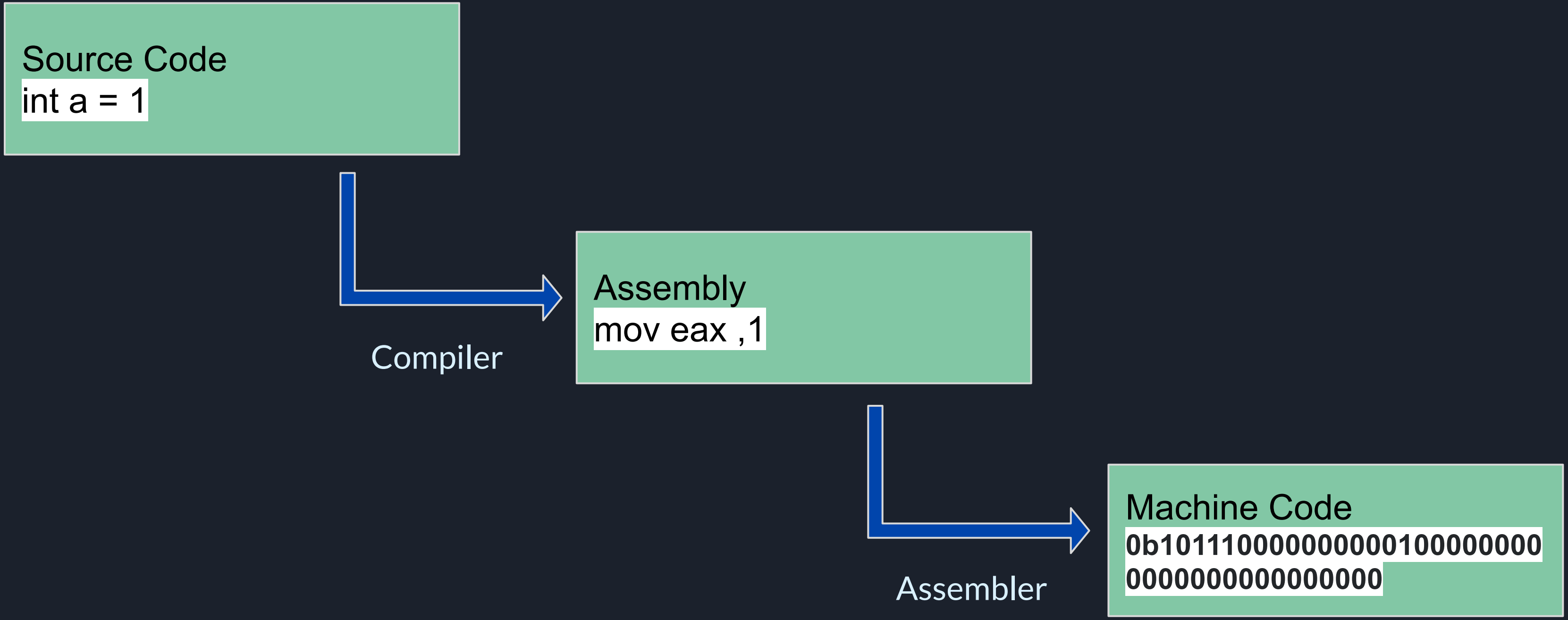
ObjDump & GDB

Contents:

Program Life Cycle [High Level]



Source Code

```
int a = 1
```

Assembly

```
mov eax ,1
```

Machine Code

```
0b1011100000000000100000000
0000000000000000
```

Important terms

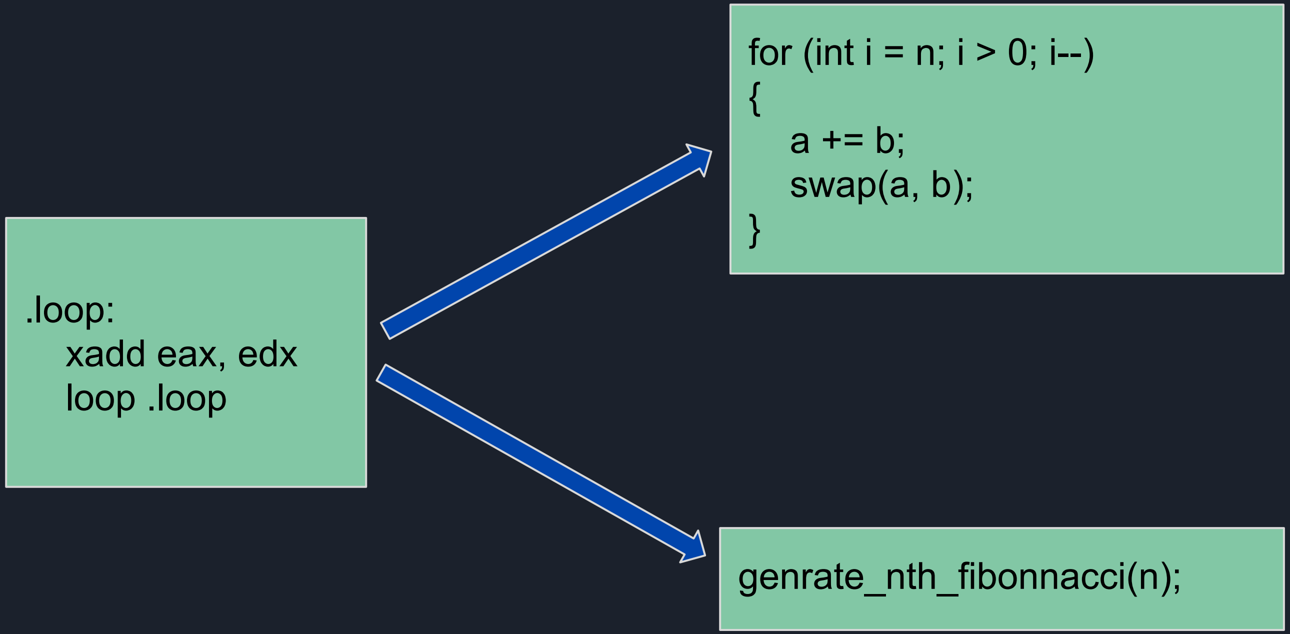
01 **Compilation :** Is language-to-language transformation in which the original meaning is preserved. [Ex : C source code to x86 Assembly] [Compilation is architecture dependent]

02 **Assembler :** Takes ASCII encoded assembly language code as input, and converts it into machine language code which the CPU can execute

03 **Disassembly :** The role of a disassembler is to display machine-language operation codes as human-readable mnemonics.

04 **Decompiler :** A decompiler, given an executable program compiled in any high-level language, attempts to produce a high-level language program that performs the same function as the executable program.

Why can’t we have perfect decompilation?



```
.loop:
    xadd eax, edx
    loop .loop
```

```
for (int i = n; i > 0; i--)
{
    a += b;
    swap(a, b);
}
```

```
genrate_nth_fibonnacci(n);
```

Tools like Ghidra, IDA Pro do this, but they are not 100% accurate in recovering the original source code

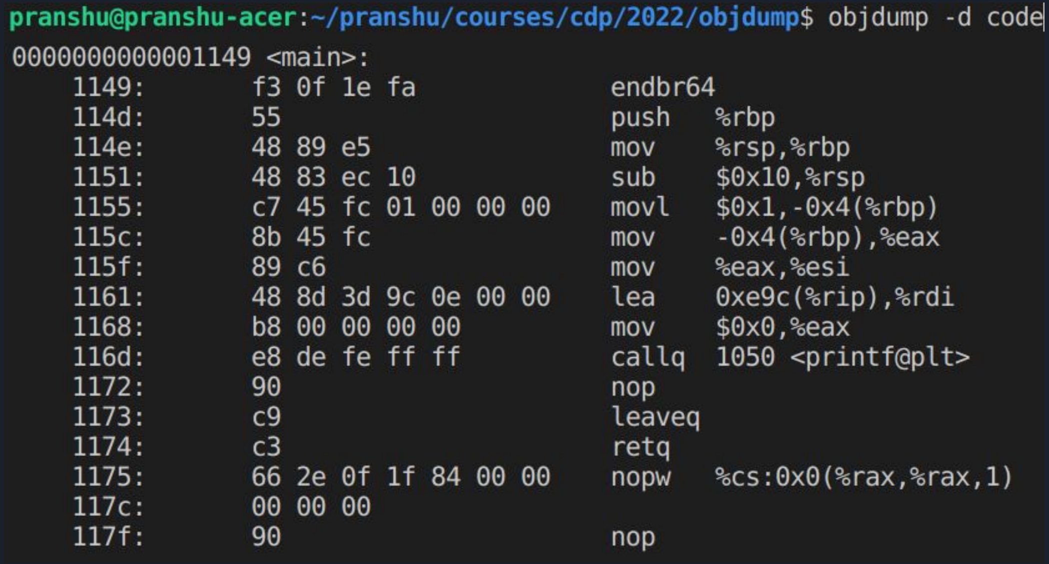
ObjDump

- objdump is a command-line program for displaying various information about object files on Unix-like operating systems.
- It can be used as a disassembler for display assembly code from binary data
- Syntax :
  objdump [options] objfile

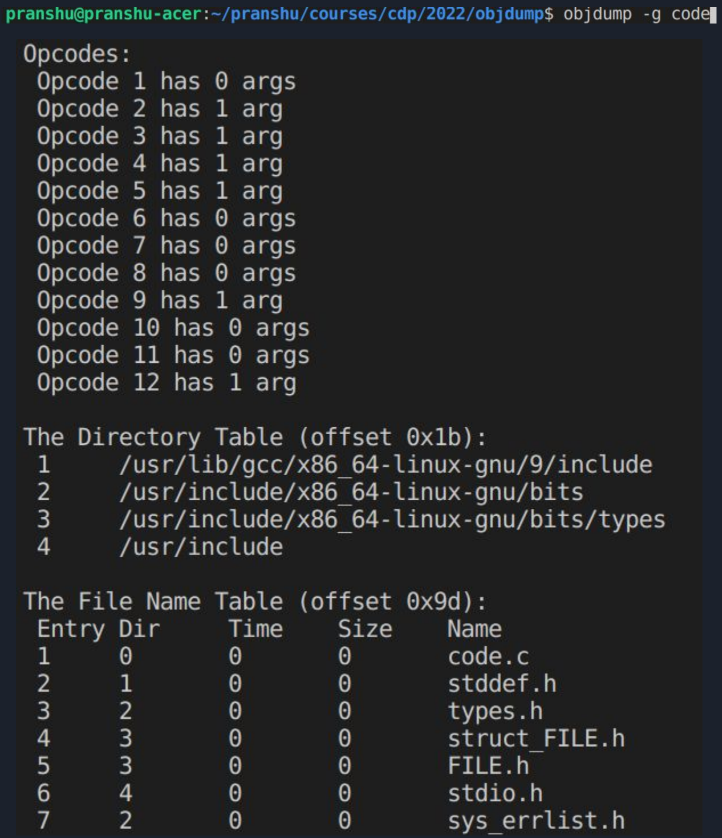
ObjDump - Important Flags

Flag	Command	Purpose
-d	objdump -d filename	Display assembler contents of executable sections
-f	objdump -f filename	Display the contents of the overall file header
-p	objdump -p filename	Display object format specific file header contents
-h	objdump -h filename	Display the contents of section headers
-g	objdump -g filename	Display debug information
-t	objdump -t filename	Display the contents of symbol table

ObjDump - Screenshots and examples



```
pranshu@pranshu-acer:~/pranshu/courses/cdp/2022/objdump$ objdump -d code
0000000000001149 <main>:
    1149:       f3 0f 1e fa             endbr64
    114d:       55                      push   %rbp
    114e:       48 89 e5                mov    %rsp,%rbp
    1151:       48 83 ec 10             sub    $0x10,%rsp
    1155:       c7 45 fc 01 00 00 00    movl   $0x1,-0x4(%rbp)
    115c:       8b 45 fc                mov    -0x4(%rbp),%eax
    115f:       89 c6                   mov    %eax,%esi
    1161:       48 8d 3d 9c 0e 00 00    lea    0xe9c(%rip),%rdi
    1168:       b8 00 00 00 00          mov    $0x0,%eax
    116d:       e8 de fe ff ff          callq  1050 <printf@plt>
    1172:       90                      nop
    1173:       c9                      leaveq
    1174:       c3                      retq
    1175:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
    117c:       00 00 00
    117f:       90                      nop
```



```
pranshu@pranshu-acer:~/pranshu/courses/cdp/2022/objdump$ objdump -g code
 Opcodes:
  Opcode 1 has 0 args
  Opcode 2 has 1 arg
  Opcode 3 has 1 arg
  Opcode 4 has 1 arg
  Opcode 5 has 1 arg
  Opcode 6 has 0 args
  Opcode 7 has 0 args
  Opcode 8 has 0 args
  Opcode 9 has 1 arg
  Opcode 10 has 0 args
  Opcode 11 has 0 args
  Opcode 12 has 1 arg

 The Directory Table (offset 0x1b):
  1     /usr/lib/gcc/x86_64-linux-gnu/9/include
  2     /usr/include/x86_64-linux-gnu/bits
  3     /usr/include/x86_64-linux-gnu/bits/types
  4     /usr/include

 The File Name Table (offset 0x9d):
  Entry Dir     Time    Size    Name
  1     0       0       0       code.c
  2     1       0       0       stddef.h
  3     2       0       0       types.h
  4     3       0       0       struct_FILE.h
  5     3       0       0       FILE.h
  6     4       0       0       stdio.h
  7     2       0       0       sys_errlist.h
```

What is GDB ?

Debugger - A computer program that assists in the detection and correction of errors in other computer programs.

GDB - GNU Debugger

Features:

- Provides an interactive shell
- Learn once, Debug anywhere

```
Printf("BAD\n");
```

img source

Why GDB ?

- Watch and modify variables during runtime
- Why and where did the program quit or fail
- Check the current state of the program
- Change the execution flow dynamically
- Can be configured to provide a GUI of the flow with IDE/TEs like VS code.

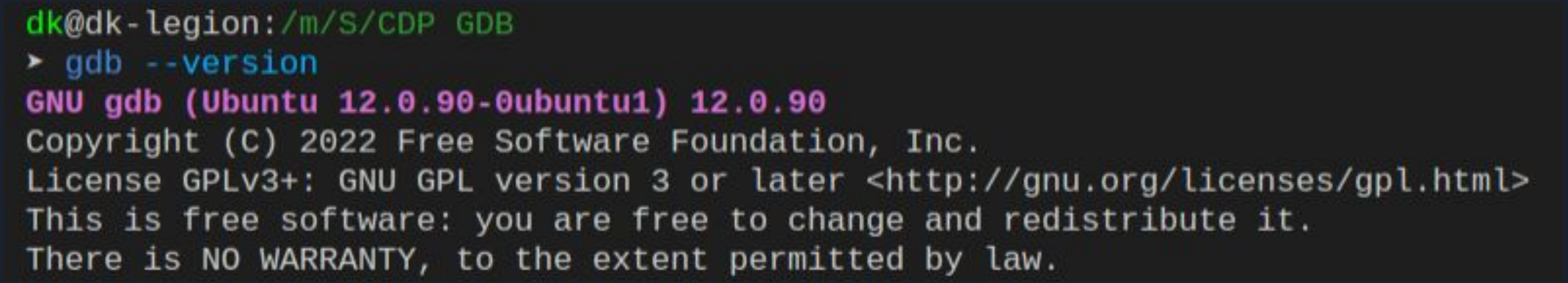
Basic GDB Commands

run	[args]
start	[args]
break	[line/function] or [cond]
delete	<breakpoint>
clear	
enable/disable	<breakpoint>
continue	
next	

step	
list	[line/function]
print	[exp] or [var=val]
condition	<breakpoint> <condition>
backtrace	
help	[subcommand]
finish	
kill	

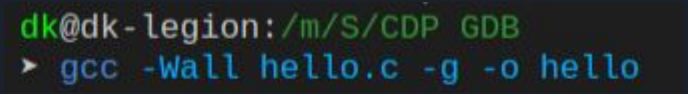
Running GDB

Checking if installed:



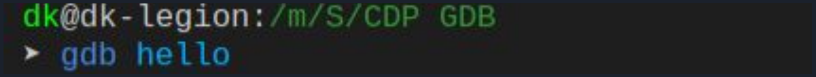
```
dk@dk-legion:/m/S/CDP GDB
➤ gdb --version
GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

Compiling for using GDB:

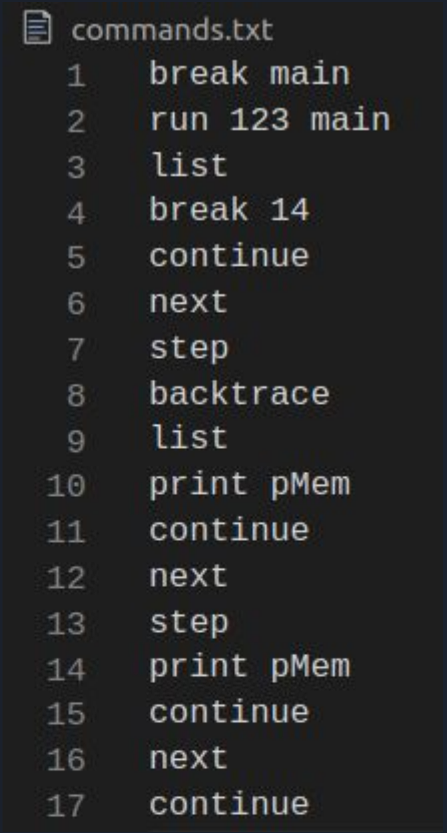


```
dk@dk-legion:/m/S/CDP GDB
➤ gcc -Wall hello.c -g -o hello
```

Using GDB:

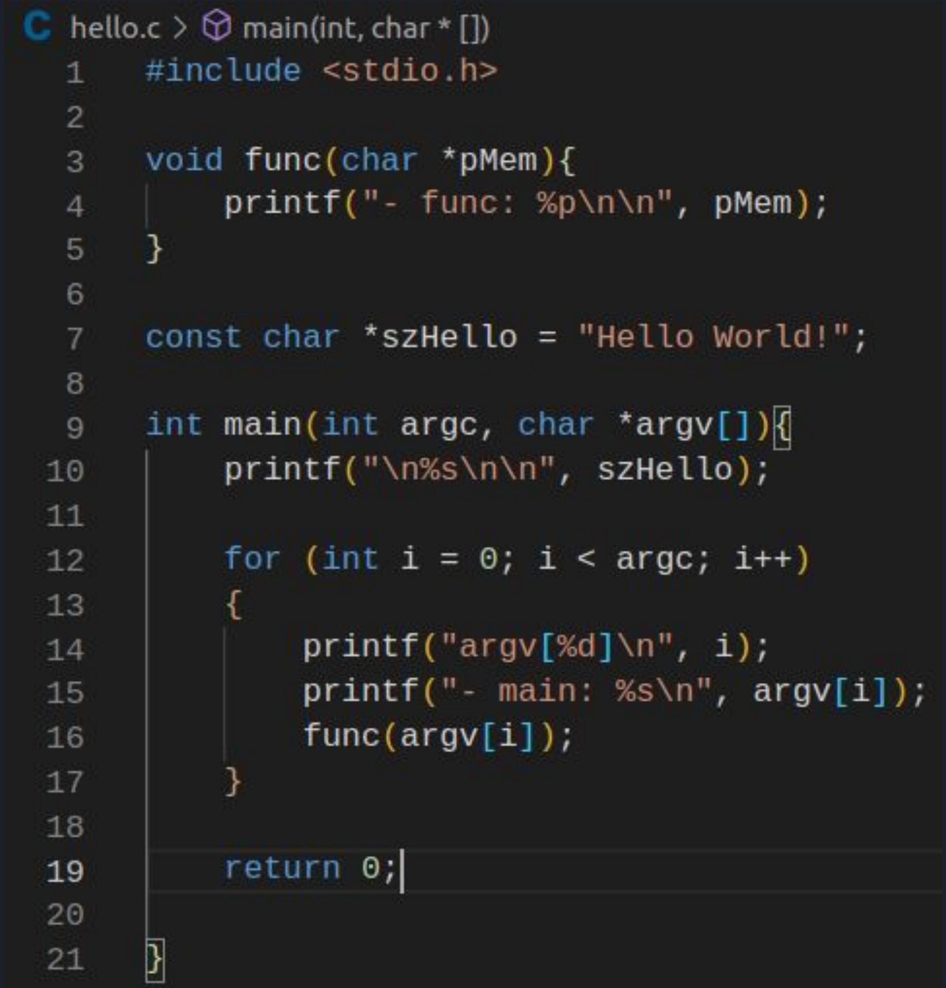


```
dk@dk-legion:/m/S/CDP GDB
➤ gdb hello
```



commands.txt

```
break main
run 123 main
list
break 14
continue
next
step
backtrace
list
print pMem
continue
next
step
print pMem
continue
next
continue
```



hello.c > main(int, char * [])

```
 1  #include <stdio.h>
 2
 3  void func(char *pMem){
 4      printf("- func: %p\n\n", pMem);
 5  }
 6
 7  const char *szHello = "Hello World!";
 8
 9  int main(int argc, char *argv[]){
10      printf("\n%s\n\n", szHello);
11
12      for (int i = 0; i < argc; i++)
13      {
14          printf("argv[%d]\n", i);
15          printf("- main: %s\n", argv[i]);
16          func(argv[i]);
17      }
18
19      return 0;
20
21  }
```