SOUTHERN
UNIVERSITY COLLEGE
南方大学学院

CSIS3003
Project 2

CSIS2083
Computer Organization Architecture
Group Project

Square Root- Assembly

Ho Weng Yin D210044A

Edmond Coh Rui En D210022A

Yap Zhi Hao D210252C

2023

Create an assembly language program to calculate the result of square root.

The report requires to explain your platform (Operating system, tool, compiler), assembly language type.

1	2	3	4
Report formatting (20%)	No standardize the report format, no cover page, table of content, page number and so on	Report with cover page, table of content and page number	Report come with cover page, table of content, page number and all justify all paragraph and the font formatting are same.
Development environment (40%)	Provide the basic info about system platform	Explain about the assembly language development environment	Explain in the detail about the advantage and disadvantage of the platform
Assembly application (40%)	Program able to start running	Program running with minor error	Complete running application without error

Table of Content

Introduction

SOLUTION 1:GCC TO Assembly Code (100% working) 4

SOLUTION 2: NASM with Microsoft WSL (100% Work but may require debug) 5

SOLUTION 3: TASM (Work but no output) 14

SOLUTION 4: MASM Intel 8086 (100% work) 16

SOLUTION 5: EMU8086 21

SOLUTION 6: MIPS software to write assembly code (100% Working) 22

SOLUTION 7: MASM Microsoft Visual Studio (Work but may require debug) 26

SOLUTION 8: MASM editor (Work but may require debug) 26

SOLUTION 9: ARM Keil (100% Working) with 8051 Assembly Program Code to find Square Root - AT89C51 – Keil 26

SOLUTION 10: IDA (100% but reverse engineering) 28

Conclusion: 31

Introduction

Create an assembly language program to calculate the result of square root

Our project creates this program through several solutions and methods. Therefore, we can classify our solution into NASM, MASM, TASM, GCC to assembly code, MASM with Intel 8086. First and foremost, NASM is used for Linux. MASM and TASM are used for Microsoft Windows. GCC to assembly could be used cross platform. Another one is using MASM by using Intel 8086. One last solution is using IDA software to do the reverse engineering and we know the assembly code. Besides, we are using simulator and Keil as well as the MIPS to do the assembly code.

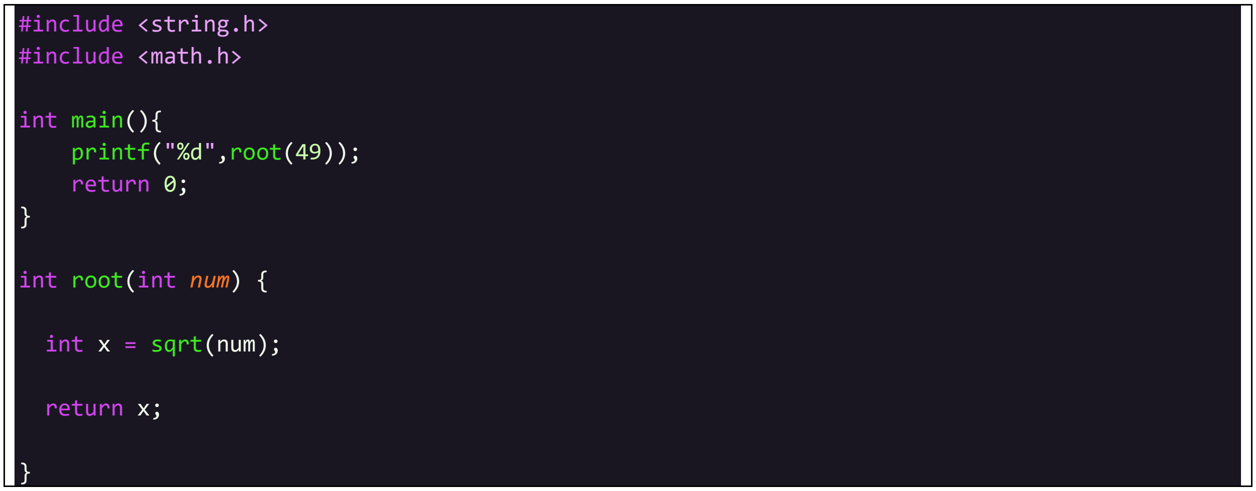
There are many solutions some of them working well and running and showing the output for square root of number.

The easiest solution is using GCC to assembly code (Run smooth without any error). First and foremost, create a main.c file as C programming source code. MinGW should be installed before running these.

Link: https://www.mingw-w64.org/

SOLUTION 1: GCC TO Assembly Code (100% working)

Main.c



```
#include <string.h>
#include <math.h>

int main(){
    printf("%d",root(49));
    return 0;
}

int root(int num) {

  int x = sqrt(num);

  return x;

}
```

In the source code, we are doing the square root of 49. The result will be 7. Then, we use GCC to convert this to assembly code.

Main.s

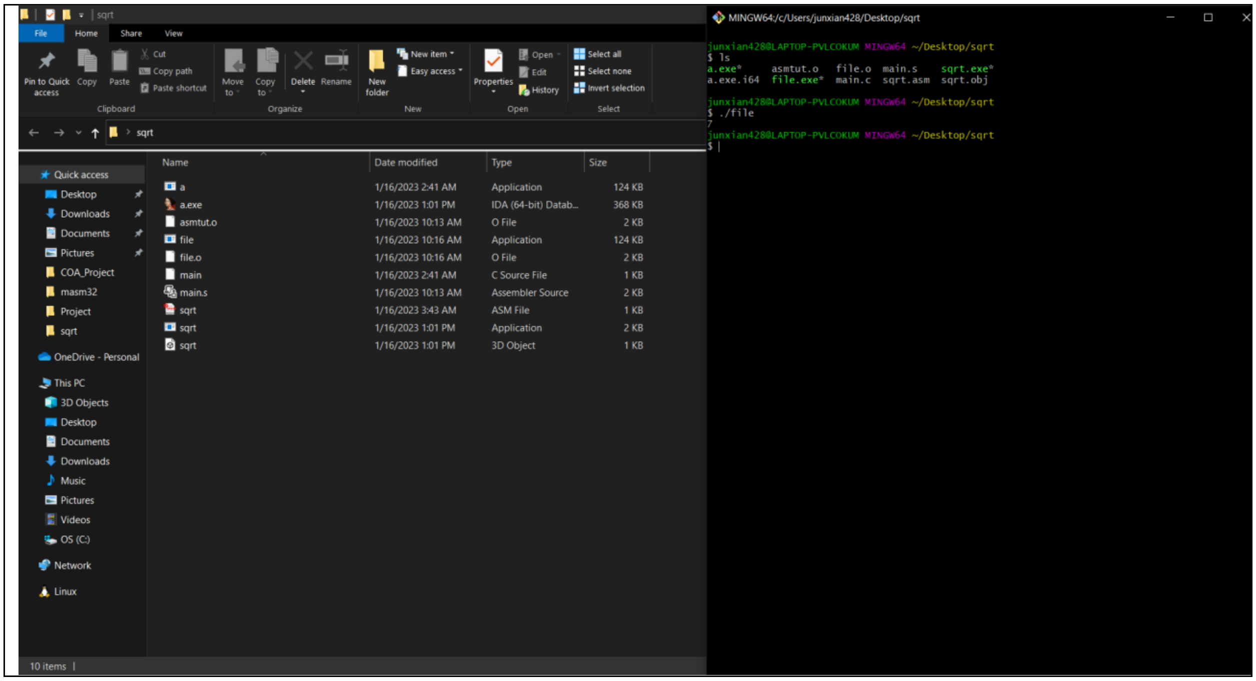
```
	.file	"main.c"
	.text
	.def	__main;	.scl	2;	.type	32;	.endef
	.section .rdata,"dr"
.LC0:
	.ascii "%d\0"
	.text
	.globl	main
	.def	main;	.scl	2;	.type	32;	.endef
	.seh_proc	main
main:
	pushq	%rbp
	.seh_pushreg	%rbp
```

```
        movq    %rsp, %rbp
        .seh_setframe   %rbp, 0
        subq    $32, %rsp
        .seh_stackalloc 32
        .seh_endprologue
        call    __main
        movl    $49, %ecx
        call    root
        movl    %eax, %edx
        leaq    .LC0(%rip), %rax
        movq    %rax, %rcx
        call    printf
        movl    $0, %eax
        addq    $32, %rsp
        popq    %rbp
        ret
        .seh_endproc
        .globl  root
        .def    root;   .scl    2;      .type   32;     .endef
        .seh_proc       root
root:
        pushq   %rbp
        .seh_pushreg    %rbp
        movq    %rsp, %rbp
        .seh_setframe   %rbp, 0
        subq    $48, %rsp
        .seh_stackalloc 48
        .seh_endprologue
        movl    %ecx, 16(%rbp)
        pxor    %xmm1, %xmm1
        cvtsi2sdl       16(%rbp), %xmm1
        movq    %xmm1, %rax
        movq    %rax, %xmm0
        call    sqrt
        cvttsd2sil      %xmm0, %eax
        movl    %eax, -4(%rbp)
        movl    -4(%rbp), %eax
        addq    $48, %rsp
        popq    %rbp
        ret
        .seh_endproc
        .ident "GCC: (Rev8, Built by MSYS2 project) 11.2.0"
        .def   printf; .scl    2;      .type  32;    .endef
        .def   sqrt;   .scl    2;      .type  32;    .endef
```

This solution is the easiest and quickest. Since it is high level programming, it is converted back into lower programming therefore it is quite easy.

```
assembly for sqrt root. One is by using gcc compile into assembly another
is using TASM

for first one main.s

gcc -c file.S -o file.o

gcc file.o -o file
```

Output:



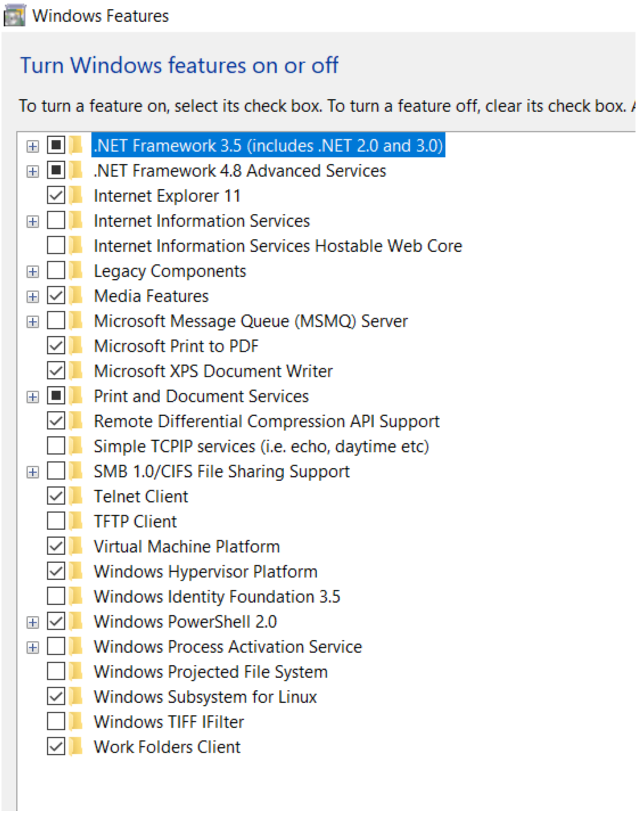
Name	Date modified	Type	Size
a	1/16/2023 2:41 AM	Application	124 KB
a.exe	1/16/2023 1:01 PM	IDA (64-bit) Datab...	368 KB
asmtut.o	1/16/2023 10:13 AM	O File	2 KB
file	1/16/2023 10:16 AM	Application	124 KB
file.o	1/16/2023 10:16 AM	O File	2 KB
main	1/16/2023 2:41 AM	C Source File	1 KB
main.s	1/16/2023 10:13 AM	Assembler Source	2 KB
sqrt	1/16/2023 3:43 AM	ASM File	1 KB
sqrt	1/16/2023 1:01 PM	Application	2 KB
sqrt	1/16/2023 1:01 PM	3D Object	1 KB

MINGW64:/c/Users/junxian428/Desktop/sqrt

```
junxian428@LAPTOP-PVLCOKUM MINGW64 ~/Desktop/sqrt
$ ls
a.exe*     asmtut.o   file.o  main.s    sqrt.exe*
a.exe.i64  file.exe*  main.c  sqrt.asm  sqrt.obj

junxian428@LAPTOP-PVLCOKUM MINGW64 ~/Desktop/sqrt
$ ./file
7
junxian428@LAPTOP-PVLCOKUM MINGW64 ~/Desktop/sqrt
$ |
```

SOLUTION 2: NASM with Microsoft WSL (100% Work but may require debug)

First and foremost, turn on Microsoft Feature,



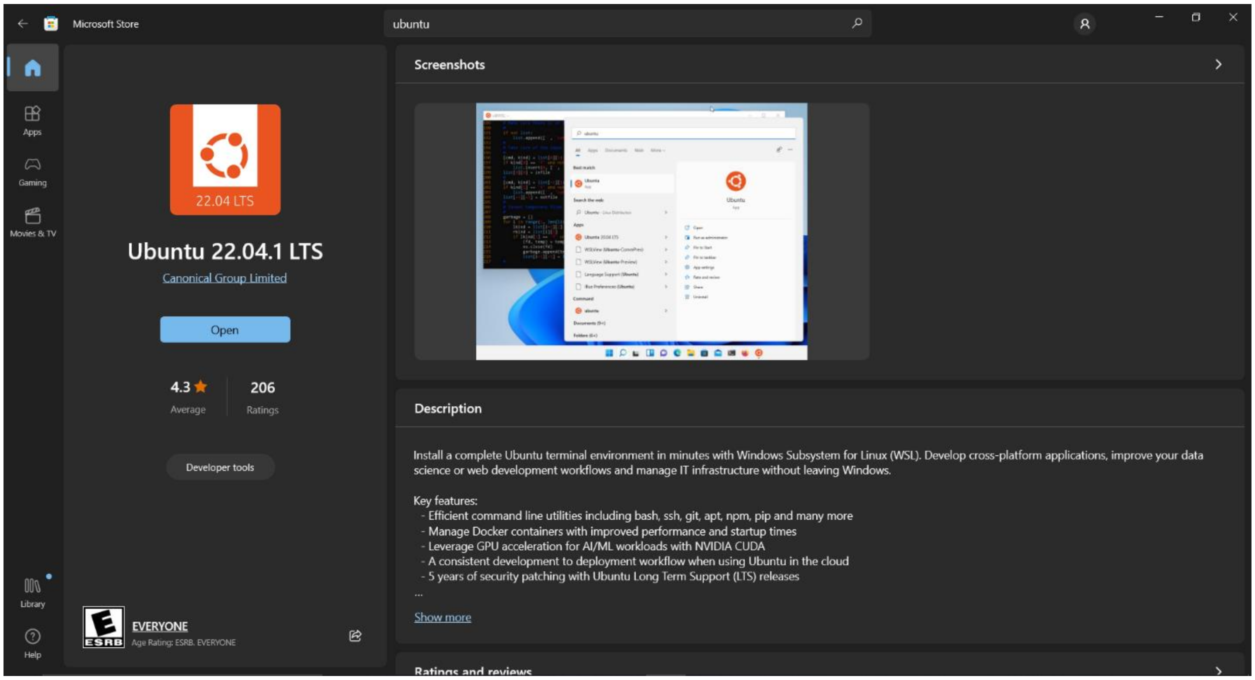
Windows Features

Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A

- ■ .NET Framework 3.5 (includes .NET 2.0 and 3.0)
- ■ .NET Framework 4.8 Advanced Services
- ☑ Internet Explorer 11
- □ Internet Information Services
- □ Internet Information Services Hostable Web Core
- □ Legacy Components
- ☑ Media Features
- □ Microsoft Message Queue (MSMQ) Server
- ☑ Microsoft Print to PDF
- ☑ Microsoft XPS Document Writer
- ■ Print and Document Services
- ☑ Remote Differential Compression API Support
- □ Simple TCPIP services (i.e. echo, daytime etc)
- □ SMB 1.0/CIFS File Sharing Support
- ☑ Telnet Client
- □ TFTP Client
- ☑ Virtual Machine Platform
- ☑ Windows Hypervisor Platform
- □ Windows Identity Foundation 3.5
- ☑ Windows PowerShell 2.0
- □ Windows Process Activation Service
- □ Windows Projected File System
- ☑ Windows Subsystem for Linux
- □ Windows TIFF IFilter
- ☑ Work Folders Client

Then, go to Microsoft Store to download WSL Ubuntu



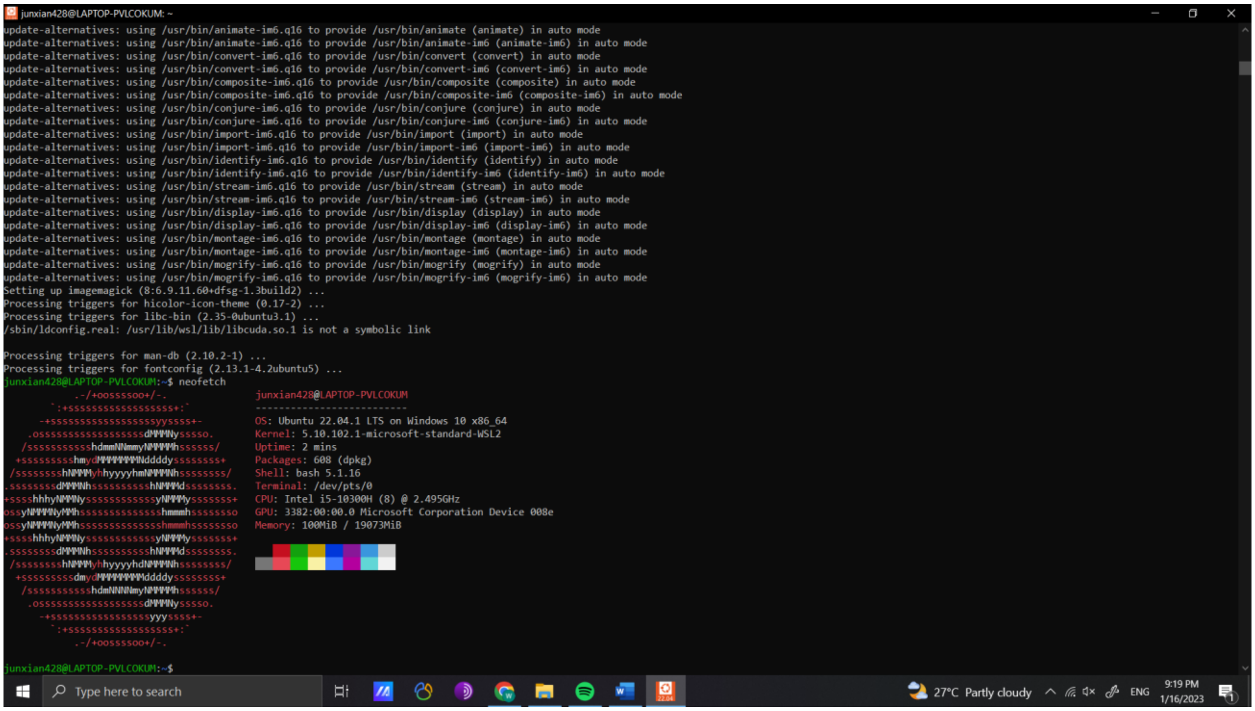
Ubuntu 22.04.1 LTS

Canonical Group Limited

Install a complete Ubuntu terminal environment in minutes with Windows Subsystem for Linux (WSL). Develop cross-platform applications, improve your data science or web development workflows and manage IT infrastructure without leaving Windows.

Key features:
- Efficient command line utilities including bash, ssh, git, apt, npm, pip and many more
- Manage Docker containers with improved performance and startup times
- Leverage GPU acceleration for AI/ML workloads with NVIDIA CUDA
- A consistent development to deployment workflow when using Ubuntu in the cloud
- 5 years of security patching with Ubuntu Long Term Support (LTS) releases
...

After installing Ubuntu 22.04 LTS, then you can launch your WSL Ubuntu



```
update-alternatives: using /usr/bin/animate-im6.q16 to provide /usr/bin/animate (animate) in auto mode
update-alternatives: using /usr/bin/animate-im6.q16 to provide /usr/bin/animate-im6 (animate-im6) in auto mode
update-alternatives: using /usr/bin/convert-im6.q16 to provide /usr/bin/convert (convert) in auto mode
update-alternatives: using /usr/bin/convert-im6.q16 to provide /usr/bin/convert-im6 (convert-im6) in auto mode
update-alternatives: using /usr/bin/composite-im6.q16 to provide /usr/bin/composite (composite) in auto mode
update-alternatives: using /usr/bin/composite-im6.q16 to provide /usr/bin/composite-im6 (composite-im6) in auto mode
update-alternatives: using /usr/bin/conjure-im6.q16 to provide /usr/bin/conjure (conjure) in auto mode
update-alternatives: using /usr/bin/conjure-im6.q16 to provide /usr/bin/conjure-im6 (conjure-im6) in auto mode
update-alternatives: using /usr/bin/import-im6.q16 to provide /usr/bin/import (import) in auto mode
update-alternatives: using /usr/bin/import-im6.q16 to provide /usr/bin/import-im6 (import-im6) in auto mode
update-alternatives: using /usr/bin/identify-im6.q16 to provide /usr/bin/identify (identify) in auto mode
update-alternatives: using /usr/bin/identify-im6.q16 to provide /usr/bin/identify-im6 (identify-im6) in auto mode
update-alternatives: using /usr/bin/stream-im6.q16 to provide /usr/bin/stream (stream) in auto mode
update-alternatives: using /usr/bin/stream-im6.q16 to provide /usr/bin/stream-im6 (stream-im6) in auto mode
update-alternatives: using /usr/bin/display-im6.q16 to provide /usr/bin/display (display) in auto mode
update-alternatives: using /usr/bin/display-im6.q16 to provide /usr/bin/display-im6 (display-im6) in auto mode
update-alternatives: using /usr/bin/montage-im6.q16 to provide /usr/bin/montage (montage) in auto mode
update-alternatives: using /usr/bin/montage-im6.q16 to provide /usr/bin/montage-im6 (montage-im6) in auto mode
update-alternatives: using /usr/bin/mogrify-im6.q16 to provide /usr/bin/mogrify (mogrify) in auto mode
update-alternatives: using /usr/bin/mogrify-im6.q16 to provide /usr/bin/mogrify-im6 (mogrify-im6) in auto mode
Setting up imagemagick (8:6.9.11.60+dfsg-1.3build2) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
Processing triggers for libc-bin (2.35-0ubuntu3.1) ...
/sbin/ldconfig.real: /usr/lib/wsl/lib/libcuda.so.1 is not a symbolic link

Processing triggers for man-db (2.10.2-1) ...
Processing triggers for fontconfig (2.13.1-4.2ubuntu5) ...
junxian428@LAPTOP-PVLCOKUM:~$ neofetch
            .-/+oossssoo+/-.               junxian428@LAPTOP-PVLCOKUM
        `:+ssssssssssssssssss+:`           --------------------------
      -+ssssssssssssssssssyyssss+-         OS: Ubuntu 22.04.1 LTS on Windows 10 x86_64
    .ossssssssssssssssssdMMMNysssso.       Kernel: 5.10.102.1-microsoft-standard-WSL2
   /ssssssssssshdmmNNmmyNMMMMhssssss/      Uptime: 2 mins
  +ssssssssshmydMMMMMMMNddddyssssssss+     Packages: 608 (dpkg)
 /sssssssshNMMMyhhyyyyhmNMMMNhssssssss/    Shell: bash 5.1.16
.ssssssssdMMMNhsssssssssshNMMMdssssssss.   Terminal: /dev/pts/0
+sssshhhyNMMNyssssssssssssyNMMMysssssss+   CPU: Intel i5-10300H (8) @ 2.495GHz
ossyNMMMNyMMhsssssssssssssshmmmhssssssso   GPU: 3382:00:00.0 Microsoft Corporation Device 008e
ossyNMMMNyMMhsssssssssssssshmmmhssssssso   Memory: 100MiB / 19073MiB
+sssshhhyNMMNyssssssssssssyNMMMysssssss+
.ssssssssdMMMNhsssssssssshNMMMdssssssss.
 /sssssssshNMMMyhhyyyyhdNMMMNhssssssss/
  +sssssssssdmydMMMMMMMMddddyssssssss+
   /ssssssssssshdmNNNNmyNMMMMhssssss/
    .ossssssssssssssssssdMMMNysssso.
      -+sssssssssssssssssyyyssss+-
        `:+ssssssssssssssssss+:`
            .-/+oossssoo+/-.

junxian428@LAPTOP-PVLCOKUM:~$
```

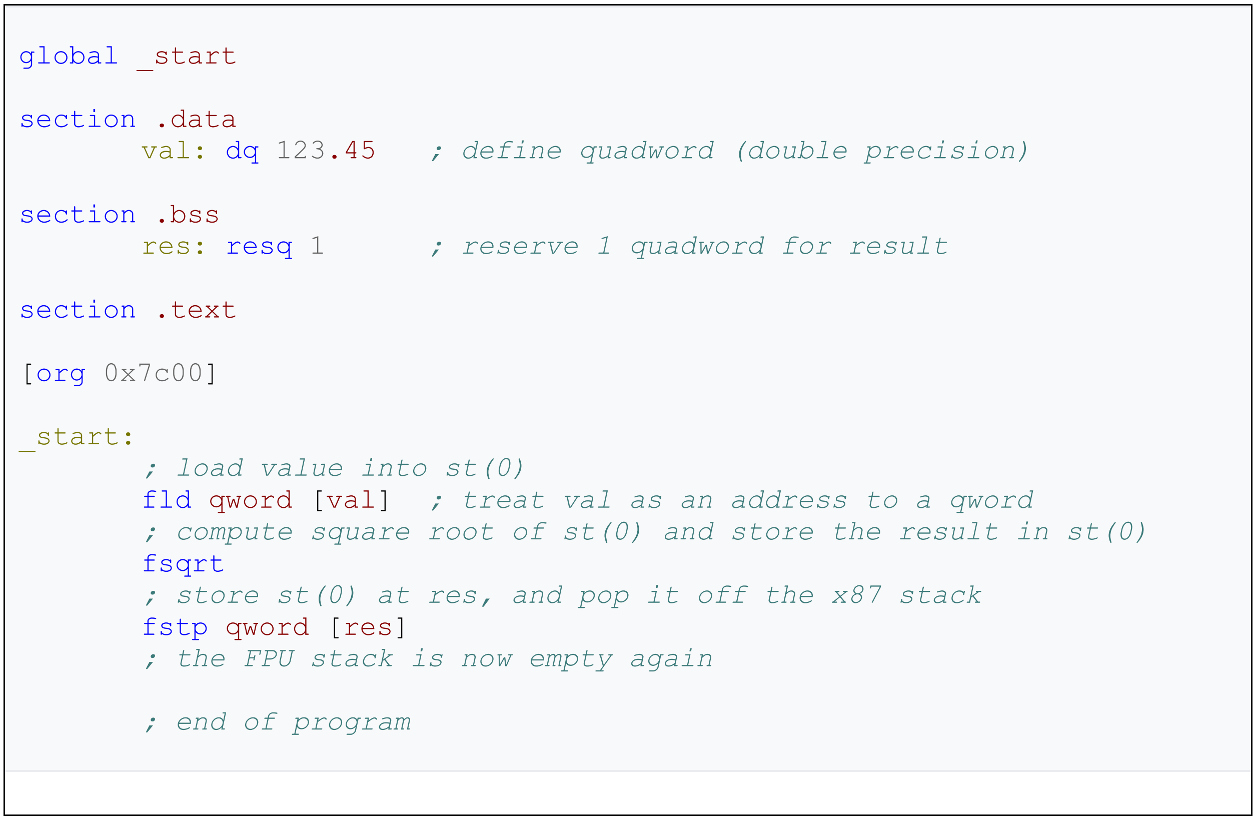
Then download NASM

Sudo apt-get install nasm

Example (Source Code)

https://en.wikibooks.org/wiki/X86_Assembly/Floating_Point

The following program (using NASM syntax) calculates the square root of 123.45.



```
global _start

section .data
       val: dq 123.45   ; define quadword (double precision)

section .bss
       res: resq 1      ; reserve 1 quadword for result

section .text

[org 0x7c00]

_start:
       ; load value into st(0)
       fld qword [val]  ; treat val as an address to a qword
       ; compute square root of st(0) and store the result in st(0)
       fsqrt
       ; store st(0) at res, and pop it off the x87 stack
       fstp qword [res]
       ; the FPU stack is now empty again

       ; end of program
```

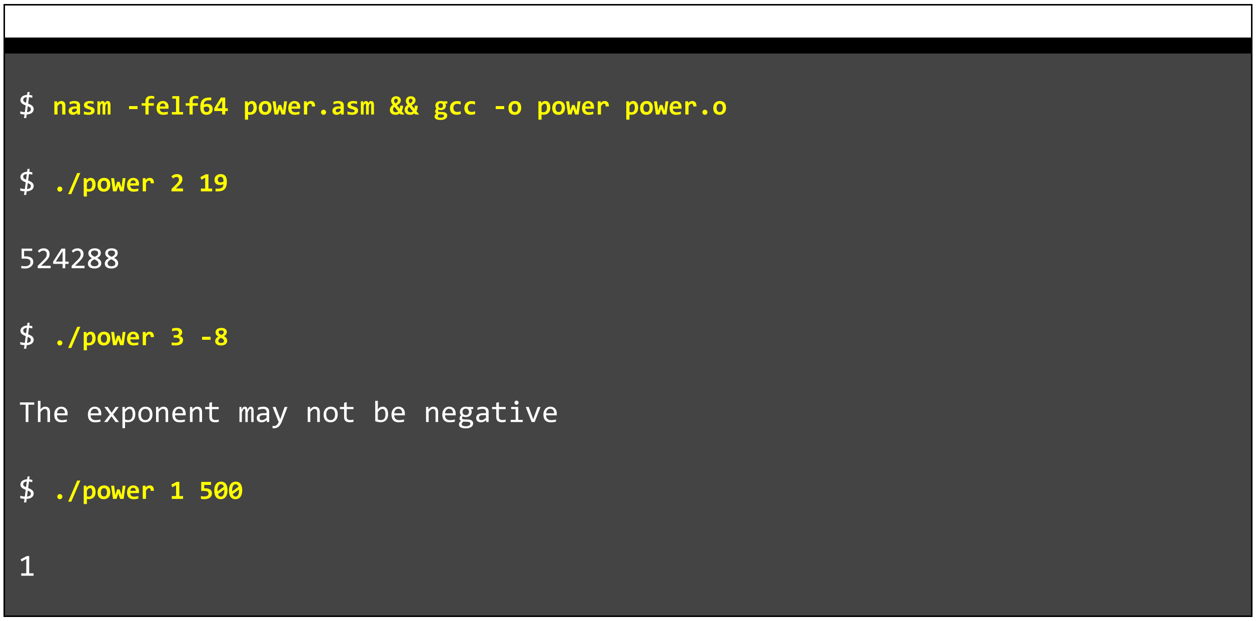
Another Solution for NASM:

https://cs.lmu.edu/~ray/notes/nasmtutorial/

Maybe we cannot find square root but the root of power 2 could be reverse back to find the square root right in the mathematics sense

```
; ----------------------------------------------------------------------------
; A 64-bit command line application to compute x^y.
;
; Syntax: power x y
; x and y are (32-bit) integers
; ----------------------------------------------------------------------------

        global  main
        extern  printf
        extern  puts
        extern  atoi

        section .text
main:
        push    r12                 ; save callee-save registers
        push    r13
        push    r14
        ; By pushing 3 registers our stack is already aligned for calls

        cmp     rdi, 3              ; must have exactly two arguments
        jne     error1

        mov     r12, rsi            ; argv

; We will use ecx to count down form the exponent to zero, esi to hold the
; value of the base, and eax to hold the running product.

        mov     rdi, [r12+16]       ; argv[2]
        call    atoi                ; y in eax
        cmp     eax, 0              ; disallow negative exponents
        jl      error2
        mov     r13d, eax           ; y in r13d

        mov     rdi, [r12+8]        ; argv
        call    atoi                ; x in eax
        mov     r14d, eax           ; x in r14d

        mov     eax, 1              ; start with answer = 1
check:
        test    r13d, r13d          ; we're counting y downto 0
        jz      gotit               ; done
        imul    eax, r14d           ; multiply in another x
        dec     r13d
        jmp     check
gotit:                              ; print report on success
        mov     rdi, answer
        movsxd  rsi, eax
        xor     rax, rax
        call    printf
        jmp     done
error1:                                 ; print error message
        mov     edi, badArgumentCount
        call    puts
        jmp     done
error2:                                 ; print error message
        mov     edi, negativeExponent
        call    puts
done:                                   ; restore saved registers
        pop     r14
        pop     r13
        pop     r12
        ret

answer:
        db      "%d", 10, 0
badArgumentCount:
        db      "Requires exactly two arguments", 10, 0
negativeExponent:
        db      "The exponent may not be negative", 10, 0
```

Console:

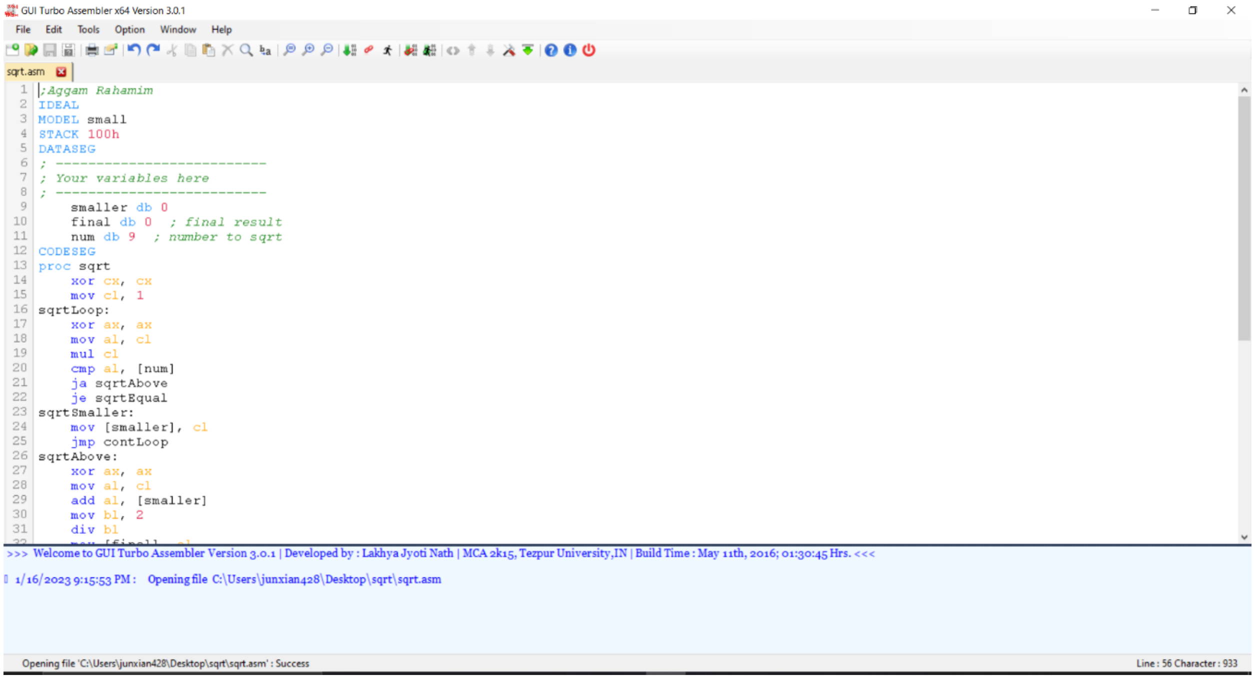


```
$ nasm -felf64 power.asm && gcc -o power power.o

$ ./power 2 19

524288

$ ./power 3 -8

The exponent may not be negative

$ ./power 1 500

1
```

```
$ ./power 1

Requires exactly two arguments
```

SOLUTION 3: TASM (Work but no output)

Download Link: https://sourceforge.net/projects/guitasm8086/



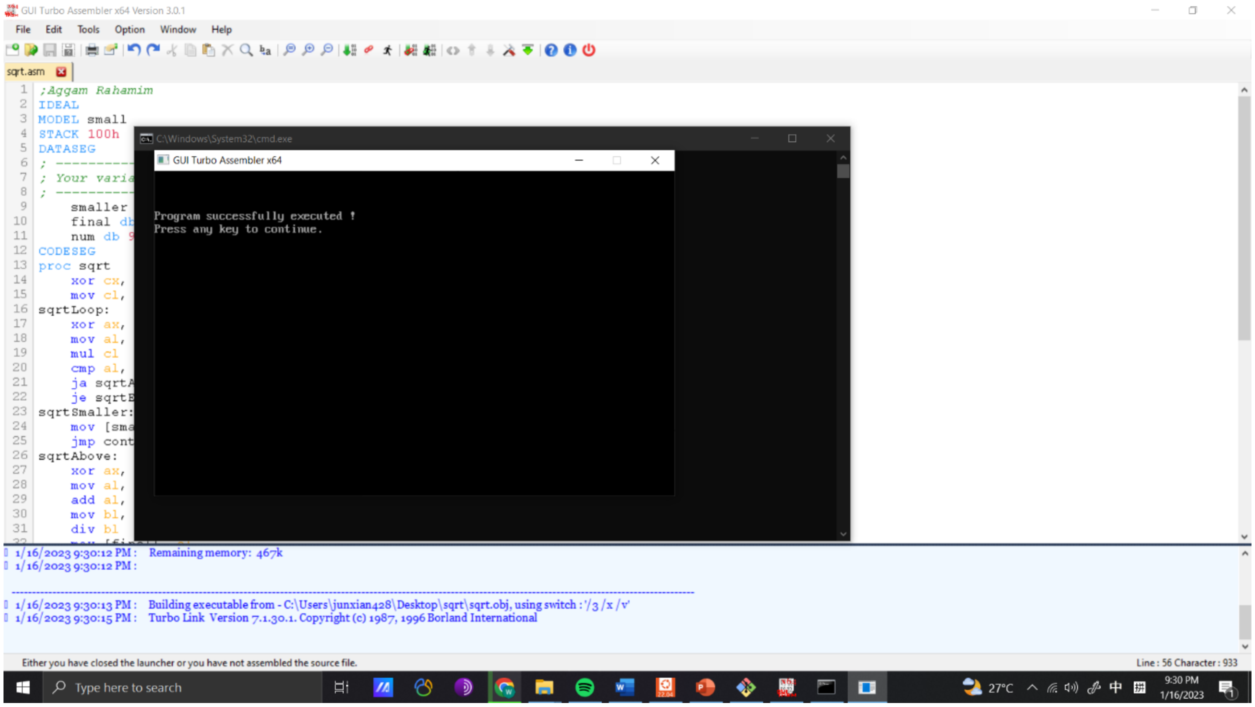
```
;Aggam Rahamim
IDEAL
MODEL small
STACK 100h
DATASEG
; --------------------------
; Your variables here
; --------------------------
    smaller db 0
    final db 0  ; final result
    num db 9  ; number to sqrt
CODESEG
proc sqrt
    xor cx, cx
    mov cl, 1
sqrtLoop:
    xor ax, ax
    mov al, cl
    mul cl
    cmp al, [num]
    ja sqrtAbove
    je sqrtEqual
sqrtSmaller:
    mov [smaller], cl
    jmp contLoop
sqrtAbove:
    xor ax, ax
    mov al, cl
    add al, [smaller]
    mov bl, 2
    div bl
```

```
>>> Welcome to GUI Turbo Assembler Version 3.0.1 | Developed by : Lakhya Jyoti Nath | MCA 2k15, Tezpur University,IN | Build Time : May 11th, 2016; 01:30:45 Hrs. <<<

1/16/2023 9:15:53 PM :   Opening file  C:\Users\junxian428\Desktop\sqrt\sqrt.asm

Opening file 'C:\Users\junxian428\Desktop\sqrt\sqrt.asm' : Success                    Line : 56 Character : 933
```

Square root source code

```
IDEAL
MODEL small
STACK 100h
DATASEG
; --------------------------
; Your variables here
; --------------------------
    smaller db 0
    final db 0  ; final result
    num db 9  ; number to sqrt
CODESEG
proc sqrt
    xor cx, cx
    mov cl, 1
sqrtLoop:
    xor ax, ax
    mov al, cl
    mul cl
    cmp al, [num]
    ja sqrtAbove
    je sqrtEqual
sqrtSmaller:
    mov [smaller], cl
    jmp contLoop
sqrtAbove:
    xor ax, ax
    mov al, cl
    add al, [smaller]
    mov bl, 2
    div bl
    mov [final], al
    jmp endLoop
sqrtEqual:
    mov [final], cl
    jmp endLoop
contLoop:
    inc cl
    cmp cl, [num]
    jb sqrtLoop
endLoop:
    ret
endp
start:
    mov ax, @data
    mov ds, ax
; -------------------------
; Your code here
; -------------------------
    mov [num], 49  ; number to sqrt
    call sqrtLoop

exit:
    mov ax, 4C00h
    int 21h
END start
```

Output



GUI Turbo Assembler x64 Version 3.0.1
File Edit Tools Option Window Help
sqrt.asm
C:\Windows\System32\cmd.exe
GUI Turbo Assembler x64
Type here to search
27°C
9:30 PM
1/16/2023

```
;Aggam Rahamim
IDEAL
MODEL small
STACK 100h
DATASEG
; ----------
; Your varia
; ----------
	smaller
	final db
	num db 9
CODESEG
proc sqrt
	xor cx,
	mov cl,
sqrtLoop:
	xor ax,
	mov al,
	mul cl
	cmp al,
	ja sqrtA
	je sqrtB
sqrtSmaller:
	mov [sma
	jmp cont
sqrtAbove:
	xor ax,
	mov al,
	add al,
	mov bl,
	div bl
```

```
Program successfully executed !
Press any key to continue.
```

```
1/16/2023 9:30:12 PM :   Remaining memory:  467k
1/16/2023 9:30:12 PM :
------------------------------------------------------------------------------------------------
1/16/2023 9:30:13 PM :   Building executable from - C:\Users\junxian428\Desktop\sqrt\sqrt.obj, using switch : '/3 /x /v'
1/16/2023 9:30:15 PM :   Turbo Link  Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International
```

Either you have closed the launcher or you have not assembled the source file.

Line : 56 Character : 933

SOLUTION 4: MASM Intel 8086 (100% work)

Tutorial to setup

https://medium.com/@axayjha/getting-started-with-masm-8086-assembly-c625478265d8

After following those, then can run MASM program with Intel 8086

This one is success

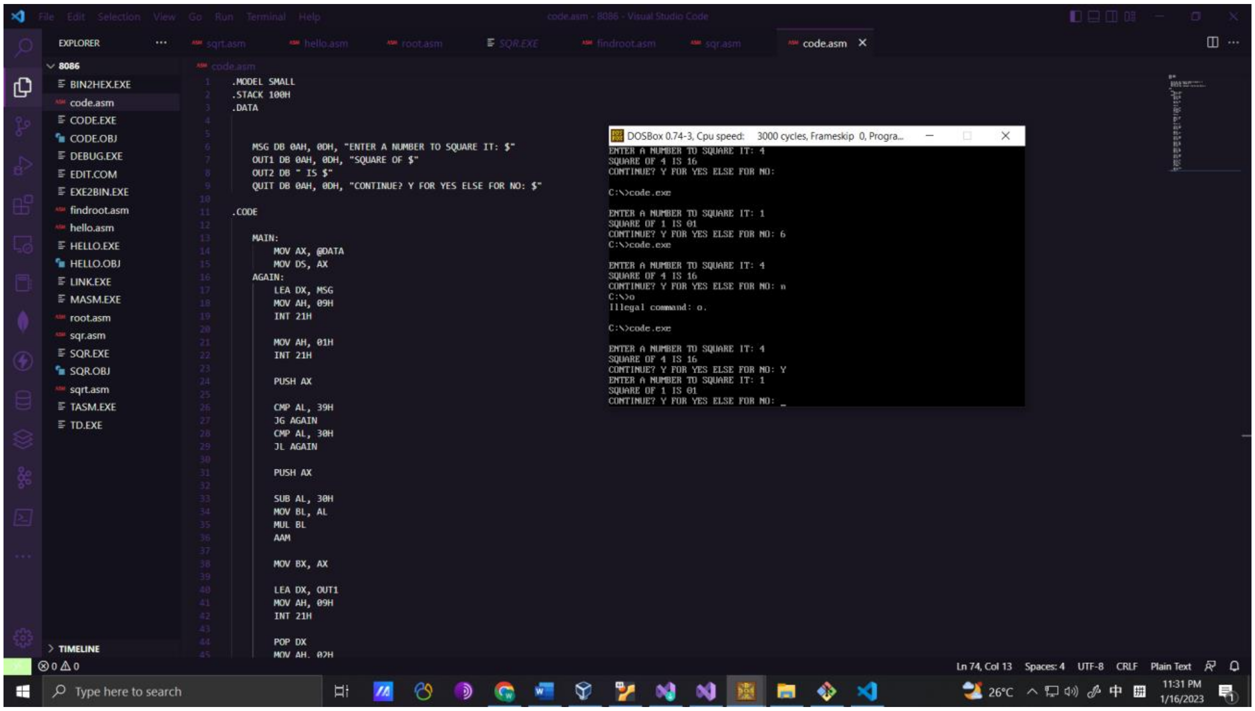
Reference Link:

https://engineering-lab.blogspot.com/

```
.MODEL SMALL
.STACK 100H
.DATA


   MSG DB 0AH, 0DH, "ENTER A NUMBER TO SQUARE IT: $"
```

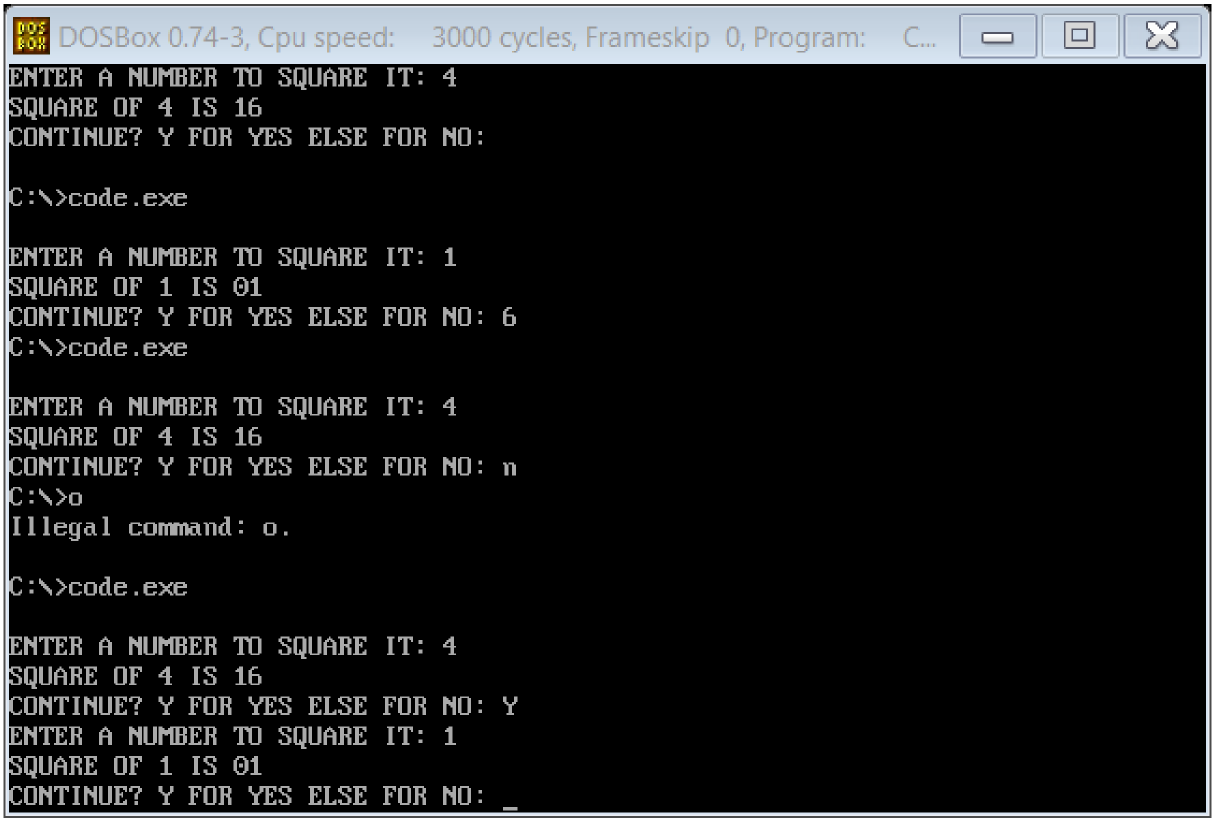
```
    OUT1 DB 0AH, 0DH, "SQUARE OF $"
    OUT2 DB " IS $"
    QUIT DB 0AH, 0DH, "CONTINUE? Y FOR YES ELSE FOR NO: $"

.CODE

    MAIN:
        MOV AX, @DATA
        MOV DS, AX
    AGAIN:
        LEA DX, MSG
        MOV AH, 09H
        INT 21H

        MOV AH, 01H
        INT 21H

        PUSH AX

        CMP AL, 39H
        JG AGAIN
        CMP AL, 30H
        JL AGAIN

        PUSH AX

        SUB AL, 30H
        MOV BL, AL
        MUL BL
        AAM

        MOV BX, AX

        LEA DX, OUT1
        MOV AH, 09H
        INT 21H

        POP DX
        MOV AH, 02H
        INT 21H
        LEA DX, OUT2
        MOV AH, 09H
        INT 21H

        MOV DL, BH
        ADD DL, 30H
        MOV AH, 02H
        INT 21H

        MOV DL, BL
        ADD DL, 30H
        MOV AH, 02H
        INT 21H

        LEA DX, QUIT
        MOV AH, 09H
        INT 21H

        MOV AH, 01H
        INT 21H
        OR AL, 20H
        CMP AL, 'y'
        JE AGAIN

        MOV AH, 04CH
        INT 21H
      END MAIN
```

Output



```
.MODEL SMALL
.STACK 100H
.DATA


    MSG DB 0AH, 0DH, "ENTER A NUMBER TO SQUARE IT: $"
    OUT1 DB 0AH, 0DH, "SQUARE OF $"
    OUT2 DB " IS $"
    QUIT DB 0AH, 0DH, "CONTINUE? Y FOR YES ELSE FOR NO: $"

.CODE

    MAIN:
        MOV AX, @DATA
        MOV DS, AX
    AGAIN:
        LEA DX, MSG
        MOV AH, 09H
        INT 21H

        MOV AH, 01H
        INT 21H

        PUSH AX

        CMP AL, 39H
        JG AGAIN
        CMP AL, 30H
        JL AGAIN

        PUSH AX

        SUB AL, 30H
        MOV BL, AL
        MUL BL
        AAM

        MOV BX, AX

        LEA DX, OUT1
        MOV AH, 09H
        INT 21H

        POP DX
        MOV AH, 02H
```

```
ENTER A NUMBER TO SQUARE IT: 4
SQUARE OF 4 IS 16
CONTINUE? Y FOR YES ELSE FOR NO:

C:\>code.exe

ENTER A NUMBER TO SQUARE IT: 1
SQUARE OF 1 IS 01
CONTINUE? Y FOR YES ELSE FOR NO: 6
C:\>code.exe

ENTER A NUMBER TO SQUARE IT: 4
SQUARE OF 4 IS 16
CONTINUE? Y FOR YES ELSE FOR NO: n
C:\>o
Illegal command: o.

C:\>code.exe

ENTER A NUMBER TO SQUARE IT: 4
SQUARE OF 4 IS 16
CONTINUE? Y FOR YES ELSE FOR NO: Y
ENTER A NUMBER TO SQUARE IT: 1
SQUARE OF 1 IS 01
CONTINUE? Y FOR YES ELSE FOR NO: _
```



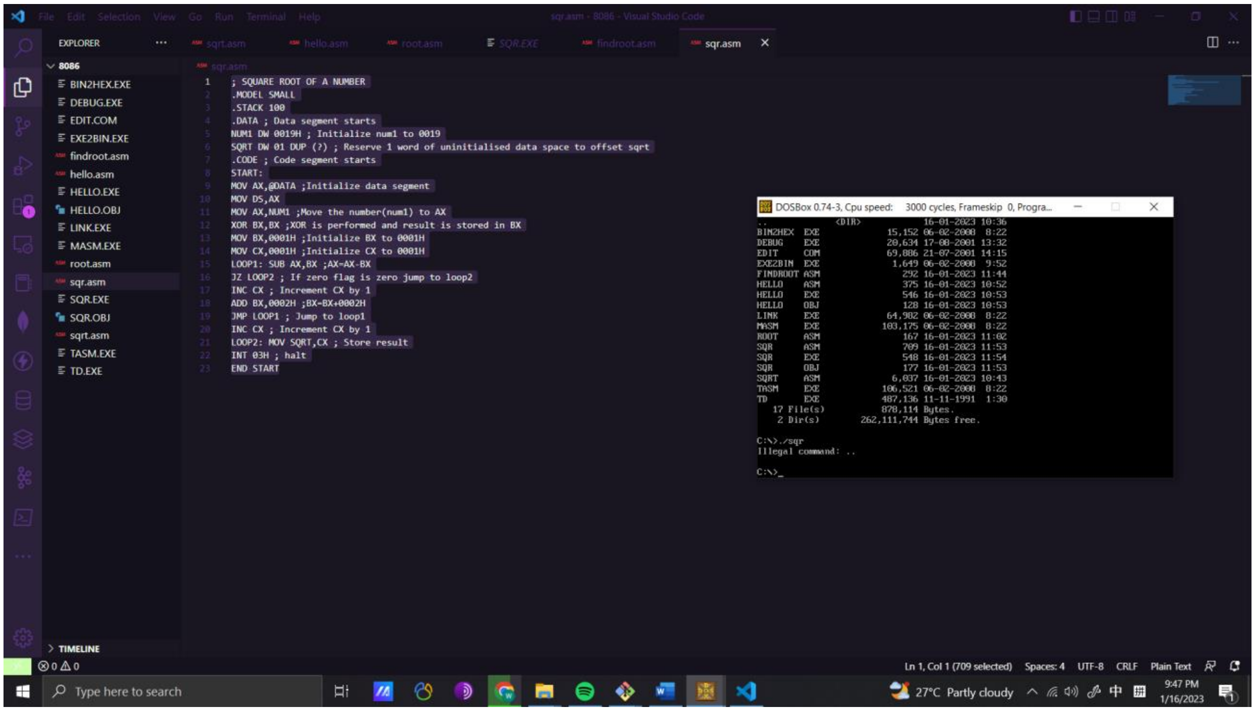
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: C...

```
ENTER A NUMBER TO SQUARE IT: 4
SQUARE OF 4 IS 16
CONTINUE? Y FOR YES ELSE FOR NO:

C:\>code.exe

ENTER A NUMBER TO SQUARE IT: 1
SQUARE OF 1 IS 01
CONTINUE? Y FOR YES ELSE FOR NO: 6
C:\>code.exe

ENTER A NUMBER TO SQUARE IT: 4
SQUARE OF 4 IS 16
CONTINUE? Y FOR YES ELSE FOR NO: n
C:\>o
Illegal command: o.

C:\>code.exe

ENTER A NUMBER TO SQUARE IT: 4
SQUARE OF 4 IS 16
CONTINUE? Y FOR YES ELSE FOR NO: Y
ENTER A NUMBER TO SQUARE IT: 1
SQUARE OF 1 IS 01
CONTINUE? Y FOR YES ELSE FOR NO: _
```

Another method is Fail to run

```
; SQUARE ROOT OF A NUMBER
.MODEL SMALL
.STACK 100
.DATA ; Data segment starts
NUM1 DW 0019H ; Initialize num1 to 0019
SQRT DW 01 DUP (?) ; Reserve 1 word of uninitialised data space to offset sqrt
.CODE ; Code segment starts
START:
MOV AX,@DATA ;Initialize data segment
MOV DS,AX
MOV AX,NUM1 ;Move the number(num1) to AX
XOR BX,BX ;XOR is performed and result is stored in BX
MOV BX,0001H ;Initialize BX to 0001H
MOV CX,0001H ;Initialize CX to 0001H
LOOP1: SUB AX,BX ;AX=AX-BX
JZ LOOP2 ; If zero flag is zero jump to loop2
INC CX ; Increment CX by 1
ADD BX,0002H ;BX=BX+0002H
JMP LOOP1 ; Jump to loop1
INC CX ; Increment CX by 1
LOOP2: MOV SQRT,CX ; Store result
INT 03H ; halt
END START
```

The output exe cannot be run and it will crush the emulation

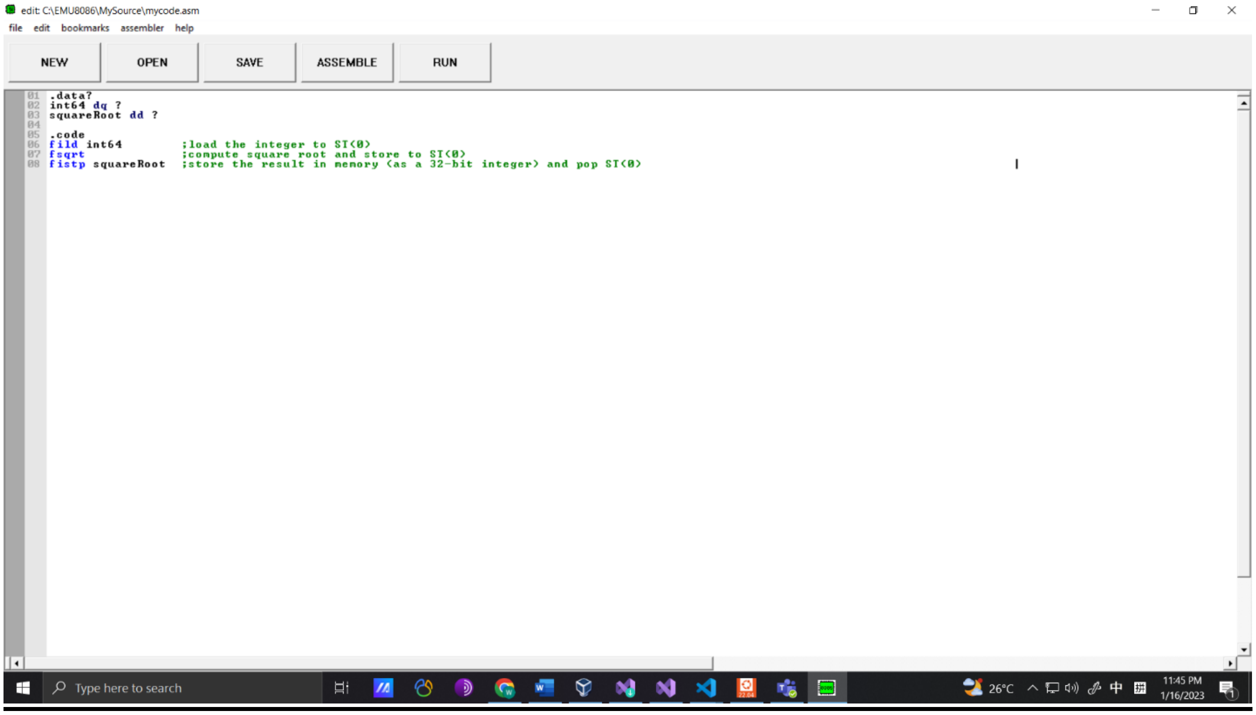


```
; SQUARE ROOT OF A NUMBER
.MODEL SMALL
.STACK 100
.DATA ; Data segment starts
NUM1 DW 0019H ; Initialize num1 to 0019
SQRT DW 01 DUP (?) ; Reserve 1 word of uninitialised data space to offset sqrt
.CODE ; Code segment starts
START:
MOV AX,@DATA ;Initialize data segment
MOV DS,AX
MOV AX,NUM1 ;Move the number(num1) to AX
XOR BX,BX ;XOR is performed and result is stored in BX
MOV BX,0001H ;Initialize BX to 0001H
MOV CX,0001H ;Initialize CX to 0001H
LOOP1: SUB AX,BX ;AX=AX-BX
JZ LOOP2 ; If zero flag is zero jump to loop2
INC CX ; Increment CX by 1
ADD BX,0002H ;BX=BX+0002H
JMP LOOP1 ; Jump to loop1
INC CX ; Increment CX by 1
LOOP2: MOV SQRT,CX ; Store result
INT 03H ; halt
END START
```

```
..               <DIR>            16-01-2023 10:36
BIN2HEX  EXE            15,152 06-02-2008  8:22
DEBUG    EXE            20,634 17-08-2001 13:32
EDIT     COM            69,886 21-07-2001 14:15
EXE2BIN  EXE             1,649 06-02-2008  9:52
FINDROOT ASM               292 16-01-2023 11:44
HELLO    ASM               375 16-01-2023 10:52
HELLO    EXE               546 16-01-2023 10:53
HELLO    OBJ               128 16-01-2023 10:53
LINK     EXE            64,982 06-02-2008  8:22
MASM     EXE           103,175 06-02-2008  8:22
ROOT     ASM               167 16-01-2023 11:02
SQR      ASM               709 16-01-2023 11:53
SQR      EXE               548 16-01-2023 11:54
SQR      OBJ               177 16-01-2023 11:53
SQRT     ASM             6,037 16-01-2023 10:43
TASM     EXE           106,521 06-02-2008  8:22
TD       EXE           487,136 11-11-1991  1:30
   17 File(s)          878,114 Bytes.
    2 Dir(s)       262,111,744 Bytes free.

C:\>./sqr
Illegal command: ..

C:\>_
```

SOLUTION 5: EMU8086

Download: https://softfamous.com/postdownload-file/emu8086/7377/3239/



```
.data?
int64 dq ?
squareRoot dd ?

.code
fild int64          ;load the integer to ST(0)
fsqrt               ;compute square root and store to ST(0)
fistp squareRoot    ;store the result in memory (as a 32-bit integer) and pop ST(0)
```

SOLUTION 6: MIPS software to write assembly code (100% Working)

GitHub Repository:

https://github.com/Weava/square_root_asy

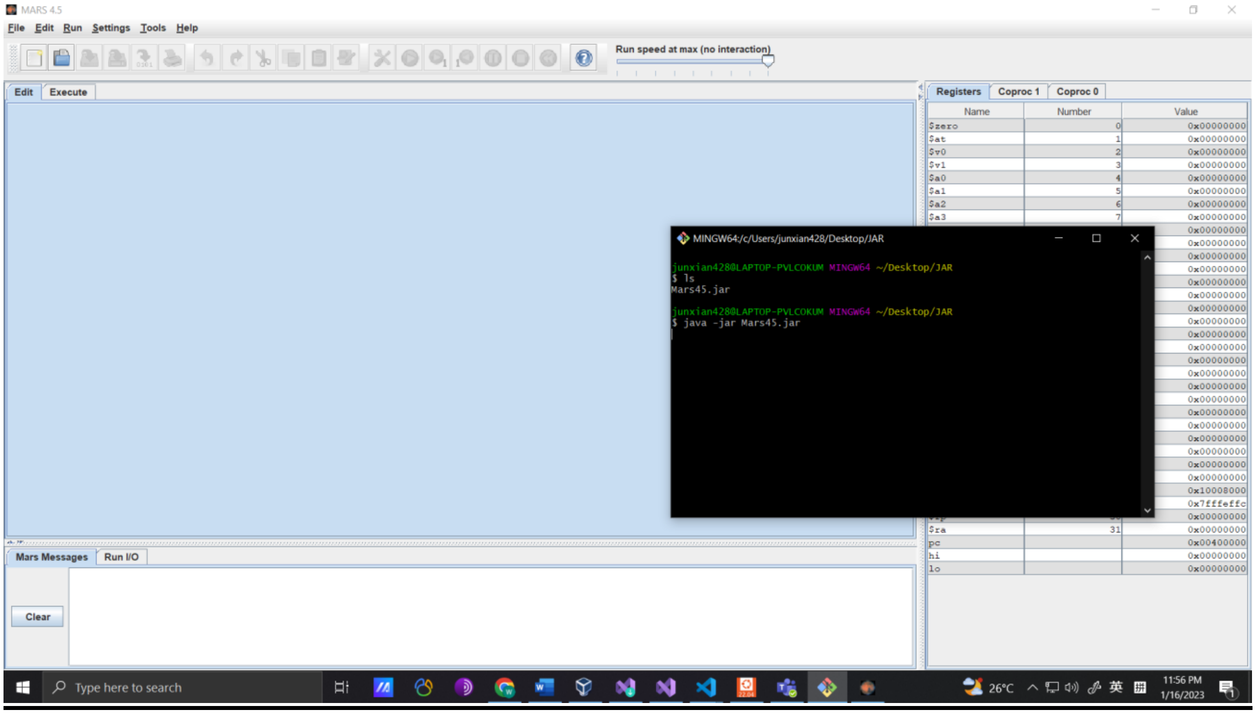
Article:

https://sweetcode.io/building-first-simple-program-mips-assembly-language/#:~:text=MIPS%20assembly%20language%20simply%20refers,an%20organization%20called%20MIPS%20Technologies.

Source Code:

```
#SquareRoot.s

#DATA

.data

square: .asciiz "Enter the number you wish to find the square root for: "
answer: .asciiz "The answer is: "
newline: .asciiz "\n"

#Text

.text
.globl main

main:
        li $v0, 4           #Prompt user for input
        la $a0, square
        syscall

        li $v0, 5                   #Receive said input
        syscall
        move $a0, $v0

        move $t4, $zero             #Move variables to t registers
        move $t1, $a0

        addi $t0, $zero, 1              #Set $t0 to 1
        sll $t0, $t0, 30            #Bit Shift $t0 left by 30

        #For loop
        loop1:
                slt $t2, $t1, $t0
                beq $t2, $zero, loop2
                nop

                srl $t0, $t0, 2                 #Shift $t0 right by 2
                j loop1

        loop2:
                beq $t0, $zero, return
                nop

                add $t3, $t4, $t0       #if $t0 != zero add t0 and t4 into t3
                slt $t2, $t1, $t3
                beq $t2, $zero, else1
                nop

                srl $t4, $t4, 1                 #shift $t4 right by 1
                j loopEnd

        else1:
                sub $t1, $t1, $t3       #Decrement $t1 by $t3
                srl $t4, $t4, 1                 #Shift $t4 right by 1
                add $t4, $t4, $t0       #then add $t0 to that

        loopEnd:
                srl $t0, $t0, 2                 #shift $t0 to the right
                j loop2

        return:
                li $v0, 4                       #print out the answer then exit
                la $a0, answer
                syscall
```

```
li $v0, 1
move $a0, $t4
syscall

li $v0, 10
syscall
```



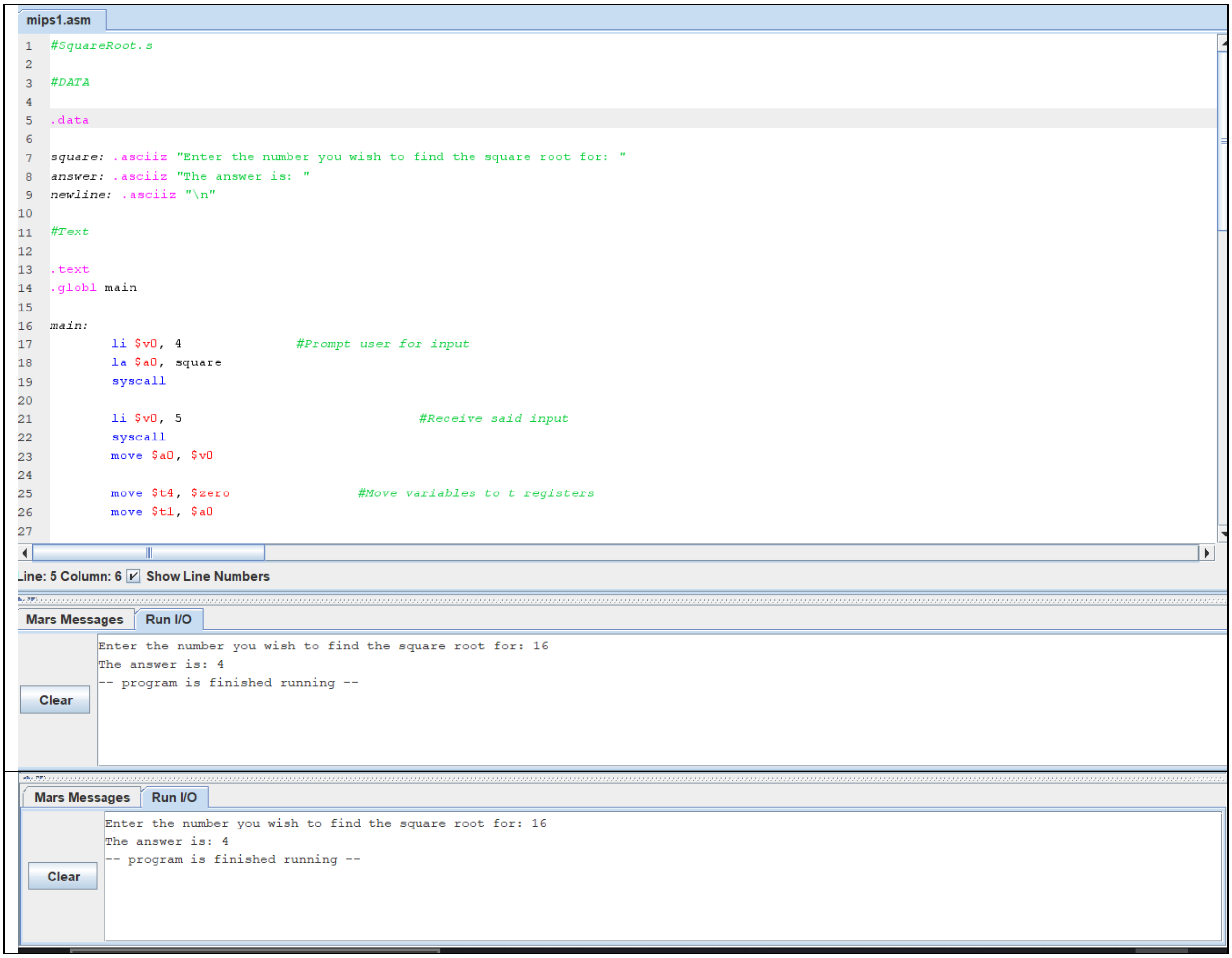
MARS 4.5
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Mars Messages Run I/O
Clear
Registers Coproc 1 Coproc 0
Type here to search
26°C
11:56 PM
1/16/2023

MINGW64:/c/Users/junxian428/Desktop/JAR

```
junxian428@LAPTOP-PVLCOKUM MINGW64 ~/Desktop/JAR
$ ls
Mars45.jar

junxian428@LAPTOP-PVLCOKUM MINGW64 ~/Desktop/JAR
$ java -jar Mars45.jar
```

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x00000000
		0x10008000
		0x7fffeffc
		0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Output:



```
#SquareRoot.s

#DATA

.data

square: .asciiz "Enter the number you wish to find the square root for: "
answer: .asciiz "The answer is: "
newline: .asciiz "\n"

#Text

.text
.globl main

main:
        li $v0, 4               #Prompt user for input
        la $a0, square
        syscall

        li $v0, 5                               #Receive said input
        syscall
        move $a0, $v0

        move $t4, $zero                 #Move variables to t registers
        move $t1, $a0

```

```
Enter the number you wish to find the square root for: 16
The answer is: 4
-- program is finished running --
```

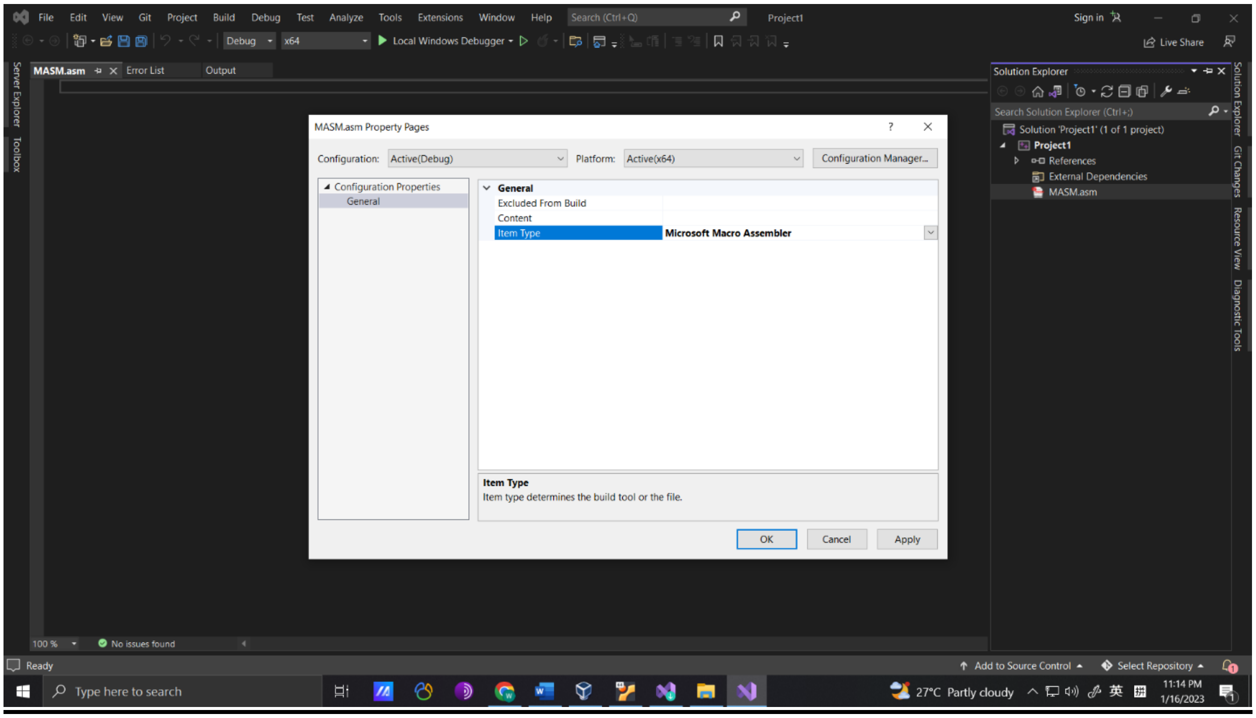
```
Enter the number you wish to find the square root for: 16
The answer is: 4
-- program is finished running --
```

Enter the number you wish to find the square root for: 16

The answer is: 4

-- program is finished running –

SOLUTION 7: MASM Microsoft Visual Studio (Work but may require debug)



MASM.asm Property Pages

Configuration: Active(Debug)   Platform: Active(x64)   Configuration Manager...

Configuration Properties
- General

General
- Excluded From Build
- Content
- Item Type: Microsoft Macro Assembler

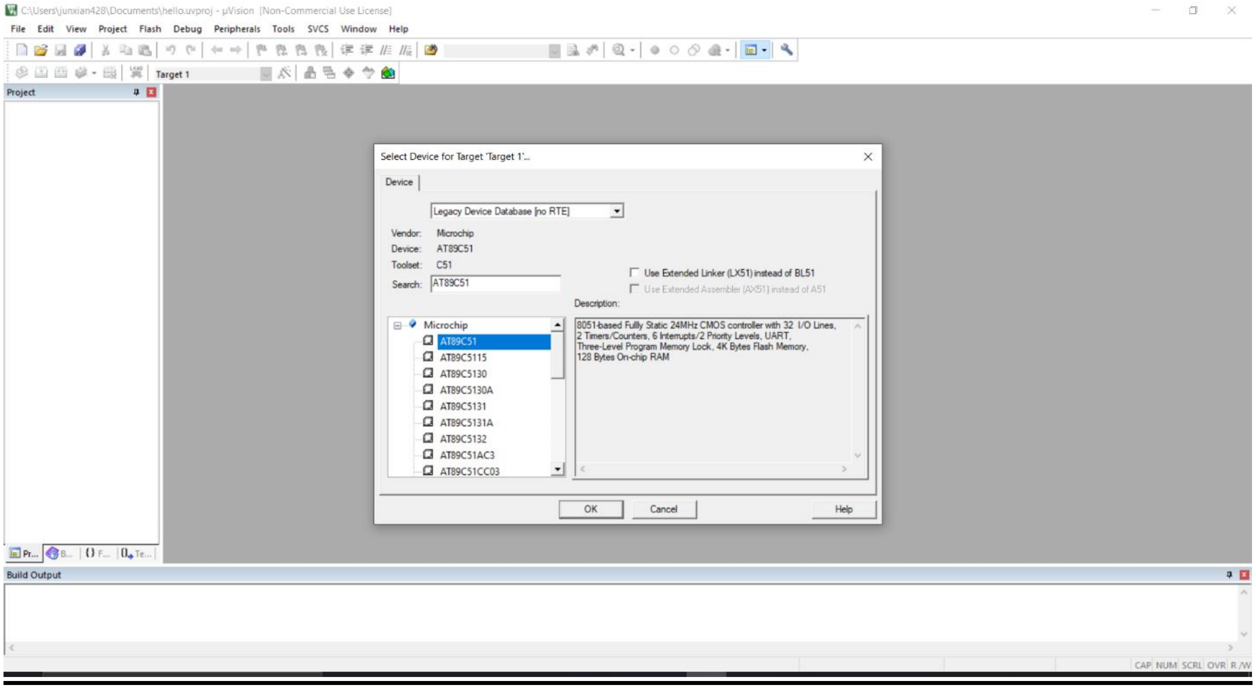
Item Type
Item type determines the build tool or the file.

OK   Cancel   Apply

SOLUTION 8: MASM editor (Work but may require debug)

https://www.masm32.com/

SOLUTION 9: ARM Keil (100% Working) with 8051 Assembly Program Code to find Square Root - AT89C51 - Keil


C:\Users\junxian428\Documents\hello.uvproj - µVision [Non-Commercial Use License]
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
Project
Build Output
CAP NUM SCRL OVR R/W

Select Device for Target 'Target 1'...

Device

Legacy Device Database [no RTE]

Vendor: Microchip
Device: AT89C51
Toolset: C51
Search: AT89C51

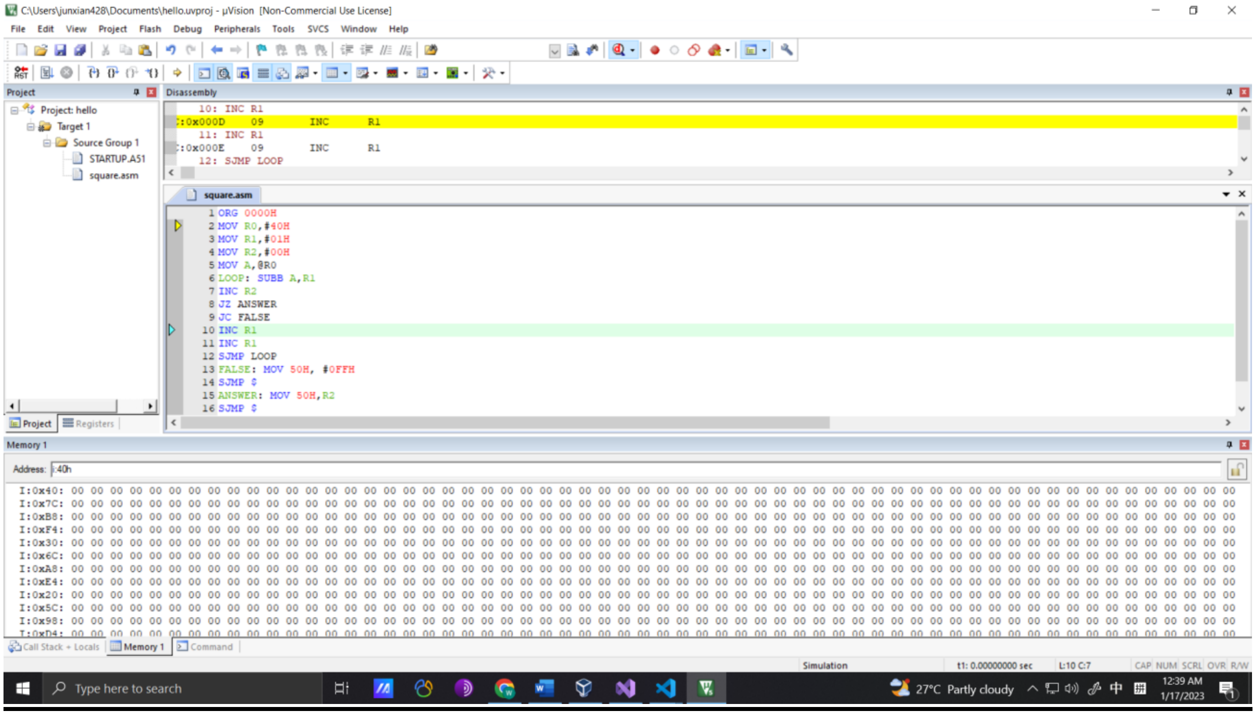
□ Use Extended Linker (LX51) instead of BL51
□ Use Extended Assembler (AX51) instead of A51

Microchip
- AT89C51
- AT89C5115
- AT89C5130
- AT89C5130A
- AT89C5131
- AT89C5131A
- AT89C5132
- AT89C51AC3
- AT89C51CC03

Description:
8051-based Fully Static 24MHz CMOS controller with 32 I/O Lines,
2 Timers/Counters, 6 Interrupts/2 Priority Levels, UART,
Three-Level Program Memory Lock, 4K Bytes Flash Memory,
128 Bytes On-chip RAM

Reference

https://archive.thebearsenal.com/2016/01/8051-assembly-program-code-to-find_11.html



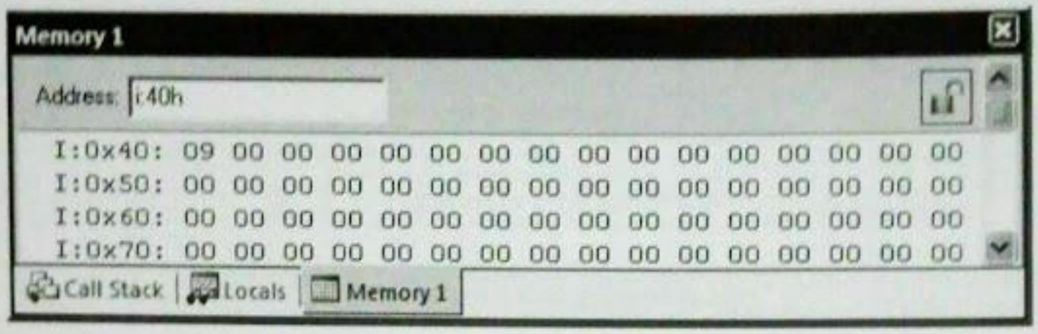
```
    10: INC R1
C:0x000D    09       INC      R1
    11: INC R1
C:0x000E    09       INC      R1
    12: SJMP LOOP
```

```
ORG 0000H
MOV R0,#40H
MOV R1,#01H
MOV R2,#00H
MOV A,@R0
LOOP: SUBB A,R1
INC R2
JZ ANSWER
JC FALSE
INC R1
INC R1
SJMP LOOP
FALSE: MOV 50H, #0FFH
SJMP $
ANSWER: MOV 50H,R2
SJMP $
```

Address: i:40h

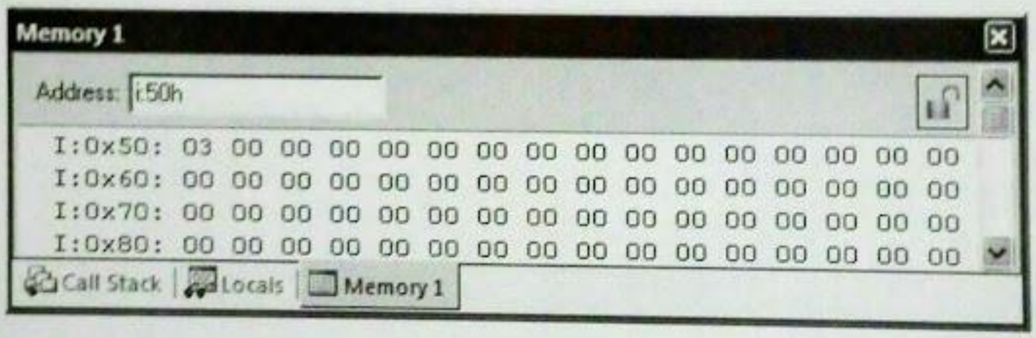
```
I:0x40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0x7C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0xB8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0xF4: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0x30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0x6C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0xA8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0xE4: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0x20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0x5C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0x98: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0xD4: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Input window:



```
I:0x40: 09 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0x50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0x60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0x70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Output window:



```
I:0x50: 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0x60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0x70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
I:0x80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

SOLUTION 10: IDA (100% but reverse engineering)

First and foremost,

Write Main.c code then decompile

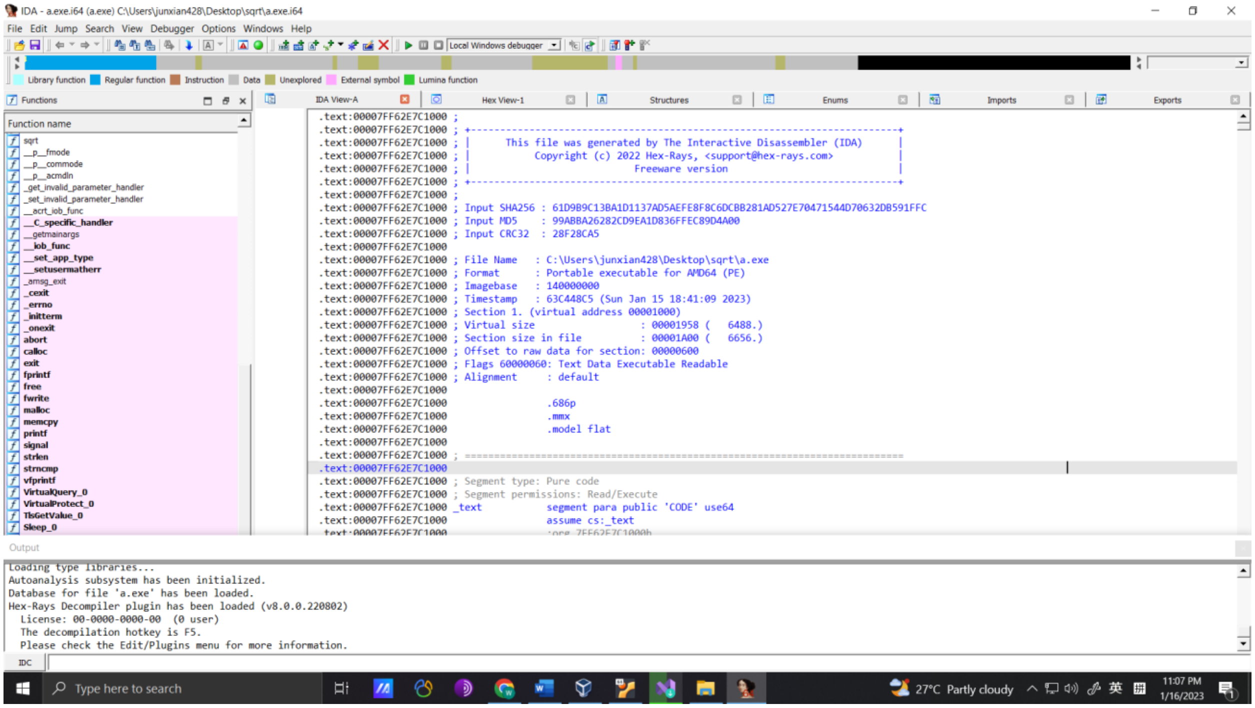
```
#include <string.h>
#include <math.h>

int main(){
    printf("%d",root(49));
    return 0;
}

int root(int num) {
```

```
int x = sqrt(num);

return x;
}
```

Decompile

Too long, not suitable to be displayed

Text view

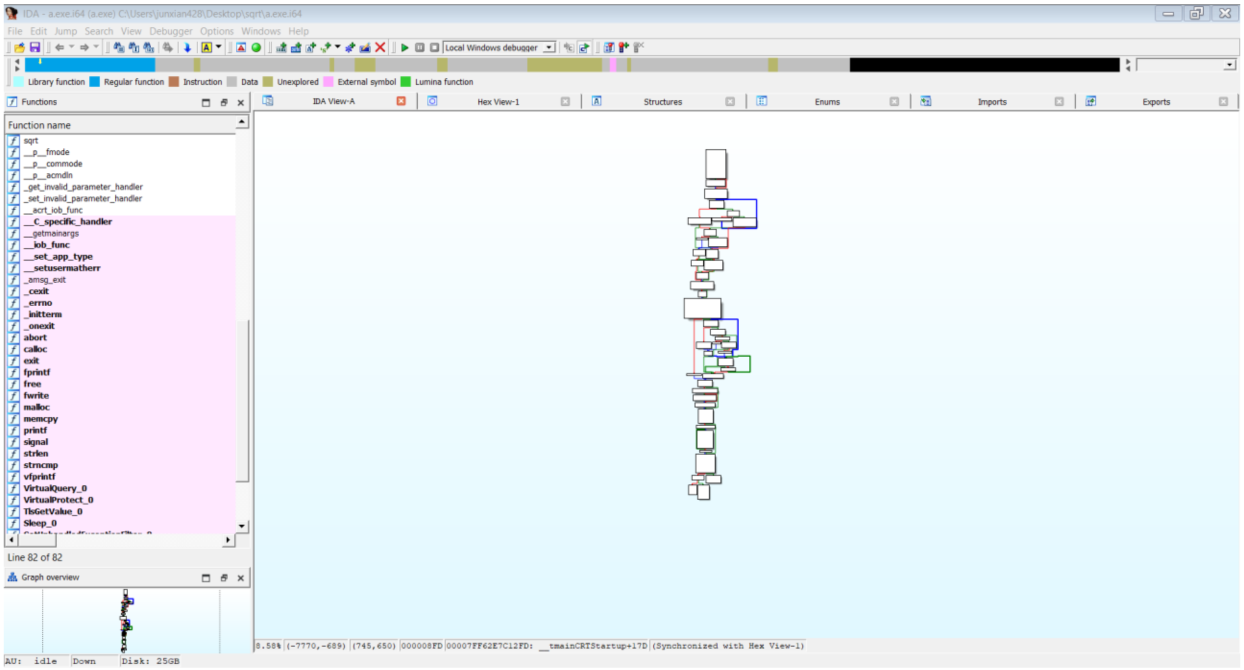


```
Function name
sqrt
__p__fmode
__p__commode
__p__acmdln
_get_invalid_parameter_handler
_set_invalid_parameter_handler
__acrt_iob_func
__C_specific_handler
__getmainargs
__iob_func
__set_app_type
__setusermatherr
_amsg_exit
_cexit
_errno
_initterm
_onexit
abort
calloc
exit
fprintf
free
fwrite
malloc
memcpy
printf
signal
strlen
strncmp
vfprintf
VirtualQuery_0
VirtualProtect_0
TlsGetValue_0
Sleep_0
```

```
.text:00007FF62E7C1000 ;
.text:00007FF62E7C1000 ; +-------------------------------------------------------------------------+
.text:00007FF62E7C1000 ; |      This file was generated by The Interactive Disassembler (IDA)      |
.text:00007FF62E7C1000 ; |           Copyright (c) 2022 Hex-Rays, <support@hex-rays.com>           |
.text:00007FF62E7C1000 ; |                            Freeware version                             |
.text:00007FF62E7C1000 ; +-------------------------------------------------------------------------+
.text:00007FF62E7C1000 ;
.text:00007FF62E7C1000 ; Input SHA256 : 61D9B9C13BA1D1137AD5AEFE8F8C6DCBB281AD527E70471544D70632DB591FFC
.text:00007FF62E7C1000 ; Input MD5    : 99ABBA26282CD9EA1D836FFEC89D4A00
.text:00007FF62E7C1000 ; Input CRC32  : 28F28CA5
.text:00007FF62E7C1000
.text:00007FF62E7C1000 ; File Name   : C:\Users\junxian428\Desktop\sqrt\a.exe
.text:00007FF62E7C1000 ; Format      : Portable executable for AMD64 (PE)
.text:00007FF62E7C1000 ; Imagebase   : 140000000
.text:00007FF62E7C1000 ; Timestamp   : 63C448C5 (Sun Jan 15 18:41:09 2023)
.text:00007FF62E7C1000 ; Section 1. (virtual address 00001000)
.text:00007FF62E7C1000 ; Virtual size                  : 00001958 (   6488.)
.text:00007FF62E7C1000 ; Section size in file          : 00001A00 (   6656.)
.text:00007FF62E7C1000 ; Offset to raw data for section: 00000600
.text:00007FF62E7C1000 ; Flags 60000060: Text Data Executable Readable
.text:00007FF62E7C1000 ; Alignment     : default
.text:00007FF62E7C1000
.text:00007FF62E7C1000                 .686p
.text:00007FF62E7C1000                 .mmx
.text:00007FF62E7C1000                 .model flat
.text:00007FF62E7C1000
.text:00007FF62E7C1000 ; ===========================================================================
.text:00007FF62E7C1000
.text:00007FF62E7C1000 ; Segment type: Pure code
.text:00007FF62E7C1000 ; Segment permissions: Read/Execute
.text:00007FF62E7C1000 _text           segment para public 'CODE' use64
.text:00007FF62E7C1000                 assume cs:_text
.text:00007FF62E7C1000                 ;org 7FF62E7C1000h
```

```
Loading type libraries...
Autoanalysis subsystem has been initialized.
Database for file 'a.exe' has been loaded.
Hex-Rays Decompiler plugin has been loaded (v8.0.0.220802)
  License: 00-0000-0000-00  (0 user)
  The decompilation hotkey is F5.
  Please check the Edit/Plugins menu for more information.
```

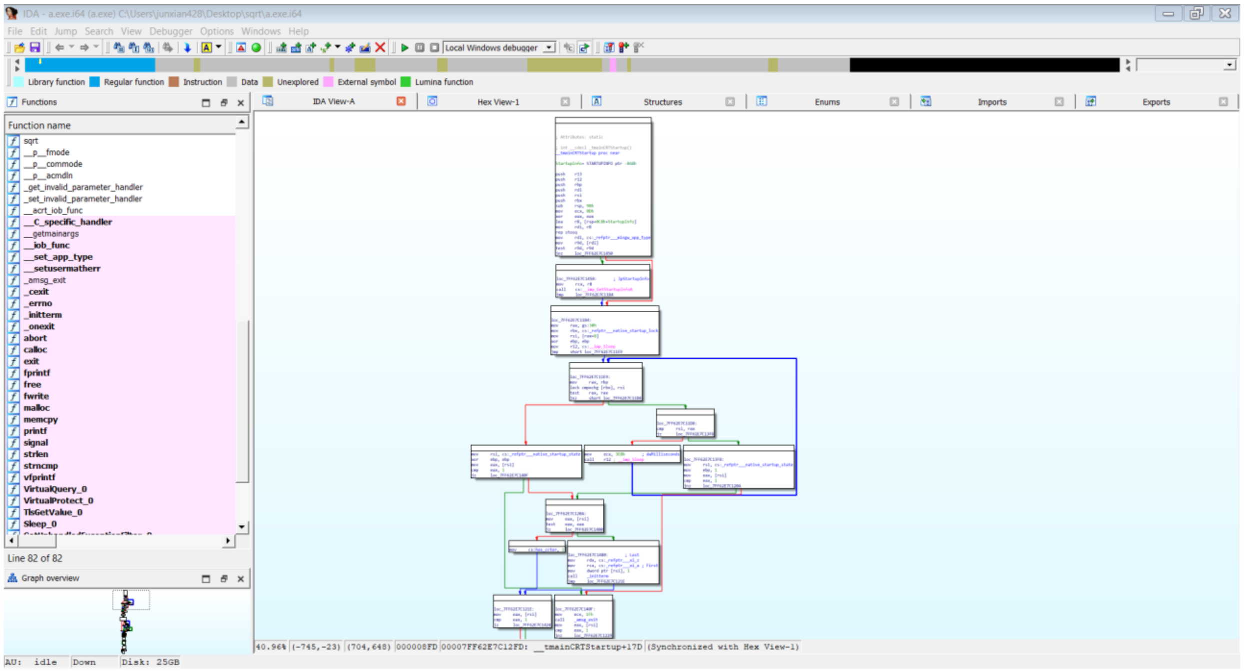
Graph View



```
sqrt
__p__fmode
__p__commode
__p__acmdln
_get_invalid_parameter_handler
_set_invalid_parameter_handler
__acrt_iob_func
__C_specific_handler
__getmainargs
__iob_func
__set_app_type
__setusermatherr
_amsg_exit
_cexit
_errno
_initterm
_onexit
abort
calloc
exit
fprintf
free
fwrite
malloc
memcpy
printf
signal
strlen
strncmp
vfprintf
VirtualQuery_0
VirtualProtect_0
TlsGetValue_0
Sleep_0
```

Line 82 of 82

```
8.58% (-7770,-689) (745,650) 000008FD 00007FF62E7C12FD: __tmainCRTStartup+17D (Synchronized with Hex View-1)
AU:  idle  Down  Disk: 25GB
```



```
sqrt
__p__fmode
__p__commode
__p__acmdln
_get_invalid_parameter_handler
_set_invalid_parameter_handler
__acrt_iob_func
__C_specific_handler
__getmainargs
__iob_func
__set_app_type
__setusermatherr
_amsg_exit
_cexit
_errno
_initterm
_onexit
abort
calloc
exit
fprintf
free
fwrite
malloc
memcpy
printf
signal
strlen
strncmp
vfprintf
VirtualQuery_0
VirtualProtect_0
TlsGetValue_0
Sleep_0
```

Line 82 of 82

```
40.96% (-745,-23) (704,648) 000008FD 00007FF62E7C12FD: __tmainCRTStartup+17D (Synchronized with Hex View-1)
AU:  idle  Down  Disk: 25GB
```

Conclusion:

There are many applications require assembly code even though we are not usually and often to use but it is quite considered everywhere especially cybersecurity and embedded system industry. Square root assembly code is done through several methods and some of them are considered running quite well and some do not. As a conclusion, we already achieve the objective and requirement asked by the project.