Introduction to Assembly Programming - A Forgotten Art

Mohan Raman

July 20, 2025

Contents

1 Intro To Assembly

1.1 Basics of a Process

1.1.1 What is mean by an Executable

Set of instructions to CPU organized in a specific format.

1.1.2 Executable formats

PE Portable Executable

Mach-O Used in Apple Machines

ELF Executable Linkable Format

1.1.3 CPU Architectures

x86 and x86-64 Originate from Intel, also provided by AMD

ARM and ARM64 Owned by ARM Holdings, licenced to Apple, Qualcomm, Meditek, Samsung etc.

1.1.4 How to create an Executable using C programming

```
// helloworld.c
#include<stdio.h>
int main(int argc,
         char **argv,
         char **envp) {
        printf("hello world\n");
        return 0;
}

gcc helloworld.c
```

1.2 What happens when we invoke C compiler

1.2.1 Pre Processing

```
gcc -E -o helloworld.i helloworld.c
```

1.2.2 Compiling

```
gcc -S -o helloworld.s helloworld.i
```

1.2.3 Assembling

```
as -o helloworld.o helloworld.s
```

1.2.4 Linking

```
ld -I /lib64/ld-linux-x86-64.so.2 -o helloworld /usr/
    lib/x86-64-linux-gnu/crt1.o helloworld.o -lc
```

1.2.5 Dynamic Linker

- Responsible to load shared objects into memory and start the executable, usually available as /lib64/ld-linux-x86-64.so.2 in 64bit linuxmint.
- Parses ELF executable to load shared objects
- sets function addresses GOT (for PIE executables)
- calls main

1.3 Basics of Digital Logic

1.3.1 Combinational Circuits

Logic Gates AND, OR, NOT, NAND, NOR, XOR, XNOR

Truth Table

Boolean Algebra $A + B . C = A . C + B . C$

Adder, Multiplexer, Demultiplexer, Decoder, Encoder, Comparitor

1.3.2 Sequential Circuits

- SR Latch
- Flip-Flop
- Registers
- Clock Cycle

1.3.3 ASICs (Application Specific Integrated Circuits)

- RTL (Register Transfer Logic) and HDL (Hardware Description Language, SystemVerilog (iverilog))
- Gate-Level Netlist (Yosys)
- OpenLane (RTL to GDSII)
- Foundries

  Pure Foundries (TSMC, UMC) Components, American Components, Russian Components ALL MADE IN TAIWAN !!!!! – One Crazy Russian Cosmonaut

  IDMs (Intel, Samsung) Integrated Device Manufacturers

- ASML (Manufacturer of EUV (Extreame UltraViolet) Lithography)

1.4 Assembly Programming (x86-64)

Giving Instructions to an x86-64 CPU

1.4.1 Registers

Writable

- AX (RAX, EAX, AX, AH, AL)
- BX (RBX, EBX, BX, BH, BL)
- CX (RCX, ECX, CX, CH, CL)
- DX (RDX, EDX, DX, DH, DL)
- SP (RSP, ESP, SP)

- BP (RBP, EBP, BP)
- SI (RSI, ESI, SI)
- DI (RDI, EDI, DI)
- R8 - R15 (Rn, RnD, RnW, RnB)

Non Writable

- IP (RIP, EIP, IP)
- FLAGS (RFLAGS, EFLAGS, FLAGS)

1.4.2 Endianness

Big Endian 0x12345678 = 0x12 0x34 0x56 0x78

Little Endian 0x12345678 = 0x78 0x56 0x34 0x12

1.4.3 Opcodes

- mov
- cmp
- cmov
- jmp
- push
- pop
- call
- ret

1.4.4 Opcode Suffixes

- 8 bit (b, byte)
- 16 bit (w, word)
- 32 bit (l, doubleword)
- 64 bit (q, quadword)

1.4.5 Operand Specifiers

imm movq $10, %rax

%reg movq %rbx, %rax

(%reg) movq (%rbx), %rax

offset(%reg, %indexreg, multiplier) movq 8(%rbx, %rcx, 4), %rax (value in memory pointed by calculating %rbx + 8 + (%rcx * 4))

1.5 First Assembly Program

1.5.1 Interrupts

1. Hardware Interrupts

   ```
   cat /proc/interrupts
   ```

2. Software Interrupts

   ```
   int 0x80
   ```

1.5.2 Simple Program with Software Interrupt

```
        .global _start
        .section .text
_start:
        movq $241, %rbx
        movq $1, %rax
        int $0x80
```

```
as -o assembly.o assembly.s; ld -static -o assembly
    assembly.o; ./assembly; echo $?
```

- `exit` syscall (0x01) [x86 architecture, not x86-64]

1.5.3 Stack and Functions

```
        .global _start
        .section .text
```

```
addfunc:
        pushq %rbp
        movq %rsp, %rbp
        subq $16, %rsp
        movq %rdi, 8(%rsp)
        movq %rsi, (%rsp)
        movq 8(%rsp), %rax
        addq (%rsp), %rax
        movq %rbp, %rsp
        popq %rbp
        ret

_start:
        movq $10, %rdi
        movq $20, %rsi
        call addfunc
        movq %rax, %rdi
        movq $60, %rax
        syscall

as -o assembly.o assembly.s; ld -static -o assembly
   assembly.o; ./assembly; echo $?
```

- `exit` syscall (0x3c) [x86-64 architecture]

1.5.4 Calling C library functions from Assembly

```
        .global _start

        .section .data
hellostring:    .asciz "helloworld"

        .section .text
_start:
        movq $hellostring, %rdi
        call puts
        movq %rax, %rdi
        movq $60, %rax
        syscall
```

```
as -o assembly.o assembly.s; ld -I /lib64/ld-linux-x86
    -64.so.2 -o assembly assembly.o -lc; ./assembly;
    echo $?
```

1.5.5 Heap

- `mmap` syscall (0x09)
- `munmap` syscall (0x0b)

1.5.6 GDB - GNU Debugger

- starti
- break
- nexti
- stepi
- disassemble
- info registers