/EIP-ին վերագրելով պրոցեդուրայի սկզբի բացարձակ հասցեն:

Հեռու կանչի դեպքում պրոցեսորը պահունակի մեջ գրում է CS և IP/EIP ռեգիստրների ընթացիկ արժեքները, ապա CS և IP/EIP ռեգիստրներին վերագրում կանչվող պրոցեդուրայի սեգմենտի և սեգ-

մենտի սկզբից ունեցած շեղման արժեքները համապատասխանաբար:

Օրինակ 1

CALL AX

Եթե (AX) = 1234H, ապա ղեկավարությունը կվոխանցվի CS:1234H հասցեին: 1234H-ը չի դիտարկվում որպես հարաբերական շեղում:

Օրինակ 2

CALL WORD PTR 2000H[BX]

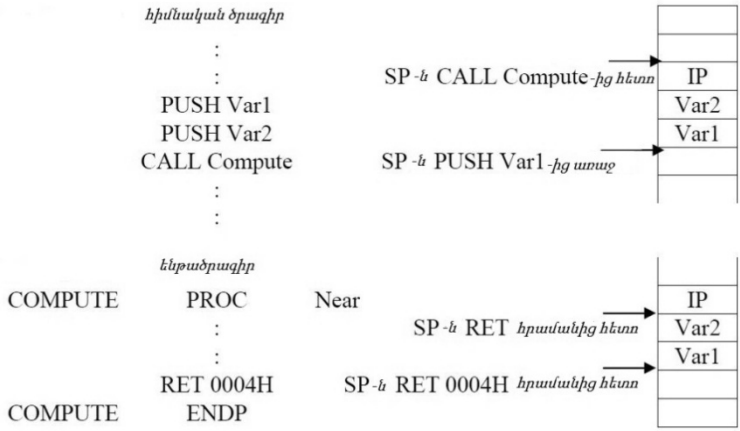
Եթե (BX) = 1234H, DS:3234H-ի պարունակությունը՝ 5678H, ապա ղեկավարությունը կվոխանցվի CS:5678H հասցեով ենթածրագրին:

RET / RETN /RETF – Վերադարձ պրոցեդուրայից / Return from Procedure

Շարահյուսություն:	**RET/ RET K / RETN / RETN K / RETF / RETF K**
Նկարագրություն:	վերադարձ պրոցեդուրայից հիմնական ծրագիր
Դրոշներ:	----------

RET հրամանը կատարում է վերադարձ պրոցեդուրայից հիմնական ծրագիր: Կախված պրոցեդուրայի տեսակից՝ NEAR կամ FAR, ասեմբլերը RET հրամանը փոխարինում է RETN կամ RETF հրամաններով: RETN-ի դեպքում պահունակից կարդացվում է 1 բառ կամ կրկնակի բառ և գրվում IP/EIP-ի մեջ, իսկ RETF-ի դեպքում՝ 2 կամ 3 բառ և գրվում IP/EIP և CS-ի մեջ համապատասխանաբար:

RET K (RETN K/ RETF K) հրամանն աշխատում է նույն կերպ, ինչ-որ RET-ը, միայն թե վերադարձի հասցեն (IP/EIP կամ IP/EIP և CS) պահունակից հանելուց հետո պահունակից հանվում է նաև K բայթ: Սա օգտագործվում է այն դեպքում, երբ պրոցեդուրային պարամետրեր են փոխանցվել պահունակով:



Նկար 3. RET K հրամանի իրականացումը

INT– Ընդհատման կանչ / Call to Interrupt Procedure

Շարահյուսություն: **INT N**

Նկարագրություն: ընդհատումը մշակող պրոցեդուրայի կանչ

Դրոշներ: ---------

N-ը ընդհատման համարն է՝ [0,255] միջակայքի թիվ, և INT N հրամանով կատարվում է N համարի ընդհատումը մշակող ենթածրագրի կանչ, այլ կերպ ասած՝ ծրագրի ղեկավարությունը փոխանցվում է N համարով որոշվող հասցեում տեղադրված ենթածրագրին, որին անվանում են ընդհատումը մշակող ենթածրագիր (**interrupt service routine** (**ISR**)): Այս հրամանի իրականացումը կախված է պրոցեսորի ռեժիմից:

Իրական ռեժիմում ընդհատումը մշակող ենթածրագրերի/պրոցեդուրաների հասցեները պահվում են ընդհատման վեկտորների աղյուսակում, իսկ պաշտպանված ռեժիմում՝ ընդհատումների նկարագրիչների (descriptor) աղյուսակում (IDT – Interrupt Descriptor Table):

Կդիտարկենք INT հրամանի կատարումը իրական ռեժիմում: Ընդհատման վեկտորների աղյուսակը տեղադրված է 0000:0000h

հասցեից, ունի 1ԿԲ երկարություն (1024 = 400H), որտեղ յուրաքանչյուր ընդհատման վեկտորի հատկացված է 4 բայթ: Ընդհատման վեկտորի առաջին 2 բայթում գրված է ընդհատումը մշակող պրոցեդուրայի սեգմենտի սկզբի հասցեն, իսկ հաջորդ 2 բայթում՝ այդ սեգմենտում ընդհատումը մշակող պրոցեդուրայի շեղման արժեքը: Աղյուսակի առաջին 4 բայթը համապատասխանում են „INT 0" ընդհատմանը, հաջորդ քառյակը՝ „INT 1" ընդհատմանը և այլն:

INT N հրամանն իրական ռեժիմում իրականացնում է հետևյալ քայլերը.

1. պահունակի մեջ գրվում է FLAGS/EFLAGS արժեքը (PUSH FLAGS/ PUSH EFLAGS),
2. պահունակի մեջ գրվում է հեռու վերադարձի սեգմենտ:շեղում հասցեն (PUSH CS; PUSH IP/EIP),
3. պրոցեսորը 0:N*4 հասցեից որոշում է ընդհատումը մշակող ենթածրագրի հասցեն,
4. IF ← 0, TF ←0,
5. ղեկավարությունը փոխանցվում է ընդհատումը մշակող ենթածրագրին (CS← 0:[4 * N], IP ← 0:[4 * N + 2]):

IRET / IRETD (80386+) – Վերադարձ ընդհատումից /Interrupt Return

Շարահյուսություն:	**IRET /IRETD**
Նկարագրություն:	վերադարձ ընդհատումը մշակող պրոցեդուրայից հիմնական ծրագիր
Դրոշներ:	բոլոր դրոշների արժեքները փոխվում են

IRET/IRETD հրամանը կազմակերպում է վերադարձ ընդհատումը մշակող պրոցեդուրայից հիմնական ծրագիր: Դրա համար հրամանը պահունակից կարդում է 3 արժեք և գրում դրանք IP, CS և FLAGS ռեգիստրների մեջ IRET-ի դեպքում կամ EIP, CS և EFLAGS ռեգիստրների մեջ IRETD-ի դեպքում:

Խնդիրների լուծման օրինակներ

1. Հաշվել և CL ռեգիստրում պահել տրված բայթային A և B բնական թվերի ամենամեծ ընդհանուր բաժանարարը՝ ԱԸԲ.

```
.MODEL SMALL
.DATA
    A       DB      20
    B       DB      30
.CODE
BEGIN:  MOV         AL, A
        MOV         BL, B
CIRCLE: CMP         AL, BL
        JE          FINISH
        JA          X1
        SUB         BL, AL
        JMP         CIRCLE
X1:     SUB         AL, BL
        JMP         CIRCLE
FINISH: MOV         CL, AL          ; ⇔ MOV CL, BL
        MOV         AX, 4C00H
        INT         21H
        END         BEGIN
```

2. Հաշվել և CL ռեգիստրում պահել N տարրից բաղկացած A միաչափ բայթային զանգվածի դրական տարրերի քանակը.

```
.MODEL  SMALL
.DATA
    N       EQU     5
    A       DB      -27, 45, -5, 22, 67
.CODE
BEGIN:  XOR         SI, SI
```

```
            XOR     CL, CL
CIRCLE:     CMP     SI, N
            JE      FINISH
            CMP     A[SI], 0
            JLE     NEXT
            INC     CL
NEXT:       INC     SI
            JMP     CIRCLE
FINISH:     MOV     AX, 4C00H
            INT     21H
            END     BEGIN
```

3. Հաշվել և BX ռեգիստրում պահել առանց նշանի N տարրից բաղկացած A միաչափ բառային զանգվածի զույգ տարրերի գումարը.

```
.MODEL  SMALL
.DATA
   N    EQU   5
   A    DW    24, 31, 45, 18, 74
.CODE
BEGIN:  XOR     SI, SI
        MOV     CX, N
        XOR     BX, BX
        MOV     AX, A[SI]
CIRCLE: TEST    AX,1
        JNZ     NEXT
        ADD     BX, AX
NEXT:   INC     SI
        INC     SI
        LOOP    CIRCLE
        MOV     AX, 4C00H
        INT     21H
        END     BEGIN
```

4. Հաշվել և BL ռեգիստրում պահել N տարրերից բաղկացած A միաչափ բայթային զանգվածի այն տարրերի քանակը, որոնց երկուական ներկայացման մեջ 1-երի քանակը զույգ է:

```
.MODEL  SMALL
.DATA
    N       EQU     5
    A       DB      -42, 20, -5, 11, 7
.CODE
BEGIN:  XOR         SI, SI
        MOV         CX, N
        XOR         BL, BL
CIRCLE: TEST        byte ptr A[SI], 0FFh
        JNP         NEXT
        INC         BL
NEXT:   INC         SI
        LOOP        CIRCLE
        MOV         AX, 4C00H
        INT         21H
        END         BEGIN
```

5. SI ռեգիստրում ստանալ N (N = 5) տարրերից բաղկացած A միաչափ բայթային զանգվածի առաջին 0 արժեք ունեցող տարրի ինդեքսը: BL ռեգիստրին տալ 1 արժեք, եթե նման տարր կա, հակառակ դեպքում BL-ին վերագրել 0:

```
.MODEL SMALL
.DATA
    N       EQU     5
    A       DB      -4, 0, -5, 1, 7
.CODE
BEGIN:  MOV         SI, -1
        MOV         CX, N
        MOV         BL, 1
REPEAT: INC         SI
```

```
        CMP     A[SI], 0
        LOOPNE  REPEAT
        JE      SKIP
        MOV     BL, 0           ; փնտրվող տարրը չկա
SKIP:   MOV     AX, 4C00H
        INT     21H
        END     BEGIN
```

Խնդիրներ և վարժություններ

1. Նշանով բայթային ամբողջ թվերի տրված միաչափ զանգվածում հաշվել 7-ից մեծ և 4-ին պատիկ թվերի գումարը:
2. Նշանով բայթային ամբողջ թվերի տրված միաչափ զանգվածում հաշվել միանիշ թվերի քանակը:
3. Նշանով բառային ամբողջ թվերի տրված միաչափ զանգվածից ստանալ զույգ թվերից կազմված նոր ենթազանգված:
4. Տրված սիմվոլային միաչափ զանգվածից ստանալ մեծատառերից կազմված նոր ենթազանգված:
5. Նշանով բայթային ամբողջ թվերի տրված միաչափ զանգվածում գտնել առաջին կենտ թիվը և հաշվել դրան նախորդող թվերի գումարը:
6. Նշանով բայթային ամբողջ թվերի տրված միաչափ զանգվածում գտնել [5; 10] միջակայքին պատկանող արժեքով վերջին թիվը և հաշվել դրան հաջորդող թվերի քանակը:
7. Նշանով բառային ամբողջ թվերի տրված միաչափ զանգվածում գտնել առաջին երկնիշ թիվը և հաշվել դրան հաջորդող թվերի միջին թվաբանականը:
8. Ստեղնաշարից ներածել սիմվոլներ մինչև ´*´-ի հանդիպելը: Դրանցում հաշվել թվանշանների քանակը և այն արտածել էկրանին: Թվանշան-սիմվոլների որոնումը և քանակի արտածումն իրականացնել առանձին պրոցեդուրաներով:

9. Ստեղնաշարից ներածել սիմվոլներ մինչև ՛%՛-ի հանդիպելը: Դրանցում հաշվել թվանշանների գումարը և այն արտածել էկրանին: Թվանշան-սիմվոլների որոնումը և գումարի արտածումն իրականացնել առանձին պրոցեդուրաներով:
10. Ստեղնաշարից ներածել սիմվոլներ մինչև ՛#՛-ի հանդիպելը: Դրանցում հաշվել մեծատառերի քանակը և այն արտածել էկրանին: Մեծատառերի որոնումը և քանակի արտածումն իրականացնել առանձին պրոցեդուրաներով:
11. Ստեղնաշարից ներածել սիմվոլներ մինչև ՛$՛-ի հանդիպելը (առավելագույնը 20 սիմվոլ): Դրանցում գտնել փոքրատառերը և առանձին տողով արտածել էկրանին: Փոքրատառերի որոնումն իրականացնել պրոցեդուրայով:
12. Ստեղնաշարից ներածել 15 սիմվոլներից կազմված զանգված: Պրոցեդուրայի միջոցով այդ զանգվածում որոնել փոքրատառերը և դրանք դարձնել համապատասխան մեծատառեր: Դրանցից կազմել նոր երկչափանի զանգված և այն նոր տողից արտածել էկրանին:
13. Ստեղնաշարից ներածել 20 սիմվոլներից կազմված զանգված: Պրոցեդուրայի միջոցով այդ զանգվածում որոնել [20H; 70H] միջակայքի սիմվոլները: Դրանցից կազմել նոր երկչափանի զանգված և այն նոր տողից արտածել էկրանին: Զանգվածում այդպիսի սիմվոլներ չլինելու դեպքում արտածել համապատասխան հաղորդագրություն:
14. Ստեղնաշարից ներածել 26 սիմվոլներից կազմված զանգված: Պրոցեդուրայի միջոցով այդ զանգվածում որոնել թվանշանները: Հաշվել նրանց գումարը և այն պրոցեդուրայի միջոցով նոր տողից արտածել էկրանին: Զանգվածում այդպիսի սիմվոլներ չլինելու դեպքում արտածել համապատասխան հաղորդագրություն:

Տողային հրամաններ

Այս հրամաններն աշխատում են տողերի հետ: Տողը (string) բայթերից/ բառերից/ կրկնակի բառերից (386+) կազմված հաջորդականություն է: Տողային հրամաններն իրականացնում են տվյալների տեղափոխություն հիշողության մի տիրույթից մյուսը (MOVS), հիշողության տիրույթների համեմատություն (CMPS), որոնում հիշողության տիրույթում (SCAS), հիշողության տիրույթի սկզբնարժեքավորում (STOS) և այլն: Որոշ տողային հրամաններում (MOVS, CMPS) երկու օպերանդներն էլ հիշողության հասցեներ են: Տողային հրամանները յոթն են, և նրանց հուշանունները վերջանում են S(String) տառով կամ S-ին հաջորդող B,W,D(386+) տառերից մեկով (որով էլ որոշվում է հրամանում մասնակցող միավորի տիպը): Տողային հրամաններն ունեն գրության երկու եղանակ՝ առանց օպերանդների և օպերանդների բացահայտ տրմամբ: Եթե հրամանի հուշանունը վերջանում է B/W/D տառով, ապա օպերանդները չեն գրվում: Հրամանում օպերանդները ձևական են, և դրանք կարող են ցույց չտալ իրական հասցեն:

Տողային հրամաններով աշխատելիս պետք է հիշել հետևյալը.

- ընդունիչը որոշվում է ES: DI/EDI -ի, իսկ աղբյուրը՝ DS: SI/ESI հասցեներով,
- օպերանդներն օգտագործվում են միավորի տիպը որոշելու համար (ասեմբլեր թարգմանիչը որոշում է տվյալ հրամանը ‘B’, ‘W’ թե ‘D’ տարբերակի թարգմանի): Օպերանդների բացահայտ օգտագործումը հնարավորություն է տալիս լռությամբ օգտագործվող DS սեգմենտային ռեգիստրը փոխարինելու այլ սեգմենտային ռեգիստրով (ES, GS, FS, CS, SS) սեգմենտի վերաբեռնման նախդիրի միջոցով: Ընդունիչի սեգմենտային ռեգիստրը՝ ES-ը, չի թույլատրվում փոխարինել:
- Տողային հրամանները մեկ միավորի հետ գործողությունն իրականացնելուց հետո, կախված DF դրոշից, հրամանում

աշխատող ինդեքսային ռեգիստրների արժեքներն ավտոմատ մեծացնում (DF = 0 դեպքում) կամ փոքրացնում (DF = 1 դեպքում) են միավորի չափով՝ 1, 2 կամ 4-ով:

- Տողային հրամանների հետ կարող են օգտագործվել կրկնման նախդիրներ (REP,REPE/REPZ, REPNE/REPNZ): Առանց կրկնման նախդիրների՝ այս հրամաններն աշխատում են միայն մեկ միավորի՝ բայթի (byte), բառի (word) կամ կրկնակի բառի (dword, 386+) հետ: Եթե հրամանն ունի կրկնման նախդիր, այդ դեպքում գործողությանը կարող են մասնակցել ամենաշատը CX/ECX-ի չափով միավորներ, և յուրաքանչյուր միավորը մշակելուց հետո CX/ECX-ի արժեքը նվազում է 1-ով: Կրկնությունն իրականացվում է ըստ հետևյալ աղյուսակի.

Կրկնման նախդիր	**Կրկնության ավարտի պայման1**	**Կրկնության ավարտի պայման2**
REP (կրկնել քանի դեռ ECX-ը 0 չի / Repeat while ECX not zero)	ECX/CX = 0	-
REPE/REPZ (կրկնել քանի դեռ հավասար է / Repeat while equal/Repeat while zero	ECX/CX = 0	ZF = 0
REPNE/REPNZ (կրկնել քանի դեռ հավասար չէ/Repeat while not equal/Repeat while not zero	ECX/CX = 0	ZF = 1

REP նախդիրի դեպքում յուրաքանչյուր միավորի մշակումից հետո CX/ECX ռեգիստրի արժեքը նվազում է 1-ով, և եթե այն զրո չէ (զրոյի դեպքում հրամանն ավարտվում է), հրամանի իրականացումը շարունակվում է: REPE/REPNE/REPZ/REPNZ նախդիրների դեպքում յուրաքանչյուր միավորի մշակումից հետո CX/ECX ռեգիստրի արժեքը նվազում է 1-ով, և եթե այն զրո չէ (զրոյի դեպքում հրամանն ավարտվում է), հրամանի իրականացումը շարունակվում է՝ կախված նախդիրից և, ըստ դրա, ZF դրոշից: Յուրաքանչյուր միավորի մշա-

կումից հետո հրամանում կիրառվող ինդեքսային ռեգիստրները (DI/EDI, SI/ESI) ավազում (DF = 1 դեպքում) կամ աճում (DF = 0 դեպքում) են միավորի չափով:

Սովորաբար REP նախդիրն օգտագործվում է INS, OUTS, MOVS, LODS, STOS հրամանների հետ, իսկ REPE, REPNE, REPZ, REPNZ նախդիրները՝ CMPS, SCAS հրամանների հետ: Կրկնման նախդիրներն օգտագործվում են միայն տողային հրամանների հետ:

MOVS/MOVSB/MOVSW/MOVSD	Տողի տեղափոխում/ Move string
CMPS/CMPSB/CMPSW/CMPSD	Տողի համեմատում/ Compare string
SCAS/SCASB/SCASW/SCASD	Տողում որոնում /Scan string
LODS/LODSB/LODSW/LODSD	Տողի բեռնում/Load string
STOS/STOSB/STOSW/STOSD	Գրառում տողում/Store string
INS/INSB/INSW/INSD	Տողի ներածում կայանից/ Input string from port
OUTS/OUTSB/INSD	Տողի արտածում կայան/ Output string to port

MOVS /MOVSB / MOVSW /MOVSD(386+) – Տեղափոխել տող / Mov String

Շարահյուսություն:	**MOVS ընդունիչ, աղբյուր**
Նկարագրություն:	Իրականացնել MOVSB /MOVSW /MOVSD կախված ընդունիչի և աղբյուրի տիպից
Շարահյուսություն:	**MOVSB / MOVSW / MOVSD (386+)**
Նկարագրություն:	BYTE / WORD / DWORD PTR ES:EDI/DI ← BYTE / WORD / DWORD PTR DS:ESI/SI EDI/DI ← EDI/DI + 1 / 2 / 4, եթե DF = 0 ESI/SI ← ESI/SI + 1 / 2 / 4, եթե DF = 0 EDI/DI ← EDI/DI - 1 / 2 / 4, եթե DF = 1 ESI/SI ← ESI/SI – 1 / 2 / 4, եթե DF = 1
Դրոշներ:	----------

Ընդունիչը և աղբյուրը հիշողության հասցե (mem8/mem16 /mem32) են: Հրամանի կատարման արդյունքում աղբյուրով որոշվող հասցեից միավորի չափ բայթ (1, 2 կամ 4) պատճենվում է ընդունիչով որոշվող հասցեի մեջ, և փոխվում են ինդեքսային ռեգիստրների արժեքները միավորի չափով և ըստ DF դրոշի:

Հրամանը կարող է ունենալ կրկնման REP նախդիր, որի դեպքում հրամանը կրկնվում է CX/ECX ռեգիստրի արժեքի չափով: Հրամանի ավարտին CX/ECX ռեգիստրի արժեքը զրո է:

Օրինակ

Հիշողության T հասցեից 6 բառ պատճենել հիշողության K հասցե:

```
.DATA
   T    DW     5, 6, 7, 56, -24, 37
   K    DW     6  dup(?)
.CODE
        …
       PUSH    DS
       POP     ES      ; DS - ը և ES - ը նույն արժեքն ունեն
       CLD             ; մաքրում ենք DF դրոշը, ինդեքսն աճելու է
       LEA     SI, T
       LEA     DI, K
       MOV     CX, 6
REP  MOVSW
```

CMPS /CMPSB /CMPSW /CMPSD(386+) –Տողերի համեմատում /Compare String

Շարահյուսություն: **CMPS ընդունիչ, աղբյուր**

Նկարագրություն: Իրականացնել CMPSB /CMPSW /CMPSD՝ կախված ընդունիչի և աղբյուրի տիպից

Շարահյուսություն: **CMPSB / CMPSW / CMPSD (386+)**

Նկարագրություն: CMP BYTE / WORD / DWORD PTR ES:EDI/DI, BYTE / WORD / DWORD PTR DS:ESI/SI

EDI/DI ← EDI/DI + 1 / 2 / 4, եթե DF = 0
ESI/SI ← ESI/SI + 1 / 2 / 4, եթե DF = 0
EDI/DI ← EDI/DI - 1 / 2 / 4, եթե DF = 1
ESI/SI ← ESI/SI - 1 / 2 / 4, եթե DF = 1

Դրոշներ: AF CF OF PF SF ZF

Ընդունիչը և աղբյուրը հիշողության հասցե (mem8/mem16/ mem32) են: Այս հրամանն ընդունիչի արժեքից հանում է աղբյուրի արժեքը և ըստ դրա՝ փոփոխում է վիճակի դրոշները:

Հրամանը կարող է ունենալ կրկնման REPE/REPZ կամ REPNE/ REPNZ նախդիրը, որի դեպքում հրամանը կրկնվում է ամենաշատը CX/ECX ռեգիստրի արժեքի չափով՝ հաջորդաբար համեմատելով միավորի չափ բայթեր ընդունիչից և աղբյուրից: Աշխատանքն ավարտվում է կամ CX/ECX ռեգիստրի արժեքը զրո դառնալու կամ ստուգվող պայմանը խախտվելու դեպքում: Հրամանը կարելի է օգտագործել՝ 2 տողերում առաջին (DF = 0 դեպքում) կամ վերջին (DF = 1 դեպքում) համընկնող (REPNE/REPNZ նախդիրով) կամ չհամընկնող (REPE/REPZ նախդիրով) միավորի տեղը որոշելու համար:

Նշում

Եթե տողերում վերջին միավորը չի բավարարում նախդիրի պայմանին, ապա դա CX/ECX-ի արժեքից չի երևում, քանի որ այս դեպքում այն զրո է, և տողն ավարտվել է: Այնպես որ պետք է լրացուցիչ ստուգվի պայմանին բավարարող միավորի առկայությունը:

SCAS/SCASB/SCASW/SCASD(386+) – Որոնում տողում /Scan String

Շարահյուսություն: **SCAS ընդունիչ**
Նկարագրություն: Իրականացնել SCASB / SCASW / SCASD՝ կախված ընդունիչի և կուտակիչի տիպից

Շարահյուսություն: **SCASB / SCASW / SCASD (386+)**
Նկարագրություն: CMP BYTE / WORD / DWORD PTR ES:EDI/DI, Al / AX / EAX

EDI/DI ← EDI/DI + 1 / 2 / 4, եթե DF = 0
EDI/DI ← EDI/DI - 1 / 2 / 4, եթե DF = 1

Դրոշներ: AF CF OF PF SF ZF

Ընդունիչը հիշողության հասցե (mem8/mem16/mem32) է: Որպես երկրորդ օպերանդ՝ հանդես է գալիս կուտակիչը՝ AL/AX/EAX ռեգիստրը: Այս հրամանն ընդունիչի արժեքից հանում է կուտակիչի՝ AL/AX/EAX արժեքը և ըստ դրա՝ փոփոխում է վիճակի դրոշները:

Հրամանը կարող է ունենալ կրկնման REPE/REPZ կամ REPNE/REPNZ նախդիրը, որի դեպքում հրամանը կրկնվում է ամենաշատը CX/ECX ռեգիստրի արժեքի չափով՝ հաջորդաբար համեմատելով ընդունիչի միավորի չափ բայթերի արժեքները կուտակիչի արժեքի հետ: Աշխատանքն ավարտվում է կամ CX/ECX ռեգիստրի արժեքը զրո դառնալու կամ ստուգվող պայմանը խախտվելու դեպքում: Յուրաքանչյուր միավորը համեմատելիս ինդեքսային DI/EDI ռեգիստրի արժեքը փոխվում է: Կրկնման նախդիրով SCAS հրամանը կարելի է օգտագործել կուտակիչի արժեքի հետ տողում առաջին (DF = 0 դեպքում) կամ վերջին (DF = 1 դեպքում) համընկնող (REPNE/REPNZ) կամ չհամընկնող (REPE/REPZ) միավորի տեղը որոշելու համար:

Նշում

Եթե տողերում վերջին միավորը չի բավարարում նախդիրի պայմանին, ապա դա CX/ECX-ի արժեքից չի երևում, քանի որ այս դեպքում այն զրո է, և տողն ավարտվել է: Այնպես որ պետք է լրացուցիչ ստուգվի պայմանին բավարարող միավորի առկայությունը:

Օրինակ

Հիշողության T հասցեում գրված 6 բայթերի արժեքներից գտնել վերջին 15 թիվը:

```
DATA  SEGMENT
      T       DB      15, 6, 7, 15,6, -24,
DATA  ENDS
COD   SEGMENT
```

```
ASSUME      CS:COD, DS:DATA
    START:  ……
            STD                 ; DF = 1, ինդեքսը նվազելու է
            LEA     SI, T
            ADD     SI, 5       ; SI-ն տողի վերջն է
            MOV     AL, 15      ; փնտրվող արժեքը
            MOV     CX, 6       ; ընդհանուր քանակը
    REPNE   SCASB               ; փնտրում
            ……..
```

LODS /LODSB /LODSW /LODSD(386+) –Տողի բեռնում/Load String

Շարահյուսություն: **LODS աղբյուր**

Նկարագրություն: Իրականացնել LODSB/LODSW/LODSD՝ կախված կուտակիչի և աղբյուրի տիպից

Շարահյուսություն: **LODSB / LODSW / LODSD (386+)**

Նկարագրություն: AL/AX/EAX ← BYTE /WORD/DWORD PTR DS:ESI/SI

ESI/SI ← ESI/SI + 1 / 2 / 4, եթե DF = 0
ESI/SI ← ESI/SI – 1 / 2 / 4, եթե DF = 1

Դրոշներ: AF CF OF PF SF ZF

Աղբյուրը հիշողության հասցե (mem8/mem16/mem32) է: Այս հրամանն աղբյուրով որոշվող հասցեից արժեքը պատճենում է կուտակիչի (AL/AX/EAX) մեջ: Միավորը պատճենելիս ինդեքսային ռեգիստրը՝ ESI/SI, նվազում (DF = 1 դեպքում) կամ աճում (DF = 0 դեպքում) է միավորի չափով:

Հրամանը կարող է ունենալ կրկնման REP նախդիր, որի դեպքում կուտակիչի մեջ կմնա տողի վերջին միավորի արժեքը, սակայն դա կարելի է անել առանց REP նախդիրի, հետևաբար նախդիր այս հրամանի համար չարժի օգտագործել: Հրամանը կարելի է օգտագործել ցիկլում՝ տողից հերթական միավորը կուտակիչի մեջ բեռնելու համար:

STOS/STOSB/STOSW/STOSD(386+) - Գրառում տողում/ Store String

Շարահյուսություն: **STOS ընդունիչ**
Նկարագրություն: Իրականացնել STOSB/STOSW/STOSD՝ կախված ընդունիչի և կուտակիչի տիպից

Շարահյուսություն: **STOSB / STOSW / STOSD (386+)**
Նկարագրություն: BYTE/WORD/DWORD PTR ES:EDI/DI ←AL/AX/EAX

EDI/DI ← EDI/DI + 1 / 2 / 4, եթե DF = 0
EDI/DI ← EDI/DI – 1 / 2 / 4, եթե DF = 1

Դրոշներ: AF CF OF PF SF ZF

Ընդունիչը հիշողության հասցե (mem8/mem16/mem32) է: Այս հրամանը կուտակիչի (AL/AX/EAX) արժեքը պատճենում է ընդունիչում, և ինդեքսային EDI/DI ռեգիստրի արժեքը ավելացնում (DF = 1 դեպքում) կամ ավելացնում (DF = 0 դեպքում) է միավորի չափով:

Հրամանը կարող է ունենալ կրկնման REP նախդիր, որի դեպքում հրամանը կրկնվում է CX/ECX ռեգիստրի արժեքի չափով՝ կուտակիչի արժեքը հաջորդաբար գրանցելով ընդունիչ տողում: Հրամանը կարելի է օգտագործել ցիկլում՝ կուտակիչի արժեքը տողում հերթական միավորի տեղը գրանցելու համար:

Օրինակ

Հիշողության T հասցեում գրված 6 բառային տողից անջատել կենտ թվերը և գրել K հասցեում:

```
DATA  SEGMENT
        T       DW      5, 6, 7, 59, 24, 38
        K       DW      6 dup(?)
DATA  ENDS
COD   SEGMENT
        ASSUME          CS:COD, DS:DATA, ES:DAT
```

```
START:  . . . . . .
        PUSH    DS
        POP     ES      ; DS –ը և ES –ը նույն արժեքն ունեն
        CLD             ; մաքրում ենք DF դրոշը, ինդեքսն
                        ; աճելու է
        LEA     SI, T
        LEA     DI, ES:K
        MOV     CX, 6
N:      LODSW
        TEST    AX, 1
        JE      H
        STOSW
        LOOP    N
        . . . . .
```

INS /INSB /INSW /INSD(386+) –Տողի ներածում կայանից /Input String

Շարահյուսություն: **INS ընդունիչ, DX**

Նկարագրություն: Իրականացնել INSB/INSW/INSD(386+)՝ կախված ընդունիչի տիպից

Շարահյուսություն: **INSB / INSW / INSD (386+)**

Նկարագրություն: BYTE / WORD / DWORD PTR ES:EDI/DI ← DX-ով որոշվող կայանից արժեքը

EDI/DI ← EDI/DI + 1 / 2 / 4, եթե DF = 0
EDI/DI ← EDI/DI – 1 / 2 / 4, եթե DF = 1

Դրոշներ: AF CF OF PF SF ZF

Ընդունիչը հիշողության հասցե (mem8/mem16/mem32) է: Այս հրամանով DX ռեգիստրի արժեքով որոշվող կայանից միավորի չափով բայթի արժեքը գրվում է ընդունիչում և ինդեքսային EDI/DI ռեգիստրի արժեքը նվազեցվում (DF = 1 դեպքում) կամ ավելացվում (DF = 0 դեպքում) է միավորի չափով:

Հրամանը կարող է ունենալ կրկնման REP նախդիր, որի դեպքում հրամանը կրկնվում է CX/ECX ռեգիստրի արժեքի չափով՝ հաջորդաբար ընդունիչ տողում գրանցելով կայանից կարդացված հերթական միավորը:

OUTS/OUTSB/OUTSW/OUTSD(386+) – Տողի արտածում կայան /Output String

Շարահյուսություն:	**OUTS DX, աղբյուր**
Նկարագրություն:	Իրականացնել OUTSB/OUTSW/OUTSD(386+)՝ կախված աղբյուրի տիպից
Շարահյուսություն:	**OUTSB / OUTSW / OUTSD (386+)**
Նկարագրություն:	DX-ով որոշվող կայան ← BYTE / WORD /DWORD PTR DS:ESI/SI ESI/SI ← ESI/SI + 1 / 2 / 4, եթե DF = 0 ESI/SI ← ESI/SI – 1 / 2 / 4, եթե DF = 1
Դրոշներ:	AF CF OF PF SF ZF

Աղբյուրը հիշողության հասցե (mem8/mem16/mem32) է: Այս հրամանով աղբյուրի տողից միավորի չափով բայթի արժեքը գրվում է DX ռեգիստրի արժեքով որոշվող կայանում, և ինդեքսային ESI/SI ռեգիստրի արժեքը նվազեցվում (DF = 1 դեպքում) կամ ավելացվում (DF = 0 դեպքում) է միավորի չափով:

Հրամանը կարող է ունենալ կրկնման REP նախդիր, որի դեպքում հրամանը կրկնվում է CX/ECX ռեգիստրի արժեքի չափով՝ հաջորդաբար աղբյուրի տողից միավորները պատճենելով DX ռեգիստրի արժեքով որոշվող կայանում:

Խնդիրների լուծման օրինակներ

1. Տրված տողում յուրաքանչյուր ‘B’ սիմվոլ փոխարինել ‘A’ սիմվոլով: Արտածել ստացված տողն էկրանին:

```
.MODEL SMALL
.STACK 100h
.DATA
   STRING1          DB     ‘ABCDEFGHIJBBAB’, ’$’
   STRING1_LEN      EQU    $ - STRING1
.CODE
MAIN:          MOV          AX , @DATA
               MOV          DS, AX
               MOV          ES, AX
               LEA          DI, STRING1
               CLD
               MOV          CX, STRING1_LEN
               DEC          CX
L1:            MOV          AL, ‘B’
REPNE          SCASB
               JNZ          L2
               DEC          DI
               MOV          AL, ‘A’
               STOSB
               INC          CX
               LOOP         L1
L2:            LEA          DX, STRING1
               MOV          AH, 9
               INT          21H
               MOV          AX, 4C00H
               INT          21H
END            MAIN
```

2. Գրել ծրագրի հատված, որը ASCIIZ string (այսինքն՝ տող, որն ավարտվում է 0 բայթով) տողից կհեռացնի վերջի բացատանիշերը:

```
.DATA
        TNAME     DB     'DHAHRAN    ' , 0
        LENGTH    EQU    $ - TNAME
        . . .
        MOV       AX, @DATA
        MOV       DS, AX
        MOV       ES, AX
        STD
        LEA       DI, TNAME + (LENGTH - 2)
        MOV       CX, LENGTH - 1
        MOV       AL , ' '
REPE    SCASB
        JE        L2
        MOV       BYTE PTR [DI + 2] , 0
        JMP       EXIT
L2:  . . .
```

3. Գրել ծրագիր, որը տրված տողի մեջ փնտրի տրված ենթատողը: Եթե ենթատողը գտնվել է, արտածել 'MATCH' հաղորդագրությունը, հակառակ դեպքում՝ 'No MATCH' հաղորդագրությունը:

```
.MODEL  SMALL
.STACK  100h
.DATA
  Sen_max       DB    127
  Sen_real      DB    ?
  Sentence      DB    127 dup (?)
  Key_max       DB    20
  Key_real      DB    ?
  Keyword       DB    20 DUP(?)
  msg1          DB    0dh, 0ah, "Enter keyword: $"
  msg2          DB    0dh, 0ah, "Enter String: $"
```

```
        msg3            DB      0dh, 0ah, “NO MATCH.$”
        msg4            DB      0dh, 0Ah, “MATCH.$”
.CODE
main:
                MOV             AX, @DATA
                MOV             DS, AX
                MOV             ES, AX
;ներածել ենթատողը
                MOV             DX, offset msg1
                MOV             AH, 9
                INT             21H
                MOV             DX, offset key_max
                MOV             AH, 10
                INT             21H
; ներածել տողը
New_sent:
                MOV             DX, offset msg2
                MOV             AH, 9
                INT             21H

                MOV             DX, offset sen_max
                MOV             AH, 10
                INT             21H
; փնտրում է տողի մեջ ենթատողը
; SI -ում ենթատողի հասցեն է
; DI- ում տողի հասցեն է
; BX – ում տողի մեջ ընթացիկ դիրքը
; DX-ում տողի մեջ սիմվոլների քանակը
                CLD
                MOV             AL, sen_real
                SUB             AL, key_real
                JL              no_match
```

```
                CBW
                MOV             DX, AX
                C               DX
; BX- ը տողի առաջին սիմվոլն է ցույց տալիս
                MOV    BX, offset sentence
Compare:
                MOV             DI, BX
                MOV             SI, offset keyword
                MOV             AL, key_real
                CBW
                MOV             CX, AX
REPE            CMPSB                   ; համեմատել սիմվոլները,
                                        ; կրկնել մինչև CX = 0 կամ
                                        ; հավասարության խախտվելը
                JZ              match
                INC             BX
                DEC             DX
                JZ              no_match
                JMP             compare
Match:
                MOV             DX, offset msg4
                MOV             AH, 9
                INT             21H
                JMP             finish
No_match:
                MOV             DX, offset msg3
                MOV             AH, 9
                INT             21H
Finish:         MOV             AX, 4c00H
                INT             21H
END main
```

Խնդիրներ և վարժություններ

1. Որոշել A փոփոխականի արժեքը հետևյալ հրամանների կատարումից հետո.

```
.DATA
    A       DB      '123456789   $'
.CODE
            MOV   AX, @DATA
            MOV   DS, AX
            MOV   ES, AX
            MOV   CX, 4
            LEA   SI, A + 2
            LEA   DI, ES:A + 4
            CLD
REP         MOVSB
```

2. Որոշել A և B տողերի և AL ռեգիստրի արժեքները հետևյալ հրամանների կատարումից հետո.

ա)
```
.DATA
    B  DB       'Fgd74'
    C  DB       5 dup(?)

.CODE
        . . .
        PUSH    DS
        POP     ES
        CLD
        LEA     DI, C
        LEA     SI, B
        MOV     CX, 5
REP     MOVSB
        LEA     DI, C
        MOV     AL, 'K'
        STOSB
        . . .
```

բ)
```
.DATA
    A      DB       'ba7ec'
    B      DB       'ba4fe'

.CODE
        . . .
        PUSH    DS
        POP     ES
        CLD
        LEA     DI, B
        LEA     SI, A
        MOV     CX, 5
REPE    CMPSB
        LODSB
        . . .
```

գ) .DATA

```
A     DB      'eB5a5'
C     DB      5 dup('0')

.CODE
      . . .
      PUSH    DS
      POP     ES
      CLD
      LEA     DI, A
      MOV     AL, '5'
      MOV     CX, 5
REPNE SCASB
      LEA     SI, C
      MOVSB
```

դ) .DATA

```
A     DB      'ba7ec'
B     DB      'ba4fe'

.CODE
      . . .
      PUSH    DS
      POP     ES
      CLD
      LEA     DI, C
      LEA     SI, A
      LODSB
      MOV     CX, 5
REP   STOSB
      . . .
```

3. Ներածել 2 սիմվոլային տող: Ստանալ և արտածել նրանց «միակցում» տողը:
4. Ներածել 2 սիմվոլային տող: Արտածել տողերի առաջին համընկող զույգը: Համընկնող զույգ չլինելու դեպքում արտածել հաղորդագրություն:
5. Ներածել 2 սիմվոլային տող: Արտածել տողերի չհամընկնող զույգերը: Չհամընկնող զույգ չլինելու դեպքում արտածել հաղորդագրություն:
6. Ներածել սիմվոլային տող (առավելագույնը 20 սիմվոլ): Արտածել տողում ‘K’ սիմվոլի հանդիպումների քանակը: Տողում ‘K’ սիմվոլ չլինելու դեպքում արտածել հաղորդագրություն:
7. Ներածել փոքրատառերից բաղկացած սիմվոլային տող: Ստանալ և արտածել համապատասխան մեծատառերից բաղկացած տողը:
8. Ներածել սիմվոլային տող: Գտնել վերջին ‘C’ սիմվոլը և տողի մնացած մասը լցնել ‘*’ սիմվոլներով: Արտածել ստացված տողը: Տողում ‘C’ սիմվոլ չլինելու դեպքում արտածել հաղորդագրություն:

9. Ներածել սիմվոլային տող: Ստանալ ենթատող միայն տառերից և արտածել այն: Տողում տառ չլինելու դեպքում արտածել հաղորդագրություն:
10. Հիշողության C հասցեում սահմանված է 75 սիմվոլներից կազմված տող: Եթե այն կարգավորված է աճման կարգով, ապա ստանալ և արտածել տողի վերջին սիմվոլի քառակի կրկնումով ստացվող տողը, հակառակ դեպքում՝ ստանալ և արտածել տողի առաջին սիմվոլի քառակի կրկնումով ստացվող տողը:
11. Հիշողության A և B հասցեներում սահմանված են 50 սիմվոլներից կազմված 2 տողեր: Դրանցում որոնել ‘Z’ սիմվոլը (ենթադրվում է, որ այն 2 տողերում էլ կա) և դրան հաջորդող սիմվոլներից սկսած՝ համեմատել տողերը: AL ռեգիստրում գրանցել 1, եթե դրանք հավասար են, 0՝ հակառակ դեպքում:
12. Ստեղնաշարից մուտքագրել 20 սիմվոլներից կազմված 2 տողեր և դրանք համեմատել: Եթե հավասար են, ապա ստանալ և արտածել ‘XXXXXYYYYY’ տողը: Հակառակ դեպքում՝ ստանալ և արտածել 2 տողերի առաջին չհամընկնող սիմվոլների հնգակի կրկնումով ստացվող տողը:
13. Ստեղնաշարից մուտքագրել 25 սիմվոլներից կազմված տող: Դրանում որոնել ’A’ սիմվոլը: Եթե այն առկա է, ապա ստանալ և արտածել ´ ***AAA*** ´ տողը: Հակառակ դեպքում արտածել ‘There isn’t symbol A’ հաղորդագրությունը:
14. Տրված է. K DB 25 DUP(16 DUP(?)):
    Դիտարկելով K-ն որպես 16-սիմվոլանոց 25 բառերից կազմված զանգված՝ լուծել հետևյալ խնդիրները.
    ա) Հաշվել զանգվածի սիմետրիկ տողերի քանակը և գրանցել այն AL ռեգիստրում:
    բ) Հաշվել սիմվոլների նվազման կարգով դասավորված տողերի քանակը:
    գ) Որոշել՝ զանգվածում կան արդյոք գոնե երկու միանման տողեր:

Դրոշների մշակման հրամաններ

Դրոշների հետ աշխատող հրամաններն առանց օպերանդի հրամաններ են, որոնք աշխատում են դրոշների ռեգիստրի կամ նրա որոշակի դրոշների հետ:

STC, CLC, CMC, CLD, STD, STI և CLI հրամանների դեպքում փոխվում է միայն հրամանի մեջ օգտագործվող դրոշը, մյուս դրոշների արժեքները չեն փոխվում:

CLD և STD հրամանները կիրառվում են տողերի հետ աշխատելիս և որոշում են ինդեքսների աճելը (CLD) կամ նվազելը (STD):

CLI և STI հրամաններն իրական ռեժիմում աշխատում են IF դրոշի հետ: CLI հրամանի արդյունքում պրոցեսորը անտեսում է դիմակավորվող արտաքին ընդհատումները:

STI-ն անում է CLI-ի հակառակ գործողությունը՝ պրոցեսորը կարող է ընդունել արտաքին սարքերից ընդհատումները:

STC – CF դրոշի 1-ով արժեքավորում /Set Carry Flag

Շարահյուսություն: **STC**

Նկարագրություն: CF ← 1

CLC – CF դրոշի 0-ով արժեքավորում /Clear Carry flag

Շարահյուսություն: **CLC**

Նկարագրություն: CF ← 0

CMC – CF դրոշի հակադարձում /Complement Carry Flag

Շարահյուսություն: **CMC**

Նկարագրություն: CF ← NOT(CF)

CLD – DF դրոշի 0-ով արժեքավորում /Clear Direction Flag

Շարահյուսություն: **CLD**

Նկարագրություն: DF ← 0

STD – DF դրոշի 1-ով արժեքավորում /Set Direction Flag

Շարահյուսություն: **STD**

Նկարագրություն: DF ← 1

STI – IF դրոշի 1-ով արժեքավորում /Set Interrupt Flag

Շարահյուսություն: **STI**
Նկարագրություն: IF ← 1

CLI – IF դրոշի 0-ով արժեքավորում /Clear Interrupt Flag

Շարահյուսություն: **CLI**
Նկարագրություն: IF ← 0

LAHF – AH ռեգիստրի բեռնում վիճակի դրոշներով /Load Status Flags into AH Register

Շարահյուսություն: **LAHF**
Նկարագրություն: AH ← FLAGS –ի կրտսեր բայթը
Դրոշներ: ----------

Բեռնում է AH ռեգիստրը FLAGS դրոշների ռեգիստրի կրտսեր բայթի արժեքով: Այսպիսով՝ AH ռեգիստրի 1 բիթը ստանում է 1 արժեք, 3 և 5 բիթերը՝ 0 արժեք, իսկ մյուս բիթերը՝ CF (0 բիթ), PF (2 բիթ), AF (4 բիթ), ZF (6 բիթ) և SF (7 բիթ) դրոշների արժեքները:

SAHF – Վիճակի ռեգիստրների բեռնում AH ռեգիստրից /Store AH into Flags

Շարահյուսություն: **SAHF**
Նկարագրություն: AF, CF, PF, SF, ZF ← AH
Դրոշներ: AF CF PF SF ZF

Բեռնում է FLAGS/ EFLAGS դրոշների ռեգիստրի կրտսեր բայթի CF, PF, AF, ZF և SF դրոշները AH ռեգիստրի 0, 2, 4, 6 և 7 բիթերի արժեքներով համապատասխանաբար: FLAGS/EFLAGS դրոշների ռեգիստրի 1, 3 և 5 բիթերի արժեքները չեն փոխվում:

PUSHF/PUSHFD – Դրոշների EFLAGS ռեգիստրի արժեքը ուղարկել պահունակ /Push EFLAGS Register onto the Stack

Շարահյուսություն: **PUSHF / PUSHFD**
Նկարագրություն: STACK ← FLAGS/EFLAGS
Դրոշներ: ----------

Պահունակի մեջ է գրում FLAGS (PUSHF-ի դեպքում) կամ EFLAGS (PUSHFD-ի դեպքում) դրոշների ռեգիստրի պարունակությունը:

POPF/POPFD – Դրոշների EFLAGS ռեգիստրի արժեքը հանել պահունակից /Pop Stack into EFLAGS Register

Շարահյուսություն:	**POPF / POPFD**
Նկարագրություն:	FLAGS/EFLAGS ← STACK
Դրոշներ:	բոլոր դրոշները

POPF հրամանը կարդում է պահունակից 2 բայթ և գրում EFLAGS-ի կրտսեր 2 բայթի՝ FLAGS ռեգիստրի մեջ, իսկ POPFD հրամանը կարդում է պահունակից 4 բայթ և գրում EFLAGS ռեգիստրի մեջ:

Այլ հրամաններ

NOP – Դատարկ հրաման/No Operation

Շարահյուսություն:	**NOP**
Նկարագրություն:	++[E]IP
Դրոշներ:	---------

LOCK – Կապուղու փակում/ Assert Lock

Շարահյուսություն:	**LOCK**
Նկարագրություն:	---------
Դրոշներ:	---------

Այս նախդիրն օգտագործվում է տվյալ հրամանի կատարման ժամանակ կապուղին այլ նպատակների չծառայեցնելու համար: Նախդիրը կարելի է օգտագործել հետևյալ հրամաններից առաջ և այն դեպքում, երբ ընդունիչ օպերանդը հիշողություն է՝ ADD, ADC, INC, SUB, SBB, DEC, NEG, AND, OR, XOR, NOT, BTC, BTR, BTS, XCHG: Եթե LOCK նախդիրն օգտագործվել է նշված հրամաններից որևէ մեկի համար, և աղբյուր օպերանդը հիշողություն է, ապա հրամանի կատարման ժամանակ հնարավոր է գեներացվի «անորոշ գործողության կոդ» (#UD) բացառությունը: Այլ հրամաններից առաջ LOCK նախդիրն օգտագործելու դեպքում գեներացվում է «անորոշ գործողության կոդ» (#UD) բացառությունը:

ՀՐԱՄԱՆԱԳՐԵՐ

Հրամանագրերն (directive) ասեմբլեր ծրագրի այն նախադասություններն են, որոնք չեն թարգմանվում մեքենայական հրամանի: Դրանք ղեկավարում են ոչ թե միկրոպրոցեսորի, այլ ասեմբլեր թարգմանչի կամ կապերի խմբագրիչի (компоновщик, linker*)* կամ բեռնիչի աշխատանքը: Հրամանագրերը, օրինակ, կարող են օգտագործվել՝ ասեմբլեր թարգմանչին հաղորդելու համար, թե ինչպիսի հաստատուններ և փոփոխականներ են օգտագործվում ծրագրում, թե որոնք են տվյալների սեգմենտներ և այլն: Հրամանագրերը նաև հնարավորություն են տալիս ղեկավարելու լիսթինգի ձևավորման գործընթացը: Հրամանագրերին անվանում են նաև կեղծ հրամաններ (պսեւդոհրամաններ): Հրամանագրի շարահյուսությունը հետևյալն է.

[անուն] <հրամանագրի_անվանում> [<օպերանդ[ներ]>] [; մեկնաբանություն]

Ինչպես տեսնում ենք, հրամանագրի կառուցվածքը հիմնականում համընկնում է ասեմբլերի հրամանի հետ: Միակ տարբերությունը այն է, որ անունից հետո ‘:’ չի օգտագործվում: <Հրամանագրի_անվանում>-ը, ինչպես և հրամանի հուշանունը, բանալի բառեր են:

Հրամանագրերն ունեն տարբեր շարահյուսություն՝ կախված օգտագործված ռեժիմից (MASM թե IDEAL): Ե՛վ Microsoft Assembler (masm.exe), և՛ Borland (Turbo) Assembler (tasm.exe) կոմպիլյատորներն ունեն MASM ռեժիմ, իսկ IDEAL ռեժիմ ունի միայն tasm կոմպիլյատորը:

Հրամանագրերի օգտագործումը կախված է ասեմբլեր-թարգմանչի ընտրությունից, ինչպես նաև թարգմանչի տարբերակից: Կդիտարկենք առավել հաճախ օգտագործվող հրամանագրերը, որոնց կարելի է օգտագործել և՛ tasm, և՛ masm թարգմանիչների համար:

Անունների սահմանման հրամանագրեր

EQU հրամանագիր

Շարահյուսություն: **<Անուն> EQU <օպերանդ>**

EQU հրամանագիրն օգտագործվում է հաստատուն սահմանելու նպատակով: Այս հրամանագիրն անվանը վերագրում է արժեք, որը որոշվում է աջակողմյան օպերանդով: Օպերանդը կարող է լինել կա՛մ հաստատուն, կա՛մ հաստատուն արտահայտություն, կա՛մ ցանկացած տեքստ, կա՛մ անուն:

Օրինակ 1

օպերանդը հաստատուն է կամ հաստատուն արտահայտություն:

```
N    EQU   100
K    EQU   2 * N + 1        ; K = 201
```

Օրինակ 2

օպերանդը տեքստ է:

```
T     EQU   [SI][BX + 20]  ; այս եղանակն օգտագործվում է հա-
                           ; ճախ հանդիպող երկար տողին կարճ
                           ; նշանակում տալու նպատակով

WP   EQU   WORD PTR
INC  WP    [BX]            ; համարժեք է INC WORD PTR [BX]

Msg  EQU   “HELLO World$”      ; Msg = “HELLO World$”
```

Օրինակ 3

օպերանդն անուն է:

```
A    DB    ?
B    EQU   A
C    DW    B               ; համարժեք է C  DW  A
```

EQU հրամանագիրը կրում է տեղեկատվական բնույթ, և այն չի ազդում մեքենայական կողի վրա: Այդ պատճառով այն կարող է տեղադրվել ասեմբլեր ծրագրի ցանկացած տեղում:

= հրամանագիր

Շարահյուսություն: **Անուն= <հաստատուն_ամբողջ_արտահայտություն>**

Այս հրամանագիրը նման է EQU հրամանագրին, սակայն, ի տարբերություն EQU հրամանագրի, անվան արժեքը հետագայում կարող է փոխվել:

Օրինակ

```
K = 10
A DW  K        ; համարժեք է 'A  DW  10'
K = K + 4
B DB   K       ; համարժեք է 'B  DW  14'
```

LABEL հրամանագիր

Շարահյուսություն: **<Անուն> LABEL <տիպ>**

Որտեղ <տիպ>-ը կարող է ընդունել BYTE, WORD, DWORD, FWORD, QWORD, TBYTE, NEAR, FAR արժեքները: Այս հրամանագիրը հնարավորություն է տալիս սահմանելու կամ վերասահմանելու փոփոխականի կամ նշիչի հետ կապված տիպը:

Օրինակ 1

```
ARRAY1      LABEL       WORD
 ARRAY2     DB          100 DUP(0)
```

Այստեղ ARRAY1-ը սահմանված է որպես 50-բառային զանգված, իսկ ARRAY2-ը՝ որպես 100-բայթանի զանգված. երկու զանգվածներն ունեն հիշողության միևնույն տիրույթը: Այսպիսով՝ հիշողության միևնույն տիրույթը կարող է դիտարկվել և՛ որպես բայթային (ARRAY1), և՛ որպես բառային (ARRAY2) զանգված:

Օրինակ 2

```
VAR3        LABEL       DWORD
WORD1       LABEL       WORD
BYTE1       DB          ?
BYTE2       DB          ?
WORD2       LABEL       WORD
BYTE3       DB          50H
BYTE4       DB          66H
```

Այս օրինակում կարելի է կատարել դիմում VAR3 կրկնակի բառի յուրաքանչյուր բառի կամ բայթի:

Սեգմենտի հրամանագիր

Սեգմենտի սահմանումն ունի հետևյալ տեսքը.

<Սեգմենտի_անուն> SEGMENT [հավասարեցում] [միավորում] [բիթայնություն] [դաս]

<նախադասություններ>

<Սեգմենտի_անուն> ENDS

Դիտարկենք SEGMENT հրամանագրի տրման դաշտերի նշանակությունը.

[հավասարեցում] (align) – սեգմենտի հավասարեցման դաշտն է, որը հաղորդում է կապերի խմբագրին (linker) սեգմենտի սկզբի հասցեի տեղադրության մասին: Ընդունում է հետևյալ հնարավոր արժեքները.

- **BYTE** – հավասարեցում չի կատարվում: Սեգմենտը կարող է սկսվել հիշողության կամայական հասցեից,
- **WORD** – սեգմենտը սկսվում է 2-ին պատիկ հասցեից (այսինքն՝ հավասարեցումը կատարվում է բառի սահմանից),
- **DWORD** – սեգմենտը սկսվում է 4-ին պատիկ հասցեից (այսինքն՝ հավասարեցումը կատարվում է կրկնակի բառի սահմանից),

- **PARA** – սեգմենտը սկսվում է 16-ին պատիկ հասցեից,
- **PAGE** – սեգմենտը սկսվում է 256-ին պատիկ հասցեից,
- **MEMPAGE** – սեգմենտը սկսվում է 4ԿԲ-ին պատիկ հասցեից (այսինքն՝ 16-ական ներկայացման վերջին 3 թվանշանները 0 են):

Լռությամբ հավասարեցման տիպը PARA է:

[Միավորում] (combine) – Միավորման դաշտն է, որը հաղորդում է կապերի խմբագրիչին, թե ինչպես պետք է միավորել նույն անուն ունեցող տարբեր մոդուլների սեգմենտները: Միավորման դաշտն ընդունում է հետևյալ հնարավոր արժեքները.

- **PRIVATE** – սեգմենտը չի միավորվում այլ մոդուլում գտնվող նույն անունով այլ սեգմենտների հետ,
- **PUBLIC** – ստիպում է կապերի խմբագրիչին միավորել նույն անունով բոլոր սեգմենտները: Նոր միավորված սեգմենտը ստացվում է անընդհատ: Օբյեկտների բոլոր հասցեները (շեղումները) հաշվարկվում են նոր սեգմենտի սկզբից,
- **COMMON** – նույն անունով բոլոր սեգմենտները տեղադրում է միևնույն հասցեից: Այսինքն՝ տվյալ անունով սեգմենտները համատեղ են օգտագործում հիշողության միևնույն տիրույթը: Արդյունքում միավորված սեգմենտն ունենում է ամենամեծ սեգմենտի չափը,
- **AT xxxx** – սեգմենտը տեղադրվում է պարագրաֆի՝ x*16 բացարձակ հասցեով (պարագրաֆը 16-ին պատիկ հիշողության տիրույթ է),
- **STACK** – որոշում է պահունակ սեգմենտ: Նման է PUBLIC-ին՝ բացառությամբ նրա, որ SS ռեգիստրը դառնում է պահունակ սեգմենտի սեգմենտային ռեգիստրը: SP ռեգիստրը ցույց է տալիս միավորված սեգմենտի վերջի վրա: Եթե ոչ մի պահունակ սեգմենտ սահմանված չէ, կապերի խմբագիրը նախազգուշացնում է, որ պահունակ սեգմենտը չի գտնվել:

Լռությամբ միավորման դաշտն ունի PRIVATE արժեք:

[Բիթայնություն] – Սեգմենտի բիթայնության դաշտը 386+ պրոցեսորների համար կարող է լինել 16 կամ 32-բիթանոց: Կարող է ընդունել հետևյալ արժեքները.

- **USE16** – նշանակում է սեգմենտի հասցեավորման եղանակը 16-բիթանի է: Ֆիզիկական հասցեի ձևավորման ժամանակ կարող է օգտագործվել միայն 16-բիթանոց շեղում:
- **USE32** – սեգմենտը 32-բիթանի է: Ֆիզիկական հասցեի ձևավորման համար օգտագործվում է 32-բիթանոց շեղում: Այդ պատճառով սեգմենտը կարող է պարունակել 4ԳԲ կոդ կամ տվյալներ:

[Դաս] - դասը չակերտների մեջ պարփակված սիմվոլների հաջորդականություն է: Այս դաշտն օգնում է կապերի խմբագրիչին որոշել սեգմենտների տեղադրման կարգը տարբեր մոդուլների սեգմենտները դասավորելիս: Կապերի խմբագրիչը միավորում է հիշողության մեջ նույն դասի բոլոր սեգմենտները: Դասի անունը կարող է լինել կամայական, սակայն լավ է, եթե այն արտացոլում է սեգմենտի նշանակությունը: Սովորաբար կոդ սեգմենտի համար օգտագործվում է 'code' անվանումը:

ENDS հրամանագրով նշվում է սեգմենտի վերջը:

SEGMENT և **ENDS** հրամանագրերը պետք է ունենան նույն <սեգմենտի_անուն>-ը:

GROUP հրամանագիր

Շարահյուսություն: **անուն** GROUP **սեգմենտ [, սեգմենտ]...**

Այս հրամանագիրը կարող է օգտագործվել սեգմենտները խմբավորելու համար: Խմբերը հնարավորություն են տալիս խմբի բոլոր սեգմենտներին դիմելու համար օգտագործել մեկ անվանում:

ASSUME հրամանագիր

Շարահյուսություն:

ASSUME սեգմենտային_ռեգիստր:արժեք,…

Կամ

ASSUME NOTHING

որտեղ սեգմենտային ռեգիստրը CS, DS, ES, SS, FS, GS ռեգիստրներից մեկն է, իսկ արժեքը կարող է լինել սեգմենտի անուն, խմբի անուն, SEG արտահայտություն կամ NOTHING:

Նշված հրամանագիրը կապում է սեգմենտային ռեգիստրը սեգմենտի, խմբի, կոնկրետ արտահայտության հետ կամ չեղարկում է տվյալ սեգմենտային ռեգիստրի արժեքը:

'ASSUME NOTHING' օգտագործելու դեպքում չեղարկվում է սեգմենտային ռեգիստրների և սեգմենտի միջև հայտարարված կապը:

Եթե օգտագործվել են սեգմենտի նկարագրության պարզեցված հրամանագրեր, ասեմբլերը լռությամբ նշանակում է ASSUME հրամանագիր, և հետագայում ASSUME չօգտագործելու դեպքում այդ արժեքները չեն փոխվում:

Հիշողության մոդելի և սեգմենտի սահմանման պարզեցված հրամանագրեր

Պարզ ծրագրերի համար, որոնք կազմված են միայն կոդ, տվյալների և պահունակ սեգմենտներից, հարմար է օգտագործել սեգմենտների նկարագրության պարզեցված հրամանագրերը (.DATA, .CODE, .STACK): Այս դեպքում ծրագրավորողը պարտադրված չէ նշելու սեգմենտի ատրիբուտները: Ծրագրի վերջը նշելու համար հրամանագիր չի օգտագործվում: Սեգմենտն ավարտվում է, երբ սկսվում է այլ սեգմենտի սահմանում: Սեգմենտի նկարագրության պարզեցված հրամանագրեր օգտագործելու համար նախ անհրաժեշտ է սահմանել հիշողության մոդելը .MODEL հրամանա-

գրի միջոցով: Այս հրամանագիրը ղեկավարում է սեգմենտների տեղաբաշխումը և իրականացնում ASSUME հրամանագրի ֆունկցիոնալությունը:

Շարահյուսություն:

.MODEL <հիշողությանՄոդել> [կոդ-սեգմենտի-անուն] [,<լեզու>] [,<մոդիֆիկատոր>]

<Հիշողության Մոդել> պարամետրը պարտադիր է: Այդ պարամետրի միջոցով է որոշվում ծրագրի սեգմենտների մոդելը:

Մոդել	Կոդի տիպ	Տվյալների տիպ	Մոդելի նշանակությունը
TINY	NEAR	NEAR	Կոդը և տվյալները միավորված են մեկ խմբում, որի անունը DGROUP է: Օգտագործվում է COM ձևաչափի ծրագրեր ստեղծելու համար:
SMALL	NEAR	NEAR	Կոդը զբաղեցնում է մեկ սեգմենտ, տվյալները միավորված են մեկ խըմբում, որի անունը DGROUP է:
MEDIUM	FAR	NEAR	Կոդը զբաղեցնում է մի քանի սեգմենտ: Ղեկավարումը փոխանցող բոլոր հղումները ունեն FAR տիպ: Տվյալները միավորված են մեկ խըմբում: Տվյալների վրա բոլոր հղումները ունեն NEAR տիպ:
COMPACT	NEAR	FAR	Կոդը զբաղեցնում է մեկ սեգմենտ: Տվյալների վրա բոլոր հղումները ունեն FAR տիպ:
LARGE	FAR	FAR	Ե՛վ կոդը, և՛ տվյալները զբաղեցնում են մի քանի սեգմենտ:
FLAT	NEAR	NEAR	Կոդը և տվյալները գտնվում են մեկ 32-բիթանոց սեգմենտում:

Աղյուսակ 1. Հիշողության մոդելի տեսակները

<Լեզու> պարամետրը թույլ է տալիս որոշ չափով դյուրացնել ասեմբլեր մոդուլն այլ ալգորիթմական լեզուներով գրված մոդուլների հետ միավորելուց առաջացող ինտերֆեյսային պրոբլեմները: Այս պարամետրի անհրաժեշտությունը կապված է տարբեր բարձր մակարդակի լեզուներում պրոցեդուրայի կանչի ժամանակ պարամետրերի փոխանցման հերթականության և պահունակը պարամետրերից ազատելու կանոնների տարբերության հետ: Այս պարամետրը կարող է ընդունել C, CPP, STDCALL, BASIC, PASCAL, FORTRAN, PROLOG արժեքները: Եթե այս պարամետրը նշված չէ, այն համարվում է NOLANGUAGE:

Օրինակ

```
.MODEL    Large, C
.MODEL    Small, Pascal
```

<Մոդիֆիկատոր>-ը կարող է ընդունել NEARSTACK, FARSTACK, USE16, USE32, DOS, OS2 արժեքներ: Այս ոչ պարտադիր պարամետրը կարող է փոխել մոդելի որոշ հատկություններ, ընդ որում՝ կարելի է օգտագործել մի քանի մոդիֆիկատորներ միաժամանակ:

NEARSTACK – ցույց է տալիս, որ պահունակ սեգմենտը կներառվի DGROUP խմբում (եթե DGROUP խումբը կա) և SS-ը ցույց կտա DGROUP-ը,

FARSTACK – պահունակ սեգմենտը չպետք է ներառվի DGROUP-ում, և SS-ը չի սկզբնարժեքավորվի,

USE16 – ցույց է տալիս (եթե ընտրվել է 80386+ պրոցեսորը), որ ընթացիկ մոդելի բոլոր սեգմենտները 16-բիթանի են,

USE32 – ցույց է տալիս, որ ընթացիկ մոդելի բոլոր սեգմենտները 32-բիթանի են,

DOS, OS_DOS – ցույց է տալիս, որ տվյալ ծրագիրը նախատեսված է DOS օգտագործման համար,

OS2, OS_OS2 – ցույց է տալիս, որ տվյալ ծրագիրը նախատեսված է OS/2-ի օգտագործման համար:

Լռությամբ TASM-ը օգտագործում է NEARSTACK DOS և USE16 (USE32, եթե ընտրվել է 80386+ պրոցեսորը):

Ոչ պարտադիր <***կոդ-սեգմենտի-անուն***> պարամետրը կարող է օգտագործվել LARGE մոդելում՝ լռությամբ ընդունված անվանումը փոխելու նպատակով:

Պարզեցված հրամանագրերով սեգմենտների սահմանման դեպքում ծրագիրը կարող է ունենալ միայն սահմանված տիպի սեգմենտներ: Սեգմենտների սահմանման հրամանագրերը տրված են Աղյուսակ 2-ում:

Հրամանագրի ձևաչափը	**Նշանակությունը**
.CODE [անուն]	Կոդ սեգմենտի սկիզբն է կամ շարունակությունը:
.DATA	Սկզբնարժեքավորված տվյալների սեգմենտի սկիզբն է կամ շարունակությունը: Օգտագործվում է նաև NEAR տիպի տվյալների համար:
.CONST	Ծրագրի հաստատունների սեգմենտի սկիզբն է կամ շարունակությունը:
.DATA?	Չսկզբնարժեքավորված տվյալների սեգմենտի սկիզբն է կամ շարունակությունը: Օգտագործվում է նաև NEAR տիպի տվյալների համար:
.STACK [չափ]	Ծրագրի պահունակ սեգմենտի սկիզբն է կամ շարունակությունը: [Չափ] պարամետրի միջոցով տրվում է պահունակի չափը:
.FARDATA [անուն]	FAR տիպի սկզբնարժեքավորված տվյալների սեգմենտի սկիզբն է կամ շարունակությունը:
.FARDATA? [անուն]	FAR տիպի չսկզբնարժեքավորված տվյալների սեգմենտի սկիզբն է կամ շարունակությունը:

Աղյուսակ 2. Սեգմենտի հայտարարման պարզեցված հրամանագրեր

Որոշ սեգմենտների նկարագրությունը հնարավորություն է տալիս ունենալու մի քանի անվանական սեգմենտներ:

.MODEL հրամանագիրն օգտագործելու դեպքում թարգմանիչը ստեղծում է մի քանի նախապես սահմանված անուններով հաստատուններ (օրինակ՝ @DATA – տվյալների սեգմենտի հասցեն):

Պրոցեսորի հրամանագրեր

Լռությամբ ասեմբլեր թարգմանիչը օգտագործում է 8086 պրոցեսորի հրամանների համակարգը և տալիս է հաղորդագրություն սխալի մասին, եթե օգտագործվել է հրաման, որը այդ պրոցեսորը չի սատարում: Այլ պրոցեսորներում ի հայտ եկած հրամանների օգտագործման թույլտվության համար առաջարկվում են ասեմբլերի հետևյալ հրամանագրերը.

***.8086**- օգտագործվում է լռությամբ: Թույլատրում է օգտագործել միայն 8086-ի հրամանները,*
***.186**- թույլատրվում են 80186-ի հրամանները,*
*.**286** և .**286c**- թույլատրվում են 80286-ի ոչ արտոնյալ հրամանները,*
***.286p**- թույլատրվում են 80286-ի բոլոր հրամանները,*
*.**386** և .**386c**- թույլատրվում են 80386-ի ոչ արտոնյալ հրամանները,*
***.386p**- թույլատրվում են 80386-ի բոլոր հրամանները,*
***.486** և .**486c**- թույլատրվում են 80486-ի ոչ արտոնյալ հրամանները,*
***.486p**- թույլատրվում են 80486-ի բոլոր հրամանները,*
*.**586** և **.586c**- թույլատրվում են P5 (Pentium) ոչ արտոնյալ հրամանները,*
***.586p**- թույլատրվում են P5 (Pentium)-ի բոլոր հրամանները,*
***.686** – թույլատրվում են P6 (Pentium Pro, Pentium II)-ի ոչ արտոնյալ հրամանները,*
***.686p**- թույլատրվում են P6 (Pentium Pro, Pentium II)-ի բոլոր հրամանները,*
***.8087** – թույլատրվում են 8087 կից պրոցեսորի հրամանները,*
***.MMX** – թույլատրվում են IA MM հրամանները,*
***.K3D** – թույլատրվում են AMD 3D հրամանները:*

Պայմանական թարգմանության հրամանագրեր

Պայմանական թարգմանության հրամանագրերը թույլ են տալիս ղեկավարել ասեմբլեր թարգմանության գործընթացը: Կախված պայմանից՝ ասեմբլերը կարող է ընտրել կամ բաց թողնել նախադասությունների մի ամբողջ բլոկ:

IFxxx հրամանագրի շարահյուսությունը

```
    IFxxx
      True-բլոկի-մարմին
    EMDIF
```

կամ

```
    IFxxx
     True-բլոկի-մարմին
    ELSE
      False-բլոկի-մարմին
    ENDIF
```

Որտեղ IFxxx կարող է լինել հետևյալ հրամանագրերից որևէ մեկը.
IF, IFDEF, IFNDEF, IFB, IFNB, IFIDN, IFIDNI, IFDIF, IFDIFI:

IFxxx-ում տրվում է կոնկրետ պայման: Եթե պայմանը տեղի ունի, ապա թարգմանությանը մասնակցում է True-բլոկի-մարմին-ը, հակառակ դեպքում՝ False-բլոկի-մարմին-ը (ELSE-ով տարբերակի դեպքում:

Օրինակ

```
number   EQU   253
IF       number   LE   65535
  stroka DB      32 DUP(30h)
  endstr DB      0
ENDIF
IFNDEF stroka
  stroka DB      16 DUP(30h)
  endstr DB      0
ENDIF
```

Տվյալ օրինակում թարգմանության կենթարկվի հետևյալ հատվածը.

```
stroka      DB      32 DUP(30h)
endstr      DB      0
```

xxx-ից կախված՝ պայմանները տրվում են տարբեր եղանակներով: Ստորև բերված են IFxxx-ի տրման շարահյուսությունը և բացատրություն, թե երբ պայմանը տեղի ունի:

IF հաստատուն-արտահայտություն – պայմանը տեղի ունի, եթե արտահայտության արժեքը 0 չէ:

IFDEF անուն – պայմանը տեղի ունի, եթե անունը սահմանված է:

IFNDEF անուն – պայմանը տեղի ունի, եթե անունը սահմանված չէ:

IFDIF արգումենտ_1, արգումենտ_2 – պայմանը տեղի ունի, եթե արգումենտ_1-ը և արգումենտ_2-ը **տարբեր (**different) են:

IFDIFI արգումենտ_1, արգումենտ_2 – պայմանը տեղի ունի, եթե արգումենտ_1-ը և արգումենտ_2-ը տարբեր են՝ առանց հաշվի առնելու մեծատառ-փոքրատառը (օրինակ՝ “IFDIFI <EAX>,<eaX>” պայմանը տեղի ունի):

IFIDN արգումենտ_1, արգումենտ_2 – պայմանը տեղի ունի, եթե արգումենտ_1-ը և արգումենտ_2-ը **նույնն** (identical) են:

IFIDNI արգումենտ_1, արգումենտ_2 – պայմանը տեղի ունի, եթե արգումենտ_1-ը և արգումենտ_2-ը նույնն են՝ առանց հաշվի առնելու մեծատառ-փոքրատառը:

IFB արգումենտ – պայմանը տեղի ունի, եթե արգումենտը դատարկ է (blank):

IFNB արգումենտ – պայմանը տեղի ունի, եթե արգումենտը դատարկ չէ:

Արգումենտները գրվում են <> նշանների մեջ:

Արգումենտներով տրված հրահանգներն օգտագործվում են մակրոսահմանումներում փոխանցված պարամետրերի/ արգումենտների արժեքները վերահսկելու նպատակով:

Սխալի գեներացման հրամանագիր (.ERR)

Այս հրամանագիրը հանդիպելիս ասեմբլերը սխալի մասին հաղորդագրություն է տալիս էկրանին և լիսթինգի ֆայլում ու կանխում է օբյեկտային կոդի ֆայլի ստեղծումը:

Շարահյուսություն: .ERR [հաղորդագրություն]

Օրինակ

Եթե ընթացիկ հաշվիչի արժեքը գերազանցում է 64ԿԲ-ը, ապա թարգմանությունը չի շարունակվի, և կտրվի հաղորդագրություն սխալի մասին:

```
IF      $    GT   65535
.ERR
ENDIF
```

Կան նաև .ERR հրամանագրի պայմանով տարատեսակներ (.ERRDEF, .ERRB …)՝ IF-ի բոլոր դեպքերի նման:

INCLUDE հրամանագիր

Շարահյուսություն: **INCLUDE filename**

INCLUDE հրամանագրում նշված ֆայլի պարունակությունը զետեղվում է հրամանագրի տեղում:

Օրինակ

```
INCLUDE    d:\tasm\macro.asm
INCLUDE    MyMacro.asm
```

PUBLIC, EXTERN/EXTRN հրամանագրեր

Այս հրամանագրերն օգտագործվում են, երբ անհրաժեշտություն է լինում մի մոդուլում հայտարարված պրոցեդուրան, փոփոխականը կամ հաստատունը օգտագործելու այլ մոդուլում: Մոդուլ ասելով՝ կհասկանանք առանձին թարգմանության ենթակա ծրագրի միավոր: Այն անունները, որոնք պետք է օգտագործվեն այլ մոդուլներում, հայտարարվում են PUBLIC, իսկ անունները, որոնք օգտագործվել են տվյալ մոդուլում, բայց սահմանվել են այլ մոդուլում՝ հայտարարվում են որպես EXTRN (external – արտաքին):

Շարահյուսություն: **PUBLIC [լեզու] անուն [, [լեզու] անուն]...**

EXTRN [լեզու] անուն:տիպ[,...]

PUBLIC-ը ցույց է տալիս, որ հայտարարված անունը կարող է օգտագործվել այլ մոդուլներում:

EXTRN-ը ցույց է տալիս, որ անունը սահմանվել է այլ մոդուլում:

«Լեզու»-ն (C, PASCAL, BASIC, FORTRAN, ASSEMBLER կամ PROLOG) ցույց է տալիս, որ տվյալ անվան նկատմամբ պետք է կիրառվեն տվյալ լեզվի համաձայնեցումները:

«Տիպ»-ը կարող է ընդունել NEAR, FAR, PROC, BYTE, WORD, DWORD, DATAPTR, CODEPTR, FWORD, PWORD, QWORD, TBYTE, ABS կամ կառուցվածքի անուն (STRUC) արժեքները:

Օրինակ

Ենթադրենք, ծրագիրը բաղկացած է երկու ֆայլերից: Մի ֆայլը պարունակում է մի շարք փոփոխականներ, որոնք օգտագործվելու են երկրորդում, իսկ երկրորդը օգտագործում է այդ փոփոխականները՝ չհմանալով, թե որտեղ են դրանք սահմանված:

```
;Մոդուլ #1 ֆայլի անունը = f1.asm: tasm f1
.MODEL      SMALL
PUBLIC      Var1, Var2, Proc1
.DATA
```

```
    Var1    WORD    ?
    Var2    WORD    ?
.CODE
PROC1   PROC        NEAR
        MOV         AX, Var2
        ADD         Var1, AX
        RET
PROC1   ENDP
END

; Մողուլ #2: ֆայլի անունը = f2.asm: tasm f2
.MODEL  SMALL
EXTRN   Var1:WORD, Var2:WORD, Proc1:NEAR
.CODE
        MOV         Var1, 2
        MOV         Var2, 3
        CALL        Proc1
        ...
END
```

Մողուլ #2-ը հղում է կատարում Var1, Var2 և Proc1 անուններին, այնինչ դրանք արտաքին են: Հետևաբար անհրաժեշտ է դրանք հայտարարել extern հրամանագրի օգնությամբ:

f1.asm ֆայլի թարգմանության արդյունքում կստացվի f1.obj ֆայլը, իսկ f2.asm ֆայլի թարգմանության արդյունքում՝ f2.obj-ն: f.exe կատարվող ֆայլը ստանալու համար կարելի է օգտագործել հետևյալ հրամանը՝

```
Tlink f1.obj  f2.obj f.exe
```

ՊՐՈՑԵԴՈՒՐԱ

Ասեմբլերում պրոցեդուրան ծրագրի հատված (ենթածրագիր) է, որն իրականացնում է որոշակի խնդիր: Պրոցեդուրաները օգտագործվում են.

- ծրագրի կառուցվածքն ավելի պարզ և իմաստալի դարձնելու նպատակով,
- ծրագրի ֆունկցիոնալությունն ավելի հեշտ թեստավորելու նպատակով,
- կրկնություններից խուսափելու նպատակով:

C++-ում ֆունկցիան կատարում է նմանատիպ դեր, ինչ պրոցեդուրան ասեմբլերում:

Պրոցեդուրայի նկարագրություն

Պրոցեդուրայի նկարագրությունն ունի հետևյալ տեսքը.

<*պրոցեդուրայի-անուն*> **PROC** <*պարամետրերի ցուցակ*>

<*պրոցեդուրայի մարմին*>

<*պրոցեդուրայի-անուն*> **ENDP**

PROC և ENDP հրամանագրերը նշում են պրոցեդուրայի նկարագրության սկիզբն ու վերջը: <Պրոցեդուրայի-անուն>-ը նույնարկող է:

<Պրոցեդուրայի մարմին>-ը բաղկացած է պրոցեդուրայում կատարվող հրամաններից: Վերջին կատարվող հրամանը RET հրաման է, որն ապահովում է պրոցեդուրայից վերադարձ հիմնական ծրագիր:

8086 պրոցեսորում <պարամետրերի ցուցակ>-ը պարունակում էր միայն 1 պարամետր՝ <տիպ>:

<Տիպ>-ն ունի 2 հնարավոր արժեք՝ NEAR և FAR, լռությամբ արժեքը՝ NEAR: NEAR-ի դեպքում պրոցեդուրայի կանչն ու նկարագրությունը պետք է լինեն նույն սեգմենտում, այսինքն՝ NEAR տիպ ունեցող պրոցեդուրան կարող է կանչվել միայն այն սեգմենտում, որտեղ նկարագրված է: FAR-ի դեպքում պրոցեդուրայի կանչն ու նկարագրությունը կարող են լինել նաև տարբեր սեգմենտներում, այսինքն՝ FAR տիպ ունեցող պրոցեդուրան կարող է կանչվել ոչ միայն այն սեգմենտում, որտեղ նկարագրված է:

Ներկայիս ասեմբլեր թագմանիչները, բացի <տիպ> պարամետրից, PROC հրամանագրի համար ապահովում են լրացուցիչ պարամետրեր, որոնք ծրագրորդին հնարավորություն են տալիս խուսափելու ավելորդ գործողություններից:

Ընդհանուր դեպքում <*Պարամետրերի ցուցակ*>-ն ունի հետևյալ տեսքը.

<լեզվի մոդիֆիկատոր> <լեզու> <տիպ> <ARG արգումենտների_ցուցակ> <RETURNS արժեքների_ցուցակ> <LOCAL արգումենտների_ցուցակ> <USES արժեքների_ցուցակ>

Բոլոր պարամետրերն էլ ոչ պարտադիր են:

Լեզվի մոդիֆիկատոր – հայտնում է ասեմբլերին, որ պրոցեդուրայի մարմնի մեջ պետք է ընդգրկել պրոցեդուրայի սկզբի և ավարտի հատուկ ծրագրի կոդ, որը կապահովի WINDOWS-ի կամ օվերլեյների կառավարչի՝ VROOM (overlay manager) հետ կապը (interface): Կարող է ունենալ հետևյալ արժեքները՝ NORMAL, WINDOWS, ODDNEAR, ODDFAR: Եթե պրոցեդուրայի նկարագրության մեջ լեզվի մոդիֆիկատորը բացակայում է, ապա ասեմբլերը վերցնում է .MODEL հրամանագրում հայտարարված լեզվի մոդիֆիկատորի արժեքը, իսկ .MODEL հրամանագրի բացակայության կամ NORMAL արժեքի դեպքում ասեմբլերը գեներացնում է պրոցեդուրայի սկզբի և վերջի ստանդարտ կոդ:

Լեզու – այս պարամետրը կարող է ունենալ հետևյալ արժեքները՝ NOLANGUAGE (ASSEMBLER), BASIC, PROLOG, FORTRAN, C, (C++), PASCAL: Ասեմբլերը լեզվի նշված արժեքով որոշում է պոցեդուրայի սկզբի և վերջի ավտոմատ ավելացվող լրացուցիչ կոդը, որը կազմակերպում է պրոցեդուրային պահունակի միջոցով փոխանցվող արգումենտների և լոկալ փոփոխականների հետ աշխատանքը: Լռությամբ և NOLANGUAGE արժեքի դեպքում լրացուցիչ կոդ չի ավելացվում, բայց

այս դեպքում պահունակով պրոցեդուրային արժեքներ փոխանցելը և պահունակից դրանք հեռացնելը կատարում է ծրագրողը: BASIC, FORTFAN,PASCAL արժեքների դեպքում արգումենտները պրոցեդուրային փոխանցվում են ձախից աջ հերթականությամբ և պրոցեդուրայից կանչող ծրագիր վերադառնալուց պահունակից արգումենտները պետք է հանի պրոցեդուրան, իսկ C, CPP և PROLOG արժեքների դեպքում արգումենտները պրոցեդուրային են փոխանցվում աջից ձախ հերթականությամբ, և պրոցեդուրայից կանչող ծրագիր վերադառնալիս պահունակից արգումենտները պետք է հանի կանչող ծրագիրը:

ARG արգումենտների_ցուցակ – այստեղ նշվում են պրոցեդուրային փոխանցվող արժեքները:

RETURNS արժեքների_ցուցակ – այստեղ նշվում են պրոցեդուրայից կանչող ծրագրին վերադարձվող արժեքները:

LOCAL արգումենտների_ցուցակ – այստեղ նշվում են պրոցեդուրայի լոկալ փոփոխականները:

USES արժեքների_ցուցակ (USES արժեք1 արժեք2 ...) – պրոցեդուրայի սկզբում պահունակի մեջ է գրում, իսկ պրոցեդուրայի վերջում՝ պահունակից հանում *արժեքների_ցուցակում* նշված ռեգիստրների արժեքները: USES-ի միջոցով կարող ենք պրոցեդուրայում օգտագործել ռեգիստրներ՝ առանց կանչող ծրագրում նրանց փոխվելու: Օրինակ՝ USES CX AX SI–ով կարող ենք պրոցեդուրայում օգտագործել CX AX և SI ռեգիստրները, բայց կանչող ծրագրում նրանց նախնական արժեքները չեն փոխվի:

Նշում 1. USES պարամետրը կարելի է օգտագործել միայն այն պրոցեդուրաներում, որտեղ տրված է լեզուն .MODEL հրամանագրի կամ պրոցեդուրայի «լեզու» պարամետրի միջոցով (NOLANGUAGE արժեքը չի ընդունվում):

Նշում 2. Պարամետրերի հնարավոր արժեքները տարբեր ֆիրմաների ասեմբլերներներում (TASM,MASM և այլն) կամ ասեմբլերների տարբեր տարբերակներում կարող են տարբերվել:

Պրոցեդուրայի նկարագրությունը կարող է գտնվել ծրագրում հրամանային սեգմենտի մեջ ցանկացած տեղ, միայն թե պետք է ապահովել, որ պրոցեդուրայի հրամանները սկսեն կատարվել միայն պրոցեդուրայի կանչ հանդիպելիս: Եթե այն այլ պրոցեդուրայի մեջ է, ապա դա կարելի է անել լրացուցիչ JMP հրամանի միջոցով, որով կապահովվի անցում պրոցեդուրայի նկարագրության վրայով:

```
.CODE
    …
    JMP    X
PR_NAME  PROC
    …
PR_NAME  ENDP
X:
    …
    CALL  PR_NAME
    …
    MOV   AH, 4CH
    INT    21H
END
```

Բայց սովորաբար պրոցեդուրայի նկարագրությունը ծրագրի հրամանային սեգմենտում տեղադրում են հիմնական մասից առաջ կամ հետո:

; հիմնական մասից առաջ

```
.CODE
PR_NAME   PROC
        …
PR_NAME   ENDP

        …
        CALL   PR_NAME
        …
        MOV   AH, 4CH
        INT    21H
END
```

; հիմնական մասից հետո

```
.CODE
…
CALL   PR_NAME
…
MOV    AH, 4CH
INT    21H

PR_NAME   PROC
…
PR_NAME   ENDP
END
```

Եթե ծրագիրը մեծ է և պրոցեդուրաների քանակը շատ, ապա սովորաբար պրոցեդուրաների նկարագրությունները գրվում են առանձին ֆայլում, իսկ հիմնական ծրագրին կցվում է այդ ֆայլը INCLUDE հրամանագրով:

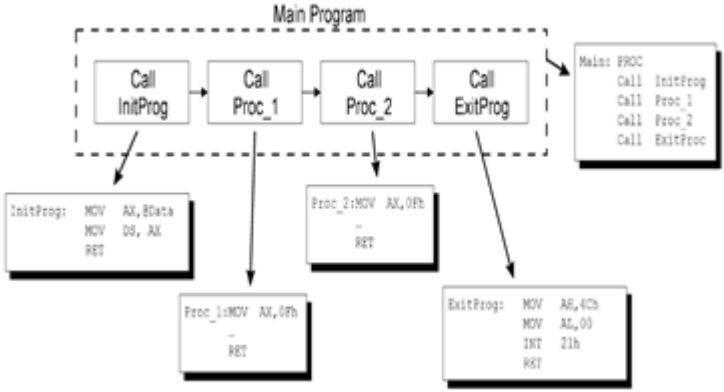
Նշում

Հաճախ օգտագործվող պրոցեդուրաները (օրինակ՝ մուտքի/ելքի հետ կապված) կարելի է դարձնել առանձին մոդուլ (թարգմանության միավոր), ստանալ նրանց օբյեկտային կոդը (.obj) և ստացված օբյեկտային կոդը կցել տարբեր ծրագրերի:

Ասեմբլերը թույլատրում է պրոցեդուրաների ներդրվածություն, սակայն խորհուրդ չի տրվում օգտագործել ներդրված պրոցեդուրաներ, քանի որ այս մոտեցման դեպքում խախտվում է ծրագրի կառուցվածքայնությունը:

Պրոցեդուրայի կանչ

Պրոցեդուրան աշխատեցնելու համար հիմնական ծրագրում գրում ենք պրոցեդուրայի կանչի CALL հրամանը: Պրոցեդուրայի NEAR տիպի դեպքում ասեմբլերը CALL հանդիպելիս կատարում է մոտիկ կանչ. պահունակի մեջ գրում է IP/EIP ռեգիստրի արժեքը, իսկ FAR տիպի դեպքում կատարվում է հեռու կանչ. պահունակի մեջ են գրվում CS և IP/EIP ռեգիստրների արժեքները: Այսպիսով՝ պահունակի մեջ գրվում է CALL-ին հաջորդող հրամանի հասցեն՝ վերադարձի հասցեն: Ապա անցում է կատարվում պրոցեդուրային՝ IP/EIP կամ CS և IP/EIP՝ ռեգիստրներին վերագրելով պրոցեդուրայի սկզբի հասցեն: Արդյունքում կատարվում է ծրագրի ղեկավարության փոխանցում հիմնական ծրագրից պրոցեդուրային, այսինքն՝ ծրագրում հաջորդ կատարվող հրամանը լինում է պրոցեդուրայի առաջին հրամանը:



Օրինակ

CALL PROC_NAME հրամանի դեպքում կկատարվի հետևյալը.

NEAR	**FAR**
1. IP/EIP → STACK	1. CS → STACK
2. IP/EIP = OFFSET PROC_NAME	2. IP/EIP → STACK
	3. CS = SEG PROC_NAME
	4. IP/EIP = OFFSET PROC_NAME

Վերադարձ պրոցեդուրայից հիմնական ծրագիր

Պրոցեդուրայից հիմնական ծրագիր վերադառնալու համար պրոցեդուրայի մարմնում գրում ենք ‘RET’ կամ ‘RET K’ հրամանը, որը պետք է լինի պրոցեդուրայի վերջին կատարվող հրամանը: K-ն թիվ է: Լրացուցիչ K պարամետրն օգտագործվում է այն դեպքում, երբ պրոցեդուրային պարամետրեր են փոխանցվել պահունակով, և հիմնական ծրագիր վերադառնալիս պետք է պահունակից հանել այդ պարամետրերը (K-ն պարամետրերի զբաղեցրած բայթերի քանակն է): Կախված պրոցեդուրայի տեսակից՝ RET հրամանը կվո-

խարինվի RETN (NEAR տիպի դեպքում) կամ RETF (FAR տիպի դեպքում) հրամանով և կկատարվեն հետևյալ գործողությունները.

NEAR	**FAR**
1. STACK → IP/EIP	1. STACK → IP/EIP
2. Եթե K-ն տրված է, ապա SP/ESP ← SP/ESP + K	2. STACK → CS
	3. Եթե K-ն տրված է, ապա SP/ESP ← SP/ESP + K

Քանի որ մինչ այդ CALL հրամանով պահունակի մեջ էին գրվել IP/EIP (NEAR տիպի դեպքում) կամ CS և IP/EIP (FAR տիպի դեպքում) ռեգիստրների արժեքները, ապա RET հրամանի այս գործողությունները կապահովեն ղեկավարության փոխանցում պրոցեդուրայից հիմնական ծրագիր, այսինքն՝ ծրագրում հաջորդ կատարվող հրամանը կլինի ծրագրում CALL հրամանին հաջորդող հրամանը:

Պարամետրերի փոխանցում պրոցեդուրային

Պրոցեդուրային պարամետրեր կարելի է փոխանցել.

- **ըստ արժեքի** – պրոցեդուրային փոխանցվում է պարամետրի արժեքը, ավելի ստույգ՝ արժեքի պատճենը և պրոցեդուրայի մեջ պարամետրի արժեքի փոփոխությունը չի ազդում հիմնական ծրագրում այդ պարամետրի ունեցած արժեքի վրա,
- **ըստ հղման** – պրոցեդուրային փոխանցվում է պարամետրի հասցեն, որտեղից պրոցեդուրան կարդում է պարամետրի արժեքը,
- **ըստ վերադարձվող արժեքի** – սա նախորդ երկուսի միավորումն է. պրոցեդուրային փոխանցվում է պարամետրի հասցեն, պրոցեդուրան կարդում է այդ հասցեից պարամետրի արժեքը, աշխատում նրա հետ և արդյունքը գրանցում նույն հասցեով,

- **ըստ արդյունքի** – պրոցեդուրային փոխանցվում է հասցե, որտեղ պրոցեդուրան գրում է իր աշխատանքի արդյունքը: Այս տարբերակը նախորդից տարբերվում է նրանով, որ փոխանցված հասցեից պրոցեդուրան արժեք չի կարդում,
- **ծրագրի կոդի մեջ** – պրոցեդուրային փոխանցվող պարամետրերը գտնվում են անմիջապես կոդ սեգմենտի մեջ՝ պրոցեդուրայի կանչի CALL հրամանից անմիջապես հետո: Այդ պարամետրերի հասցեն գրվում է պահունակի մեջ որպես վերադարձի կետի հասցե և հասանելի է BP/EBP ռեգիստրով: Այս դեպքում պրոցեդուրան աշխատանքն ավարտելուց առաջ պետք է փոխի վերադարձի կետի արժեքը՝ դարձնելով այն պարամետրերից հետո ծրագրում առաջինի հասցեն:

Պարամետրեր պրոցեդուրային կարելի է փոխանցել 3 եղանակով՝

- ռեգիստրի միջոցով,
- հիշողության (փոփոխականների) միջոցով,
- պահունակի միջոցով:

Փոխանցում ռեգիստրի միջոցով – այս եղանակով (ինչպես DOS կանչերում) տվյալները (արժեք, ցուցիչ) պրոցեդուրային փոխանցելու համար օգտագործվում են պրոցեսորի ռեգիստրները: Ռեգիստրով փոխանցելը ամենապարզ ձևն է, բայց ռեգիստրների սահմանափակ քանակի պատճառով այս ձևով հնարավոր չէ փոխանցել շատ պարամետրեր: Մեծածավալ տվյալների փոխանցման համար կարելի է օգտագործել ցուցիչներ:

Փոխանցում հիշողության միջոցով – այս եղանակը համարժեք է գլոբալ փոփոխականներ օգտագործելուն: Այս եղանակը խորհուրդ չի տրվում օգտագործել, քանի որ առկա են գլոբալ փոխխականների օգտագործման բոլոր վտանգները:

Փոխանցում պահունակի միջոցով – այս եղանակի օգտագործման դեպքում նախքան պրոցեդուրայի կանչը պարամետրերը ուղարկվում են պահունակ, իսկ պրոցեդուրան օգտագործում է պահունակը պարամետրերին դիմելու համար: Պահունակին դիմելուց անհրաժեշտ է ի նկատի ունենալ, որ վերադարձի հասցեն պրոցեդուրա-

յի կատարման ժամանակ գտնվում է պահունակում (այդ պատճառով պարամետրերին հասանելիությունը պրոցեդուրայում ապահովվում է բազային ցուցիչ (BP/EBP) ռեգիստրի միջոցով, որը հենց նախատեսված է պահունակի կաղրի հետ աշխատելու համար): Պահունակի միջոցով պարամետրերի փոխանցումը C լեզվում պարամետրերի փոխանցման ամենատարածված եղանակն է:

Քանի որ պրոցեդուրայի կանչից հետո պահունակում գրվում է վերադարձի հասցեն՝ IP/EIP կամ CS և IP/ EIP, ապա պարամետրերին հասանելիությունը պրոցեդուրայում կարելի է ապահովվել BP/EBP ռեգիստրով հետևյալ կերպ.

1. BP/EBP-ի արժեքը պահվում է պահունակում,
2. BP/EBP-ին վերագրվում է SP/ESP-ի արժեքը,
3. 16-բիթանոց պահունակի դեպքում BP-ն ավելացվում է 4 կամ 6-ով՝ կախված պրոցեդուրայի տիպից՝ NEAR / FAR (4 բայթ՝ NEAR-ի դեպքում, որպեսզի անցում կատարվի պահունակում գրված IP-ի և BP-ի արժեքների վրայով, 6 բայթ՝ FAR-ի դեպքում, որպեսզի անցում կատարվի պահունակում գրված IP-ի, CS-ի և BP-ի արժեքների վրայով): 32-բիթանոց պահունակի դեպքում EBP-արժեքը ավելանում է 8-ով (NEAR) կամ 10-ով (FAR) համապատասխանաբար,
4. [BP]-ում է պրոցեդուրային փոխանցված վերջին պարամետրը, [BP + 2/4]-ում՝ նախավերջին պարամետրը և այսպես շարունակ: Պրոցեդուրան կարդում է դրանք և աշխատում:

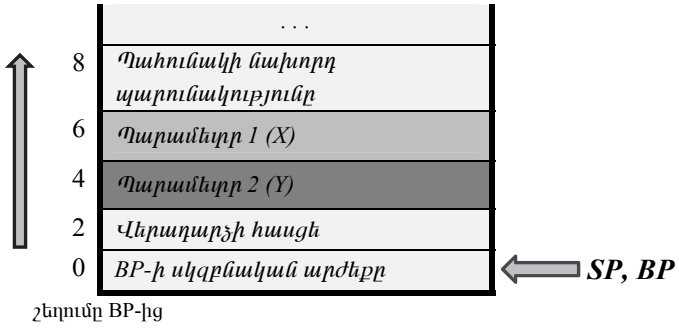
Պրոցեդուրայի աշխատանքի ավարտին պետք է պահունակից հանել ոչ միայն IP/EIP-ի կամ CS-ի և IP/ EIP-ի արժեքները, այլև պրոցեդուրային փոխանցված պարամետրերը: Դա արվում է RET K հրամանով:

Օրինակ 1

Պարամետրերի փոխանցում ըստ արժեքի, պահունակով:

MAX պրոցեդուրան պահունակով ստանում է երկու առանց նշանի բառային արժեքներ և վերադարձնում նրանցից մեծի արժեքը AX ռեգիստրում.

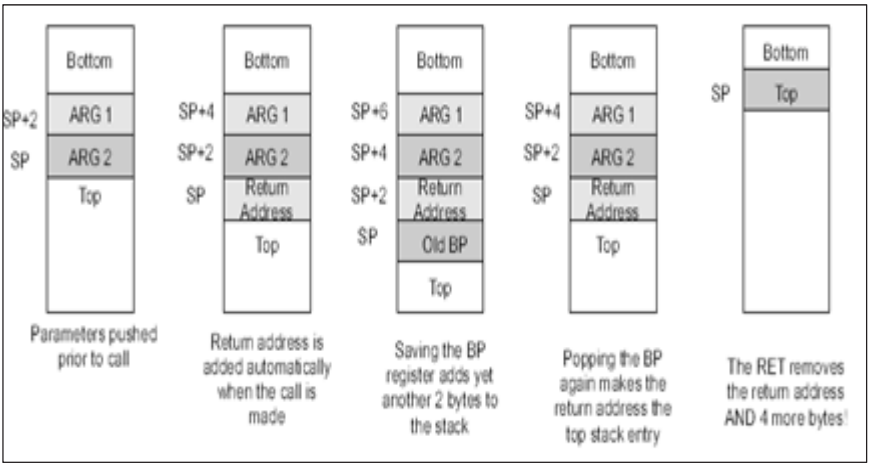
```
MAX             PROC
; Ստանալ պահունակով փոխանցված 2 առանց նշանի
; բառային արժեքներից առավելագույնը
; Մուտք – 1-ին պարամետր՝ 2 բայթ՝ պահունակում (X)
;               2-րդ պարամետր՝ 2 բայթ՝ պահունակում (Y)
; ելք  –  AX   (AX = max (X, Y))
                PUSH        BP
                MOV         BP, SP          ; պահունակի կաղրը տես
                                            ; Նկար 1-ում
                PUSH        BX
                MOV         AX, [BP + 4]    ; [BP + 4] - ում Y փոփո-
                                            ; խականի արժեքն է
                MOV         BX, [BP + 6]    ; [BP + 6] - ում X փոփո-
                                            ; խականի արժեքն է
                CMP         AX, BX
                JAE         L1
                MOV         AX, BX
        L1:     POP         BX
                POP         BP
                RET         4
MAX             ENDP
```



Նկար 1. Պահունակի կաղրը՝ MOV BP, SP հրամանից հետո

MAX պրոցեդուրայի կանչը կարելի է իրականացնել հետևյալ կերպ.

```
.DATA
        X       DW      50
        Y       DW      70
.CODE
        …
        PUSH    X
        PUSH    Y
        CALL    MAX
        …
```



Նկար 2. Պահունակի կադրերն ըստ կատարվող գործողությունների

Օրինակ 2

Պարամետրի փոխանցում ըստ արդյունքի, ռեգիստրով:

INPUT պրոցեդուրան ներածում է սիմվոլային տող: Պրոցեդուրան որպես պարամետր DX-ում ստանում է հասցե, որտեղ պահում է ներածված տողը:

```
INPUT           PROC
; մուտք - ներածվող տողի հասցեն DX-ում
; ելք – ներածված տողը, որի հասցեն տրված էր DX-ում
```

```
                PUSH        AX
                MOV         AH, 10
                INT         21H
                POP         AX
                RET
INPUT           ENDP
```

INPUT պրոցեդուրայի կանչը կարելի է իրականացնել հետևյալ կերպ.

```
.DATA
                String  DB      10, 11 dup(?)
.CODE
                …
                LEA         DX, string
                CALL        INPUT
                …
```

Օրինակ 3

Պարամետրի փոխանցում ըստ հղման, ռեգիստրով:

OUTPUT պրոցեդուրան արտածում է սիմվոլային տող: Պրոցեդուրան որպես պարամետր DX-ում ստանում է արտածվող տողի հասցեն:

```
OUTPUT          PROC
; մուտք - արտածվող տողի հասցեն DX-ում
; ելք - -
                PUSH        AX
                MOV         AH, 9
                INT         21H
                POP         AX
                RET
OUTPUT          ENDP
```

OUTPUT պրոցեդուրայի կանչը կարելի է իրականացնել հետևյալ կերպ.

```
.DATA
            String   DB     “This string will be printed$”
.CODE
            …
            LEA              DX, string
            CALL             OUTPUT
            …
```

Օրինակ 4

Պարամետրի փոխանցում ըստ վերադարձվող արժեքի, ռեգիստրով:

SUM պրոցեդուրան AX-ում ստանում է առանց նշանի թիվ և վերադարձնում թվի թվանշանների գումարը AX-ում:

```
SUM   PROC
; մուտք - AX-ում առանց նշանի թիվ
; ելք – AX-ում թվի թվանշանների գումարը
            PUSH        BX      CX      DX
            XOR         BX, BX

            MOV         CX, 10
CIRCLE:     CMP         AX, 0
            JE          FINISH
            XOR         DX, DX
            DIV         CX
            ADD         BX, DX
            JMP         CIRCLE
FINISH:     MOV         AX, BX
            POP         DX      CX      BX
            RET
SUM         ENDP
```

SUM պրոցեդուրայի կանչը կարելի է իրականացնել հետևյալ կերպ.

```
.DATA
        X       DW      12345
.CODE
        …
        MOV             AX, X
        CALL            SUM
        …
```

Օրինակ 5

Պարամետրի փոխանցում ծրագրի կողի մեջ:

PrintString պրոցեդուրան հիմնական ծրագրի կողից ստանում է տողի հասցեն և արտածում այն: Ապա փոխում է պահունակում գրված վերադարձի հասցեն այնպես, որ IP-ում հայտնվի փոխանցված տողին հաջորդող հասցեն.

```
PrintString     PROC
                PUSH            BP
                MOV             BP, SP
                PUSH            AX      DX      SI
                MOV             AH, 2
                MOV             SI, [BP + 2]
CIRCLE:         CMP             byte ptr CS:[SI], '$'
                JE              FINISH
                MOV             DL, CS:[SI]
                INT             21H
                INC             SI
                JMP             CIRCLE
FINISH:         INC             SI
                MOV             [BP + 2], SI
                POP             SI      DX      AX
                POP             BP
                RET
PrintString     ENDP
```

PrintString պրոցեդուրայի կանչը կարելի է իրականացնել հետևյալ կերպ.

```
.CODE
        …
        CALL        PrintString
        DB          “This is my string$”
        …
```

Անդրադարձ (ռեկուրսիվ) պրոցեդուրաներ

Ասեմբլերը թույլատրում է օգտագործել անդրադարձ պրոցեդուրաներ, երբ պրոցեդուրան կարող է ուղղակի կամ անուղղակի դիմում կատարել ինքն իրեն:

Դիտարկենք անդրադարձ պրոցեդուրաների մի քանի օրինակ:

Օրինակ 1

Ֆակտորիալ հաշվելու անդրադարձ պրոցեդուրա

```
Factorial       PROC
; մուտք – 2-բայթանի n պարամետր պահունակում
; ելք – AX (ax = n! )
                PUSH        BP
                MOV         BP, SP
                MOV         AX, [BP + 4]    ; ստանալ  n-ը
                CMP         AX, 0            ; ստուգենք n > 0, թե ոչ
                JA          L1              ; այո՝ շարունակել
                MOV         AX, 1           ; ոչ՝ վերադարձ
                JMP         L2
L1:             DEC         AX
                PUSH  AX                    ; Factorial(n-1)
                CALL        Factorial

;  ներքևի հրամանները կատարվում են, երբ տեղի է ունենում
```

```
; վերադարձ անդրադարձ կանչից

                MOV     BX, [BP + 4]    ; ստանալ n-ը
                MUL     BX              ; (DX:AX) = AX * BX
L2:             POP     BP              ;
                RET     2               ; մաքրել պահուսակը
Factorial       ENDP
```

Օրինակ 2

Անդրադարձ պրոցեդուրա, որը տպում է մուտքային AX ռեգիստրի արժեքը՝ որպես առանց նշանի տասական թիվ.

```
Print PROC
; մուտք – AX
; ելք –  ---
                PUSH    BX      DX      AX
                MOV     BX, 10
                XOR     DX, DX
                DIV     BX
                OR      AX, AX
                JZ      done
                CALL    print
    Done:       ADD     DL, ‘0’
                MOV     AH, 2
                INT     21h
                POP     AX      DX      BX
                RET
    Print       ENDP
```

Խնդիրների լուծման օրինակներ

1. Ստեղնաշարից ներածել 2 սիմվոլային տող, ստանալ նրանց միակցում տողը և արտածել այն: Տողերի միակցումը և “Enter” գործողության իրականացումը կազմակերպել պրոցեդուրայով (CONCATE և ENTR պրոցեդուրաներ): Տողերի հասցեները CONCATE պրոցեդուրային փոխանցել պահունակով (պարամետրերի փոխանցում՝ ըստ հղման, պահունակով).

```
STEK    SEGMENT   STACK      ‘STACK’
        DW   256  DUP(?)
STEK    ENDS
DATA    SEGMENT   ‘DATA’
        String1   DB   10, 11 DUP(?)
        String2   DB   10, 11 DUP(?)
        String3   DB   19 DUP(?)
        Entr      DB   10, 13, ’$’
DATA    ENDS

COD     SEGMENT   ‘CODE’
ASSUME  CS:COD, DS:DATA, SS:STEK
MAIN    PROC  FAR
        PUSH  DS
        XOR       AX, AX
        PUSH      AX
        MOV       AX, DATA
        MOV       DS, AX
; առաջին տողի ներածում
        MOV       AH, 10
        LEA       DX, string1
        INT       21H
        CALL      ENTR
;երկրորդ տողի ներածում
```

```
        LEA     DX, string2
        INT     21h
        CALL    ENTR

; տողերի հասցեները հիշենք պահուստակում
        PUSH    OFFSET  string3
        PUSH    OFFSET  string1
        PUSH    OFFSET  string2
; տողերի միակցում
        CALL    CONCATE
        CALL    ENTR

; ստացված տողի արտածում
        MOV     AH, 9
        LEA     DX, string3
        INT     21H
        RET
        MAIN    ENDP

ENTR    PROC
        PUSH    AX      DX
        MOV     AH, 2
        MOV     DL, 10
        INT     21H
        MOV     DL, 13
        INT     21H
        POP     DX      AX
        RET
ENTR    ENDP

CONCATE         PROC
    ; մուտք– տողերի հասցեները պահուստակում են
    ; ելք – առաջինի մեջ հաջորդ երկուսի միակցումը
        PUSH    BP
        MOV     BP, SP
```

```
        ADD     BP, 4
        PUSH    SI      DI      CX
        MOV     DI, [BP + 4]    ; DI = string3 տողի հասցե
        MOV     SI, [BP + 2]    ; SI = string1 տողի հասցե
        PUSH    DS
        POP     ES
        CLD
        XOR     CH, CH
        MOV     CL, [SI + 1]    ; CX = string1 տողի տար-
                                ; րերի քանակ
        ADD     SI, 2           ; SI = string1 տողի իրա-
                                ; կան սկզբի հասցե
        REP     MOVSB           ; string3-ում ստացանք
                                ; string1-ի պատճենը՝
                                ; առաջին տողը կցվեց
        MOV     SI, [BP]        ; SI = string2 տողի հասցե
        MOV     CL, [SI + 1]    ; CX = string2 տողի տար-
                                ; րերի քանակ
        ADD     SI, 2           ; SI = string2 տողի իրա-
                                ; կան սկզբի հասցե
        REP     MOVSB           ; կցվեց նաև երկրորդ տողը
        MOV     AL, '$'
        STOSB                   ; ստացված տողի վերջում
                                ; ավելացնում ենք '$'
        POP     CX      DI      SI
        POP     BP
        RET     6
CONCATE ENDP
COD     ENDS

        END     MAIN
```

2. Ներածել առանց նշանի տասական թիվ ([0,65535] միջակայքի) սիմվոլային տողի տեսքով և AX ռեգիստրում ստանալ թիվը: Տողից թվի ստացումը կազմակերպել պրոցեդուրայով: Ներածված տողում ոչ թվանշանների առկայության կամ թվի արժեքի պահանջվող միջակայքին չպատկանելու դեպքում տալ սխալի մասին հաղորդագրություն.

```
.MODEL SMALL
.DATA
            Number      DB      6, 7 dup(?)
            Mes1        DB      “It is not a number$”
            Mes2        DB      “this number is out of range$”
.CODE
Start:      MOV         AX, @DATA
            MOV         DS, AX
            MOV         AH, 10
            LEA         DX, number
            INT         21H
            CALL        To_Number
            CMP         DX, 0
            JE          finish
            CMP         DX, -1
            JE          ERR1
            CMP         DX, -2
            JE          ERR2
            JMP         finish
Err1:       MOV         AH, 9
            LEA         DX, mes1
            INT         21H
            JMP         finish
Err2:       MOV         AH, 9
            LEA         DX, mes2
            INT         21H
```

```
Finish:     MOV     AX, 4C00H
            INT     21H
To_Number   PROC
; Մուտք – DX՝ սիմվոլային տողի հասցեն
; ելք –    DX՝ սխալի կոդը, եթե սխալ կա (-1՝ թիվ չէ, -2՝ միջակայքից
;           դուրս է): Եթե սխալ չկա՝ DX = 0 և AX = թիվը

            PUSH    CX      DI      SI
            XOR     CX, CX
            MOV     SI, DX
            MOV     CL, [SI + 1]
            ADD     SI, 2
            MOV     DI, 10
            XOR     AX, AX
Circle:     MUL     DI
            TEST    DX, DX
            JNZ     p2
            MOV     DL, [SI]
            CMP     DL, '0'
            JB      p1
            CMP     DL, '9'
            JA      p1
            AND     DL, 0FH
            ADD     AX, DX
            INC     SI
            LOOP    circle
            XOR     DX, DX
            JMP     pr_fin
P2:         MOV     DX, -2
            JMP     pr_fin
P1:         MOV     DX, -1
Pr_fin:     POP     SI      DI      CX
            RET
```

```
To_Number       ENDP
        END     Start
```

3. Ներածել երկչափ սիմվոլային զանգված՝ (5 x 6) չափի: Կարգավորել զանգվածի տողերը աճման կարգով «պղպջակի» մեթոդով և արտածել ստացված երկչափ զանգվածը: Զանգվածի տողի արտածումը և աճման կարգով կարգավորումն իրականացնել պրոցեդուրաներով:

```
.MODEL SMALL
.DATA
        N       EQU     5
        M       EQU     6
        mas     DB      N dup (M dup(?))
.CODE
Begin:          MOV     AX, @Data
                MOV     DS, AX
; Զանգվածի ներածում էկրանից
                MOV     AH, 1
                XOR     BX, BX
                MOV     CX, N
circle1:        PUSH    CX
                XOR     SI, SI
                MOV     CX, M
circle2:        INT     21H
                MOV     mas[BX][SI], AL
                INC     SI
                LOOP    circle2
                CALL    ENTR
                POP     CX
                ADD     BX, SI
                LOOP    circle1
                CALL    ENTR
; Զանգվածի տողերի կարգավորում աճման կարգով
```

```
                LEA             BX, mas
                MOV             CX, N
circle3:        CALL            Buble_sort_string
                ADD             BX, M
                LOOP            circle3
; Զանգվածի արտածում էկրանին
                LEA             BX, mas
                MOV             CX, N
circle4:        CALL            PrintString
                CALL            ENTR
                ADD             BX, M
                LOOP            circle4
                MOV             AX, 4C00H
                INT             21H
PrintString     PROC
; Մուտք –       BX՝ զանգվածի տողի հասցեն
; ելք  – ----
                PUSH  AX        BX      CX      SI
                XOR             SI, SI
                MOV             CX, M
                MOV             AH, 2
P1:             MOV             DL, [BX][SI]
                INT             21h
                MOV             DL, ' '
                INT             21H
                INC             SI
                LOOP            P1
                POP   SI        CX      BX      AX
                RET
PrintString     ENDP
ENTR            PROC
                PUSH  AX        DX
```

```
                MOV             AH, 2
                MOV             DL, 10
                INT             21H
                MOV             DL,13
                INT             21H
                POP             DX AX
                RET
ENTR            ENDP
Bubble_sort_string      PROC
; «Պղպջակի» մեթոդը հետևյալն է.
; for(i = 0; i < N - 1;  i ++)   for(j = I + 1; j < N; j ++)
;                       if (A[i] > A[j])  change(A[i], A[j])

; Մուտք – BX՝ զանգվածի տողի հասցեն
; ելք      – կարգավորված տող
                PUSH  AX        SI      DI      BX
                XOR             SI, SI
A2:             CMP             SI, M - 1
                JE              FIN
                MOV             DI, SI
                INC             DI
                MOV             AL, [BX][SI]
A1:             CMP             DI, M
                JE              FIN1
                CMP             AL, [BX][DI]
                JLE             NO
                XCHG            AL, [BX][DI]
                MOV             [BX][SI], AL
                INC             DI
                JMP             A1
fin1:           INC             SI
                JMP             A2
fin:            POP     BX      DI      SI      AX
```

```
                RET
Buble_sort_string       ENDP
                END     BEGIN
```

Խնդիրներ և վարժություններ

1. Նկարագրել ոչ անդրադարձ պրոցեդուրա, որը տպում է մուտքային AX ռեգիստրի արժեքը՝
   ա) որպես առանց նշանի տասական թիվ,
   բ) որպես նշանով տասական թիվ:
2. Ներածել նշանով տասական թիվ ([-32768,32767] միջակայքի) սիմվոլային տողի տեսքով և AX ռեգիստրում ստանալ թիվը: Տողից թվի ստացումը կազմակերպել պրոցեդուրայով:
3. Կազմել ծրագիր, որը հաշվում է բնական թվերից կազմված բայթային զանգվածի տարրերի ամենամեծ ընդհանուր բաժանարարը: Երկու թվերի ամենամեծ ընդհանուր բաժանարարը հաշվել պրոցեդուրայով:
4. Ներածել 35 սիմվոլներից բաղկացած զանգված: Դրանից առանձնացնել ենթազանգված այն սիմվոլներից, որոնց 0 և 1 համարի բիթերը 1 են, իսկ 7 և 5 համարի բիթերը՝ 0: Մեկ սիմվոլի համար պայմանը ստուգել պրոցեդուրայով:
5. Ներածել 27 սիմվոլներից բաղկացած զանգված: Դրանում հաշվել այն սիմվոլների քանակը, որոնցում 1 և 3 համարի բիթերում 0 է, իսկ 2 և 5 համարի բիթերում՝ 1: Այդպիսի սիմվոլներում 4 և 7 համարի բիթերը դարձնել 0, իսկ 6 համարի բիթի արժեքը՝ հակադարձել: Մեկ սիմվոլի հետ աշխատանքն իրականացնել պրոցեդուրայով:
6. Նկարագրել պրոցեդուրա, որը, ստանալով X նշանով բառը (X DW ?), վերադարձնում է այն ամենամեծ N-ը, որի դեպքում.
   ա) 2-ի N աստիճանը փոքր կամ հավասար է X-ի բացարձակ արժեքից,
   բ) 4-ի N աստիճանը փոքր կամ հավասար է X-ի բացարձակ արժեքից,

գ) 8-ի N աստիճանը փոքր կամ հավասար է X-ի բացարձակ արժեքից:

Օգտագործել այդ պրոցեդուրան տրված A բառի համար և արտածել N-ը:

7. Նկարագրել PRINTW16 պրոցեդուրան, որը տպում է տրված բառ-փոփոխականի (X DW ?) արժեքը քառանիշ 16-ական առանց նշանի թվի տեսքով: Պրոցեդուրային պարամետրը փոխանցել.

   ա) ռեգիստրով, բ) պահունակով:

   Գրել ծրագրի հատված, որն այդ պրոցեդուրայի միջոցով տպում է տրված Y կրկնակի բառի (Y DD ?) արժեքը 16-ական տեսքով:

8. Տրված է հետևյալ նկարագրությունը.

```
D       DB      ?       ; օր
M       DB      ?       ; ամիս
Y       DW      ?       ; տարի
```

   Նկարագրել TRUEDATE(DD, MM, YY, P) պրոցեդուրան, որը վերադարձնում է P = 1, եթե DD.MM.YY-ը ճիշտ ամիս-ամսաթիվ է, և P = 0՝ հակառակ դեպքում (ուշադրություն դարձնել նահանջ տարվա ստուգմանը): Պրոցեդուրային պարամետրերը փոխանցել.

   ա) ռեգիստրներով, բ) պահունակով:

   Կիրառել պրոցեդուրան D, M, Y եռյակների տարբեր արժեքների համար:

9. Տրված է 40 տարրից բաղկացած նշանով բառ-փոփոխականների զանգված: Արտածել զանգվածի տարրերը: Մի տարրի արտածումն իրականացնել պրոցեդուրայով (պահունակ չօգտագործել):

10. Տրված է

```
A       DB      80 DUP(?)       ; նշանով թվերի զանգված
B       DW      ?
```

   նկարագրությունը:

   Գրել SUM(X, N, S) պրոցեդուրան, որը S պարամետրին վերագրում է N հատ նշանով բայթերի X զանգվածի տարրերի գումարը: Կիրառել այն՝ SUM(A,80,B) կանչն իրագործելու համար:

11. Նկարագրել F(X, N, P) պրոցեդուրան, որը հաշվում է N հատ բայթերի զանգվածի այն տարրերի քանակը, որոնք հավասար են P բայթին, և արդյունքը վերադարձնում AL ռեգիստրով: Կիրառել այդ պրոցեդուրան հետևյալ հաշվարկի համար.

    K = F(A, 50, F(B, 30, K)) ,

    որտեղ A-ն 50 բայթից բաղկացած զանգված է, B-ն 30 բայթից բաղկացած զանգված է, իսկ K-ն՝ բայթային փոփոխականը:

12. Տրված են 70 բառ-փոփոխականից կազմված X զանգվածը և 40 բառ-փոփոխականից կազմված Y զանգվածը: Եթե X-ում կան կրկնվող տարրեր, իսկ Y-ի բոլոր տարրերը իրարից տարբեր են, ապա DL ռեգիստրին վերագրել 1, հակառակ դեպքում՝ 0:

    Նկարագրել համապատասխան պրոցեդուրա և այն օգտագործելով՝ գրել գլխավոր ծրագիր, որը լուծում է նշված խնդիրը:

13. Նկարագրել պրոցեդուրա, որն արտածում է AX ռեգիստրի արժեքը՝ որպես առանց նշանի թիվ՝ թվարկության այն համակարգով, որի հիմքը (2-ից մինչև 10) փոխանցվում է BL ռեգիստրով (ուշադրություն դարձնել բաժանման ժամանակ գերհագեցման հնարավորությանը): Գրել գլխավոր ծրագիր, որը ստուգում է պրոցեդուրայի աշխատանքը տարբեր դեպքերի համար:

14. Գրել SUMA(BX, CX, AX) պրոցեդուրան, որտեղ BX-ով փոխանցվում է զանգվածի սկզբնական հասցեն, իսկ CX-ով՝ տարրերի քանակը: Զանգվածի տարրերն առանց նշանի բառեր են, որոնք հանդիսանում են տվյալների սեգմենտի ինչ-որ նշանային բայթերի հասցեներ: Պրոցեդուրան պետք է հաշվի այդ բոլոր բայթերի գումարը և վերադարձնի AX ռեգիստրի միջոցով: Գրել գլխավոր ծրագրի հատված՝ տվյալ պրոցեդուրայի կիրառմամբ:

15. Տրված է առանց նշանի թվերի A զանգվածը.

    A DW DUP 25(?)

    Կարգավորել զանգվածն ըստ տարրերի առաջին թվանշանների նվազման կարգի: Նկարագրել համապատասխան պրոցեդուրա(ներ) և կիրառել խնդիրը լուծող գլխավոր ծրագրում:

ՄԱԿՐՈ

Մակրոն ասեմբլերի հզոր միջոց է: Մակրոներն ապահովում են տեքստի փոխարինման ճկուն մեխանիզմ: Մակրոյի օգնությամբ ծրագրի ինչ-որ հատվածի կարելի է անուն տալ և հետագայում այդ հատվածը կրկնել բազմիցս՝ օգտագործելով մակրոկանչի հրամանագիրը: Թարգմանիչը յուրաքանչյուր մակրոկանչ փոխարինում է մակրոնկարագրության տեքստով: Մակրոն կարող է ունենալ պարամետրեր:

Մակրոյի հետ կապվում են հետևյալ հասկացությունները.

1. մակրոնկարագրություն,
2. մակրոկանչ,
3. մակրոգեներացում,
4. մակրոընդլայնում:

Մակրոնկարագրություն

Մակրոնկարագրությունն ունի հետևյալ տեսքը.

<մակրոյի անուն> MACRO <ձևական պարամետրերի ցուցակ>
<մակրոյի մարմին>
ENDM

որտեղ <մակրոյի անուն>-ը և <ձևական պարամետրերի ցուցակ>-ը նույնարկողներ են, իսկ <մակրոյի մարմին>-ը՝ հրամաններ կամ մակրոկանչեր:

Մակրոկանչ

Մակրոկանչի տեսքը հետևյալն է.

< մակրոյի անուն > < փաստացի պարամետրերի ցուցակ >,

որտեղ < փաստացի պարամետրերի ցուցակ>-ը արժեքների ցուցակ է: Հանդիպելով մակրոկանչին՝ թարգմանիչը ծրագրի մեջ՝ նրա տեղում, զետեղում է համապատասխան մակրոյի մարմինը՝ նախապես մարմնի մեջ ձևական պարամետրերը փոխարինելով փաստացի պարամետրերով: Արդյունքում ստացված տեքստը կոչվում է **մակրոընդլայնում**: Պրոցեսը, որը փոխարինում է կանչը մակրոընդլայնումով, կոչվում է **մակրոգեներացում**:

Ընդհանուր դեպքում մակրոնկարագրությունը կարող է տեղադրվել ծրագրի ցանկացած հատվածում նախքան մակրոկանչը: Սակայն առավել ընդունված է մակրոնկարագրությունների տեղադրումը կատարել հետևյալ երկու եղանակով.

1. անմիջապես ծրագրում. սովորաբար ծրագրի սկզբում՝ բոլոր սեգմենտներից դուրս:
2. ֆայլում, որը կկցվի ծրագրին include հրամանագրով:

Ֆայլի մեջ մակրոյի նկարագրությունը կարելի է գրել տեքստային որևէ խմբագրիչով (օրինակ՝ Notepad, ոչ Microsoft Word): Եթե ֆայլի մեջ կան «ավելորդ» մակրոնկարագրություններ, ապա include-ից հետո կարելի է օգտագործել “PURGE <անունների ցուցակ >” հրամանագիրը, որով էլ այդ անուններով մակրոնկարագրությունները կչեղարկվեն:

Մակրոյի և պրոցեդուրայի տարբերությունը.

1. Մակրոյում կան պարամետրեր, իսկ պրոցեդուրայում չկան:
2. Պրոցեդուրայի կանչը մեքենայական հրաման է, մակրոյի կանչը՝ հրամանագիր:
3. Մակրոկանչի փոխարինումն ընդլայնումով՝ մակրոգեներացումը, կատարում է թարգմանիչը, իսկ պրոցեդուրայի կանչն իրականացվում է պրոցեսորի կողմից ծրագրի կատարման ժամանակ:

Օրինակ

```
; init1 մակրոնկարագրություն
init1    MACRO
         PUSH  DS
         XOR   AX, AX
         PUSH  AX
         ENDM

      ; init2 մակրոնկարագրություն
init2    MACRO  d_n
         init1
         IFNFB  <d_n>
                MOV   AX, d_n
                MOV   DS, AX
         ENDIF
       ENDM
```

Ծրագրի մեջ կարելի է գրել կա՛մ *INIT1,* կա՛մ դրան համարժեք *INIT2,* կա՛մ էլ *INIT2 DATA,* եթե ունենք *DATA* անունով տվյալների սեգմենտ: Եթե մակրոնկարագրությունը պարունակում է նշիչ, ապա մակրոկանչը մեկ անգամից ավել օգտագործելու դեպքում կառաջանա սխալ. նշիչը կրկնվում է: Սխալից խուսափելու համար, մակրոյի նկարագրության մեջ՝ անմիջապես *MACRO*-ի տողից հետո, պետք է օգտագործել "*LOCAL* <նշիչների ցուցակ>" հրամանագիրը: Սրանով թարգմանչին ստիպում ենք, որ յուրաքանչյուր գեներացման ժամանակ նշիչները փոխվեն:

Կրկնման հրամանագրեր

Կրկնման հրամանագրերը մակրոմիջոցների տարատեսակ են և ապահովում են տեքստի փոխարինման և կրկնման հնարավորություն:

Գոյություն ունեն չորս տիպի կրկնման հրամանագրեր.

1. Տեսքը

REPT **ամբողջատիպ արտահայտություն**
մարմին
ENDM

Այս հրամանագրով մարմինը կրկնվում է արտահայտության արժեքի չափով:

Օրինակ

```
REPT    3       ; կրկնել 3 անգամ
  SHL   AX, 1
ENDM
```

Վերոհիշյալ REPT հրամանագիրը կվոխարինվի՝

```
SHL   AX, 1
SHL   AX, 1
SHL   AX, 1
```

տեքստով:

2. Տեսքը

WHILE ***արտահայտություն***
մարմին
ENDM

Այս հրամանագրով մարմինը կրկնվում է, քանի դեռ արտահայտության արժեքը հավասար չէ 0-ի:

Օրինակ

```
I = 0
WHILE I < 4
    DB   20      ; կրկնել չորս անգամ
    I = I + 1
ENDM
```

3. Տեսքը

IRP *պարամետր, <արժեքների_ցուցակ>*
մարմին
ENDM

Այս հրամանագրով թարգմանիչը գեներացնում է տեքստ՝ կրկնելով մարմինը արժեքների ցուցակի արժեքների քանակով և յուրաքանչյուր կրկնության ժամանակ մարմնում հանդիպող պարամետրի փոխարեն տեղադրելով ցուցակի հերթական արժեքը:

Օրինակ

```
IRP     t, <10, 30, 7>
    DB  t
ENDM
```

4. Տեսքը

```
IRPC  արգումենտ, սիմվոլային տող
        մարմին
ENDM
```

Տողը սիմվոլների հաջորդականություն է: Եթե տողում, տառից ու թվանշանից բացի, նաև հանդիպում է այլ սիմվոլ, ապա < > նշաններ պետք է դրվեն:

Օրինակ

```
1. IRPC   t, ab4   ;                    DB   'a'
     DB   '&t&'    ; նույնն է ինչ սա    DB   'b'
   ENDM            ;                    DB   '4'
2. IRPC   t, ab4
          a&t   DB    20
   ENDM
Կգեներացվի՝
   aa     DB    20
   ab     DB    20
   a4     DB    20


3. IRPC   t, BA
          MOV   E&t&X, 50
   ENDM

Կգեներացվի՝
```

```
MOV     EBX, 50
MOV     EAX, 50
```

EXITM հրամանագիր

Եթե մակրոգեներացման ժամանակ թարգմանիչը հանդիպում է *EXITM* հրամանագրին, ապա մակրոգեներացումն ավարտվում է:

Խնդիրներ և վարժություններ

1. Նկարագրել կրկնման հրամանագիր, որը լուծում է հետևյալ խնդիրը.
   ա) DH ռեգիստրում ստանալ AL, BL, CL և DL ռեգիստրներում տրված թվերի գումարը,
   բ) զրոյացնել DWORD տիպի A, B և C փոփոխականները,
   գ) DB հրամանագրի միջոցով հիշողությունում 40 բայթի համար տեղ հատկացնել՝ դրանք սկզբնարժեքավորելով առաջին 40 կենտ թվերով (1, 3, 5, .., 79):
2. X, Y, Z կրկնակի բառային փոփոխականների նկատմամբ հետևյալ գործողությունները նկարագրել այջ կողմում տրված մակրոների միջոցով (L-ը նշիչ է).
   ա) X = Y : DMOV X, Y
   բ) X = Y + Z : DADD X, Y, Z
   գ) X = Y - Z : DSUB X, Y, Z
   դ) Եթե X != Y՝ անցնել L-ին : DJNE X, Y, L

3. Գրել վերջնական ծրագրի տեքստը, որը կկառուցի մակրոգեներատորը՝ ըստ հետևյալ ծրագրային հատվածի.

ա)
```
N     EQU  5
IRP OP, <N, %N, N % N>
      ADD  AX, OP
ENDM
```

բ)
```
IRP  C, <INC A, JE L>
       C
ENDM
```

գ) IRP W, <SW, <A, 2>>
MOV&W
ENDM

դ) IRPC CH, <! % “ >
ADD AL, '&CH'
ENDM

ե) IRP T, <AB, C>
IRPC U, T
DW U, T&U, T&&U
ENDM
DW U, T&U
ENDM

զ) IRP C, <K, LL, M>
C EQU C&C&CC
DB 'C&C&CC'
ENDM

4. Նկարագրել DEF X, T, N, V մակրո, որը սահմանում է N հատ V մեծություններից կազմված X զանգված: Զանգվածի տարրերի տիպը տրվում է T պարամետրով՝ T = B-ն BYTE տիպի է, T = W-ն WORD տիպի է, իսկ T = D-ն՝ DWORD տիպի:

Գրել ծրագրի վերջնական տեքստը, որը կկառուցի մակրոգեներատորը՝ ըստ հետևյալ ծրագրային հատվածի.

```
K       EQU   4
DEF     C, B,  , ' * '
DEF     W, W, K + 1,<TYPE C>
DEF     D, %K + 1, %(TYPE W)
DEF     A, B, 1, <1, 2, 3>
```

5. Գրել ծրագիր, որը մուտքագրում է H, M և S թվերը և ստուգում է՝ արդյոք դրանք բավարարում են հետևյալ պայմաններին՝ $0 \le H \le 23$, $0 \le M \le 59$ և $0 \le S \le 59$: Եթե բավարարում են, ապա ընդունելով այդ թվերը որպես օրվա որոշակի պահի՝ ժամ (H), րոպե (M) և վայրկյան (S)՝ ծրագիրը պետք է արտածի օրվա ժամը՝ 1 վայրկյանով ավելի:

Ծրագրում սահմանել 2 մակրո, որոնցից մեկը կստուգի $a \le X \le b$ պայմանը, իսկ մյուսը՝ X-ի արժեքը կավելացնի 1-ով, և, եթե $X > b$՝ X-ը կզրոյացնի:

6. Նշանով բայթային թվերի նկատմամբ հետևյալ գործողությունները նկարագրել ձախ կողմում տրված մակրոների միջոցով.

ա) ABS R,Z : R = abs(Z), որտեղ R-ը ռեգիստր է, Z-ը՝ փոփոխական,

բ) SUM Z, N : AX = Z զանգվածի բայթային տարրերի գումարը (N > 0),

գ) MAX Z, N : AL = Z զանգվածի N քանակով բայթային տարրերից մեծագույնի արժեքը:

7. Հետևյալ հրամանները սահմանել մակրոյի միջոցով.

ա) PUSH L (L-ը բառային փոփոխական է),

բ) POP L (L-ը բառային փոփոխական է),

գ) CALL P (P - ն near պարամետրով պրոցեդուրա է),

դ) RET N (մոտիկ վերադարձ պրոցեդուրայից),

ե) LOOP L (L - ը նշիչ է):

8. Տրված է.

```
X       DW          1234H
A       MACRO       R, U
        MOV         R, U
        ENDM
B       MACRO       R, V
        A           R, <V>
        ENDM
C       MACRO       R, V
        A           R, V
        ENDM
```

Որոշել AH և AL ռեգիստրների արժեքները ծրագրի հետևյալ հատվածի կատարման արդյունքում.

```
        B       AH, <BYTE PTR X>
        C       AL, <BYTE PTR X>:
```

9. Սահմանել MAX2 M, X, Y մակրոն՝ M = MAX(X, Y)-ը հաշվելու համար: Ապա դրա հիման վրա սահմանել.

- MAX3 M, X, Y, Z մակրոն՝ M = MAX(X, Y, Z)-ը հաշվելու համար, որտեղ M, X, Y, Z-ը նշանով բայթային փոփոխականներ են:

- Գրել MAX3 A, <BYTE PTR B>, C + 1, D մակրոհրամանի մակրոընդլայնումը:

10. Նկարագրել SIGN X (X-ը բայթային, բառային կամ կրկնակի բառային փոփոխական է) մակրոն, որը AL ռեգիստրում ստանում է 1, եթե X > 0, ստանում է 0, եթե X = 0, և -1, եթե X < 0:
    Գրել մակրոընդլայնումներ SIGN W և SIGN D մակրոհրամանների համար, որտեղ W-ն բառային փոփոխական է, իսկ D-ն՝ կրկնակի բառային փոփոխական:
11. Նկարագրել NULL X, N, T մակրոն (X-ը N քանակով բայթային տարրերից կազմված զանգված է, N > 0, T = FIRST, եթե զրոյացվում է զանգվածի առաջին տարրը, T = LAST, եթե զրոյացվում է զանգվածի վերջին տարրը):
    Գրել մակրոընդլայնում NULL A, 100, LAST մակրոհրամանի համար:
12. Նկարագրել SUM X մակրոն (X-ը զանգված է՝ կազմված k > 0 քանակով բայթային փոփոխականներից), որը հաշվում է X զանգվածի զույգ տարրերի գումարը և գրանցում DX ռեգիստրում:
    Գրել SUM <A, B, C> մակրոհրամանի մակրոընդլայնումը:
13. Նկարագրել MIN X մակրոն (X-ը զանգված է՝ կազմված k > 0 քանակով բայթային փոփոխականներից), որը գտնում է X զանգվածի ամենափոքր տարրը և գրանցում AL ռեգիստրում:
    Գրել MIN <A, B, A> մակրոհրամանի մակրոընդլայնումը:

ԸՆԴՀԱՏՈՒՄ

Ընդհատումը իրադարձություն է, որը ստիպում է պրոցեսորին դադարեցնել իր հերթական հրամանի կատարումը և ղեկավարությունը տալ ընդհատումը մշակող ենթածրագրին: Իրական ռեժիմում առանձնացնում են ընդհատումների երկու տեսակ՝ ծրագրային (INT N) և ապարատային: Ապարատային ընդհատման օրինակ է ստեղնի սեղմումը, մկնիկի դիրքի փոխելը և այլն: Պաշտպանված ռեժիմում ընդհատումների տեսակին ավելանում է ևս մեկը՝ բացառություն (INT հրամանի կատարման մասին տե՛ս «Ղեկավարությունը փոխանցող հրամաններ» բաժինը):

INT հրամանի միջոցով հասանելի են BIOS-ի և DOS-ի օգտակար ենթածրագրեր:

INT 21h ընդհատման մի քանի ֆունկցիաներ

Իրական ռեժիմում ասեմբլերով մուտքի/ելքի կազմակերպման համար կարելի է օգտվել DOS ընդհատումներից: 21H ընդհատումը բազմաֆունկցիոնալ է, և նրա ֆունկցիոնալությունը որոշվում է AH ռեգիստրի արժեքով:

INT 21h – AH = 1 DOS – Սիմվոլի ներածում ստեղնաշարից

Մուտք – չկա

Վերադարձ – AL-ում ներածված սիմվոլն է (այսինքն՝ սիմվոլի ASCII կոդը)

Օրինակ

```
MOV     AH, 1
INT     21H     ; համակարգիչը սպասում է մինչև ստեղնաշարից
                ; որևէ սիմվոլ ներածվի և ներածված սիմվոլը գրում
                ; է AL ռեգիստրում օրինակ՝ ՛A՛ սիմվոլը սեղմելու
                ; դեպքում AL ռեգիստրը կընդունի 41h արժեք,
                ; այսինքն՝ ՛A՛ սիմվոլի ASCII կոդը
```

INT 21h – AH = 2 կամ AH = 6 DOS – Սիմվոլի արտածում էկրան

Մուտք – DL ռեգիստրում արտածվող սիմվոլն է (այսինքն՝ սիմվոլի ASCII կոդը)

Վերադարձ – չկա

Օրինակ

```
MOV     AH, 2
MOV     DL, 'A'
INT     21H             ; էկրանին կարտածվի ՛A՛ սիմվոլը
```

INT 21h – AH = 9 DOS – Սիմվոլային տողի արտածում էկրան

Մուտք – DS:DX-ում արտածվող սիմվոլային տողի հասցեն է: Տողը պետք է ավարտվի ՛$՛-ով:

Վերադարձ – չկա

Օրինակ

```
.DATA
  MyString    DB      "Hello$"
              ...
              MOV     AH, 9
              LEA     DX, MyString
              INT     21H     ; էկրանին կարտածվի "Hello"
```

INT 21h – AH = 10 DOS – Սիմվոլային տողի մուտք ստեղնաշարից

Մուտք – DS:DX = ներածվող սիմվոլային տողի հասցեն

Վերադարձ – DS:DX = ներածված սիմվոլային տողը

Ներածում է սիմվոլային տող (մինչև ՛ENTER՛ ստեղնի ներածումը) և պահում այն DS:DX-ով որոշվող տողում, որի առաջին բայթը մինչև ներածումը պետք է պարունակի ներածվող տողի սիմվոլների առավելագույն քանակը (ներառյալ ՛ENTER՛-ը), իսկ երկրորդ բայթը՝ ներածումից հետո պարունակում է ներածված տողի սիմվոլների փաստացի քանակը (առանց ՛ENTER՛-ի):

Օրինակ

Ներածել սիմվոլային տող՝ առավելագույնը 10 սիմվոլ:

```
.DATA
   N          EQU   10
   MyString   DB    N+1, ?, N + 1 dup (?)
.CODE
        . . .
            MOV   AH, 10
            LEA   DX, MyString
            INT   21H   ; „ABC↵” մուտքագրելուց
                        ; հետո MyString + 1 հասցեում կգրվի 3
                        ; [Mystring + 2] ← ‘A’
                        ; [Mystring + 3] ← ‘B’
                        ; [Mystring + 4] ← ‘C’
                        ; [Mystring + 5] ← 13 (Enter-ի կողը)
            . . .
```

INT 21h – AH = 4CH DOS – Ծրագրի աշխատանքի ավարտ

Մուտք – AL = ավարտի կոդ

Վերադարձ – չկա

Կազմակերպում է ծրագրի աշխատանքի ավարտը և ղեկավարությունը փոխանցում է օպերացիոն համակարգին: Ավարտի կոդի 0 արժեքը նշանակում է ծրագրի նորմալ ավարտ:

Խնդիրների լուծման օրինակներ

1. Ներածել 1 սիմվոլ, և եթե այն մեծատառ է, արտածել համապատասխան փոքրատառը, հակառակ դեպքում արտածել “It is not a capital letter” տողը:

```
.MODEL    SMALL
.STACK    100h
.DATA
     message        DB     ‘There is not a capital letter$’
.CODE
MAIN:     MOV            AX, @DATA
          MOV            DS, AX
          MOV            AH, 1
          INT            21H
          CMP            AL, ’A’
          JB             no_letter
          CMP            AL, ’Z’
          JA             no_letter
          OR             AL, 32          ; ⇔ ADD AL, 32
          MOV            AH, 2
          MOV            DL, AL
          INT            21H
          JMP            finish
no_letter: MOV           AH, 9
          LEA            DX, message
          INT            21H
Finish:   MOV            AX, 4C00H
          INT            21H
          END            MAIN
```

2. Ներածել սիմվոլային տող: Առանձնացնել թվանշանների ենթատող և արտածել այն էկրանին նոր տողի սկզբից: Թվանշան չլինելու դեպքում արտածել հաղորդագրություն:

```
.MODEL    SMALL
.STACK    100h
 .DATA
     message      DB    'There is not a number$'
     entr         DB    10,13,'$'
     string       DB    10,11 dup(?)
     substring    DB    10 dup(?)
 .CODE
 MAIN:    MOV          AX, @DATA
          MOV          DS, AX
          MOV          AH, 10
          LEA          DX, string
          INT          21H
          MOV          AH, 9
          LEA          DX, entr
          INT          21H
          XOR          CX, CX
          MOV          CL, string[1]
          MOV          SI, 2
          XOR          DI, DI
 CYRCLE:  MOV          AL, string[SI]
          CMP          AL, '0'
          JB           next
          CMP          AL, '9'
          JA           next
          MOV          substring[DI], AL
          INC          DI
 next:    INC          SI
```

```
                LOOP        CYRCLE
                TEST        DI, DI
                JZ          no_number
                MOV         byte ptr substring[DI], '$'
                LEA         DX, substring
                INT         21H
                JMP         finish
no_number:      LEA         DX, message
                INT         21H
    finish:     MOV         AX, 4C00H
                INT         21H
                END         MAIN
```

Խնդիրներ և վարժություններ

1. Գրել ծրագիր, որն արտածում է լատինական այբուբենի տառերը հետևյալ հաջորդականությամբ՝ AaBb…Zz:
2. Ներածել 2 սիմվոլային տող: Ստանալ և արտածել նոր տող, որտեղ առաջին տողի բոլոր 'A' սիմվոլների փոխարեն տեղադրված է երկրորդ տողը: Առաջին տողում 'A' սիմվոլ չլինելու դեպքում արտածել հաղորդագրություն:
3. Ներածել 20 սիմվոլներից բաղկացած միաչափ զանգված: Ստեղծել սիմվոլային տող զանգվածի տառ չհանդիսացող տարրերից և արտածել այն էկրանին նոր տողի սկզբից: Եթե զանգվածի մեջ միայն տառեր են, արտածել հաղորդագրություն այդ մասին:
4. Ներածել սիմվոլային (5 x 4) չափի երկչափ զանգված: Էկրանին նոր տողի սկզբից արտածել զանգվածի [35h, 75h] միջակայքին պատկանող ASCII կոդով սիմվոլները: Այդպիսի սիմվոլներ չլինելու դեպքում արտածել հաղորդագրություն:

ՀԱՎԵԼՎԱԾ 1
ՏՎՅԱԼՆԵՐԻ ՆԵՐԿԱՅԱՑՈՒՄԸ ԷՀՄ-ՈՒՄ

Թվարկության դիրքային համակարգեր

Հայտնի է, որ թվի գրառման մեջ օգտագործվում են թվանշաններ: Գոյություն ունեն հաշվարկման ոչ դիրքային և դիրքային համակարգեր: Ոչ դիրքային համակարգերում թվանշանի արժեքը հաստատուն է և իր դիրքից կախված չէ: Այդպիսի հաշվարկման համակարգի օրինակ է հայերեն տառերով թվերի գրառումը (ա-1, բ-2, …, թ-9, ի-10, ժ- 20, իգ-23, ռթ-1009), հռոմեական համակարգը (I-1, II-2, III-3, …, X-10, XX-20): Դիրքային համակարգերում թվանշանի դիրքից կախված այն որոշում է տարբեր արժեքներ: Հաշվարկման դիրքային համակարգի օրինակ է հանրածանոթ հաշվարկման 10-ական համակարգը, որտեղ օրինակ 575 թվի մեջ առաջին 5-ը որոշում է 500 միավոր, իսկ վերջին 5-ը՝ 5 միավոր:

Կդիտարկենք հաշվարկման դիրքային համակարգեր: Դրանցում օգտագործվող թվանշանների քանակը կանվանենք հաշվարկման համակարգի հիմք, իսկ հաշվարկման համակարգը՝ հաշվարկման «հիմք»-ական համակարգ: Հաշվարկման p-ական համակարգ անվանում են p հատ սիմվոլներով (որոնց անվանում են թվանշաններ) և որոշակի օրենքներով թվերի գրառման կարգը: p-ն բնական թիվ է և հավասար չէ 1-ի (1 թվանշանով թվի գրառումը անիմաստ է): Որպես թվանշան՝ ընդունված է օգտագործել 0, 1, 2, … $p \le 10$-ի դեպքում, իսկ $p > 10$-ի դեպքում՝ 0, 1, …, 9, A, B, …, որտեղ A-ն 10 միավոր պարունակող թվանշանն է, B-ն՝ 11 և այլն: Օրինակ՝ հաշվարկման 3-ական համակարգում թվանշաններն են 0, 1, 2, իսկ հաշվարկման 12-ական համակարգում՝ 0, 1, …, 9, A, B:

Հայտնի է, որ յուրաքանչյուր A թիվ կարելի է միակ ձևով ներկայացնել P թվի աստիճանների տեսքով՝

$$A = a_n \cdot P^n + a_{n-1} \cdot P^{n-1} + \cdots + a_1 \cdot P^1 + a_0 + b_1 \cdot P^{-1} + b_2 \cdot P^{-2} + \ldots + b_m \cdot P^{-m} \ldots$$

որտեղ՝ $P \geq 2$, $a_i, b_j = \{0, 1, \ldots, P-1\}$ $(i = 0, 1, \ldots, n;\ j = 1, 2, \ldots, m, \ldots)$

Այդ դեպքում A թիվը P-ական համակարգում գրում են հետևյալ ձևով՝

$$A=(a_n a_{n-1} \cdots a_1 a_0, b_1\, b_2 \ldots b_m \ldots)_P,$$

թվի թվանշանները գրվում են փակագծերում և փակվող փակագծից հետո, որպես ներքևի ինդեքս՝ գրվում է հաշվարկման համակարգի հիմքը (որպեսզի տարբերվի, թե թիվը որ համակարգում է գրված): Բացառություն է կազմում 10-ական համակարգը:

Թվերի թարգմանությունը P-ական համակարգից Q-ական համակարգի

Ամբողջ թվերի թարգմանություն

Դիցուք, A_P ամբողջ թիվը P-ական համակարգի թիվ է, և P-ական համակարգում կարող ենք կատարել գործողություններ: Ենթադրենք, A_P թիվը Q-ական համակարգում ունի հետևյալ տեսքը.

$$a_n \cdot Q^n + a_{n-1} \cdot Q^{n-1} + \cdots + a_1 \cdot Q^1 + a_0;$$

$a_i = \{0, 1, \ldots, Q-1\}$ անհայտ գործակիցները գտնելու համար գրենք.

$$A_P = a_n \cdot Q^n + a_{n-1} \cdot Q^{n-1} + \cdots + a_1 \cdot Q^1 + a_0; \text{ և}$$

հավասարության երկու կողմը բաժանենք Q-ի վրա, որտեղ Q-ն ձախ կողմում գրված է P-ական համակարգով, կստանանք.

$$\frac{A_P}{Q_P} = a_n \cdot Q^{n-1} + a_{n-1} \cdot Q^{n-2} + \cdots + a_1 + \frac{a_0}{Q};$$

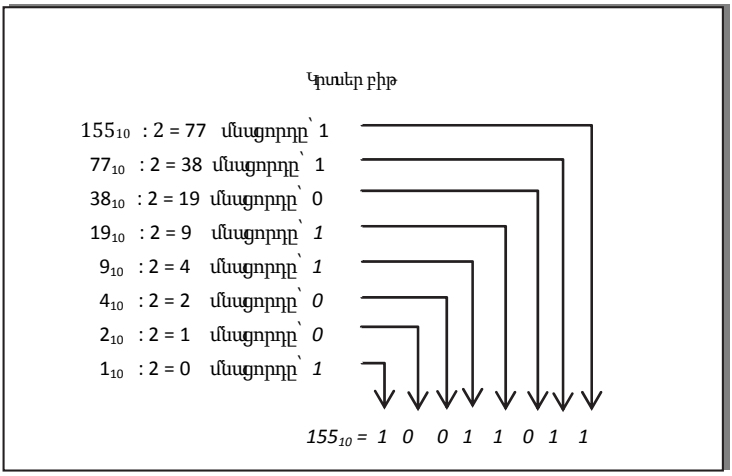
Հավասարությունից հետևում է.

$$\left[\frac{A_P}{Q_P}\right] = a_n \cdot Q^{n-1} + a_{n-1} \cdot Q^{n-2} + \cdots + a_1; \quad \left\{\frac{A_P}{Q_P}\right\} = \frac{a_0}{Q};$$ որտեղից կստանանք a_0-ն: a_0-ն $\frac{A_P}{Q_P}$ բաժանումից ստացված մնացորդի արժեքն է (a_0-ն կստանանք որպես P-ական համակարգի թիվ, այն անհրաժեշտ է փոխարինել Q-ական համակարգի թվանշանով:)

Կոտորակային մասերի հավասարությունից կորոշենք a_0-ն:

Այժմ նշանակենք $A'_P = \left[\frac{A_P}{Q_P}\right]$,

և նույնը կատարենք A'_P-ի հետ, կստանանք a_1-ը: Եվ այսպես կրկնելով նշված քայլերը n անգամ՝ կստանանք բոլոր անհայտ գործակիցները: Նշենք, որ n-ի արժեքը պարտադիր չէ նախօրոք որոշել. այն ստացվում է ավտոմատ՝ նշված գործողությունները շարունակելով այնքան, մինչև ստացվի զրո քանորդ:



Նկար 1. 155 տասական թիվը ներկայացնել 2-ական համակարգով:

P-ականից Q-ական համակարգ անցնելիս, եթե մեկը մյուսի աստիճան է, ապա անցումը կարելի է կատարել ավելի հեշտ պետք է եղանակով:

Անցում 16-ական համակարգից 2-ականի: 16-ականից 2-ականի անցնելու համար անհրաժեշտ է 16-ականի յուրաքանչյուր թվանշան փոխարինել իր 2-ական համարժեքով ըստ հետևյալ աղյուսակի.

16-ական	2-ական	16-ական	2-ական
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Անցում 8-ական համակարգից 2-ականի: 8-ականից 2-ականի անցնելու համար անհրաժեշտ է յուրաքանչյուր թվանշանը փոխարինել իր երկուական համարժեքով.

8-ական	2-ական
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Անցում 2-ական համակարգից 8-ականի: 2-ական համակարգից 8-ականի անցնելու համար անհրաժեշտ է աջից ձախ 2-ական թվանշանները բաժանել եռյակների՝ անհրաժեշտության դեպքում ձախից 0-ներ ավելացնելով, որից հետո յուրաքանչյուր եռյակ փոխարինել իր 8-ական համարժեքով:

<u>Անցում 2-ական համակարգից 16-ականի</u>: 2-ական համակարգից 8-ականի անցնելու համար անհրաժեշտ է աջից ձախ 2-ական թվանշանները բաժանել քառյակների՝ անհրաժեշտության դեպքում ձախից 0-ներ ավելացնելով, հետո յուրաքանչյուր քառյակ փոխարինել իր 16-ական համարժեքով:

Կոտորակային թվերի թարգմանություն

Ենթադրենք $0 \le B_P < 1$

Ընդունենք. $B = b_1 \cdot Q^{-1} + b_2 \cdot Q^{-2} + \cdots + b_m \cdot Q^{-m}$ (1)

Ենթադրենք P-ական համակարգի մեջ կոտորակային թվի թվանշանների քանակը n է, այսինքն՝ ճշտությունը՝ P^{-n} է: Վերցնելով Q-ական համակարգում ճշտությունը Q^{-m}՝ $P^{-n} = Q^{-m}$, կպահպանենք ճշտությունը՝ որտեղից.

$$m = \frac{n \cdot \lg P}{\lg Q}, \quad \text{քանի որ } m \in N \text{ վերցնենք}$$

$$m = \left[\frac{n \cdot \lg P}{\lg Q}\right] ([]^* \text{ - ճշանակում է վերցնել ամբողջ մասը՝ արժեքը կլորացնելով դեպի վեր})$$

Եթե P = 10, ապա $m = \left[\frac{n}{\lg Q}\right]$:

b_i $(j = 1, 2, \dots, m)$ անհայտ գործակիցները գտնելու համար (1) հավասարության երկու կողմը բազմապատկենք Q-ով:

$$B \cdot Q_p \approx b_1 + b_2 \cdot Q^{-1} + \cdots + b_m \cdot Q^{-m+1}$$

Հավասարությունից հետևում է.

$$[B \cdot Q_p] = b_1$$

$$\{B \cdot Q_p\} = b_1 Q^{-1} + \cdots + b_m Q^{-m+1} = B'_p$$

B'_p-ը նորից բազմապատկելով Q-ով, կստանանք b_2: Նույնը կրկնելով m անգամ՝ կստանանք բոլոր անհայտ գործակիցները:

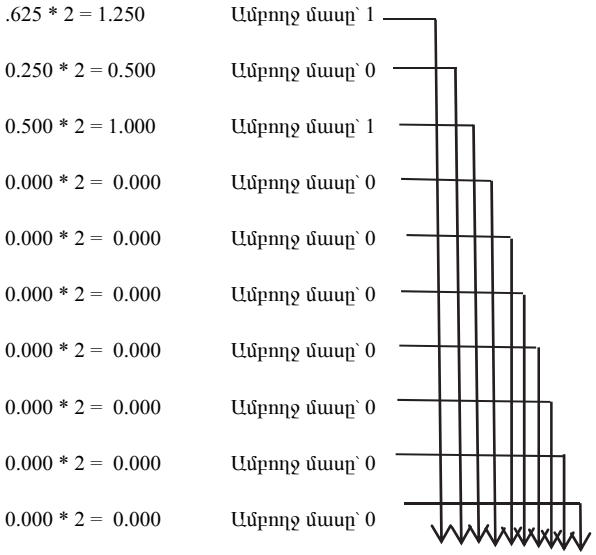
Օրինակ

0.625 թիվը ներկայացնել 2-ական համակարգով:

Նախ որոշենք՝ 2-ական համակարգում քանի թվանշան է անհրաժեշտ:

$$m = \left[\frac{3}{lg\ 2}\right] = 10\ (lg2\ \approx 0.3)$$

Կիրառելով 10-ական համակարգից 2-ականի անցման ալգորիթմը՝ կստանանք.



$0.625_{10} = 0.1\ 010000000_2$

Երկուական թվերի գումարում և հանում

Երկուական թվերի գումարումը և հանումը նման են տասական թվերի գումարմանը և հանմանը, տարբերությունն այն է միայն, որ գումարման և հանման համար օգտագործվում են այլ աղյուսակներ:

Գումարման աղյուսակն է.

B_{out}	x	y	$X + y + B_{out}$	B_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Որտեղ x–ը և y-ը գումարելիների թվանշաններ են, իսկ B_{out}-ը՝ փոխանցումն է հաջորդ կարգ:

Հանման աղյուսակը կլինի.

B_{in}	x	y	$x - y - B_{in}$	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

Որտեղ x-ը և y-ը գումարելիների թվանշաններ են, B_{in}-ը նախորդ կարգից եկած պարտքն է, իսկ B_{out}-ը՝ հաջորդ կարգ փոխանցվող պարտքը:

Օրինակ

Երկուական գումարում և հանում

B_{out}	01011 1100	B_{in}	0001 1100
X	01011 1110	X	1111 0101
Y	01000 1101	Y	0010 1110
X + Y	10100 1011	X - Y	1100 0111

Բազմապատկման աղյուսակն է.

x	y	x * y
0	0	0
0	1	0
1	0	0
1	1	1

Առանց նշանի ամբողջ թվերի ներկայացումը ԷՀՄ-ում

N երկուական թվանշանների (բիթերի) օգնությամբ կարելի է ներկայացնել $[0, 2^N]$ միջակայքի բոլոր ամբողջ թվերը:

N = 8 դեպքում՝ [0, 255]

N=16 դեպքում՝ [0, 65535]

Առանց նշանի թվերն օգտագործվում են հասցեների ներկայացման համար, ինչպես նաև ավելի մեծ միջակայքի և ճշտությամբ թվերի հետ գործողություններ կատարելու համար:

Նշանով ամբողջ թվերի ներկայացումը ԷՀՄ-ում

Նշանով ամբողջ թվերի ներկայացման համար օգտագործվում են ուղիղ, հակադարձ (one's complement), լրացուցիչ (two's complement) և շեղումով կոդեր: n երկուական թվանշանների օգնությամբ ուղիղ կոդով նշանով ամբողջ թվերը կարող են գտնվել $[-(2^{n-1}-1), 2^{n-1}-1]$ միջակայքում: Ավագ թվանշանը նշանի համար է, իսկ մյուս թվանշանները ցույց են տալիս թվի բացարձակ արժեքը: Նշանի թվանշանի 0-ն (սովորաբար) ցույց է տալիս "+", 1-ը "-": Ուղիղ կոդը ամբողջ թվերի ներկայացման համար հազվադեպ է օգտագործվում:

Թիվը հակադարձ կոդով ներկայացնելիս միջակայքը հետևյալն է՝ $[-(2^{n-1}-1),\ 2^{n-1}-1]$: Ուղիղ և հակադարձ կոդերի դեպքում ունենք երկու հատ 0՝ '+0' և '-0':

$$[x]_{հակ} = 2^n - 1 + x \ mod\ (2^n - 1) \ \ կամ$$

$$[x]_{հակ} = \begin{cases} x, & եթե\ x \geq 0 \\ 2^n - 1 + x, & եթե\ x \leq 0 \end{cases}$$

Հակադարձ կոդը ուղիղ կոդին զուգահեռ օգտագործվում է թվաբանական գործողություններ կատարելու համար: Գումարման և հանման գործողությունները հակադարձ կոդով կատարելը ավելի հեշտ է, իսկ ուղիղ կոդով հարմար է կատարել բազմապատկում և բաժանում:

Լրացուցիչ կոդով կարելի է ներկայացնել $[-2^{n-1},\ 2^{n-1}-1]$ միջակայքի թվերը:

X ամբողջ թվի լրացուցիչ կոդը որոշվում է հետևյալ ձևով.

$$[X]_{լր} = 2^n + X \ mod \ 2^n \ կամ [X]_{լր} = \begin{cases} X, & եթե \ X \geq 0 \\ 2^n + X, & եթե \ X < 0 \end{cases}$$

Լրացուցիչ կոդի դեպքում ունենք 0-ի մեկ ներկայացում: Լրացուցիչ կոդը հարմար է գումարման, հանման համար:

Լրացուցիչ կոդը ստանալու համար կարելի է հետևել հետևյալ կանոններին.

- ստանալ թվի երկուական ներկայացումը,
- շրջել բիթերը (հակադարձ կոդ),
- գումարել մեկ:

Օրինակ 1

Սկզբնական թվից լրացուցիչ կոդի ստացումը.

$3_{10} = 00000011_2$

$-3_{լր} = 11111100_2 + 1_2 = 11111101_2 = -3_{լր}$

Օրինակ 2

Լրացուցիչ կոդից սկզբնական թվի ստացումը.

$-3_{լր} = 11111101_2$

$3_{10} = 00000010_2 + 1_2 = 00000011_2 = 3_{10}$

Շեղումով կոդն օգտագործվում է միայն նշանով թվերի համար: Նշանով ամբողջ թիվը հանդես է գալիս $[X]_{շեղ} = X + M$:

Սովորաբար շեղումով կոդի դեպքում M-ը վերցնում են X նշանով թվի փոփոխման միջակայքի փոքրագույն արժեքի բացարձակ արժեքը:

Շեղումով կոդն օգտագործում են իրական թվերը ներկայացնելիս որպես կարգաթիվ:

Ինթել x86 պրոցեսորներում նշանով թվերը ներկայացվում են լրացուցիչ կոդով: 8086 պրոցեսորներում նախատեսված են գործողություններ 8-բիթանոց և 16-բիթանոց ամբողջ թվերի հետ, իսկ 80386-ից սկսած՝ կարող են օգտագործվել 32-բիթանոց ամբողջ թվեր (նշանով և առանց նշանի): Սահող կետով թվերի հետ գործողությունների կատարումն արագացնելու համար ստեղծված է կից պրոցեսոր

(Coprocessor) Ինթել 8087, որը կարող է օգտագործվել 8086/8088 կողմից (80286-ի համար 80287, 80386-ի համար 80387, 80486-ից սկսած՝ FPU (floating Point Unit), որը կենտրոնական պրոցեսորի բաղկացուցիչ մասն է):

Լրացուցիչ կոդով ներկայացված թվերի համար ճիշտ է հետևյալ պնդումը.

$$[x + y]_{լր} = ([x]_{լր} + [y]_{լր}) \bmod 2^n$$

եթե x–ի, y–ի և (x + y)-ի արժեքները գտնվում են $[-2^{n-1},\ 2^{n-1} - 1]$ միջակայքում:

Օրինակ

Լրացուցիչ կոդով 1-բայթանոց թվերի գումարում.

X = 10$_{10}$	$[X]_{լր}$ =	0000 1010
		+
Y = -3$_{10}$	$[Y]_{լր}$ =	1111 1101

	1	0000 0111

2^8 -ը դեն է նետվում

Հակադարձ կոդով ներկայացված թվերի համար ճիշտ է հետևյալը՝

$$[x + y]_{հակ} = ([x]_{հակ} + [y]_{հակ}) \bmod (2^n - 1)$$

(նշենք, որ $2^n = 1 \bmod (2^n - 1)$)

Այս դեպքում ավագ կարգից առաջ եկած փոխանցումը գումարվում է կրտսեր կարգին:

Օրինակ

Հակադարձ կոդով 1-բայթանոց թվերի գումարում.

X = 10$_{10}$	$[X]_{հակ}$ =	0000 1010
		+

$\mathbf{Y = -3_{10}}$ $[Y]_{հակ} = 1111\ 1100$

---

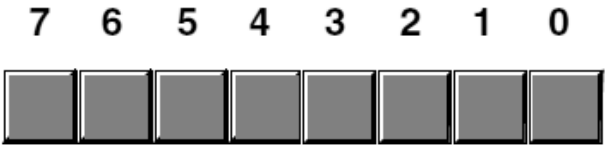
1 0000 0110

\+ 1

0000 0111

Բիթ, բայթ, բառ, կրկնակի բառ

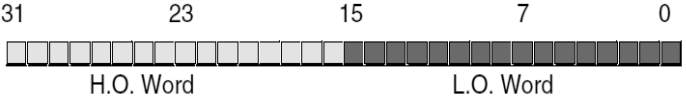
Տվյալների փոքրագույն միավորը համակարգչի համար բիթն է: Բիթը (bit/binary digit - նշանակում է երկուական թվանշան) կարող է ընդունել երկու արժեք՝ 0 կամ 1: Համակարգիչները սովորաբար աշխատում են բիթերի որոշակի քանակության հետ: Ծրագրի համար հասանելի փոքրագույն միավորը բայթն է, որը 8 բիթերի համախմբություն է: Բիթերը բայթում սովորաբար համարակալվում են 0-7:



Բիթերի համարակալումը բայթում

Ձրո բիթը **կրտսեր** բիթն է (low order bit or least significant bit), իսկ 7-րդ բիթը՝ **ավագ** (*high order bit* or *most significant bit)*: 4 բիթերի համախմբությանը անվանում են կիսաբայթ (nibble): Բառը 16 բիթերի համախմբությունն է: Բիթերը համարակալվում են 0 - 15, որտեղ 0-ն կրտսեր բիթն է, իսկ 15-րդը՝ ավագը: 0 - 7 բիթերին անվանում են կրտսեր բայթ, իսկ 8 - 15 բիթերին՝ ավագ բայթ:

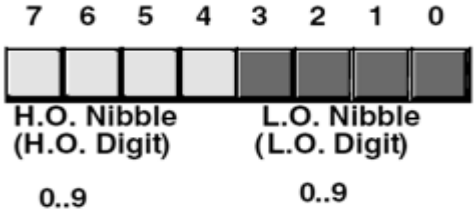
Կրկնակի բառը երկու բառերի համախումբ է:



Երկուական կոդավորված 10-ական (BCD) թվերի ներկայացումը

10-ական թվերի համար օգտագործվում են փաթեթավորված և ոչ փաթեթավորված ֆորմատներ: Ոչ փաթեթավորված ֆորմատի դեպքում յուրաքանչյուր տասական թվանշան ներկայացվում է 1 բայթում, որի ավագ կիսաբայթը 0000 է, իսկ կրտսերը թվանշանի BCD- երկուական կոդն է:

Փաթեթավորված ֆորմատի դեպքում 1 բայթում տեղավորվում է 2 10-ական թվանշան:



Օրինակ

$17_{10} = 0000\ 0001\ 0000\ 0111_{BCD}$՝ ոչ փաթեթավորված ֆորմատ,

$17_{10} = 0001\ 0111_{BCD}$՝ փաթեթավորված ֆորմատ:

Իրական թվերի ներկայացումը (IEEE ստանդարտ)

Սահող կետով կամ գիտական գրառումը օգտագործվում է, երբ անհրաժեշտ է հաշվարկներ իրականացնել շատ մեծ կամ շատ փոքր թվերի հետ:

Օրինակ

-0.35790×10^{-6} (ներկայացվում է նաև որպես -3.579E-6)

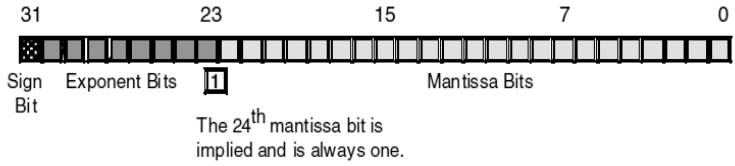
Երկուական համակարգի սահող կետով գրառումը կազմված է 3 մասերից՝

1. նշանի բիթ (“0” ոչ բացասական թվերի համար (+), “1”-ը՝ բացասական (-) թվերի համար),
2. կարգաբիվ (exponent),
3. մանտիսա:

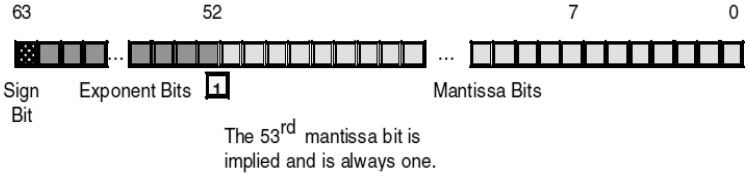
Նշում

Նորմալիզացված սահող կետով գրառման մեջ մանտիսան (այսինքն՝ մանտիսայի ավագ բիթը) պետք է սկսվի 1-ով և ենթադրվում է, որ կետը պետք է լինի մանտիսայից ձախ:

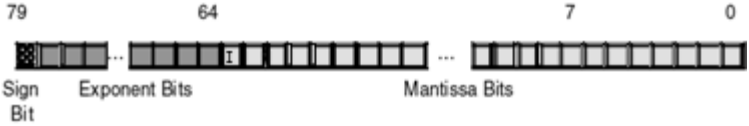
Այժմ իրական թվերի ներկայացման համար օգտագործվում է IEEE 754 (Institute of Electrical and Elecronics Engineers) ստանդարտը: Իրական թվերը ներկայացվում են որպես կարճ իրական (4 բայթ), երկար իրական (8 բայթ) և ընդլայնված/աշխատանքային իրական (10 բայթ):



Նկար 2. Կարճ իրական (սովորական ճշտությամբ) սահող կետով թվի ֆորմատը



Նկար 3. Երկար իրական (կրկնակի ճշտությամբ) սահող կետով թվի ֆորմատը



Նկար 4. Ընդլայնված իրական (կրկնակի ընդլայնված ճշտությամբ) սահող կետով թվի ֆորմատը

Այս ֆորմատներով իրական թվերի ներկայացման մեջ մանտիսան տրվում է ուղիղ կոդով, նորմալիզացված, որի բացարձակ արժեքը գտնվում է [1, 2) միջակայքում: 0 թվի ներկայացման համար կարգաթիվը (e) զրո է, մանտիսայի կոտորակային մասը ևս զրո է (f): Կարող են օգտագործվել դրական զրո և բացասական զրո, այսինքն՝ զրոյի համար ունենք $(-1)^s.0$:

Իրական թվերի միջակայքը որոշվում է e-ի փոփոխման միջակայքով: Սովորական ճշտության դեպքում՝ $0 < e < 255$, կրկնակի ճշտության դեպքում՝ $0 < e < 2047$, իսկ ընդլայնված ճշտության

դեպքում՝ $0 \le e < 32767$: Կարգաթվի (e) սահմանային արժեքները հատուկ դեպքեր են:

Ներկայացված սահող կետով թվի արժեքը (v) որոշվում է հետևյալ կերպ.

եթե 0 < e < 255, ապա թիվը կլինի՝ $v = (-1)^s \cdot 2^{e-127} \cdot (1.f)$,

եթե $e = 0, f \neq 0$, ապա թիվը կլինի՝ $v = (-1)^s \cdot 2^{e-126} \cdot (0.f)$,

եթե $e = 0, f = 0$, ապա թիվը կլինի՝ $v = (-1)^s \cdot 0$,

եթե $e = 255,\ f = 0$, ապա թիվը կլինի՝ $v = (-1)^s \cdot inf$ (անվերջություն),

եթե e = 255, f ≠ 0, ապա թիվը կլինի՝ NaN (Not a Number),

եթե 0 < e < 2047, ապա թիվը կլինի՝ $v = (-1)^s \cdot 2^{e-1023} \cdot (1.f)$;

եթե e = 0, f ≠ 0, ապա թիվը կլինի՝ $v = (-1)^s \cdot 2^{e-1022} \cdot (0.f)$;

եթե e = 0, f = 0, ապա թիվը կլինի՝ $v = (-1)^s \cdot 0$,

եթե e = 2047, f = 0, ապա թիվը կլինի՝ $v = (-1)^s \cdot Inf$ (անվերջություն),

եթե e = 2047, f ≠ 0, ապա թիվը կլինի՝ NaN (Not a Number),

եթե $0 \le e < 32767$, ապա թիվը կլինի՝ $v = (-1)^s \cdot 2^{e-16383} \cdot (i.f)$;

եթե e = 32767, f = 0, ապա թիվը կլինի՝ $v = (-1)^s \cdot Inf$ (անվերջություն),

եթե e = 32747, f ≠ 0, ապա թիվը կլինի՝ NaN (Not a Number),

Օրինակ 1

Որոշել, թե 10-ական որ թվի ներկայացումն է հետևյալ IEEE ստանդարտով տրված 32-բիթանոց սահող կետով թիվը:

0 10101100 10010100000000000000

(Բացատանիշերը բաժանում են նշանի բիթը, կարգը և մանտիսան):

Քանի որ նշանի բիթը 0 է, իրական թիվը դրական է: Կարգի բիթերը 10101100 ներկայացնում են առանց նշանի 172_{10} , հետևաբար կարգը կլինի՝ 172 - 127 = 45_{10}: Մանտիսայի բիթերն են 10010100000000000000, որը ներկայացնում է.

$$2^{-1} + 2^{-4} + 2^{-6} = 0.5781125$$

Հետևաբար իրական թիվը կլինի՝ 1.5781125×2^{45} :

Օրինակ 2

Որոշել, թե 10-ական որ թվի ներկայացումն է հետևյալ IEEE ստանդարտով տրված 32-բիթանոց սահող կետով թիվը:

1 00000000 01010000000000000000000

Քանի որ նշանի բիթը 1 է, թիվը բացասական է: Կարգի բիթերը 0-ներ են, հետևաբար կարգը -126 է: Մանտիսայի բիթերն են $01010000000000000000000 = 2^{-2} + 2^{-4} = 0.3125$:

Հետևաբար իրական թիվը կլինի՝ -0.3125×2^{-126}:

Օրինակ 3

Ներկայացնել 0.5 թիվը կարճ իրական սահող կետով (վերջնական արդյունքը ներկայացնել 16-ական համակարգով):

$$0.5 = 0.1_2 = 1.0 \cdot 2^{-1} = (-1)^0 \cdot 2^{126-127} \cdot (1.0)$$

Այստեղ՝ e = -1 + 127 = 126

Այսինքն՝ s = 0, e = 126, f = 0

Պատասխան՝ $0011\ 1111\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 3F00\ 0000_{16}$

Օրինակ 4

Ներկայացնել -1.25 թիվը կարճ իրական սահող կետով (վերջնական արդյունքը ներկայացնել 16-ական համակարգով):

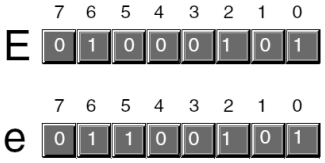
$$-1.25 = -1.01_2 = (-1)^1 \cdot 2^{127-127} \cdot (1.01)$$

Պատասխան՝ $1011\ 1111\ 1010\ 0000\ 0000\ 0000\ 0000\ 0000_2 = BFA0\ 0000_{16}$:

Սիմվոլային տվյալների ներկայացումը

Ինչպես և ցանկացած ինֆորմացիա, սիմվոլային տվյալները պահվում են երկուական ֆորմատով: Յուրաքանչյուր ֆորմատի համապատասխանության մեջ է դրվում որոշակի ոչ բացասական թիվ, որն անվանում են սիմվոլի կոդ: Սիմվոլի և կոդի միջև եղած կոնկրետ համապատասխանությունը անվանում են կոդավորում: Սովորաբար օգտագործվում են 8-բիթանոց կոդերով սիմվոլներ: Դա թույլ է տալիս կոդավորել 256 տարբեր սիմվոլներ: Օգտագործվում է ASCII (American Standard Code for Information Intechange) կոդավորումը: Նշենք այս կոդավորման որոշ առանձնահատկություններ առաջին 128 սիմվոլների համար.

- առաջին 32 սիմվոլները ղեկավարող սիմվոլներն են (օրինակ՝ 10 կոդը կուրսորը բերում է մի տող ներքև),
- պարունակում է տարբեր կետադրության սիմվոլներ, հատուկ սիմվոլներ,
- թվանշանները (ASCII 30h-39h) տարբերվում են իրենց թվային արժեքներով միայն կրտսեր կիսաբայթում,
- լատիներեն մեծատառերը (ASCII 41h-5Ah),
- լատիներեն փոքրատառերը (ASCII 61h-7Ah):



Նկար 5. e և E պառերի ASCII կոդերը

Ներկայումս ASCII կոդավորումն ամենատարածված կոդավորումն է: Մեկ այլ տարածված կոդավորում է Unicode-ը, որը վերացնում է ASCII կոդավորման սահմանափակումները՝ սիմվոլների սահմանափակ քանակությունը (առավելագույնը՝ 128/256) և միջազգային սիմվոլներ չօգտագործելը: Unicode-ը օգտագործում է 16-բիթանոց կոդեր, որը մեծ առավելություն է:

Խնդիրներ և վարժություններ

1. Չնավոխել հետևյալ 10-ական թվերը 2-ականի.
   ա) 23 բ) 139 գ) -97 դ) -246
2. Չնավոխել հետևյալ 2-ական թվերը 10-ականի.
   ա) 1011011 բ) 110101101 գ) 1001011010 դ) 110101
3. Չնավոխել 2) վարժության 2-ական թվերը 16-ականի:
4. Չնավոխել հետևյալ 16-ական թվերը 2-ականի.
   ա) 2A5D բ) EC07 գ) 25A9 դ) B3D9A
5. Կատարել հետևյալ 16-ական գործողությունները.
   ա) 35D4 + 6F11 բ) 72AD - 52B գ) B912 * 5
6. Հետևյալ 10-ական թվերը ներկայացնել 2-ական լրացուցիչ կոդով 8 բիթերի օգնությամբ և գումարել: Արդյունքը ներկայացնել 16-ական համակարգով.
   ա) -23, 45 բ) -139, 67 գ) -97, -25 դ) 246, 52
7. Հետևյալ սահող կետով թվերը ներկայացնել կարճ / երկար / ընդլայնված իրական տեսքով: Վերջնական արդյունքը ներկայացնել 16-ական համակարգով.
   ա) 39.375 բ) -92.625, գ) -97.45 դ) 46.93
8. Ներկայացնել հետևյալ երկուական կոդավորված 10-ական թվերը 10-ական համակարգով.
   ա) 0203h բ) 0139h գ) 0907h դ) 2146h
9. Հետևյալ 10-ական թվերը ներկայացնել որպես երկուական կոդավորված 10-ական թվեր.
   ա) 123 բ) 29 գ) 197 դ) 574
10. Կատարել հետևյալ 2-ական թվերի հանում.
    ա) 11001, 101 բ) 101101, 1100 գ) 10101, 1011 դ) 110111, 1010
11. Կատարել հետևյալ 2-ական թվերի բազմապատկում.
    ա) 1100, 101 բ) 10111, 1100 գ) 101, 11011 դ) 10101, 110
12. 0 սիմվոլը ASCII կոդով ներկայացվում է որպես 0011 0000, իսկ 6-ը՝ _____:

ՀԱՎԵԼՎԱԾ 2

ԾԱՆՈԹՈՒԹՅՈՒՆ MS-DOS DEBUG ԾՐԱԳՐԻՆ

DEBUG-ը (DEBUG.EXE) մեքենայական մակարդակի ճշգրտման ծրագիր (debugger) է, որը համալրում է MSDOS ՕՀ-ը: Այս կիրառության հիմնական նպատակն է ապահովել 8086 ասեմբլեր ծրագրի համար ճշգրտման հարմար միջավայր, որը ներառում է.

- թարգմանություն ասեմբլերից մեքենայական լեզվի (assembling) և հակառակը (disassembling),
- քայլային ռեժիմով ծրագրի կատարում (tracing),
- ռեգիստրների և հիշողության տիրույթների պարունակության զննում,
- հիշողության տիրույթում նոր արժեքի գրանցում,
- հիշողության մեջ ASCII արժեքների որոնում,
- հիշողության բլոկի տեղափոխում մի տիրույթից մեկ այլ տիրույթ,
- հիշողության տիրույթի արժեքավորում,
- ֆայլերի բեռնում:

Ծանոթությունը debugger-ի հետ շատ կարևոր է ասեմբլեր լեզվով ծրագրավորման համար: Այն կատարյալ միջոց է կարճ ծրագրեր գրելու և Ինքել միկրոպրոցեսորին ծանոթանալու համար: Եթե Դուք կարողանում եք աշխատել DEBUG-ով, ապա կարող եք հեշտությամբ սովորել ավելի հզոր ճշգրտման ծրագրեր, ինչպիսիք են՝ MASM:CODEVIEW-ն, TASM:TurboDebuger, որոնք DEBUG-ի կատարելագործված տարբերակներն են (32-բիթային հայտնի ճշգրտման ծրագրերից են OllyDbg, SoftICE, IDA Pro, որոնցից առաջին երկուսը միայն Windows ՕՀ-ի համար են, իսկ երրորդը՝ նաև Linux-ի):

DEBUG-ը աշխատեցնելու համար անհրաժեշտ է հրամանային տողից ներածել “debug.exe”: DEBUG-ը աշխատում է հրամանային ռեժիմում: Հրամանները մեկտառանի են: DEBUG միջավայրում բո-

լոր թվերը 16-ական համակարգով են: “-“ հրավերքի սիմվոլից հետո ներածելով “?”՝ կտեսնեք հետևյալ հրամանների ցուցակը.

```
-?
assemble    A [address]                          ; ասեմբլեր
compare     C range address                      ; համեմատել
dump        D [range]                            ; արտածել հիշողություն
enter       E address [list]                     ; ներածել
fill        F range list                         ; լրացնել
go          G [ = address] [addresses]           ; կատարել
hex         H value1 value2                      ; 16-ական արժեք
input       I port                               ; մուտք
load        L [address] [drive] [firstsector] [number] ; բեռնել
move        M range address                      ; տեղաշարժել
name        N [pathname] [arglist]               ; անվանել
output      O port byte                          ; ելք
proceed     P [ = address] [number]              ; մշակել
quit        Q                                    ; ավարտել
register    R [register]                         ; զննել ռեգիստրները
search      S range list                         ; որոնել
trace       T [ = address] [number]              ; կատարել քայլային ռեժիմով
unassemble  U [range]                            ; թարգմանել ասեմբլերի
write       W [address] [drive] [firstsector] [number] ; գրել
```

“;” սիմվոլից հետո ավելացվել է հրամանի հայերեն անվանումը:

[] փակագծերում ձշված են հրամանի ոչ պարտադիր պարամետրերը: Պարամետրերի ձշանակությունը հետևյալն է:

Հասցե/address – Հիշողության հասցե՝ տրված 16-ականով: Կարելի է օգտագործել միայն տեղաշարժը/offset (այս դեպքում լռությամբ որպես սեգմենտ կենթադրվի կոդ սեգմենտը (CS)) կամ տալ լրիվ սեգմենտ:շեղում հասցեն՝ կա՛մ 16-ականով, կա՛մ օգտագործելով սեգմենտային ռեգիստր թվի փոխարեն: Դիմացի 0-ները պարտադիր չեն: 1F հասցեն մեկնաբանվում է որպես 'CS:001F':

Օրինակ

100 DS:12 SS:0 198A:1234

Միջակայք/range – ունի տրման երկու ֆորմատ՝

1) հասցե [,հասցե]

Օրինակ՝

ա) 100, 500

բ) CS:200, 300

գ) 200

2) հասցե **L** [արժեք] (օրինակ՝ 100 L 20), որը նշանակում է 100 հասցեից սկսած 20 բայթ:

Ցուցակ/ list – բացատանիշով բաժանված 16-ական բայթեր կամ չակերտներով պարփակված ASCII տվյալներ:

Օրինակ

10, 20, 30, 40

'A', 'B', 50

Թիվ/number – Բոլոր թվերը և արժեքները DEBUG հրամաններում մեկնաբանվում են որպես 16-ական:

Դիտարկենք առավել օգտագործելի հրամանները:

Ավարտել/Quit: Q

Debug ծրագիրը ավարտում է իր աշխատանքը:

16-ական արժեք/Hex: H արժեք1 արժեք2

Շատ պարզ (միայն գումարում և հանում) 16-ական հաշվիչ:

Օրինակ

ա) **-h aaa 531** բ)**-h fff 3** գ) **-h dbf ace**

0FDB 0579 1002 0FFC 188D 02F1

դ) **-h 4 fffc** ե) **-h 100 123** զ) **-h 7fff 8000**

0000 0008 0223 FFDD FFFF FFFF

Գումարը առաջին արժեքն է՝ **AAA + 531 = 0FDB**

Տարբերությունը երկրորդ արժեքն է՝ **AAA - 531 = 579**

Ներածել/Enter: E հասցե [ցուցակ]

Մուտքագրած արժեքները անմիջապես տեղակայվում են հիշողության մեջ:

Օրինակ

-e 250 **'This is an ASCII data string.$'**

CS:250 հասցեից սկսած՝ կտեղաղրի **'This is an ASCII data string.$'** տողը:

Արտածել հիշողություն / Dump: D [range]

D [address] [length]

Էկրան է արտածվում հիշողության տիրույթ: Եթե առաջին անգամ է կիրառվում, ապա արտածումը սկսվում է 100 բայթ շեղումից: Հետագայում հրամանն առանց պարամետրի տալիս արտածվում է վերջին դուրս բերվածից հետո: Լռությամբ length-ը 128 է:

Օրինակ	**նկարագրություն**
D	արտածել 128 բայթ
D SS:0 5	արտածել SS:0-ից SS:5 բայթերը
D 915: 0	արտածել 915:0 հասցեից սկսած 128 բայթ
D 0 200	արտածել DS:0 հասցեից սկսած 200 բայթ
D 100 L 20	արտածել DS:100 հասցեից սկսած 20h բայթ

Հիշողության արտածման օրինակներ.

```
-d c000:0010
C000:0010 24 12 FF FF 00 00 00 00-60 00 00 00 00 20 49 42   $.......`.... IB
C000:0020 4D 20 43 4F 4D 50 41 54-49 42 4C 45 20 4D 41 54   M COMPATIBLE MAT
C000:0030 52 4F 58 2F 4D 47 41 2D-47 31 30 30 20 56 47 41   ROX/MGA-G100 VGA
C000:0040 2F 56 42 45 20 42 49 4F-53 20 28 56 31 2E 32 20   /VBE BIOS (V1.2
C000:0050 29 00 87 DB 87 DB 87 DB-87 DB 87 DB 87 DB 87      )..............
C000:0060 50 43 49 52 2B 10 01 10-00 00 18 00 00 00 00 03   PCIR+...........
C000:0070 40 00 12 10 00 80 00 00-38 37 34 2D 32 00 FF FF   @.......874-2...
```

```
C000:0080 E8 26 56 8B D8 E8 C6 56-74 22 8C C8 3D 00 C0 74  .&V....Vt"..=..t
-
-d 100 130
xxxx:0100 EB 24 0D 0A 54 68 69 73-20 69 73 20 6D 79 20 66       .$..This is my f
xxxx:0110 69 72 73 74 20 44 45 42-55 47 20 70 72 6F 67 72       irst DEBUG progr
xxxx:0120 61 6D 21 0D 0A 24 B4 09-BA 02 01 CD 21 B4 00 CD  am!..$......!...
xxxx:0130 21                                    !
-
```

Լրացնել/Fill: F միջակայք ցուցակ

Օրինակ

```
-f 100 12f 'BUFFER'
-d 100 12f
xxxx:0100 42 55 46 46 45 52 42 55-46 46 45 52 42 55 46 46 BUFFERBUFFERBUFF
xxxx:0110  45 52 42 55 46 46 45 52-42 55 46 46 45 52 42 55 ERBUFFERBUFFERBU
xxxx:0120  46 46 45 52 42 55 46 46-45 52 42 55 46 46 45 52 FFERBUFFERBUFFER

-f 100 ffff 0
-F 100 L 20 'A'
```

Fill 20h bytes with the letter 'A', starting at location 100.

Ասեմբլեր/Assemble: A [հասցե]

Այս հրամանից հետո օգտագործողին հնարավորություն է տրվում մուտքագրելու ասեմբլերի հրամաններ, որոնք տեղադրվում են CS:0100 հասցեից սկսած (եթե հասցեն տրված չէ): Նշիչները, մակրոհրամանները չեն ճանաչվում: Կարելի է օգտագործել 'DB' և 'DW' հրամանագրերը: 'A' հրամանը հիշում է վերջին հասցեն, որտեղ տվյալները թարգմանվել են:

Ասեմբլեր գործընթացը ավարտվում է, երբ ներածվում է դատարկ տող:

Օրինակ

(մի մուտքագրեք ‘;’ –ը և նրան հաջորդող մեկնաբանությունը).

```
-a 100
xxxx:0100 jmp 126       ; անցում  հաջորդող տվյալների վրայով
```

```
xxxx:0102 db 0d, 0a, "This is my first DEBUG program!"
xxxx:0123 db 0d, 0a, "$"
xxxx:0126  MOV AH, 9          ; INT 21h –ի 09 ֆունկցիան
xxxx:0128 MOV DX, 102         ; DS:DX→ $-ով ավարտվող տող
xxxx:012B INT   21            ; տողի արտածում
xxxx:012D MOV AH, 0           ; INT 21h – 0 ֆունկցիա
xxxx:012F   INT   21          ;  ծրագրի աշխատանքի ընդհատում
xxxx:0131
-g =100

This is my first DEBUG program!

Program terminated normally
-
```

Թարգմանել ասեմբլերի /Unassemble: U [միջակայք]

Այս հրամանը մեքենայական հրամանները թարգմանում է 8086 ասեմբլեր կոդի: Եթե լրացուցիչ [միջակայք] պարամետրը տրված չէ, օգտագործվում է **100 շեղումը որպես սկզբնակետ:**

Օրինակ

```
-u 126 12F
xxxx:0126 B409          MOV AH, 09
xxxx:0128 BA0201        MOV DX, 0102
xxxx:012B CD21          INT   21
xxxx:012D B400          MOV AH, 00
xxxx:012F CD21          INT   21
-
```

Կատարել/Go: G [=հասցե] [հասցեներ]

Այս հրամանն օգտագործվում է ծրագիրն աշխատեցնելու և ծրագրում ընդհատման կետեր տեղակայելու համար: '=հասցե' պարամետրը օգտագործվում է ծրագիրը աշխատեցնելու սկզբի հասցեն նշելու համար: Եթե օգտագործվում է միայն 'g', ապա կատարումը

սկսվում է CS:IP հասցեից: Եթե լրացուցիչ տրված են հասցեներ, ապա դրանք դիտարկվում են որպես ընդհատման կետեր, ինչը նշանակում է, որ ծրագիրն ընդհատում է իր աշխատանքը, երբ հասնում է նշված հասցեին: Ընդհատման կետը կարող է տեղակայվել միայն այն դեպքում, եթե այն 8086/8088 գործողության կողի առաջին բայթի հասցե է:

Զննել ռեգիստրները/Register: R [ռեգիստր]

' r ' հրամանը առանց պարամետրերի ցուցադրում է 8086-ի բոլոր ռեգիստրների պարունակությունները, ինչպես նաև IP ռեգիստրի պարունակությունը (այսինքն՝ հաջորդ հրամանի հասցեն) և համապատասխան հասցեի մեքենայական կողը:

Օրինակ

>debug c:\windows\command\choice.com

կիրառությունը կանչելուց և ' r ' հրամանը մուտքագրելուց հետո կարելի է տեսնել նման պատկեր.

```
AX=0000 BX=0000 CX=1437 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0ED8 ES=0ED8 SS=0ED8 CS=0ED8 IP=0100 NV UP EI PL NZ NA PO
NC
0ED8:0100 E90E01       JMP     0211
```

Վերջին տողը ցույց է տալիս ԿՊ-ի հաջորդ հրամանի (այս դեպքում փաստացի առաջին կատարվող հրամանի) տեղադրության հասցեն (0ED8:0100), հրամանի մեքենայական կողը՝ E90E0, և ասեմբլեր հրամանը՝ JMP 0211:

‘NV UP EI PL NZ NA PO NC’ անվանումները համապատասխանում են դրոշների արժեքներին:

Դրոշ	**Արժեքը = 1**	**Արժեքը = 0**
OF (Գերհագեցում)	OV (OVerflow)	NV (No OVerflow)
DF (Ուղղության)	DN (decrement)	UP (increment)
EF (Ընդհատման)	EI (enabled)	DI (disabled)
SF (Նշանի)	NG (negative)	PL (positive)

ZF (Զրո)	ZR (zero)	NZ (no zero)
AF (Օժանդակ փոխանցման)	AC	NA (no AC)
PF (Զույգության)	PE (even)	PO (odd)
CF (Փոխանցման)	CY (carry)	NC (No carry)

'rcx' հրամանը ներածելու դեպքում DEBUG-ը ցույց կտա CX ռեգիստրի պարունակությունը, հաջորդ տողում՝ ':', որով հնարավորություն է տալիս փոխելու ռեգիստրի պարունակությունը: ENTER ստեղնը սեղմելու դեպքում ռեգիստրի արժեքը չի փոխվի:

Օրինակ

```
-rcx
CX  0100
:273
```

նշանակում է որ CX ռեգիստրի արժեքը 0100-ից դարձավ 0273.

'rf' հրամանը ցուցադրում է դրոշների արժեքները և միաժամանակ թույլատրում փոխել դրոշի արժեքը հենց նույն տողում հրավերքի նշանից('-') հետո:

Օրինակ

```
-rf
 NV UP EI PL NZ NA PO NC   -zr
 -rf
NV UP EI PL ZR NA PO NC  -
-
```

(դրոշների արժեքների ցուցադրումից հետո ZF-ի արժեքը 1 դարձնելու համար մուտքագրվել է ZR):

Կատարել քայլային ռեժիմով /Trace: T [=հասցե] [թիվ]

T հրամանը օգտագործվում է հրամանները քայլային ռեժիմով կատարելու համար: Եթե մուտքագրվի միայն 'T' տառը, ապա կկատարվի մեկ հրաման՝ գրանցված CS:IP հասցեում, կդադարեցվի

ծրագրի կատարումը, և կցուցադրվի ԿՊ-ի (CPU) ռեգիստրների պարունակությունը, ինչպես նաև հաջորդ հրամանը և նրա մեքենայական կոդը: Եթե ցանկանում եք կատարել CS:0205 հասցեում տեղադրված հրամանը, ապա անհրաժեշտ է ներածել.

-t = 205 7

Մշակել /Proceed: P [=հասցե] [թիվ]

Այս հրամանը նման է Debug-ի T (Trace) հրամանին ասեմբլերի շատ հրամանների կատարման համար բացի CALL, LOOP, REP նախդիրով տողային և ընդհատման հրամաններից: Դա նշանակում է, որ ենթածրագրի կամ ընդհատումը մշակող ենթածրագրի հրամանները քայլ առ քայլ չեն կատարվի P հրամանի դեպքում:

Այս հրամանը ավելի հաճախ է օգտագործվում debug կիրառություններում, քան քայլային կատարումը (Trace), որն օգտագործվում է ենթածրագրի ֆունկցիոնալությունը ստուգելու նպատակով:

Օրինակ	**Նկարագրություն**
P = 200	կատարում է CS:0200 հասցեում գտնվող մեկ հրաման
P = 150 6	կատարում է 6 հրամաններ՝ սկսած CS:0150 հասցեից
P 5	կատարում է հաջորդ 5 հրամանները

Օրինակ

Debugging a Loop

```
-A 100
4A66:0100 MOV CX, 5 ; հաշվիչը = 5
4A66:0103 MOV AX, 0
4A66:0106 ADD AX,CX
4A66:0108 LOOP 106 ; ցիկլ 0106h հասցեից
-R
AX=000F BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=4A66 ES=4A66 SS=4A66 CS=4A66 IP=0100 NV UP EI PL NZ NA PE NC
```

```
4A66:0100 B90500 MOV CX, 0005
-P
AX=000F BX=0000 CX=0005 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=4A66 ES=4A66 SS=4A66 CS=4A66 IP=0103 NV UP EI PL NZ NA PE NC
4A66:0103 B80000 MOV AX, 0000
-P
AX=0000 BX=0000 CX=0005 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=4A66 ES=4A66 SS=4A66 CS=4A66 IP=0106 NV UP EI PL NZ NA PE NC
4A66:0106 01C8 ADD AX, CX
-P
AX=0005 BX=0000 CX=0005 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=4A66 ES=4A66 SS=4A66 CS=4A66 IP=0108 NV UP EI PL NZ NA PE NC
4A66:0108 E2FC LOOP 0106
-P
AX=000F BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=4A66 ES=4A66 SS=4A66 CS=4A66 IP=010A NV UP EI PL NZ NA PE
NC
```

Խնդիրներ և վարժություններ

1. Սկզբնարժեքավորել DS:0200 սկսած հիշողության տիրույթը 123416H արժեքով և DUMP հրամանի միջոցով ստուգել, որ հիշողության պարունակությունը փոխվել է:
2. Ջրոյացնել SI ռեգիստրի արժեքը և համոզվել դրանում՝ նորից ցուցադրելով նրա պարունակությունը:
3. IP ռեգիստրին տալ 1016 արժեք և համոզվել դրանում՝ ցուցադրելով նրա պարունակությունը:
4. Սկզբնարժեքավորել DS:0200 սկսած հիշողության 50 բայթ տիրույթը 0116 արժեքով և արդյունքը ստուգել:
5. Գրել ծրագիր, որը BX ռեգիստրում ստանա AH ռեգիստրի պարունակության քառակուսին:

ԳՐԱԿԱՆՈՒԹՅՈՒՆ

1. Зубков С. В., Assembler для DOS, Windows и Unix, Москва, издательство “ДМК”, 2006.
2. Юров В., Хорошенко С., Assembler. Учебный курс. Санкт-Петербург, издательство “Питер”,1999.
3. Рудаков П. И., Финогенов К. Г., Язык ассемблера. Уроки программирования, Москва, издательство “Диалог-МИФИ”, 2001.
4. Ирвин К., Язык ассемблера для процессоров Intel, 4-е издание, Москва, издательство “Вильямс”, 2005.
5. Скенлон Л., Персональные ЭВМ IBM PC и XT . Программирование на языке ассемблера. Москва, издательство “Радио и связь”,1989.
6. Пильщиков В. Н., Программирование на языке ассемблера IBM PC, Москва, издательство “Диалог-МИФИ”, 1996/2000.
7. Михаил Г., Процессоры Pentium II, Pentium Pro и просто Pentium. Санкт-Петербург, издательство “Питер”, 1999.
8. Михаил Г., Процессоры Intel от 8086 до Pentium II., Санкт-Петербург, издательство “Питер”, 1998.
9. Рудаков П. И., Финогенов К. Г., Программируем на языке Ассемблера IBM PC, изд. 2-е, Обнинск, издатетельство "Принтер", 1997.
10. Нортон П., Соухэ Д., Язык ассемблера для IBM PC, Москва, издательство "Компьютер", Финансы и статистика, 1992.
11. Задания практикума на ЭВМ (1 курс), методическая разработка, Москва, МГУ, 1992.
12. IA-32 Intel Architecture. Software Developer’s Manual. Volume 1: Basic Architecture.
13. IA-32 Intel Architecture. Software Developer’s Manual. Volume 2: Instruction Set Reference.
14. http://www.wasm.ru

ԵՐԵՎԱՆԻ ՊԵՏԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ

**Արա Հայկի Առաքելյան, Շուշանիկ Աշոտի Բաղդասարյան,
Կարինե Գրիշայի Գալստյան, Վարդան Արարատի Ջաքարյան,
Գևորգ Արթուրի Մարտիրոսյան, Ջավեն Արծրունու Նազարյան,
Գագիկ Ռազմիկի Սարդարյան**

ԻՆԹԵԼ x86 ԱՍԵՄԲԼԵՐ

(իրական ռեժիմ)
ՈՒՍՈՒՄՆԱՄԵԹՈԴԱԿԱՆ ՁԵՌՆԱՐԿ

Համակարգչային ձևավորումը՝ Կ. Չալաբյանի
Կազմի ձևավորումը՝ Ա. Պատվականյանի
Խմբագրումը՝ Մ. Հովհաննիսյանի

Տպագրված է «Գևորգ-Հրայր» ՍՊԸ-ում:
ք. Երևան, Գրիգոր Լուսավորչի 6

Ստորագրված է տպագրության՝ 07.12.2016:
Չափսը՝ 60x84 $^{1}/_{16}$: Տպ. մամուլը՝ 16,75:
Տպաքանակը՝ 200 օրինակ:

ԵՊՀ հրատարակչություն
ք. Երևան, 0025, Ալեք Մանուկյան 1
www.publishing.ysu.am