X86 ASSEMBLY

NASM

Sudo apt install nasm

Nasm is an *assembler*, which is similar to a compiler in that it translates assembly in to machine code.

An assembler is a program that converts assembly (low level code), into machine code. A compiler is a program that converts high level code (C, Java, C++, …) in to low level code (assembly, machine code…)

Global \_start ; defines entry point

Section .text ; section

\_start: ; label called \_start (also the entry point)

Mov eax, 1 ; move integer 1 in to eax register

Mov ebx, 2 ; move integer 2 in to ebx register

Int 0x80 ; perform interrupt

Int 0x80 – 80 is an interrupt handler for a system call. The system call which it makes is determined by the eax register value. In this example, the value 1 was moved in to the eax register, 1 is the value for the exit call function. Ebx is used as the exit status, in this example 2 is passed to ebx. Usually, you would pass 0 to ebx to show that that program ran with no errors.

To assemble in to machine code you first need to make the object file, i.e. the assemble (1). And then you need to link these object files to create an executable (2).

1. Nasm -f elf32 main.asm -o main.o
2. Ld -m elf -i386 main.o -o main

Run the program using ./main. After running the program you can view the exit status by doing: “echo $?” in the terminal.

OPERANDS

|  |  |  |
| --- | --- | --- |
| **Operand** | **Desc** | **Example** |
| Mov | Move | Mov eax, ebx |
| Int | Interrupt | Int 0x80 |
| Sub | Subtract | Sub ebx, 2 |
| Add | Add | Add ebx, ebx |
| Mul | Multiply – always applied to eax | Mul ebx (eax \*= ebx) |
| Div | Divide – always applied to eax | Div ebx (eax /= ebx) |

HELLO WORLD

Global \_start

Section .data ; data section to create constants

Msg db “Hello World!”, 0x0a ;create variable msg(db = define byte),0x0a = \n

Len equ $ - msg ; create variable len equal to length of message

Section .text ; section for actual code

Mov eax, 4 ; 4 is the system write system call

Mov ebx, 1 ; 1 is stdout file descriptor for system write call

Mov ecx, msg ; bytes to write

Mov edx, len ; number of bytes to write

Int 0x80 ; perform system write call

Mov eax, 1 ; system exit call

Mov ebx, 0 ; exit call status

Int 0x80 ; perform exit

An assembly program can be divided in to 3 sections:

* .data – declaring constants
* .bss – declaring variables
* .text – the actual code

Although these can be called anything, these section names follow the current best practices.

INSTRUCTION POINTER

The instruction pointer is the internal pointer in the processor labelled EIP which holds the location of the machine code that the processor is currently executing. This allows the processor to jump around the code by altering this pointer.

The EIP instruction pointer is changed by using the *jmp* operator.

Global \_start

Section .text

\_start:

Mov ebx, 42

Mov eax, 1

Jmp skip

Mov ebx, 13

Skip:

Int 0x80

; after compilation the exit status will be 42, as the code after where jmp was specified was skipped, and the instruction pointer never returned back to this position.

CONDITIONALS

|  |  |
| --- | --- |
| **Operand** | **Description** |
| Je A, B | Jump if equal |
| Jne A, B | Jump if not equal |
| Jg A, B | Jump if greater |
| Jge A, B | Jump if greater or equal |
| Jl A, B | Jump if lesser |
| Jls A, B | Jump if less or equal |

\_start:

Mov ebx, 42

Mov eax, 1

Mov ecx, 99

Cmp ecx, 100 ; compare ecx to the integer 100

Jl skip ; jump if ecx is less than 100 (which it is)

Mov ebx, 13

Skip:

Int 0x80

LOOPING

Global \_start

Section .text

\_start:

Main ebx, 1 ; starting value

Mov ecx, 4 ; number of iterations

Label:

Add ebx, ebx

Dec ecx ; decrease the number of iterations by 1 each loop

Cmp ecx, 0 ; compare the iterator (ecx in this eg), to 1

Jg label ; jump back to label start if ecx greater than 0

Mov eax, 1 ; prepare system exit call

Int 0x80 ; exit

MEMORY ACCESS

TEST