**LAB 5**

**AIM:**

To understand codes disassembled for exe files written in C.

**SOFTWARE REQUIRED:**

Code::Blocks, IDA Freeware

**INTRODUCTION:**

Assembly language is a low-level programming language for a computer or other programmable device.

The Interactive Disassembler (IDA) is a disassembler for computer software which generates assembly language source code from machine-executable code. It supports a variety of executable formats for different processors and operating systems.

IDA is used widely in software reverse engineering, including for malware analysis and software vulnerability research.

**Program 1**

**C code:**

int main() {

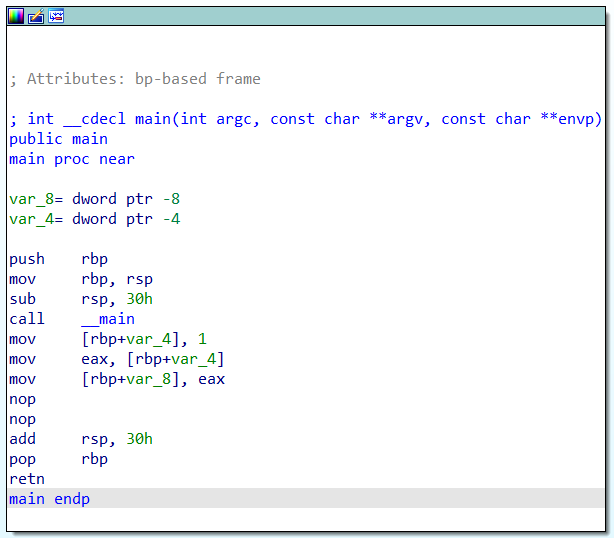
int x = 1;

int y;

y = x;

return; }

**Disassembled code:**

****

**Explanation:**

1. Defines a local variable var\_8 as a 32-bit pointer on the stack.
2. Defines a local variable var\_4 as a 32-bit pointer on the stack.
3. Pushes the base pointer onto the stack.
4. Sets the base pointer to the current stack pointer.
5. Allocates 48 bytes of memory on the stack for local variables.
6. Calls the \_\_main function for initialization.
7. Stores the value 1 in the memory location [rbp+var\_4].
8. Retrieves the value stored at the memory location [rbp+var\_4] and places it in the eax register.
9. Stores the value of eax in the memory location [rbp+var\_8].
10. No operation instruction (does nothing).
11. No operation instruction (does nothing).
12. Releases the memory allocated for local variables.
13. Restores the previous base pointer from the stack.
14. Returns from the main function.
15. Marks the end of the main procedure.

**Program 2**

**C code:**

int a;

char b;

int main() {

a = 41; b = 'A';

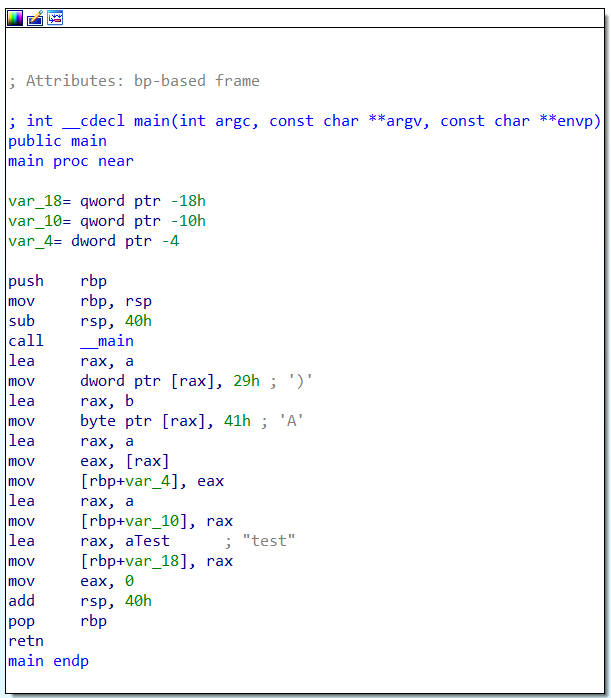
int x = a;

int \*y = &a;

char \*string = "test";

return 0; }

**Disassembled code:**

****

**Explanation:**

1. Defines a local variable var\_18 as a 64-bit pointer on the stack.
2. Defines a local variable var\_10 as a 64-bit pointer on the stack.
3. Defines a local variable var\_4 as a 32-bit pointer on the stack.
4. Pushes the base pointer onto the stack.
5. Sets the base pointer to the current stack pointer.
6. Allocates 64 bytes of memory on the stack for local variables.
7. Calls the main function for initialization.
8. Loads the address of the variable a into the register rax.
9. Stores the value ')' in the memory location pointed to by rax.
10. Loads the address of the variable b into the register rax.
11. Stores the value 'A' in the memory location pointed to by rax.
12. Loads the address of the variable a into the register rax.
13. Retrieves the value stored at the memory location pointed to by rax and places it in eax.
14. Stores the value of eax in the memory location [rbp+var\_4].
15. Loads the address of the variable a into the register rax.
16. Stores the value of rax (address of a) in the memory location [rbp+var\_10].
17. Loads the address of the string "test" into the register rax.
18. Stores the value of rax (address of the string) in the memory location [rbp+var\_18].
19. Sets the value of eax to 0.
20. Releases the memory allocated for local variables.
21. Restores the previous base pointer from the stack.
22. Returns from the main function.
23. Marks the end of the main procedure.

**Program 3**

**C code:**

int main() {

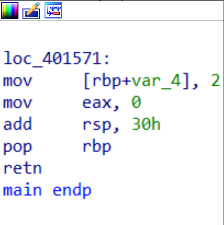
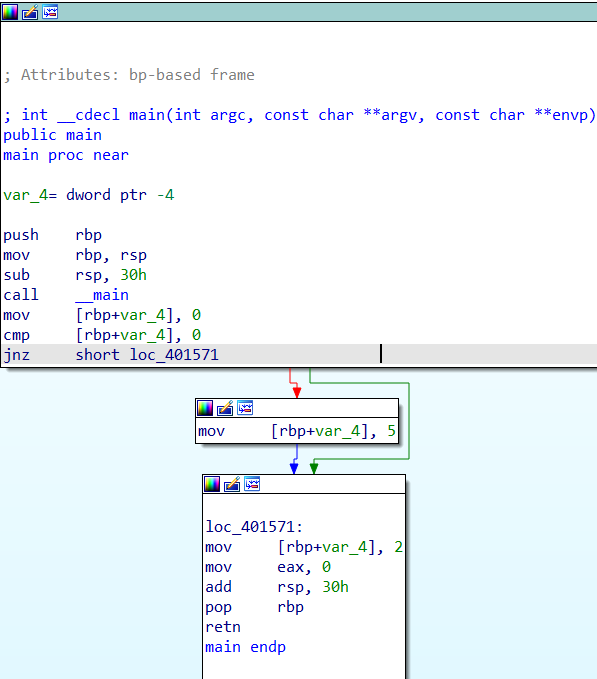
int x=0;

if (x==0) {

x=5; }

x=2; }

**Disassembled code:**

****

**Explanation:**

1. Defines a local variable var\_4 as a 32-bit pointer on the stack.
2. Pushes the base pointer onto the stack.
3. Sets the base pointer to the current stack pointer.
4. Allocates 48 bytes of memory on the stack for local variables.
5. Calls the \_\_main function for initialization.
6. Stores the value 0 in the memory location [rbp+var\_4].
7. Compares the value at the memory location [rbp+var\_4] with 0.
8. Jumps to the location loc\_401571 if the previous comparison was not equal to zero (jump if not zero).
9. Stores the value 5 in the memory location [rbp+var\_4].
10. Marks a location in the code for reference.
11. Stores the value 2 in the memory location [rbp+var\_4].
12. Sets the value of the eax register to 0.
13. Releases the memory allocated for local variables.
14. Restores the previous base pointer from the stack.
15. Returns from the main function.
16. Marks the end of the main procedure.

**Program 4**

**C code:**

void test() { }

void main() {

test(); }

**Disassembled code:**

****

**Explanation:**

1. Pushes the base pointer onto the stack.
2. Sets the base pointer to the current stack pointer.
3. Allocates 32 bytes of memory on the stack for local variables.
4. Calls the `\_\_main` function for initialization.
5. Calls the `test` function.
6. No operation instruction (does nothing).
7. Releases the memory allocated for local variables.
8. Restores the previous base pointer from the stack.
9. Returns from the `main` function.
10. Marks the end of the `main` procedure.