ՀՀ ԿՐԹՈՒԹՅԱՆ, ԳԻՏՈՒԹՅԱՆ, ՄՇԱԿՈՒՅԹԻ և ՍՊՈՐՏԻ ՆԱԽԱՐԱՐՈՒԹՅՈՒՆ

ՀԱՅԱՍՏԱՆԻ ԱԶԳԱՅԻՆ ՊՈԼԻՏԵԽՆԻԿԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ

(ՀԻՄՆԱԴՐԱՄ) 

\_\_\_\_\_\_\_\_\_ ՏՀՏԷ \_ \_ինստիտուտ/ֆակուլտետ

Քոմփյութերային համակարգեր և ցանցեր ամբիոն

Ծրագրային ճարտարագիտություն մասնագիտություն

**ՀԱՇՎԵԲԱՑԱՏՐԱԳԻՐ**

Հաշվողական համակարգերի ծրագրային ապահովում և ճարտարապետություն\_\_\_\_\_

կուրսային աշխատանք

Թեմա՝ Տրված է ամբողջ տվերի 3 զանգված, պետք է առաջինի եվ երկրորդի գումարից հանել երրորդը, արդյունքը պահել նախապես վերցված չորրորդ զանգվածում:

ՏՏ219-2 \_\_ ակադեմիական խումբ

Ուսանող \_\_\_\_\_\_\_\_\_\_ (ստորագրություն)

(ազգանուն, անուն հայրանուն)

Ղեկավար\_\_\_\_\_ Թումանյան դասախոս \_\_\_\_\_\_\_(ստորագրություն) (ազգանուն, անուն հայրանուն)

Ամբիոնի վարիչ \_\_\_\_Կիրակոսյան Գ.Տ. տ.գ.դ., պրոֆ. \_\_ (ստորագրություն) (ազգանուն, անուն հայրանուն)

ՀԱՅԱՍՏԱՆԻ ԱԶԳԱՅԻՆ ՊՈԼԻՏԵԽՆԻԿԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ

(ՀԻՄՆԱԴՐԱՄ)

ՏՀՏԷ ինստիտուտ/ֆակուլտետ

Քոմփյութերային համակարգեր և ցանցեր ամբիոն

(ամբիոնի անվանումը)

Ծրագրային ճարտարագիտություն մասնագիտություն

(մասնագիտության անվանումը)

ՏՏ219 ակադեմիական խումբ

ԿՈՒՐՍԱՅԻՆ ԱՇԽԱՏԱՆՔԻ  
ԱՌԱՋԱԴՐԱՆՔ

Հաշվողական Համակարգերի Ծրագրային Ապահովում և Ճարտարապետություն

(առառկայի անվանումը)

(ուսանողի ազգանուն, անուն, հայրանուն)

1․ Աշխատանքի թեման .   
2․ Աշխատանքի նախնական տվյալները .   
3․ Հաշվեբացատրագրի բովանդակությունը  
3․1․ Ներածություն .   
3․2․ Ծրագրի մշակում C-ով .   
3․3․ Ծրագրի մշակում Assembler-ով .   
3․4 Ծրագրի կարգաբերումը SASM փաթեթի միջոցով .   
4․ Գրաֆիկական մասի ծավալը .   
5․ Կատարման ժամանակացույցը .

6․ Աշխատանքի ղեկավար .

(ստորագրություն) (Ա․ Ա․ Հ․)

7․ Ամբիոնի վարիչ .

(ստորագրություն) (Ա․ Ա․ Հ․)

8․ Ուսանող .

(ամսաթիվ, ուսանողի ստորագրություն)

Բովանդակություն

* Ներածություն
* Խնդրի նկարագրություն
* Ծրագիրը C լեզվով
* Ծրագիրը Assembly լեզվով
* Համակարգչային ճարտարապետություն
* RISC եւ CISC ճարտարապետություններ
* Ասսեմբլեր եւ ռեգիստրներ

Ներածություն

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։

Ասեմբլեր լեզուն նախընտրում են օգտագործել հետևյալ իրավիճակներում.

* Ծրագիրն աշխատում է անմիջապես սարքերի հետ
* Անհրաժեշտ է օգտագործել պրոցեսորի հրամաններ, որոնք հասանելի չեն բարձր մակարդակի լեզուների թարգմանիչներին (compiler)։
* 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):

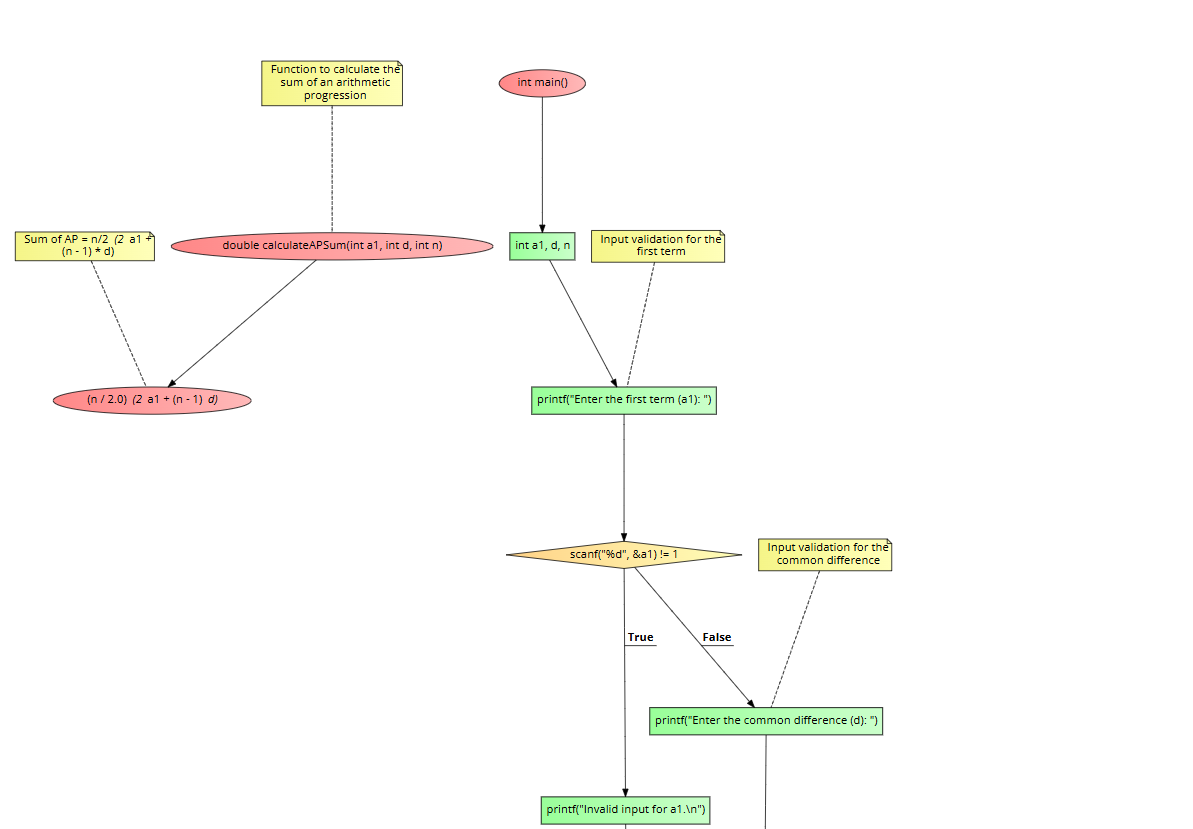
x86 ասեմբլեր լեզուն ունի երկու հիմնական շարահյուսություն` Ինթել շարահյուսություն և AT&T շարահյուսություն։ Ինթել շարահյուսությունը օգտագործվում է MS-DOS և Windows ՕՀ համար, իսկ AT&T շարահյուսությունը՝ Unix ՕՀ համար։ Ասեմբլեր լեզուն մեծատառ-փոքրատառերի նկատմամբ ոչ զգայուն (case-insensitive) լեզու է։

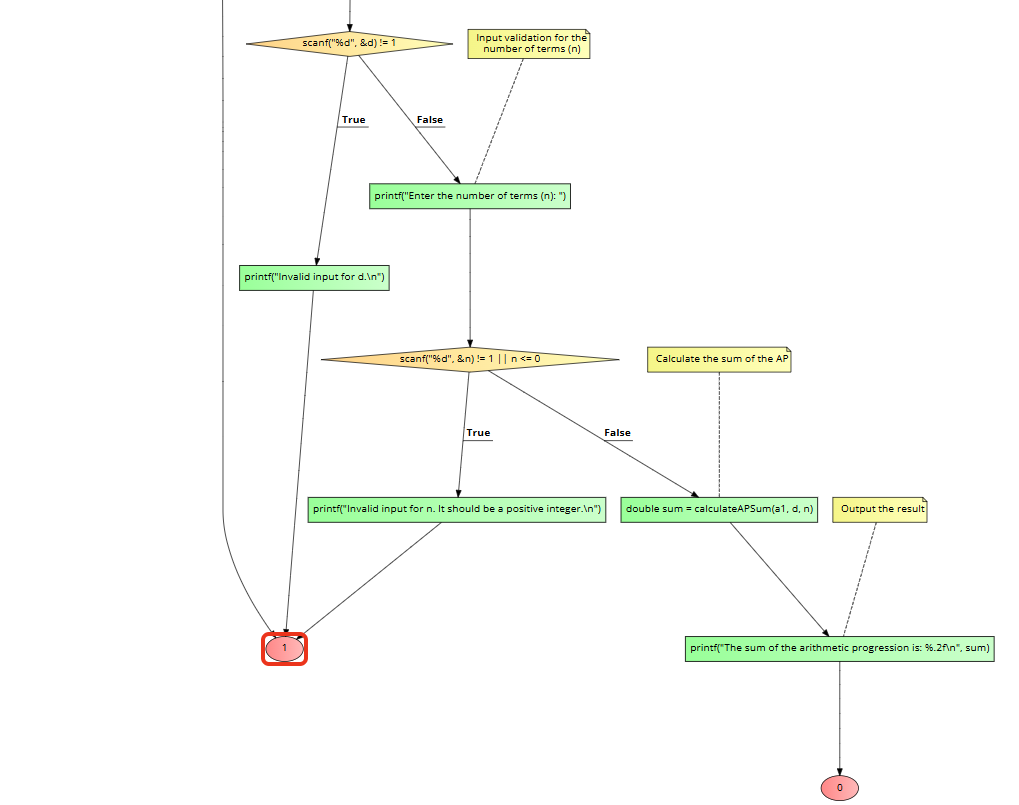
Տրված է ամբողջ տվերի 3 զանգված, պետք է առաջինի եվ երկրորդի գումարից հանել երրորդը, արդյունքը պահել նախապես վերցված չորրորդ զանգվածում:

Խնդրի նկարագրություն

Տրված է թվաբանական պրոգրեսիայի առաջին անդամը, անդամների քանակը և քայլը։ Հաշվել թվաբանական պրոգրեսիայի գումարը։ Հավելյալ օգտագործել ասեմբլերի օպերատորները վերջնական sum օպերանդի վերաօգտագործմամբ։

Ծրագրի բլոկ սխեմա





Ծրագիրը C լեզվով

#include <stdio.h>

#include <stdlib.h>

// Function to calculate the sum of an arithmetic progression

double calculateAPSum(int a1, int d, int n) {

// Sum of AP = n/2 \* (2 \* a1 + (n - 1) \* d)

return (n / 2.0) \* (2 \* a1 + (n - 1) \* d);

}

int main() {

int a1, d, n;

// Input validation for the first term (a1)

while (1) {

printf("Enter the first term (a1): ");

if (scanf("%d", &a1) == 1) {

break; // Valid input, exit the loop

} else {

printf("Invalid input for a1. Please enter a valid integer.\n");

// Clear the input buffer

while (getchar() != '\n');

}

}

// Input validation for the common difference (d)

while (1) {

printf("Enter the common difference (d): ");

if (scanf("%d", &d) == 1) {

break; // Valid input, exit the loop

} else {

printf("Invalid input for d. Please enter a valid integer.\n");

// Clear the input buffer

while (getchar() != '\n');

}

}

// Input validation for the number of terms (n)

while (1) {

printf("Enter the number of terms (n): ");

if (scanf("%d", &n) == 1 && n > 0) {

break; // Valid input, exit the loop

} else {

printf("Invalid input for n. It should be a positive integer.\n");

// Clear the input buffer

while (getchar() != '\n');

}

}

// Calculate the sum of the AP

double sum = calculateAPSum(a1, d, n);

// Output the result

printf("The sum of the arithmetic progression is: %.2f\n", sum);

return 0;

}

Ծրագիրը Assembly լեզվով

%include "io.inc"

section .bss

a1 resd 1 ; Reserve space for the first term

d resd 1 ; Reserve space for the common difference

n resd 1 ; Reserve space for the number of terms

sum resd 1 ; Reserve space for the sum

;resd 1 doubleword(4byte)

section .text

global main

print\_all:

;mov ebp, esp

PRINT\_STRING "a1 = "

PRINT\_DEC 4, [a1]

NEWLINE

PRINT\_STRING "d = "

PRINT\_DEC 4, [d]

NEWLINE

PRINT\_STRING "n = "

PRINT\_DEC 4, [n]

NEWLINE

ret

main:

mov ebp, esp ; Set up the stack frame

; Get input for the first term (a1)

GET\_DEC 4, eax ; Read first term into eax

mov [a1], eax ; Store the first term in memory

; Get input for the common difference (d)

GET\_DEC 4, eax ; Read common difference into eax

mov [d], eax ; Store the common difference in memory

; Get input for the number of terms (n)

GET\_DEC 4, eax ; Read number of terms into eax

mov [n], eax ; Store the number of terms in memory

call print\_all

; Load values into registers for calculation

mov eax, [a1] ; Load a1 into eax

mov ebx, [d] ; Load d into ebx

mov ecx, [n] ; Load n into ecx

; Calculate (n - 1) \* d

dec ecx ; ecx = n - 1

imul ebx, ecx ; ebx = (n - 1) \* d

; Calculate 2 \* a1

shl eax, 1 ; eax = 2 \* a1

; Add 2 \* a1 + (n - 1) \* d

add eax, ebx ; eax = 2 \* a1 + (n - 1) \* d

mov [sum], eax ;sum = eax

; Calculate sum = n \* (2 \* a1 + (n - 1) \* d)

mov ebx, [sum] ; ebx = 2\*a1+(n-1)\*d

mov eax, [n] ; eax = n

imul eax, ebx ; eax = n \* ebx

shr eax, 1 ; sum = eax/2

mov [sum], eax

PRINT\_STRING "Sum = "

PRINT\_DEC 4, [sum] ; Print the result

NEWLINE

; Negate the value of sum and print

mov eax, [sum] ; Load sum into eax

neg eax ; Negate eax (two's complement)

PRINT\_STRING "Negated Sum = "

PRINT\_DEC 4, eax ; Print directly from eax

NEWLINE

; Bitwise NOT of sum and print

mov eax, [sum] ; Load sum into eax

not eax ; Bitwise NOT (invert all bits)

PRINT\_STRING "Bitwise NOT of Sum = "

PRINT\_DEC 4, eax ; Print directly from eax

NEWLINE

; Add with carry (ADC) and print

mov eax, [sum] ; Load sum into eax

add eax, 10 ; Add 10 to eax

adc eax, 1 ; Add carry (CF) to eax

PRINT\_STRING "Sum + 10 + Carry = "

PRINT\_DEC 4, eax ; Print directly from eax

NEWLINE

; Subtract with borrow (SBB) and print

mov eax, [sum] ; Load sum into eax

sub eax, 5 ; Subtract 5 from eax

sbb eax, 1 ; Subtract borrow (CF) from eax

PRINT\_STRING "Sum - 5 - Borrow = "

PRINT\_DEC 4, eax ; Print directly from eax

NEWLINE

; Rotate left (ROL) by 2 bits and print

mov eax, [sum] ; Load sum into eax

rol eax, 2 ; Rotate eax left by 2 bits

PRINT\_STRING "Sum Rol 2 = "

PRINT\_DEC 4, eax ; Print directly from eax

NEWLINE

; Rotate right (ROR) by 3 bits and print

mov eax, [sum] ; Load sum into eax

ror eax, 3 ; Rotate eax right by 3 bits

PRINT\_STRING "Sum Ror 3 = "

PRINT\_DEC 4, eax ; Print directly from eax

NEWLINE

; Shift Right (SHR) by 1 and print

mov eax, [sum] ; Load sum into eax

shr eax, 1 ; Shift eax right by 1 bit

PRINT\_STRING "Sum SHR 1 = "

PRINT\_DEC 4, eax ; Print directly from eax

NEWLINE

; Exchange `sum` with another value (42) and print

mov eax, [sum] ; Load sum into eax

mov ebx, 42 ; Load another value into ebx

xchg eax, ebx ; Exchange eax and ebx

PRINT\_STRING "Sum Exchanged with 42 = "

PRINT\_DEC 4, eax ; Print directly from eax

NEWLINE

; Test if the highest bit is set

mov eax, [sum] ; Load sum into eax

test eax, 0x80000000 ; Test the highest bit (sets flags but no result stored)

PRINT\_STRING "Highest Bit Set = "

jz .bit\_not\_set ; Jump if zero flag is set (highest bit is not set)

PRINT\_STRING "Yes (1)"

jmp .end\_bit\_test

.bit\_not\_set:

PRINT\_STRING "No (0)"

.end\_bit\_test:

NEWLINE

; Compare sum with 100 and print result

mov eax, [sum] ; Load sum into eax

cmp eax, 100 ; Compare sum with 100

PRINT\_STRING "Sum > 100 = "

ja .greater ; Jump if above (unsigned comparison)

PRINT\_STRING "No (0)"

jmp .end\_compare

.greater:

PRINT\_STRING "Yes (1)"

.end\_compare:

NEWLINE

; Push and Pop operations

mov eax, [sum] ; Load sum into eax

push eax ; Push eax onto the stack

pop eax ; Pop it back into eax

PRINT\_STRING "Pushed and Popped Sum = "

PRINT\_DEC 4, eax ; Print directly from eax

NEWLINE

; Exit

xor eax, eax ; Set eax to 0 (return value)

ret

1. BSS սեկցիա

- Սահմանում է 4 փոփոխական՝ a1, d, n, sum

- Նախապատրաստում է տպելու հաղորդագրություններ

2. Փոփոխականների տպում

- Հաջորդաբար տպում է մուտքագրված փոփոխականները

- Օգտագործում է PRINT\_STRING

3. Գումարի արդյունք

- Կատարում է n \* (2 \* a1 + (n - 1) \* d)

- Պահպանում է արդյունքը sum փոփոխականում

4. Արդյունքի տպում

- Տպում է նոր ստացված sum փոփոխականը

- Օգտագործում է նույն PRINT\_STRING տպելաձևը

**Sum-ի վերաօգտագործումը ասեմբլերի օպերատորներով**

5. **Նվազեցում (Rotate Left)**:

* mov eax, [sum]: Լիցքավորում է sum-ը eax-ում։
* rol eax, 2: Դարձնում է eax-ը ձախ (2 բիթ)։
* Տպվում է «Sum Rol 2»։

6. **Դարձել (Rotate Right)**:

* mov eax, [sum]: Լիցքավորում է sum-ը eax-ում։
* ror eax, 3: Դարձնում է eax-ը աջ (3 բիթ)։
* Տպվում է «Sum Ror 3»։

7. **Դարձույթ աջ (Shift Right)**:

* mov eax, [sum]: Լիցքավորում է sum-ը eax-ում։
* shr eax, 1: Shift աջ մեկ բիթ։
* Տպվում է «Sum SHR 1»։

8. **Փոխանակում (Exchange) 42-ի հետ**:

* mov eax, [sum]: Լիցքավորում է sum-ը eax-ում։
* mov ebx, 42: Լիցքավորում է 42-ը ebx-ում։
* xchg eax, ebx: Փոխանակում է eax-ը և ebx-ը։
* Տպվում է «Sum Exchanged with 42»։

9. **Բարձրագույն բիթի ստուգում**:

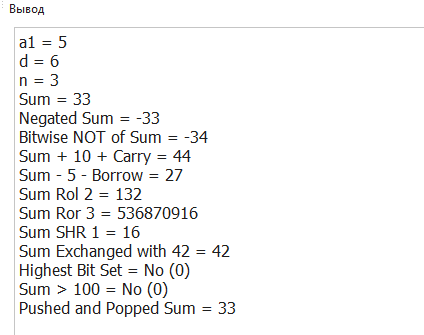
* test eax, 0x80000000: Ստուգում է բարձրագույն բիթը։
* Տպվում է «Yes (1)» կամ «No (0)»։

10. **Համեմատություն 100-ի հետ**:

* cmp eax, 100: Համեմատում է sum-ը 100-ի հետ։
* Տպվում է «Yes (1)» կամ «No (0)»։

11. **Push և Pop**:

* push eax: Տեղադրում է eax-ը ստեկում։
* pop eax: Հանում է ստեկից։
* Տպվում է «Pushed and Popped Sum»։



Համակարգչային ճարտարապետություն

Համակարգչային ճարտարապետությունը համակարգչային բաղադրիչների կառուցվածքն ու կազմակերպումն է, որը սահմանում է, թե ինչպես են դրանք փոխազդում միմյանց հետ։ Սարքավորումների եւ ծրագրային ապահովման փոխհարաբերությունները: Կենտրոնական տարրերն են՝

CPU-ն,

հիշողությունը (RAM, ROM եւ այլն),

bus-երը (data bus, address bus, control bus),

ներմուծում/արտածումը,

ճարտարապետական մոդելները՝ (Ֆոն Նեյմանի եւ Հարվարդի մոդելներ):  
Այս գաղափարները ժամանակակից հաշվարկային համակարգերի հիմքն են, եւ դրանցընկալումը կարեւոր է ծրագրերի մշակման եւ համակարգերի նախագծման համար:

RISC եւ CISC ճարտարապետություններ

RISC՝ կենտրոնանում է պարզ հրամանների արագ կատարման վրա։ CPU- ն կատարում է պարզ հրամաններ, որոնք ավարտվում են մեկ ժամային ցիկլում։

Օրինակներ են ARM եւ MIPS ճարտարապետությունները։ CISC՝ ունի ավելի բարդ հրամաններ, որոնք կարող են կատարել մի քանի գործողություն մեկ հրամանի մեջ։ Օրինակ է՝ x86 ճարտարապետությունը։

Ասսեմբլեր և ռեգիստրներ

xN8A6S-6M4 (Netwide Assembler) լեզուն նախատեսված է x86 եւ ճարտարապետությունների համար: Այն օգտագործվում է ցածր մակարդակի ծրագրերի մշակման համար: NASM-ը հայտնի է պարզ syntax-ով եւ բազմապլատֆորմ աջակցությամբ:

NASM-ի հիմնական առանձնահատկությունները  
1. Բազմապլատֆորմ աջակցություն: Աշխատում է Linux, Windows, եւ DOS համակարգերում։

2. Ճկուն սինտաքս: Պարզ syntax-ը հեշտացնում է ծրագրերի ընթեռնելիությունն ու պահպանումը։

3. Հիշողության կառավարման աջակցություն: Կոդի եւ տվյալների հատվածների հստակ բաժանում։

4. Ֆայլային ձեւաչափեր: Աջակցում է ELF, COFF եւ այլ ձեւաչափերի։

5. Ղեկավարող հրամաններ: Թույլ են տալիս հեշտացնել կոմպիլյացիայի գործընթացը։