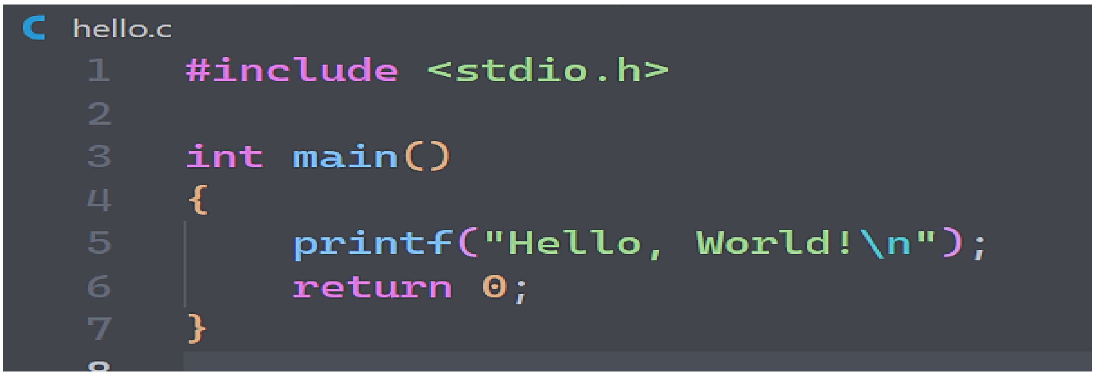
Assignment (Reverse Engineering)



hello.c

```c
#include <stdio.h>

int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

Introduction

Reverse Engineering is a process of analysing a software or a program by decompiling or disassembling the software using software like Ghidra, cutter, IDA pro, etc.

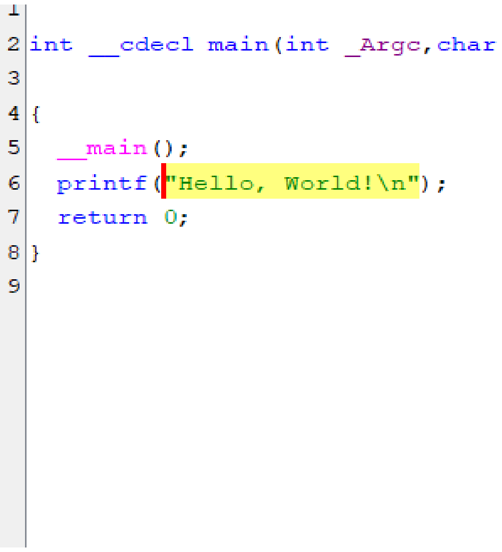
1. Purpose of this exercise : Reverse Engineer a C program executable that prints “Hello World” using Ghidra.

RBP : It is Base Pointer, that is pushed in the memory

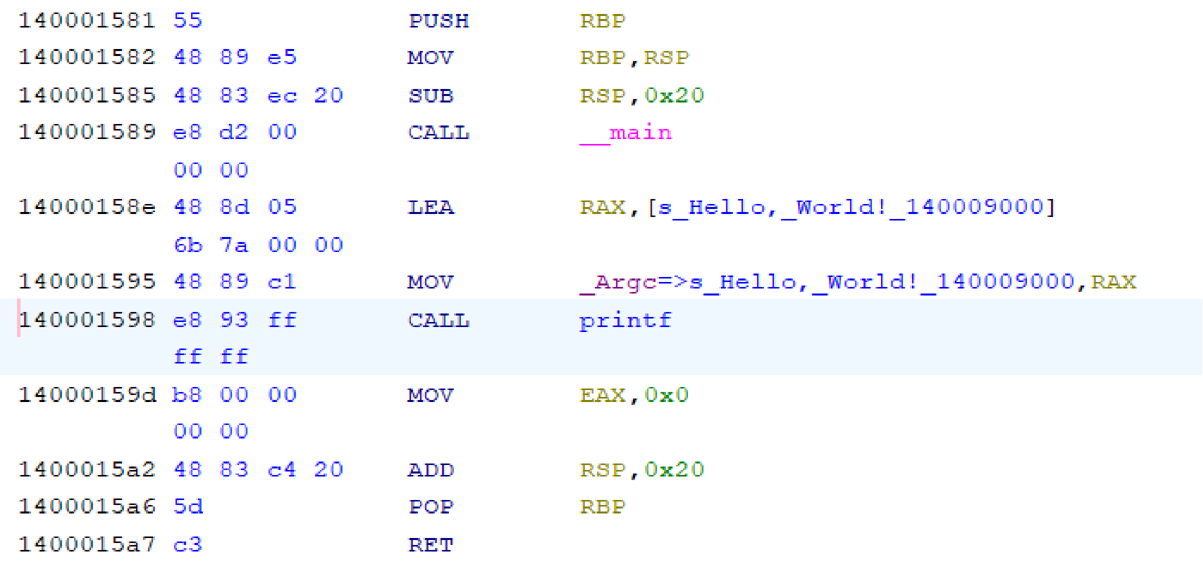
Next is, RSP : It is Stack Pointer which is assigned to the Base pointer (RBP) with MOV operation.

Then, SUB (subtract) operation of 0x20 that is 32byte. After that __main is being triggered. Next there is RAX, which is an accumulator register, used for athematic and logical operations or any special instruction. It is used here to point to the string literal which is store at some address that contains data “Hello, World! \n”.

The string literal is moved to _Argc from RAX register.



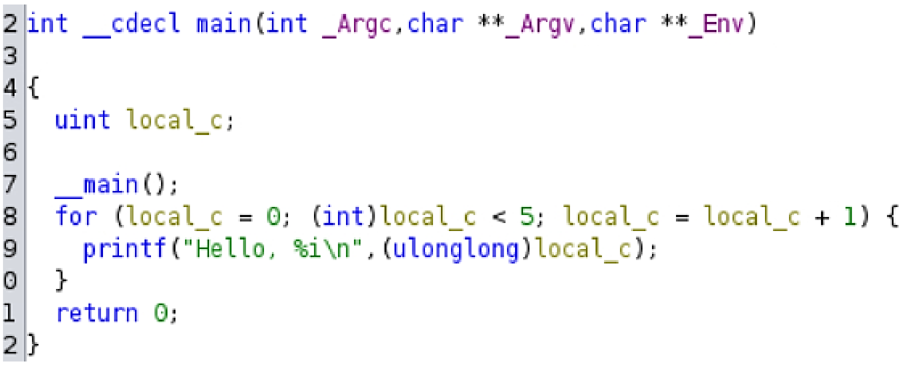
```c
int __cdecl main(int _Argc,char

{
  __main();
  printf("Hello, World!\n");
  return 0;
}
```



```
140001581 55             PUSH       RBP
140001582 48 89 e5       MOV        RBP,RSP
140001585 48 83 ec 20    SUB        RSP,0x20
140001589 e8 d2 00       CALL       __main
          00 00
14000158e 48 8d 05       LEA        RAX,[s_Hello,_World!_140009000]
          6b 7a 00 00
140001595 48 89 c1       MOV        _Argc=>s_Hello,_World!_140009000,RAX
140001598 e8 93 ff       CALL       printf
          ff ff
14000159d b8 00 00       MOV        EAX,0x0
          00 00
1400015a2 48 83 c4 20    ADD        RSP,0x20
1400015a6 5d             POP        RBP
1400015a7 c3             RET
```

Then prinf function is triggered using CALL method, where it prints the string. After that EAX assigned to Null, 0x0. Then, the Memory is been released using ADD to RSP, 0x20 that is 32byte which is been reserved before. Then POP operation on RBP, pops the value from the stack into the RBP register and updates the stack pointer. And RET to return the function, with 0.

2. Purpose of this exercise : Reverse Engineer a C program executable that prints “Hello, %i\n” for 5 times using Ghidra.



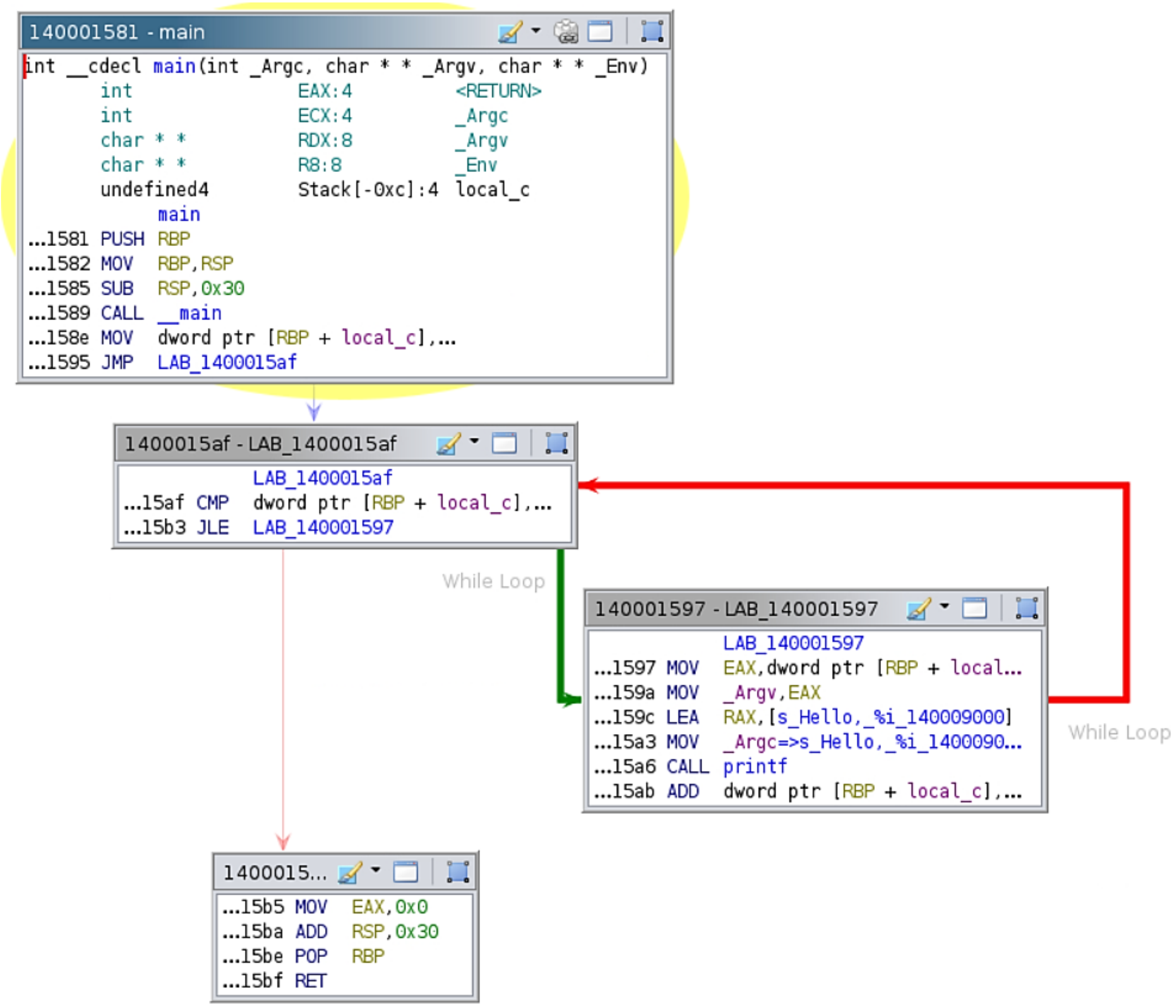
```
int __cdecl main(int _Argc,char **_Argv,char **_Env)

{
  uint local_c;

  __main();
  for (local_c = 0; (int)local_c < 5; local_c = local_c + 1) {
    printf("Hello, %i\n",(ulonglong)local_c);
  }
  return 0;
}
```

Similarly, like above first 0x30 that is 48bytes of space been made using SUB operation in the stack.

Then, calling the __main function. After that there is EAX, loop counter set to 0, it keeps coping itself to EDX. Using LEA operation, “Hello, %i\n” string address in RAX.

RCX and EDX are used as pointer and integer in printf passed through arguments, and then call printf.

The counter is for less than equal to 4, and print the string. And return 0 in the end, also POP RBP.



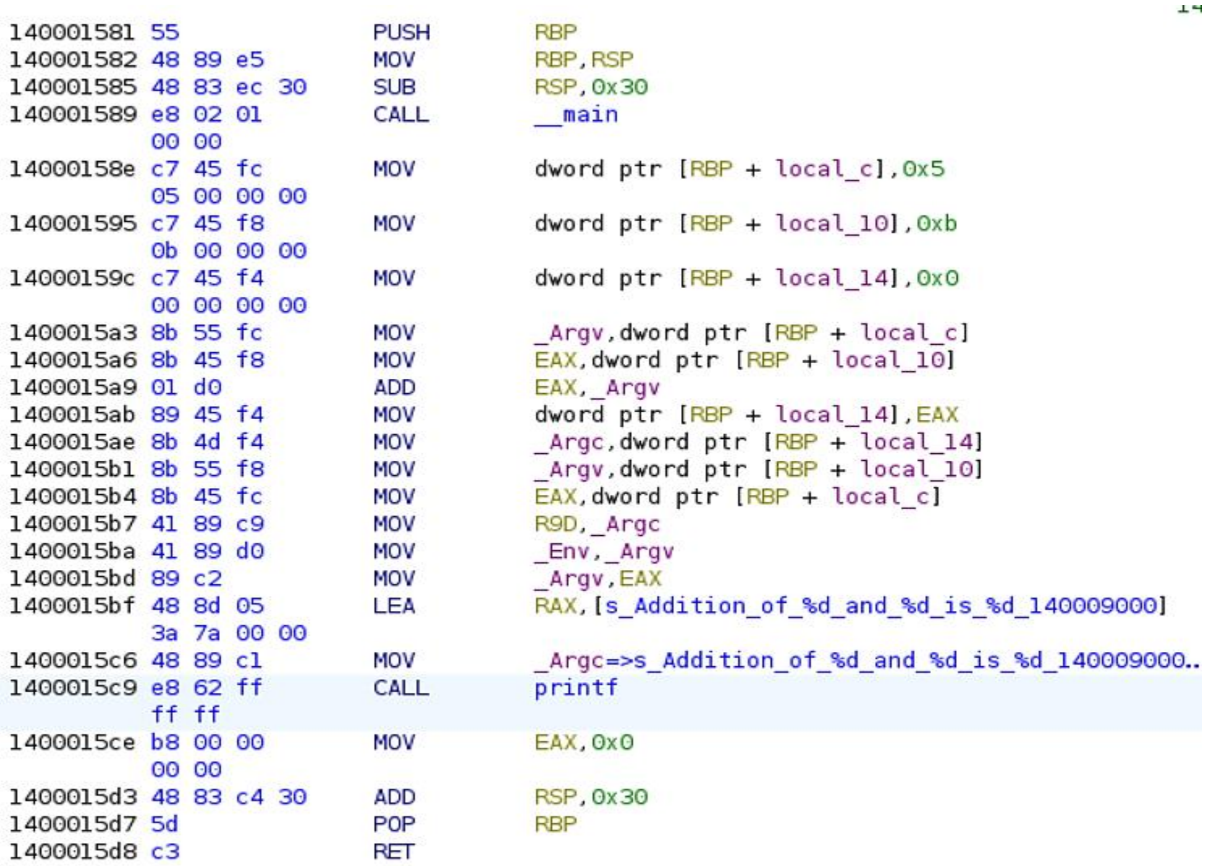
```
int __cdecl main(int _Argc, char * * _Argv, char * * _Env)
        int                  EAX:4          <RETURN>
        int                  ECX:4          _Argc
        char * *             RDX:8          _Argv
        char * *             R8:8           _Env
        undefined4           Stack[-0xc]:4  local_c
                main
...1581 PUSH  RBP
...1582 MOV   RBP,RSP
...1585 SUB   RSP,0x30
...1589 CALL  __main
...158e MOV   dword ptr [RBP + local_c],...
...1595 JMP   LAB_1400015af
```

```
              LAB_1400015af
...15af CMP   dword ptr [RBP + local_c],...
...15b3 JLE   LAB_140001597
```

```
              LAB_140001597
...1597 MOV   EAX,dword ptr [RBP + local...
...159a MOV   _Argv,EAX
...159c LEA   RAX,[s_Hello,_%i_140009000]
...15a3 MOV   _Argc=>s_Hello,_%i_1400090...
...15a6 CALL  printf
...15ab ADD   dword ptr [RBP + local_c],...
```

```
...15b5 MOV   EAX,0x0
...15ba ADD   RSP,0x30
...15be POP   RBP
...15bf RET
```

3. Purpose of this exercise : Reverse Engineer a C program executable that prints addition of two numbers using Ghidra.



```
140001581 55               PUSH     RBP
140001582 48 89 e5         MOV      RBP,RSP
140001585 48 83 ec 30      SUB      RSP,0x30
140001589 e8 02 01         CALL     __main
          00 00
14000158e c7 45 fc         MOV      dword ptr [RBP + local_c],0x5
          05 00 00 00
140001595 c7 45 f8         MOV      dword ptr [RBP + local_10],0xb
          0b 00 00 00
14000159c c7 45 f4         MOV      dword ptr [RBP + local_14],0x0
          00 00 00 00
1400015a3 8b 55 fc         MOV      _Argv,dword ptr [RBP + local_c]
1400015a6 8b 45 f8         MOV      EAX,dword ptr [RBP + local_10]
1400015a9 01 d0            ADD      EAX,_Argv
1400015ab 89 45 f4         MOV      dword ptr [RBP + local_14],EAX
1400015ae 8b 4d f4         MOV      _Argc,dword ptr [RBP + local_14]
1400015b1 8b 55 f8         MOV      _Argv,dword ptr [RBP + local_10]
1400015b4 8b 45 fc         MOV      EAX,dword ptr [RBP + local_c]
1400015b7 41 89 c9         MOV      R9D,_Argc
1400015ba 41 89 d0         MOV      _Env,_Argv
1400015bd 89 c2            MOV      _Argv,EAX
1400015bf 48 8d 05         LEA      RAX,[s_Addition_of_%d_and_%d_is_%d_140009000]
          3a 7a 00 00
1400015c6 48 89 c1         MOV      _Argc=>s_Addition_of_%d_and_%d_is_%d_140009000..
1400015c9 e8 62 ff         CALL     printf
          ff ff
1400015ce b8 00 00         MOV      EAX,0x0
          00 00
1400015d3 48 83 c4 30      ADD      RSP,0x30
1400015d7 5d               POP      RBP
1400015d8 c3               RET
```

Similar to above RBP is been pushed, and RSP been assigned to RBP. Then, main function is triggered. After that three local variables were initialised on the stack that are local_c at offset -4 from RBP, set to 5, local_10 at offset -8, set to 11, local_14 at offset -12, initialized to 0.

local_14 = local_c + local_10 is the addition. Loading a temporary register in local_c and EAX with local_10. The result is stored in local_14.

Using LEA again to load address in RAX of the format string "Addition of %d and %d is %d”. Then moves that pointer into RCX, Counter. And calls printf in the end to print the statement.

Finally, after the loop it return 0, releasing Stack pointer reserved of 48bytes.

These were are examples of x86_64 architecture based, 64bits executable for windows which is been analysed using Ghidra software for its functionality engineer and understanding how reverse engineering works using registers and operations like MOV, SUB, ADD, CALL, LEA etc as shown above.