Introduction to Reverse Engineering

00 - Introduction

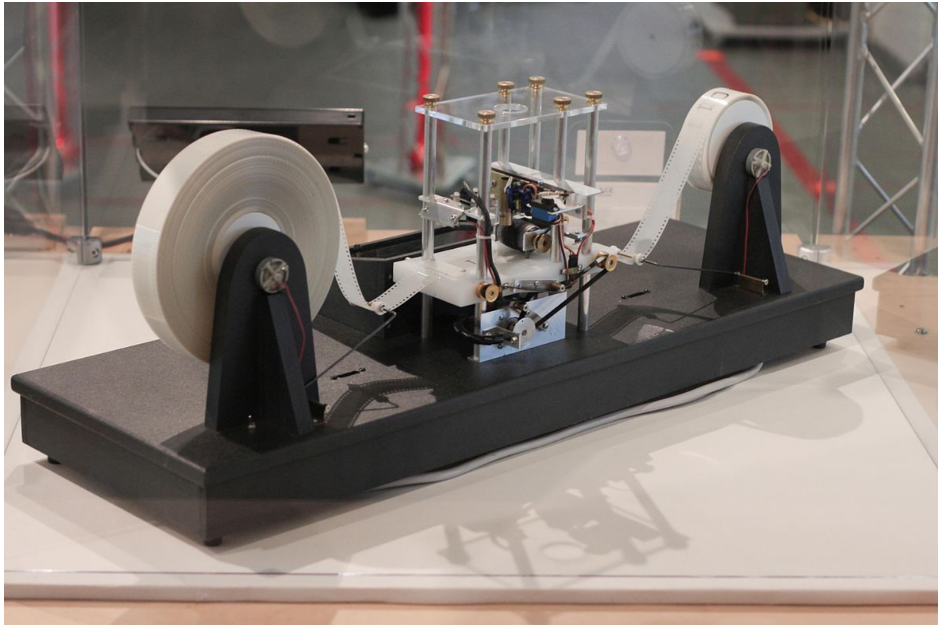
https://github.com/0x03c6/IRE/

What is a program?

- A computer program is a general term which refers a single unit of computation. This program contains a specific set of binary-encoded **instructions** for the **CPU** to execute, as well additional **data**.
- Oftentimes we can approach reverse engineering a program as a **black box**. A black box is a system which can only be viewed in terms of its **inputs** and **outputs**. We have no transparency as to what operations are performed within this box.
- A program which is currently being executed is called a **process**.
- Programs consist of **headers**, **sections** and **segments**.
- Code is data and data is code, it’s simply a matter of interpretation and context.

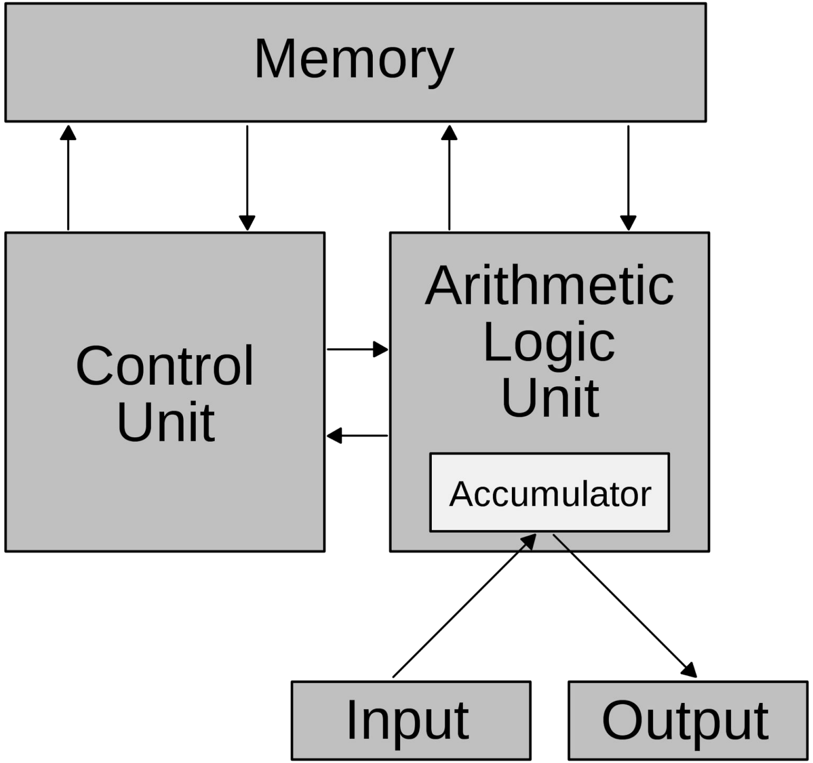
Turing Machine

- A formal model of computation which describes an **abstract machine** which is capable of executing any computable algorithm.
- It is manipulates **symbols** on an **infinite strip of tape** according to a predetermined set of **rules**.
- An **automata** which belongs to another field of research known as **computability theory**.



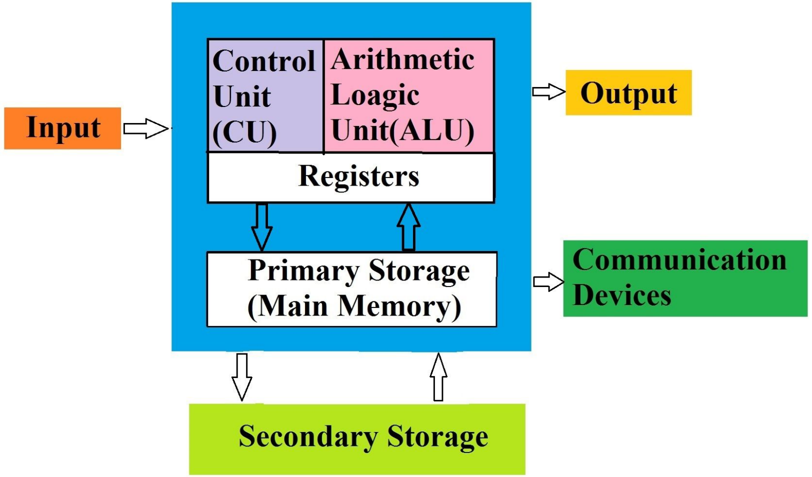
Von Neumann Architecture

- A more **practical** architecture for the modern computer.
- This is the architecture that all modern computers abide by.
- **Turing complete** (equivalent to that of a Turing machine), disregarding the lack of memory restrictions.
- Various other esoteric computer architectures such as **Harvard**, **Dataflow** & etc.



Control Unit / Arithmetic Logic Unit (CPU)

- For simplicity sake we can consider the **CU** (Control Unit) and **ALU** (Arithmetic Logic Unit) as a single component known as the **CPU** (Central Processing Unit).
- Executes **instructions**, performs **arithmetic** & **logic**, **read** and **write** from registers, memory, disk, network and more.
- Modern CPUs are more sophisticated, but not relevant.

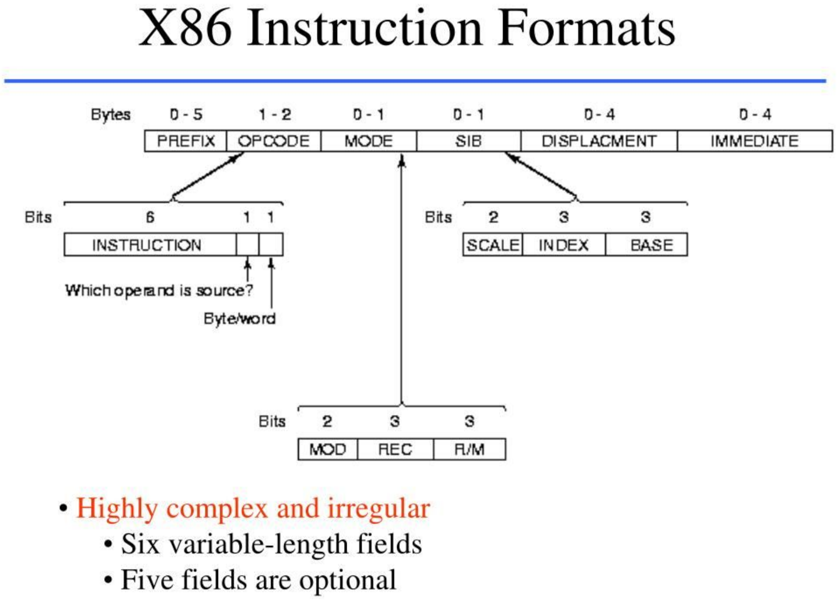


CPU (Central Processing Unit)

Image copyrights Informationq.com

Instructions

- Instructions are essentially primitive operations for the CPU to execute.
- CPU’s can have different **architectures** as well, some of these include the following families: **x86, ARM, MIPS** and much more.
- All programming languages will **compile** source into these low level instructions.
- **Interpreted** languages such as python implement abstract machines on top of hardware.

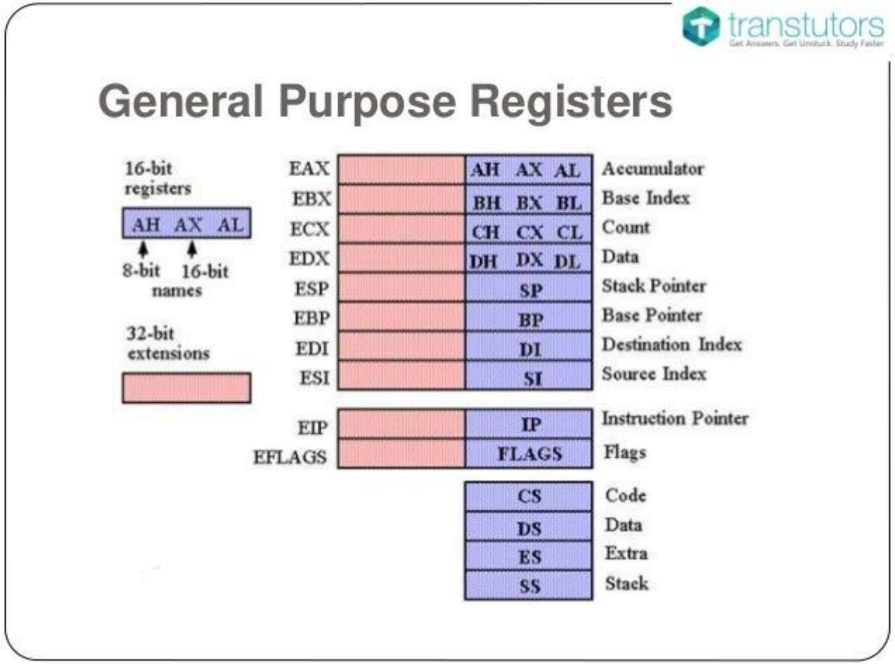


X86 Instruction Formats

- Highly complex and irregular
  - Six variable-length fields
  - Five fields are optional

Registers

- **High-speed** memory placed within the processor which allows for quick data access.
- Some registers serve a specific purpose, whether that be to store the location of some important structure within memory, or to point to the current instruction being executed by the CPU & etc.
- Some registers are **general purpose**, meaning that they don't serve a specific purpose but can act as temporary variables for userspace processes.

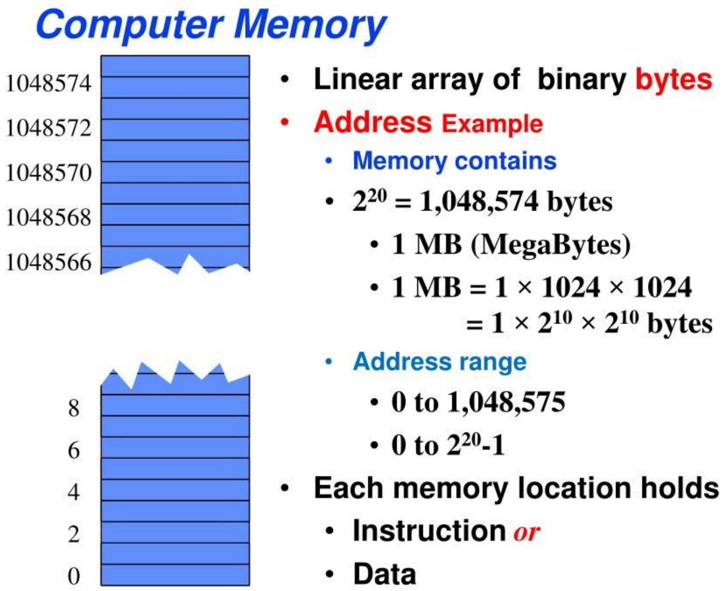


General Purpose Registers

EAX	AH AX AL	Accumulator
EBX	BH BX BL	Base Index
ECX	CH CX CL	Count
EDX	DH DX DL	Data
ESP	SP	Stack Pointer
EBP	BP	Base Pointer
EDI	DI	Destination Index
ESI	SI	Source Index
EIP	IP	Instruction Pointer
EFLAGS	FLAGS	Flags
	CS	Code
	DS	Data
	ES	Extra
	SS	Stack

Memory

- Memory, generally referred to as **RAM** (Random Access Memory) acts as our computer's **short-term** memory; meaning that it does not persist after a process executes.
- When we **execute** any program, the binary is loaded into memory by the **loader**.
- Our entire **process** is contained within its own memory. Operating systems have separate mechanisms for managing memory.

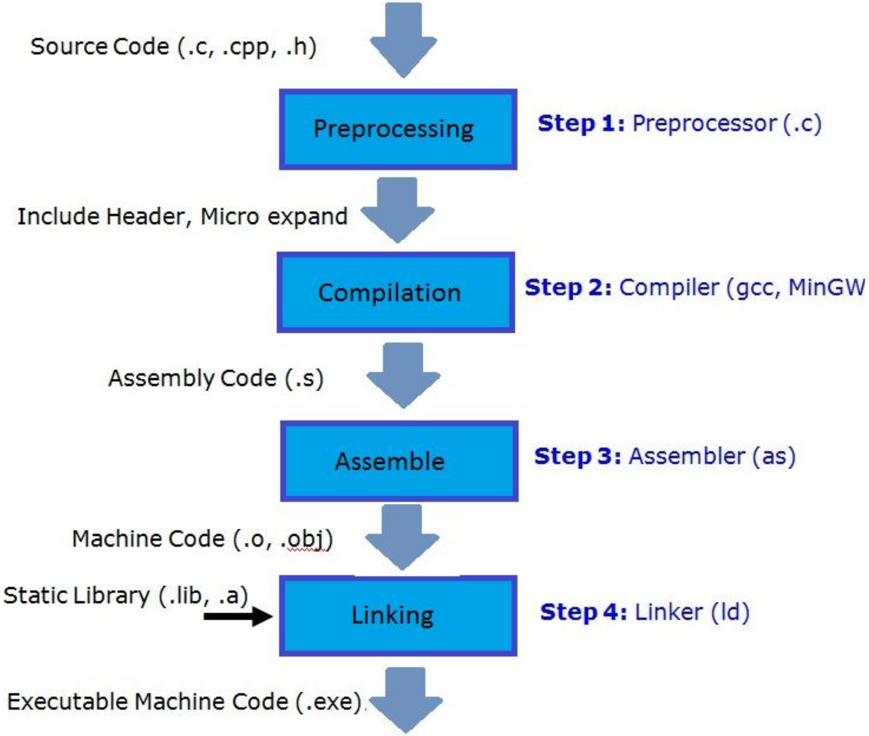


Computer Memory

- Linear array of binary bytes
- Address Example
  - Memory contains
  - 2^{20} = 1,048,574 bytes
    - 1 MB (MegaBytes)
    - 1 MB = 1 × 1024 × 1024
      = 1 × 2^{10} × 2^{10} bytes
  - Address range
    - 0 to 1,048,575
    - 0 to 2^{20}-1
- Each memory location holds
  - Instruction *or*
  - Data

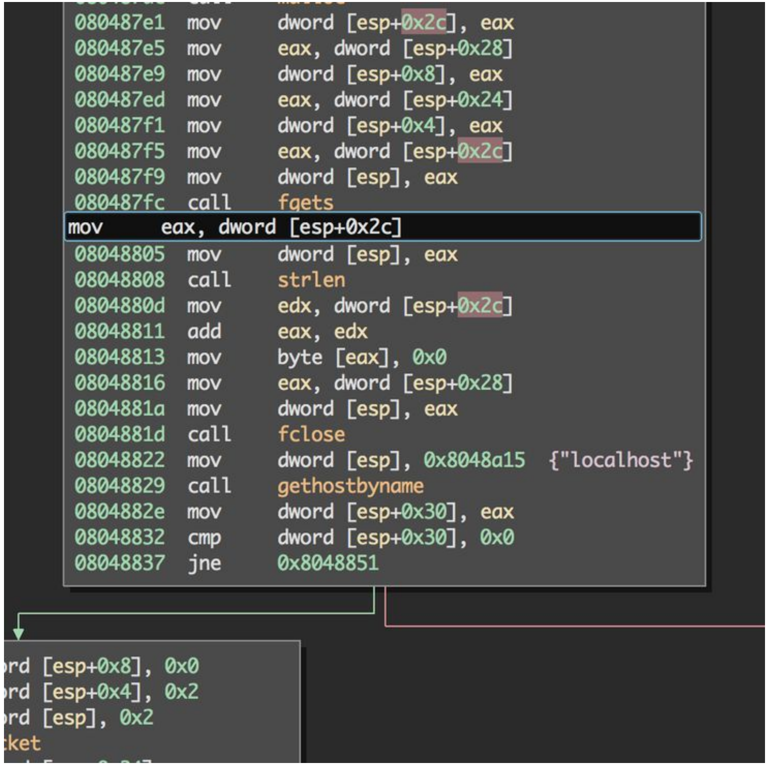
The Compilation Process

- A **compiler** is the program which converts your source code into instructions that can be executed by your computer.
- This is a high level representation of the compilation process. In truth, **compiler theory** is a very deep field of research.
- Compiler theory aids in the understanding of the programs that we are **reverse engineering**.


Source Code (.c, .cpp, .h)
Preprocessing
Step 1: Preprocessor (.c)
Include Header, Micro expand
Compilation
Step 2: Compiler (gcc, MinGW
Assembly Code (.s)
Assemble
Step 3: Assembler (as)
Machine Code (.o, .obj)
Static Library (.lib, .a)
Linking
Step 4: Linker (ld)
Executable Machine Code (.exe)

The Reverse Engineering Process

- **Reverse engineering** is not the direct opposite of program compilation, rather, it's a broader field which consists of **analyzing programs** which we do not have the source to.
- There are different means of analyzing binaries, categorized into either **static** or **dynamic analysis**.
- We will dive deeper into these specifically in the future.



```
080487e1  mov    dword [esp+0x2c], eax
080487e5  mov    eax, dword [esp+0x28]
080487e9  mov    dword [esp+0x8], eax
080487ed  mov    eax, dword [esp+0x24]
080487f1  mov    dword [esp+0x4], eax
080487f5  mov    eax, dword [esp+0x2c]
080487f9  mov    dword [esp], eax
080487fc  call   fgets
mov     eax, dword [esp+0x2c]
08048805  mov    dword [esp], eax
08048808  call   strlen
0804880d  mov    edx, dword [esp+0x2c]
08048811  add    eax, edx
08048813  mov    byte [eax], 0x0
08048816  mov    eax, dword [esp+0x28]
0804881a  mov    dword [esp], eax
0804881d  call   fclose
08048822  mov    dword [esp], 0x8048a15  {"localhost"}
08048829  call   gethostbyname
0804882e  mov    dword [esp+0x30], eax
08048832  cmp    dword [esp+0x30], 0x0
08048837  jne    0x8048851
```

```
rd [esp+0x8], 0x0
rd [esp+0x4], 0x2
rd [esp], 0x2
ket
```

Demo time!

Motivation & Real-world Applications

- Primer into any low-level field.
- Enables you to dissect binaries and understand their implementation without the source.
- **Game hacking**, **vulnerability research**, **malware analysis** and more.
- Builds on your understanding of computers at a fundamental level.
- Applicable to every field related to computers.