# **Reverse Engineering**

Assembly Instructions Class Demo

**Definitions:**

• mov : moves data into destination

mov eax, ebx # copy value in ebx into eax

• push : places data onto the stack

push[var] # push var into the stack

• pop : pulls data off the stack

pop [ebx] # pull top element of the stack into memory at EBX

• lea : loads data into identified address

lea eax,[var] # var is placed into EAX

**Instruction | Register | Data**

MOV EAX,5

MOV EBX,EAX

MOV [var1],EAX ; var1=5

MOV R12,[var1]

\*INCORRECT (memory to memory)

MOV [var2],[var1]

MOV EAX,5

MOV EBX,EAX

ADD EBX,10 ; EBX=EBX+10

ADD EBX,EAX ; EBX=EBX+EAX

EAX stays as value 5

SUB EAX,2 ; EAX=EAX-2

SUB EBX,EAX ; EBX=EBX-EAX

EBX now has a value 17

MUL/IMUL EAX,5

DIV/IDIV EAX,3

INC EBX

DEC EBX

**Jump and Compare**

MOV EAX,5

MOV EBX,6

CMP EAX,EBX EAX-EBX

JE lab1 (conditional)

MOV EAX,1

JMP lab1 (unconditional)

ret

lab1:

MOV EAX,0

ret

this will return 0

**Instruction Pointer (read-only, can’t write/modify)**

**Stack**

MOV EAX,5

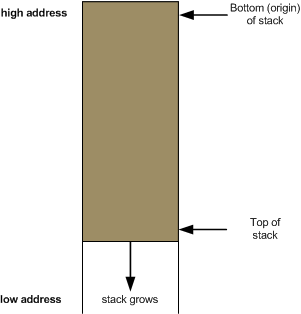
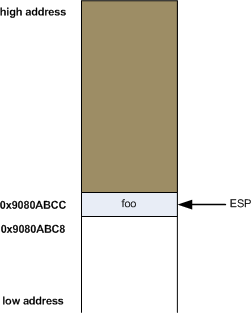
MOV EBX,120

PUSH EAX (pushes onto the \*top\* of the stack)

PUSH EBX

POP EDX

**\*Subtract to grow stack down, add to shrink up**

**Loop**:

add r12, 1

cmp r12, r13

jl loop

| **#include <stdio.h>**  **int math(char\* num1){**  **int sumOf;**  **int num2 = 66;**  **int num11 = atoi(num1); //Converts the string into an integer**  **sumOf = num11+num2;**  **return sumOf;**  **}**  **int main(void){**  **char num1[10];**  **printf("Enter number: ");**  **fgets(num1,5,stdin);**  **strtok(num1, "\n");**  **printf("%d\n",math(num1)); //Adds 66 to whatever the user inputs and then writes it to the screen**  **return 0;**  **}** |
| --- |

**Control flow**

• ret : return, pops the memory location of the stack and then execuates a unconditional jmp to

subroutine location

Example 1

main:

mov rax, 16 //16 moved into rax

push rax //push value of rax (16) onto stack. RSP is pushed up 8 bytes (64

bits)

jmp mem2 //jmp to mem2 memory location

mem1: //Subroutine

mov rax, 0 //move 0 (error free) exit code to rax

ret //return out of code

mem2: //Subroutine

pop r8 //pop value on the stack (16) into r8. RSP falls 8 bytes

cmp rax, r8 //compare rax register value (16) to r8 register value (16)

je mem1 //jump if comparison has zero bit set to mem1

Example 2

main:

mov rcx, 25 //store the value 25 in rcx register

mov rbx, 62 //store the value 62 in rbx register

jmp mem1 //jumps to mem1 location

mem1: //Subroutine

sub rbx, 40 //subtract 40 from rbx

mov rsi, rbx //copy rbx value to rsi

cmp rcx, rsi //compare the values in rcx and rsi

jmple mem2 //jumps to mem2 location if value is less than or equal

mem2: //Subroutine

mov rax, 0 //store 0 in rax

ret //return out of code