Reverse Engineering: Towards Malware Analysis

Lecture – Crash Course in x86 Disassembly

Computer Security Practice

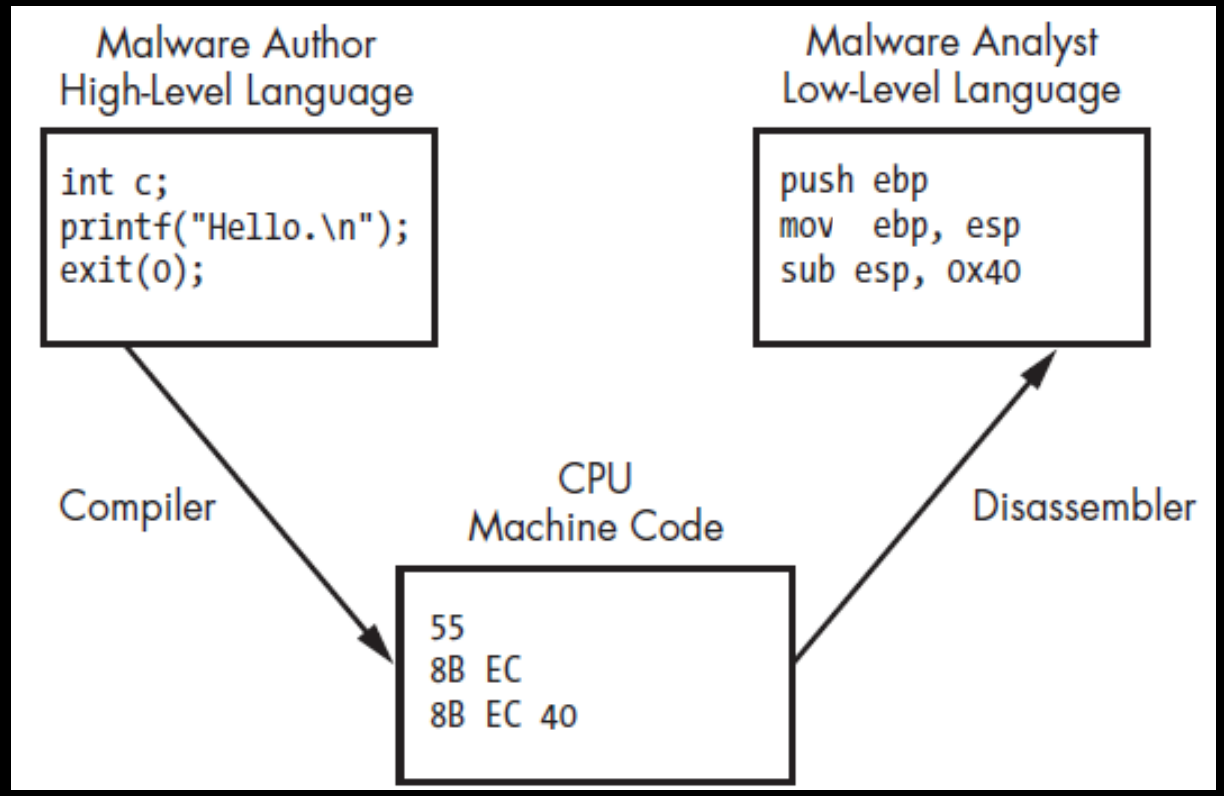
Outline

- Why learn x86 Assembly?
- Levels of Abstraction
- x86 Architecture
- CPU Instructions

Why dig deep?

- Basic Static and Dynamic can
  - Fail to tell the whole picture
  - Sometimes deceive you
- Know exactly what can be executed
- Assembly is the highest level language that can be reliably and consistently recovered from machine code

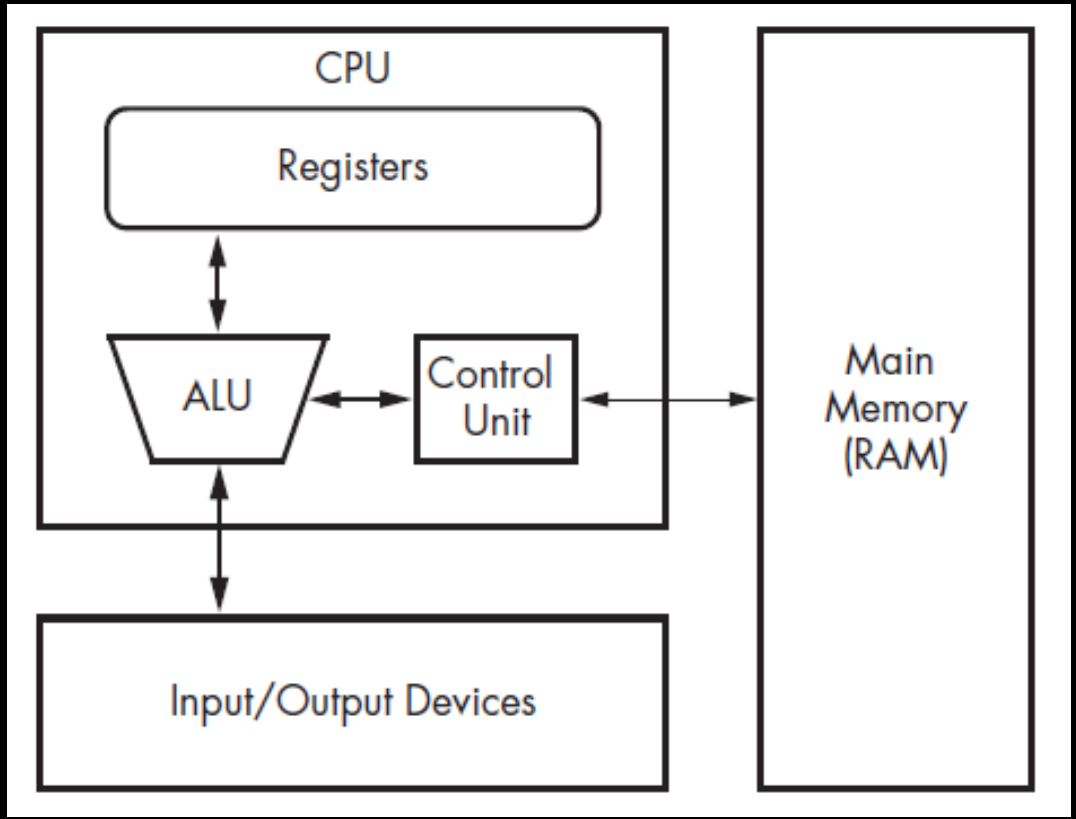
Abstraction Levels



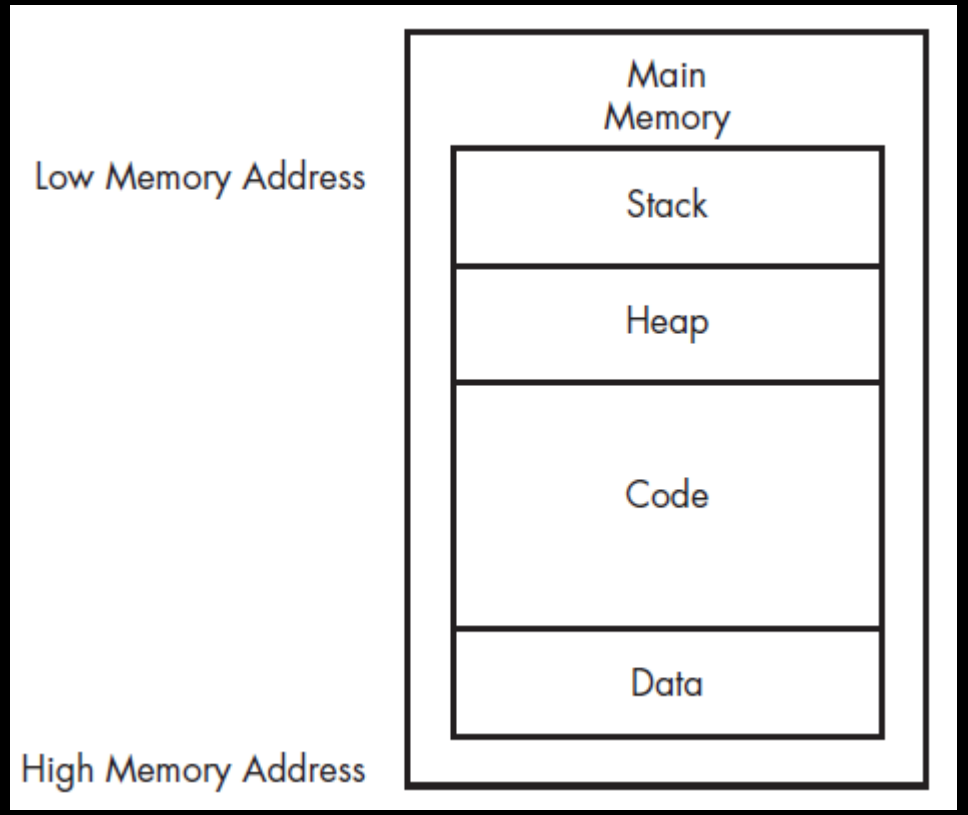
Software Reverse Engineering

- Disassembler
- Many architectures
  - x86
  - x64
  - SPARC
  - PowerPC
  - MIPS
  - ARM
- Why do we focus on x86?

Von Neumann


CPU
Registers
ALU
Control Unit
Main Memory (RAM)
Input/Output Devices

Main Memory


Main
Memory
Low Memory Address
Stack
Heap
Code
Data
High Memory Address

Instructions

- The building blocks of assembly programs

Mnemonic	Destination operand	Source operand
mov	ecx	0x42

Instruction	mov ecx,	0x42
Opcodes	B9	42 00 00 00

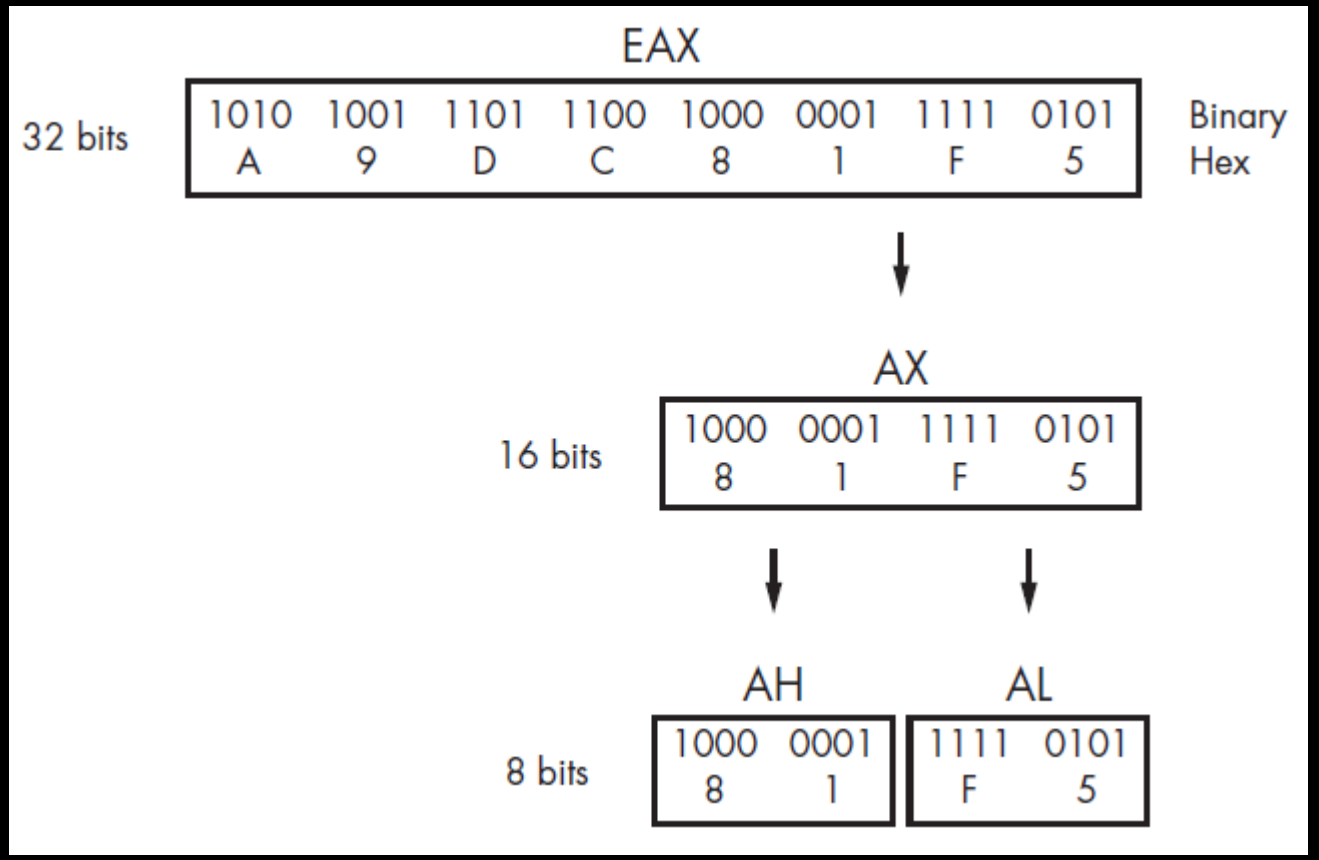
Operand Types

- Immediate – `0x42`
- Register – `EAX`
- Memory Address – `[EAX]`

Registers

- Microprocessors internal memory
- Accessed without performance penalty
- Temporary storage (RAM is used for long-term)
- General Registers
  - `EAX, EBX, ESP, EBP,` etc
- Status Register
  - `EFLAGS (ZF, CF)`
  - Set = 1, Clear = 0 (True/False)
- Instruction Pointer `EIP`

Register Breakdown



Register Conventions

- `EAX` used in multiplication and division
- `EBP` and `ESP` are used for tracking the stack
- `EAX` is used to store the return value for call
- `ECX` is used as a counting variable
- `ESI` and `EDI` are used for copying data in loops
  - (Source and Destination)

mov

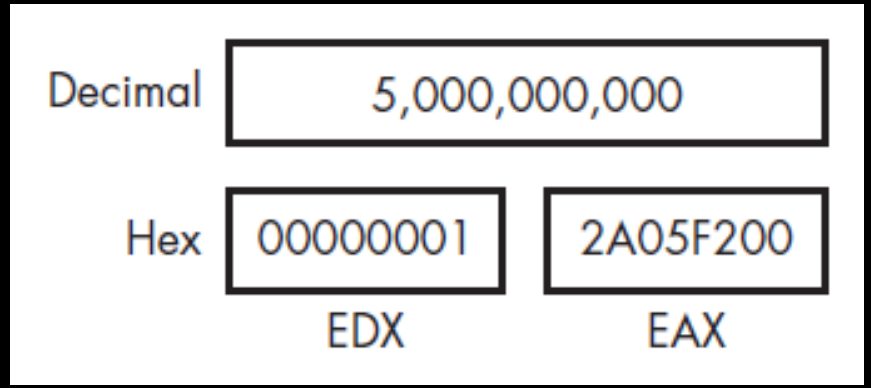
Instruction	Description
`mov eax, ebx`	Copies the contents of EBX into the EAX register
`mov eax, 0x42`	Copies the value 0x42 into the EAX register
`mov eax, [0x4037C4]`	Copies the 4 bytes at the memory location 0x4037C4 into the EAX register
`mov eax, [ebx]`	Copies the 4 bytes at the memory location specified by the EBX register into the EAX register
`mov eax, [ebx+esi*4]`	Copies the 4 bytes at the memory location specified by the result of the equation `ebx+esi*4` into the EAX register

Basic Arithmetic

Instruction	Description
`sub eax, 0x10`	Subtracts 0x10 from EAX
`add eax, ebx`	Adds EBX to EAX and stores the result in EAX
`inc edx`	Increments EDX by 1
`dec ecx`	Decrements ECX by 1

`mul` & `div`

- `mul value`
  - `eax` is multiplied
  - Result in `edx:eax`
- `div` value
  - `edx:eax` is divided
  - Result in `eax`
  - Remainder in `edx`



Logical Operators

- `xor, or, and`
  - `xor eax, eax`?
- Shift – `shr, shl`
- Rotate – `ror, rol`

Instruction
`xor eax, eax`
`or eax, 0x7575`
`mov eax, 0xA` `shl eax, 2`
`mov bl, 0xA` `ror bl, 2`

`nop`

- `xchg eax, eax`
- Literally does nothing
- Common in shellcode

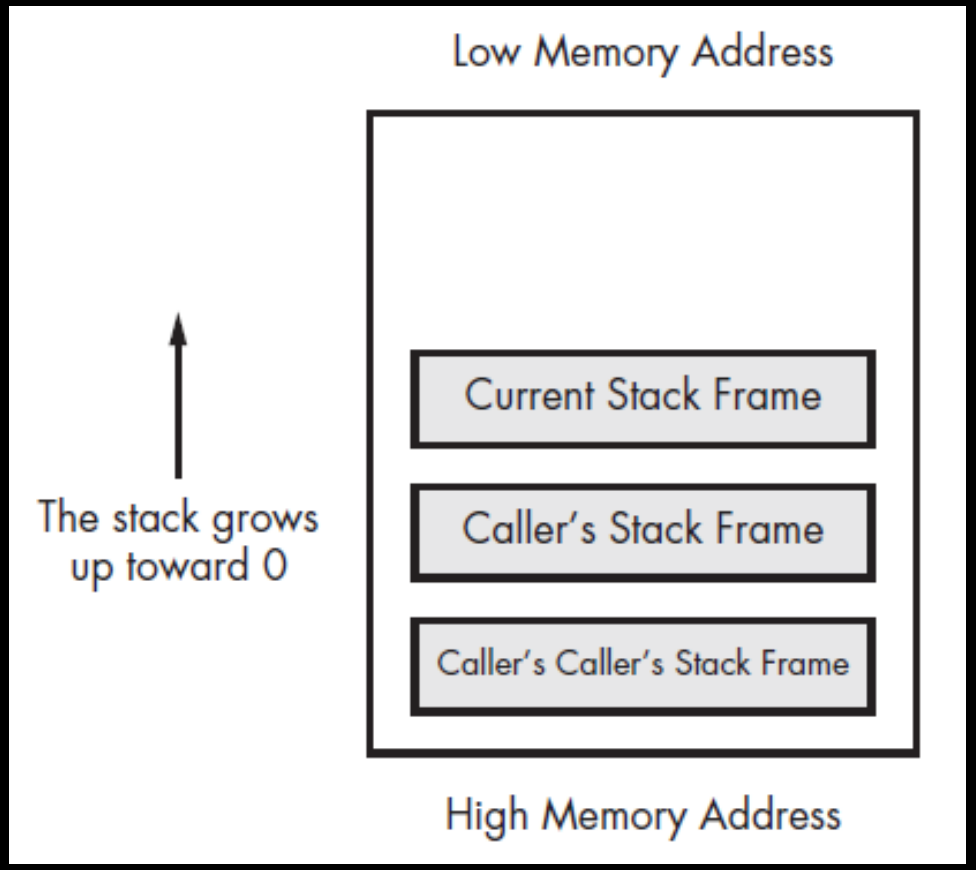
The Stack

- Data Structure
- LIFO
- Used for short-term storage
- Used for management of data exchanged between functions
- x86 built-in support
  - `ESP` and `EBP`
  - `push, pop, call, leave, enter, ret`
- Function Calls
  - Prologue
  - Epilogue

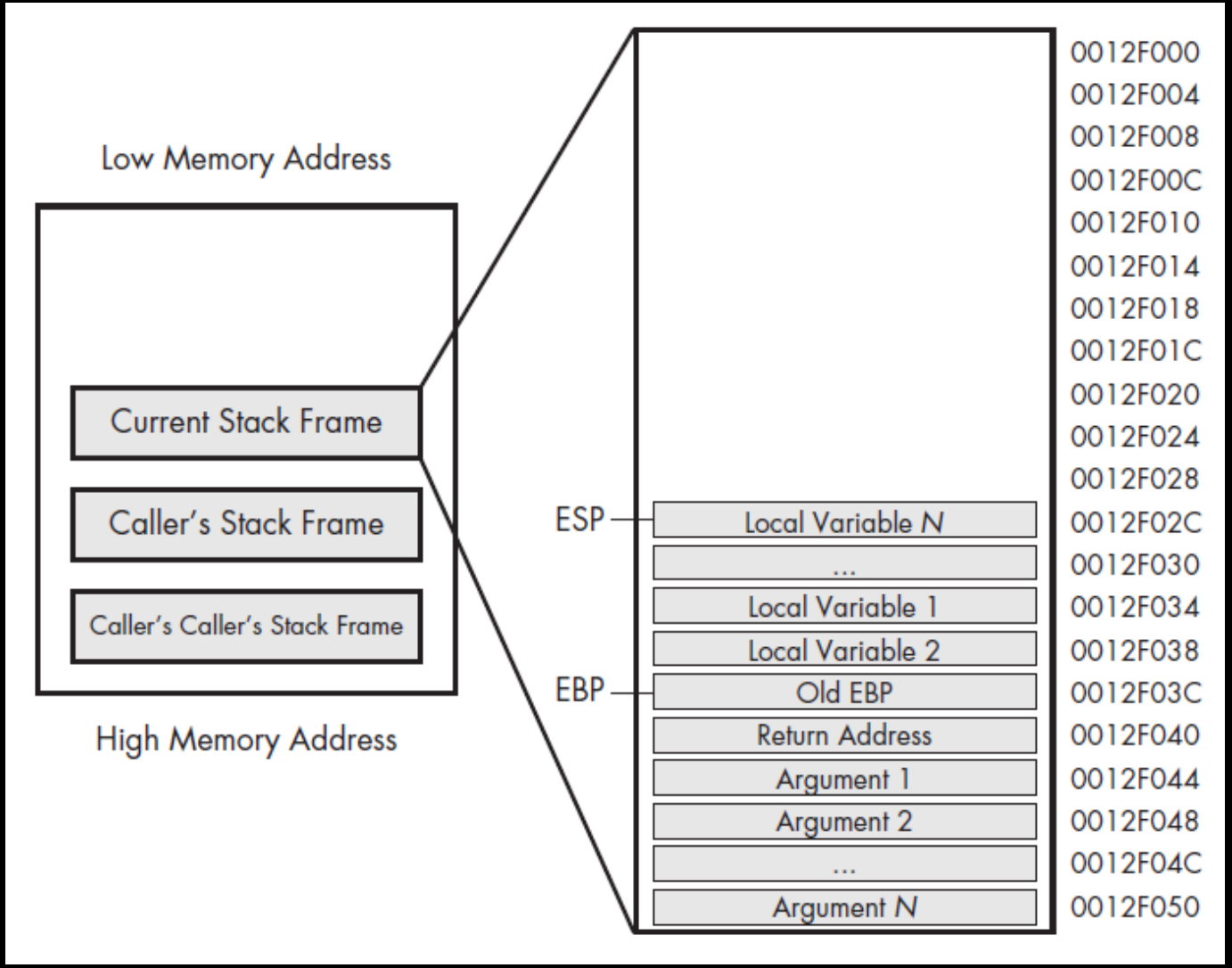
Function Calls

- Arguments placed on the stack using push
- A function is call using `call memory_location`
  - `EIP` pushed on the stack
- Through the prologue space is allocated for local variables and `EBP` (base pointer)
  - Save `EBP` for the calling function
- The function does it work
- Through the epilogue the stack is restored
- The function returns by calling the `ret` instruction
  - Pops the return address off the stack and into `EIP`
- The stack is adjusted to remove the arguments that were sent

Stack Layout



Individual Stack Frame



Conditionals

- All programming languages have the ability to make comparisons and make decisions based on those comparisons.
- Conditionals perform the comparison
- Two most popular
  - `test`
  - `cmp`
- `test` non-destructive `and`
- `cmp` non-destructive `sub`

cmp dst, src	ZF	CF
dst = src	1	0
dst < src	0	1
dst > src	0	0

Examples

```
mov  eax, 01011111b
mov  ebx, 10100001b
test eax, ebx      ; what is the value of ZF?

mov  eax, 1
cmp  eax, eax      ; what is the value of ZF?
```

Branching

- Way to control flow through a program
- Most popular is using Jump instructions
- Unconditional Jump
  - `jmp [location]`
- Conditional Jumps
  - Over 30 types

Conditional Jumps

Instruction	Description
`jz loc`	Jump to specified location if ZF = 1.
`jnz loc`	Jump to specified location if ZF = 0.
`je loc`	Same as `jz`, but commonly used after a `cmp` instruction. Jump will occur if the destination operand equals the source operand.
`jne loc`	Same as `jnz`, but commonly used after a `cmp`. Jump will occur if the destination operand is not equal to the source operand.
`jg loc`	Performs signed comparison jump after a `cmp` if the destination operand is greater than the source operand.
`jge loc`	Performs signed comparison jump after a `cmp` if the destination operand is greater than or equal to the source operand.
`ja loc`	Same as `jg`, but an unsigned comparison is performed.
`jae loc`	Same as `jge`, but an unsigned comparison is performed.
`jl loc`	Performs signed comparison jump after a `cmp` if the destination operand is less than the source operand.
`jle loc`	Performs signed comparison jump after a `cmp` if the destination operand is less than or equal to the source operand.
`jb loc`	Same as `jl`, but an unsigned comparison is performed.
`jbe loc`	Same as `jle`, but an unsigned comparison is performed.
`jo loc`	Jump if the previous instruction set the overflow flag (OF = 1).
`js loc`	Jump if the sign flag is set (SF = 1).
`jecxz loc`	Jump to location if ECX = 0.

`rep` Instructions

- Set of instructions for manipulating data buffers
- `ESI`, `EDI`, and `ECX`

Instruction	Description
`rep`	Repeat until ECX = 0
`repe, repz`	Repeat until ECX = 0 or ZF = 0
`repne, repnz`	Repeat until ECX = 0 or ZF = 1

Rep Examples

Instruction	Description
`repe cmpsb`	Used to compare two data buffers. EDI and ESI must be set to the two buffer locations, and ECX must be set to the buffer length. The comparison will continue until ECX = 0 or the buffers are not equal.
`rep stosb`	Used to initialize all bytes of a buffer to a certain value. EDI will contain the buffer location, and AL must contain the initialization value. This instruction is often seen used with `xor eax, eax`.
`rep movsb`	Typically used to copy a buffer of bytes. ESI must be set to the source buffer address, EDI must be set to the destination buffer address, and ECX must contain the length to copy. Byte-by-byte copy will continue until ECX = 0.
`repne scasb`	Used for searching a data buffer for a single byte. EDI must contain the address of the buffer, AL must contain the byte you are looking for, and ECX must be set to the buffer length. The comparison will continue until ECX = 0 or until the byte is found.

C Main Function

- `int main(int argc, char ** argv)`

```
filetestprogram.exe -r filename.txt

argc = 3
argv[0] = filetestprogram.exe
argv[1] = -r
argv[2] = filename.txt
```

C Main Function Example

```
004113CE    cmp     [ebp+argc], 3 ❶
004113D2    jz      short loc_4113D8
004113D4    xor     eax, eax
004113D6    jmp     short loc_411414
004113D8    mov     esi, esp
004113DA    push    2               ; MaxCount
004113DC    push    offset Str2     ; "-r"
004113E1    mov     eax, [ebp+argv]
004113E4    mov     ecx, [eax+4]
004113E7    push    ecx             ; Str1
004113E8    call    strncmp ❷
004113F8    test    eax, eax
004113FA    jnz     short loc_411412
004113FC    mov     esi, esp ❸
004113FE    mov     eax, [ebp+argv]
00411401    mov     ecx, [eax+8]
00411404    push    ecx             ; lpFileName
00411405    call    DeleteFileA
```