AIM: Understand modules of the compilation process with the help of a program. (Pre-processor, Compiler, Assembler, Linker/Loader)

Theory: Explain Language Processing System with System Software Pre-processor, Compiler, Assembler and Linker/Loader.

1. Pre-Processor

Role: The pre-processor handles initial code transformations before the actual compilation begins. It processes directives (e.g., #include, #define) and performs tasks like macro substitution and file inclusion.

Example: In C/C++, if you have #include <stdio.h>, the pre-processor includes the contents of stdio.h into your source file.

2. Compiler

Role: The compiler translates the pre-processed source code into intermediate code or assembly code. It performs syntax and semantic analysis to ensure the code adheres to the language's rules and generates an intermediate representation (IR) of the program.

Example: The compiler translates C code into assembly code, such as converting int x = 5; into assembly instructions.

3. Assembler

Role: The assembler converts the assembly code generated by the compiler into machine code (binary code). This machine code is a low-level code that the computer’s processor can execute directly.

Example: The assembler takes the assembly instructions and generates a corresponding object file containing machine code.

4. Linker/Loader

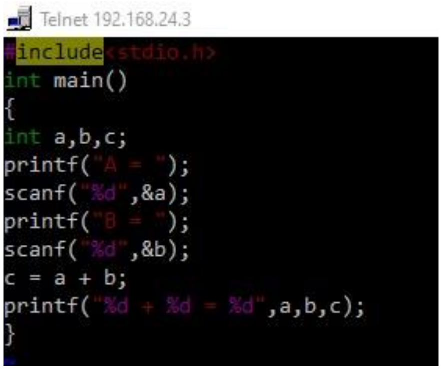
Role: The linker combines object files generated by the assembler into a single executable file, resolving external references between them. The loader then loads the executable into memory and prepares it for execution.

Example: If your program consists of multiple source files, the linker combines the object files from each source file and resolves references like function calls. The loader places the executable into memory and sets it up for execution.

- **Understand modules of the compilation process with the help of a program. (Pre-processor,**

  Compiler, Assembler, Linker/Loader)

Execution steps to see the output of each module:



Telnet 192.168.24.3

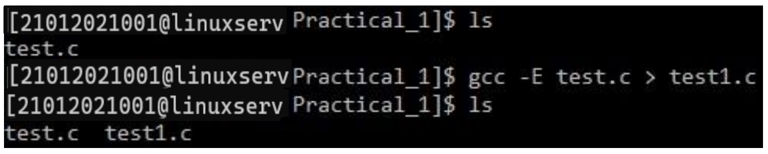
```
#include<stdio.h>
int main()
{
int a,b,c;
printf("A = ");
scanf("%d",&a);
printf("B = ");
scanf("%d",&b);
c = a + b;
printf("%d + %d = %d",a,b,c);
}
```

1. Preprocessor

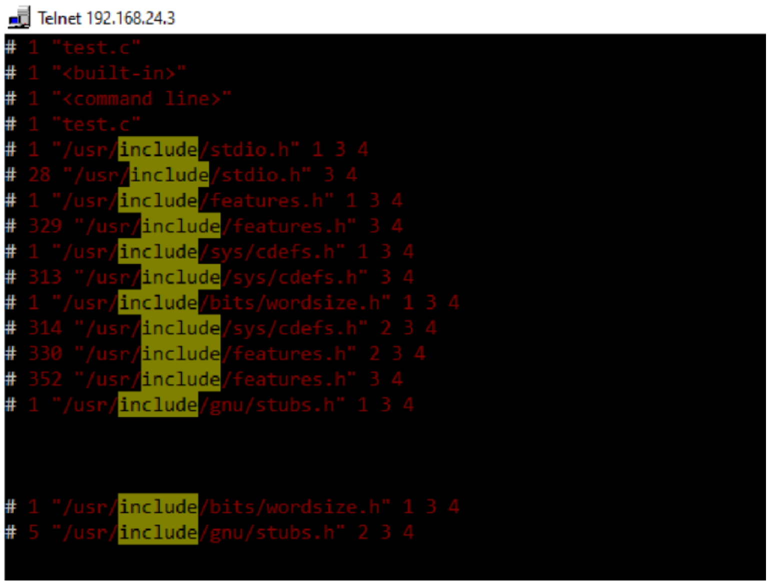
gcc –E test.c > test1.c E option is used to perform

preprocessor it performs file inclusion and macro

expansion



```
[21012021001@linuxserv Practical_1]$ ls
test.c
[21012021001@linuxserv Practical_1]$ gcc -E test.c > test1.c
[21012021001@linuxserv Practical_1]$ ls
test.c  test1.c
```

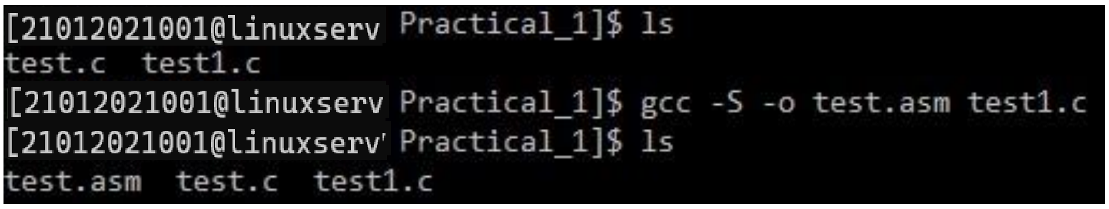


Telnet 192.168.24.3

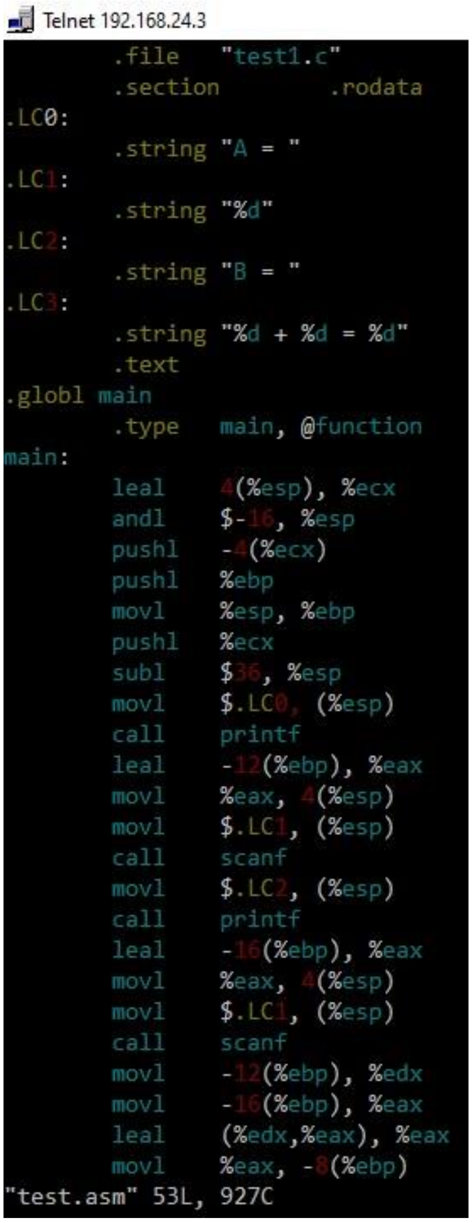
```
# 1 "test.c"
# 1 "<built-in>"
# 1 "<command line>"
# 1 "test.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 28 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/features.h" 1 3 4
# 329 "/usr/include/features.h" 3 4
# 1 "/usr/include/sys/cdefs.h" 1 3 4
# 313 "/usr/include/sys/cdefs.h" 3 4
# 1 "/usr/include/bits/wordsize.h" 1 3 4
# 314 "/usr/include/sys/cdefs.h" 2 3 4
# 330 "/usr/include/features.h" 2 3 4
# 352 "/usr/include/features.h" 3 4
# 1 "/usr/include/gnu/stubs.h" 1 3 4



# 1 "/usr/include/bits/wordsize.h" 1 3 4
# 5 "/usr/include/gnu/stubs.h" 2 3 4
```

2. Compiler

gcc –S test1.c S option generate assembly code file(test1.s) OR gcc –S –o test.asm test1.c – this generates assembly code file with the name given by the user.



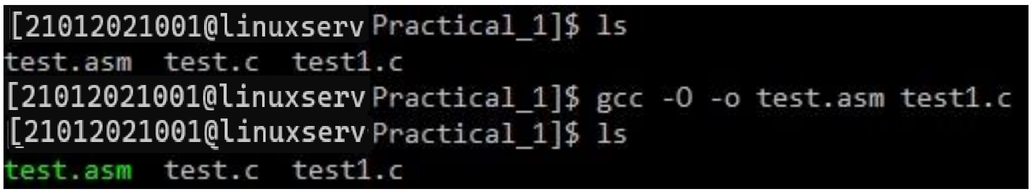
```
[21012021001@linuxserv Practical_1]$ ls
test.c  test1.c
[21012021001@linuxserv Practical_1]$ gcc -S -o test.asm test1.c
[21012021001@linuxserv' Practical_1]$ ls
test.asm  test.c  test1.c
```



```
        .file   "test1.c"
        .section        .rodata
.LC0:
        .string "A = "
.LC1:
        .string "%d"
.LC2:
        .string "B = "
.LC3:
        .string "%d + %d = %d"
        .text
.globl main
        .type   main, @function
main:
        leal    4(%esp), %ecx
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ecx
        subl    $36, %esp
        movl    $.LC0, (%esp)
        call    printf
        leal    -12(%ebp), %eax
        movl    %eax, 4(%esp)
        movl    $.LC1, (%esp)
        call    scanf
        movl    $.LC2, (%esp)
        call    printf
        leal    -16(%ebp), %eax
        movl    %eax, 4(%esp)
        movl    $.LC1, (%esp)
        call    scanf
        movl    -12(%ebp), %edx
        movl    -16(%ebp), %eax
        leal    (%edx,%eax), %eax
        movl    %eax, -8(%ebp)
"test.asm" 53L, 927C
```

3. To optimize the assembly code

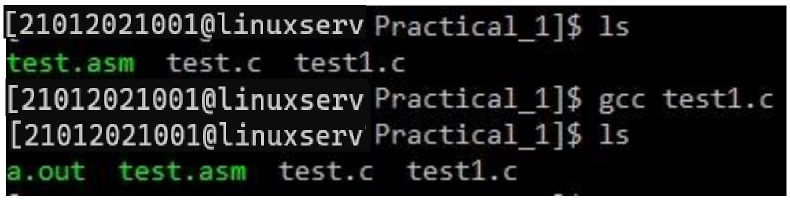
code gcc –S –O –o test.asm test1.c



```
[21012021001@linuxserv Practical_1]$ ls
test.asm  test.c  test1.c
[21012021001@linuxserv Practical_1]$ gcc -O -o test.asm test1.c
[21012021001@linuxserv Practical_1]$ ls
test.asm  test.c  test1.c
```

4. To compile the code

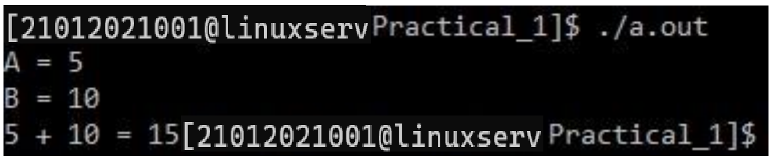
gcc –C test1.s --- generate executable code



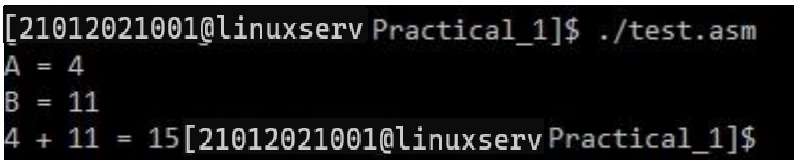
```
[21012021001@linuxserv Practical_1]$ ls
test.asm  test.c  test1.c
[21012021001@linuxserv Practical_1]$ gcc test1.c
[21012021001@linuxserv Practical_1]$ ls
a.out  test.asm  test.c  test1.c
```

5. Run the code

./a.out



```
[21012021001@linuxserv Practical_1]$ ./a.out
A = 5
B = 10
5 + 10 = 15[21012021001@linuxserv Practical_1]$
```



```
[21012021001@linuxserv Practical_1]$ ./test.asm
A = 4
B = 11
4 + 11 = 15[21012021001@linuxserv Practical_1]$
```