CSC 3210 Computer Organization and programming

LAB 3

“for” loops with an incrementing or decrementing index

INSTRUCTIONS FOR LAB 3

Tasks to be done in this lab:

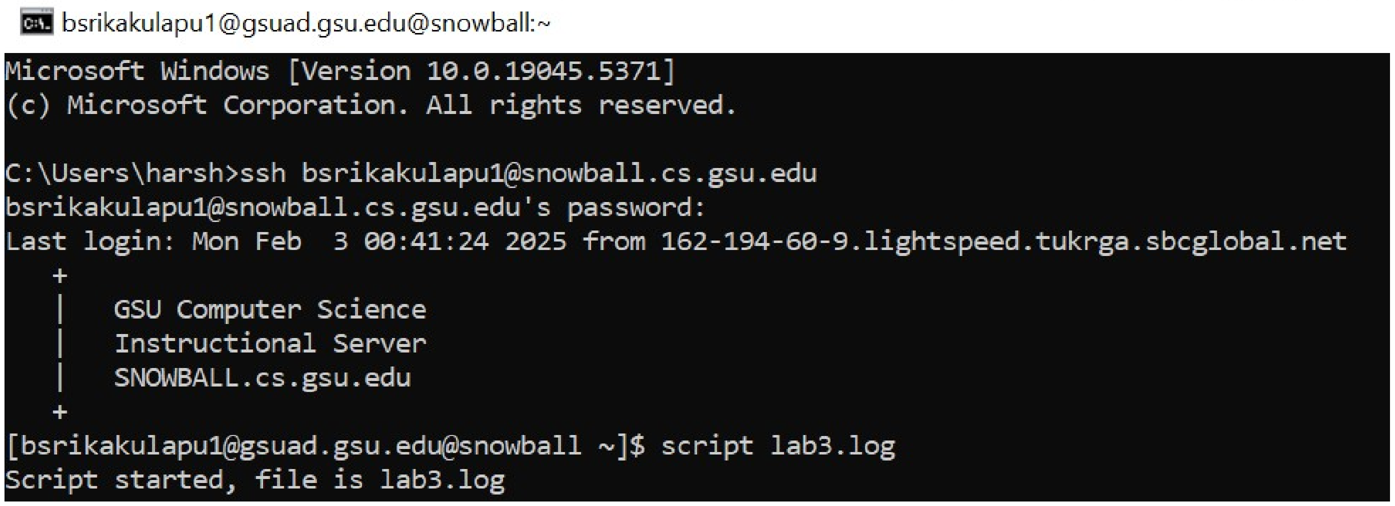
- Learn how a "for" loop works
- Produce "for" loop assembly language code with incrementing test expression
- Change the test expression to decrementing test expression
- Produce "for" loop assembly language code with decrementing test expression
- Compile an assembly language file to an executable
- Run the executable program
- Do an exercise to print even or odd numbers from 1 to 20.

ASSIGNMENT FOR LAB 3

- By now, you all are familiar with logging into SNOWBALL account, compiling a C program to get an assembly language program. In the next exercise we will be looking at how a for loop would look in assembly language.
- There are questions to answer in this lab. When turning this in (as with all other labs), you should submit the cleaned-up .txt file, as well as the lab report (i.e. a .pdf file). We will grade the lab based on the report, and look at the .txt file if the report is not clear or if there is a problem.

STEP-1

- Log in to SNOWBALL server
- Use the command “**ssh *your_account*@snowball.cs.gsu.edu**”
- Create a log file with the "**script**" command.
- Use the command “**script lab3.log**”
- You can use “**ls**” command to view the files that are already available in your server



bsrikakulapu1@gsuad.gsu.edu@snowball:~

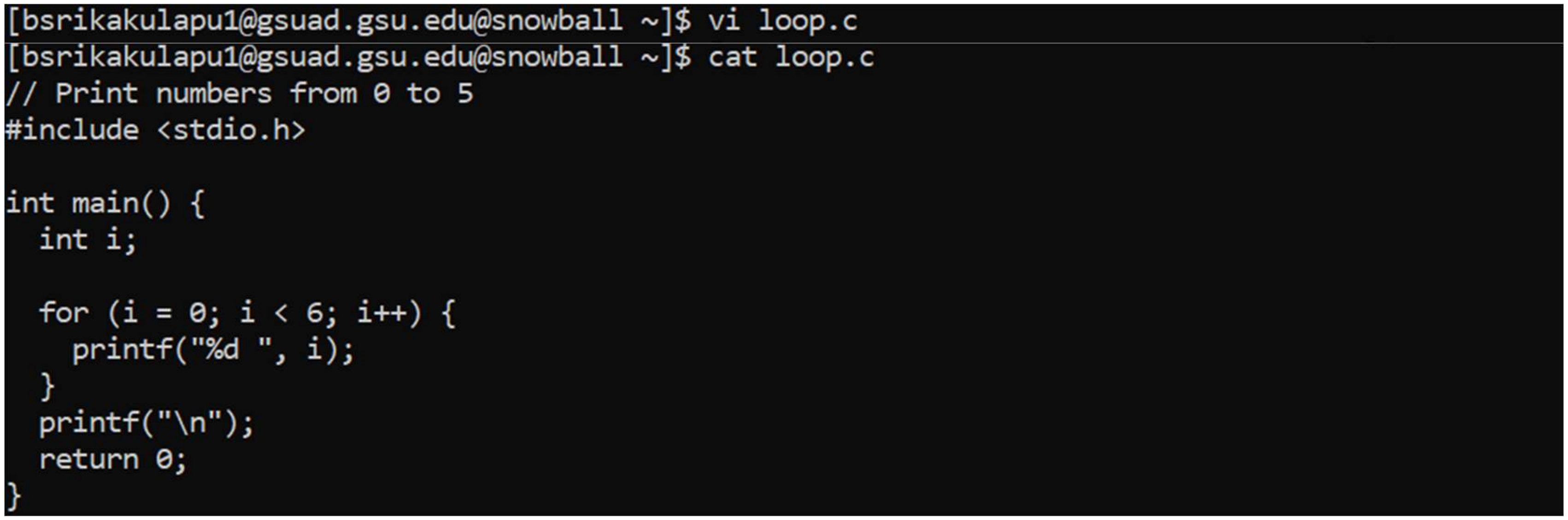
```
Microsoft Windows [Version 10.0.19045.5371]
(c) Microsoft Corporation. All rights reserved.

C:\Users\harsh>ssh bsrikakulapu1@snowball.cs.gsu.edu
bsrikakulapu1@snowball.cs.gsu.edu's password:
Last login: Mon Feb  3 00:41:24 2025 from 162-194-60-9.lightspeed.tukrga.sbcglobal.net
   +
   |   GSU Computer Science
   |   Instructional Server
   |   SNOWBALL.cs.gsu.edu
   +
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ script lab3.log
Script started, file is lab3.log
```

STEP-2

- Next, get the "loop.c" file. You could copy and paste it from this assignment, or you could even type it yourself.
- We are looking at how a "for" loop works here. In the "for" loop, we have initialization statement, test expression and update statement.
- Initialization statement is executed only once. Then, test expression is evaluated. If the test expression is evaluated to false then "for" loop is terminated.
- However if the test expression is evaluated to true then, statements inside the "for" loop body are executed and the update expression is updated. Again the test expression is evaluated to continue for the next iteration. This process goes on until the test expression is false.
- When the test expression is false, the loop terminates.
- If you have not started the log, start it now.

STEP-3

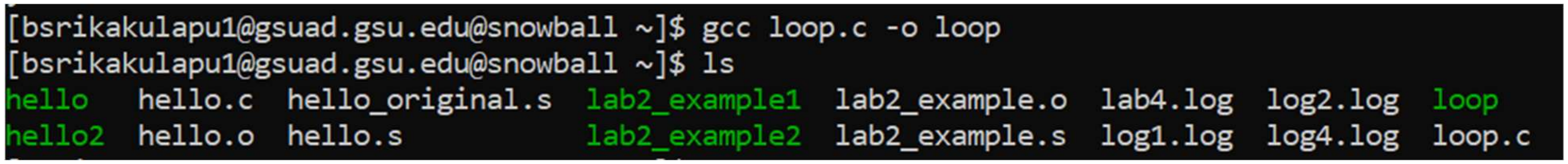
- Use “**vi loop.c**” to open the file and copy the code.
- Use “**cat loop.c**” to view the content of the file



```
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ vi loop.c
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ cat loop.c
// Print numbers from 0 to 5
#include <stdio.h>

int main() {
  int i;

  for (i = 0; i < 6; i++) {
    printf("%d ", i);
  }
  printf("\n");
  return 0;
}
```

STEP-4

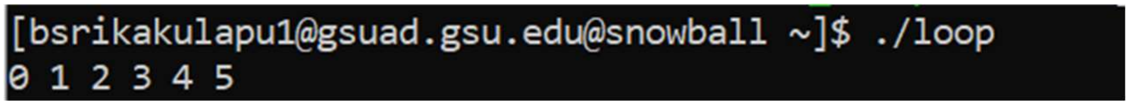
- In this step, we compile the "loop.c" program using the command "**gcc loop.c -o loop**". "gcc" is the GNU C compiler, and "-o loop" specifies that we want the output to be a new file called "loop".
- Be careful with this step; if you were to type "-o loop.c", it would over-write the C file, and you would have to start over.
- use "**ls**" command to get a list of files. Notice how the output shows the new file "**loop**".



```
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ gcc loop.c -o loop
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ ls
hello    hello.c  hello_original.s  lab2_example1  lab2_example.o  lab4.log  log2.log  loop
hello2   hello.o  hello.s           lab2_example2  lab2_example.s  log1.log  log4.log  loop.c
```

STEP-5

- The file "**loop**" is executable, run it using the command “./loop”. The "**./**" specifies where to find the file, and "**.**" is short-hand for the current directory.

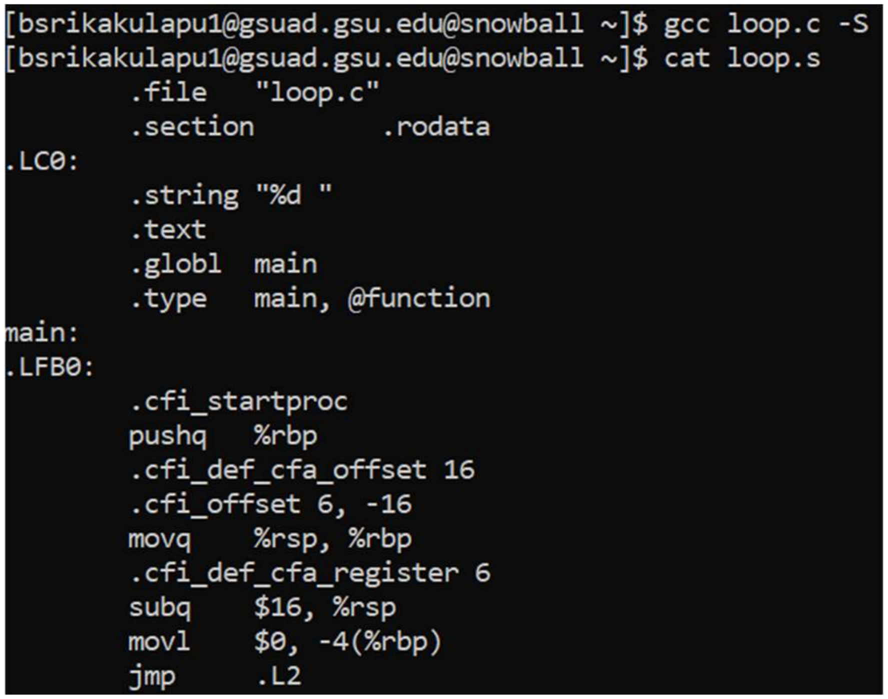


```
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ ./loop
0 1 2 3 4 5
```

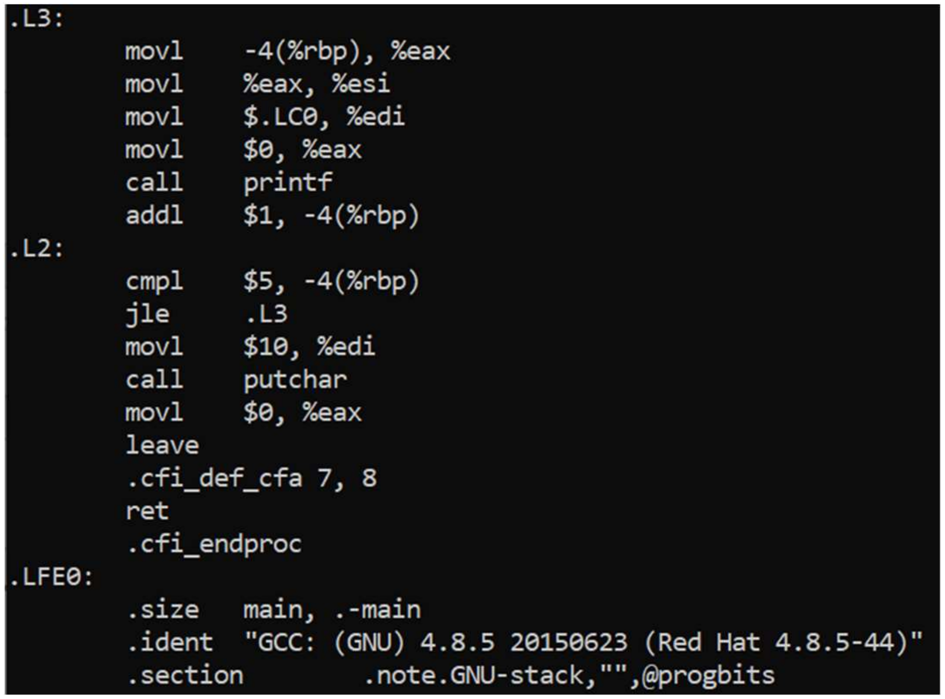
- The program ran as expected, and gave us the line of output.
- Here, in the program "i" is the initialization variable assigned with 0.
- As the test expression is i<6, the "for" loop executed from 0 to 5, incrementing i value at every iteration.
- We gave the update statement as i++, which means i=i+1.
- At i=6, i<6 expression failed as 6 is not less than 6, so "for" loop terminated.
- As we see, the output is 0 1 2 3 4 5.
- So far, we have obtained a C program, showed its contents, compiled it, and ran it.

STEP-6

- Now, we will compile it using the command "**gcc loop.c -S**". Notice the "-S" (and yes, that is a capital S). This generates a new file, "loop.s".
- To view the contents use "**cat**" command like before.



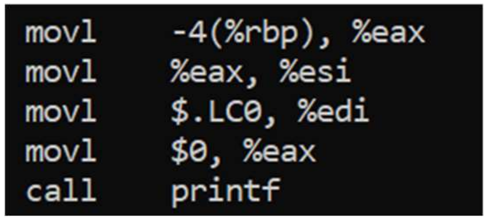
```
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ gcc loop.c -S
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ cat loop.s
        .file   "loop.c"
        .section        .rodata
.LC0:
        .string "%d "
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $0, -4(%rbp)
        jmp     .L2
```



```
.L3:
        movl    -4(%rbp), %eax
        movl    %eax, %esi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        addl    $1, -4(%rbp)
.L2:
        cmpl    $5, -4(%rbp)
        jle     .L3
        movl    $10, %edi
        call    putchar
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (GNU) 4.8.5 20150623 (Red Hat 4.8.5-44)"
        .section        .note.GNU-stack,"",@progbits
```

STEP-7

- At this point, we have an assembly-language program for the "for" loop.
- You may wonder what this code does:

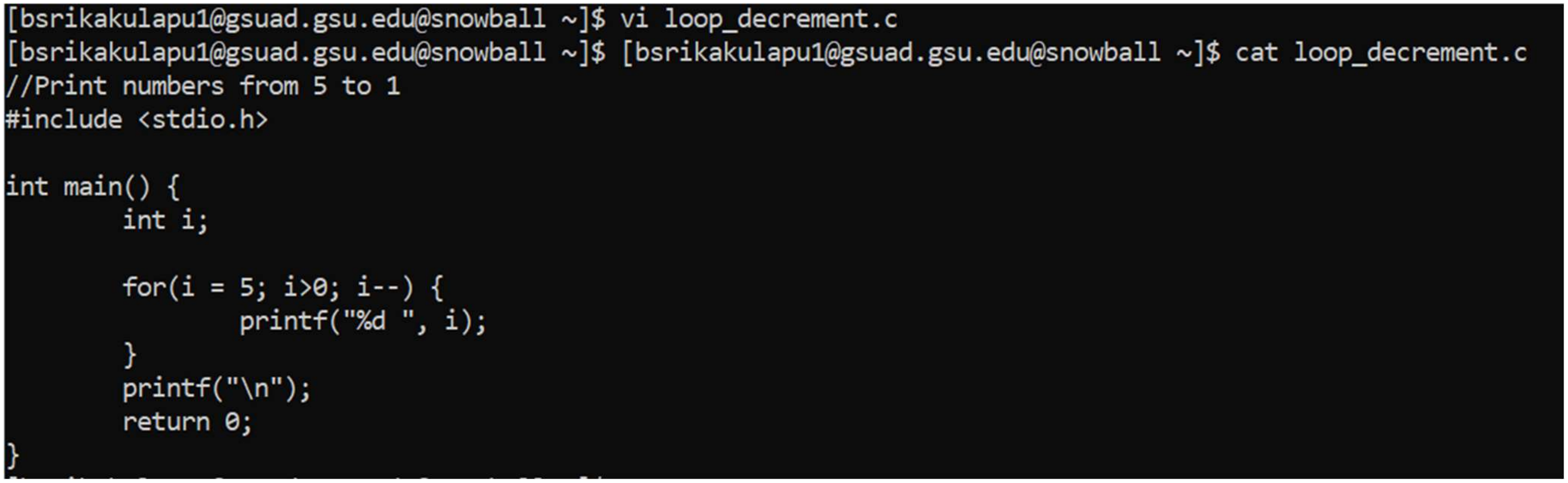


```
movl    -4(%rbp), %eax
movl    %eax, %esi
movl    $.LC0, %edi
movl    $0, %eax
call    printf
```

- Effectively, it prints out the number stored in a local variable.
- "**movl -4(%rbp), %eax**" copies the local variable's value to EAX.
- "**movl %eax, %esi**" copies the value from EAX to ESI.
- Why not copy it directly? It looks like you can.
- "**movl $.LC0, %edi**" copies the label's memory location .LC0 to EDI.
- "**movl $0, %eax**" sets EAX to 0.
- "**call printf**" then calls the function printf to output the value that ESI has, with the format specified by EDI.

STEP-8

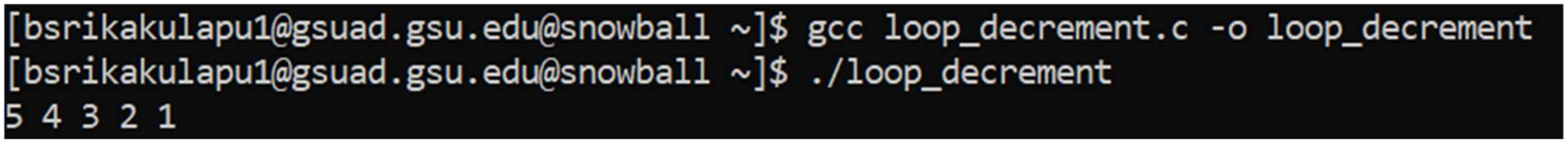
- As an exercise, we can try changing the test expression to see decrementing expression which is i--, which is evaluated as i=i-1. Create a loop_decrement.c file.
- You can add program contents using the command "**vi loop_decrement.c**"
- Use command "**cat loop_decrement.c**" to view the contents of the file.



```
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ vi loop_decrement.c
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ [bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ cat loop_decrement.c
//Print numbers from 5 to 1
#include <stdio.h>

int main() {
        int i;

        for(i = 5; i>0; i--) {
                printf("%d ", i);
        }
        printf("\n");
        return 0;
}
```

STEP-9

- Now, we follow the similar steps as above to compile the file using "gcc" and executing it.
- Use command “**gcc loop_decrement.c –o loop_decrement**” to convert the file into an executable.
- Use command “**./loop_decrement**” to run the executable file.

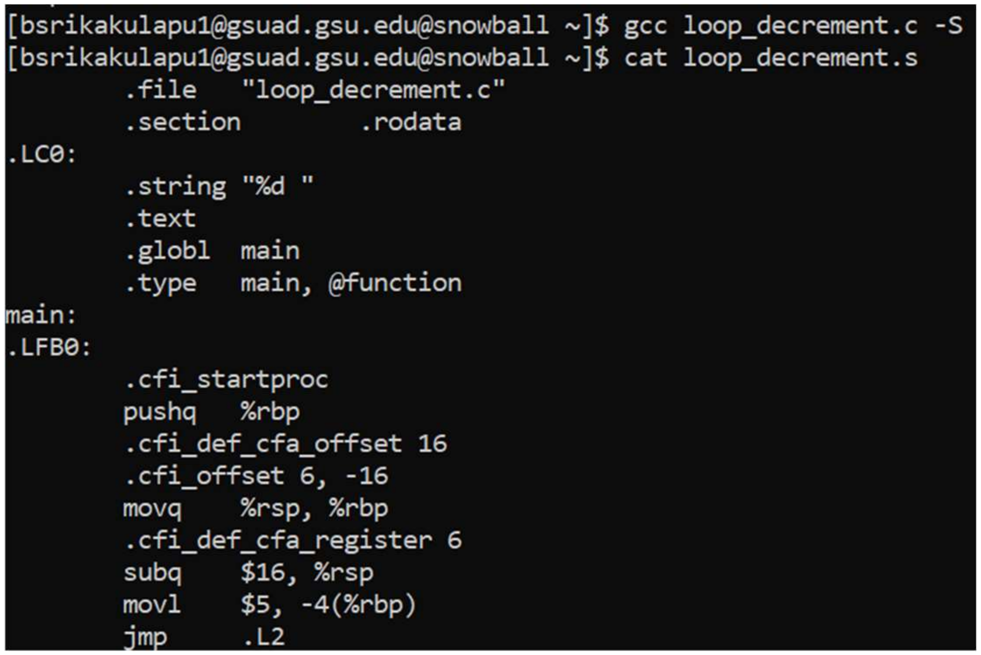


```
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ gcc loop_decrement.c -o loop_decrement
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ ./loop_decrement
5 4 3 2 1
```

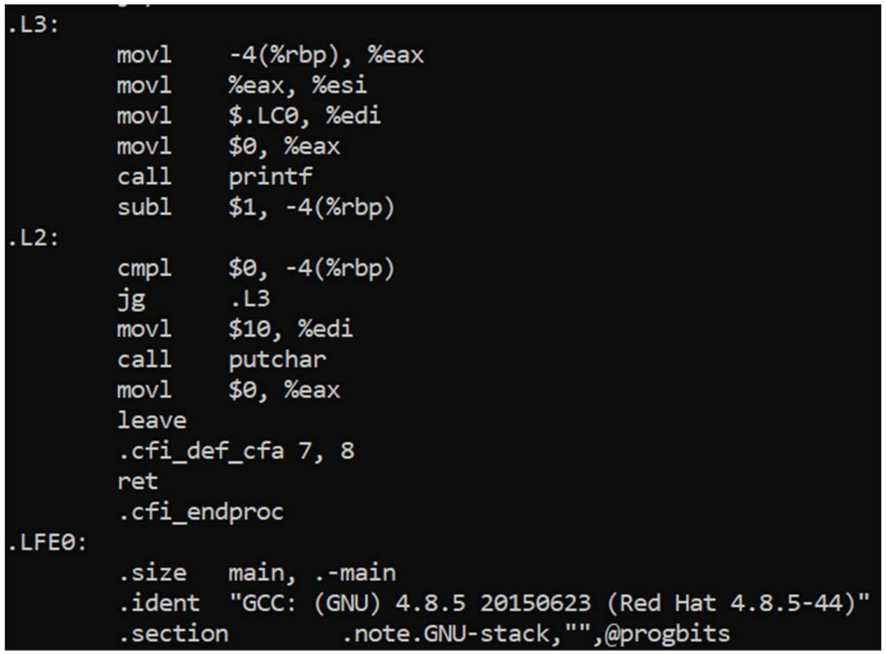
- As you can see, here we are initializing i with 5 and then decrementing its value with every iteration.
- Here, the test expression is i>0 so the loop will continue from 5 to 1 until i=1.
- As the test expression is i>0 (means 1>0) after the final iteration, the test expression is false so the loop terminates.
- Thus, the output is 5 4 3 2 1.

STEP-10

- Use command "**gcc loop_decrement.c -S**" to compile the file.
- Use command "**cat loop_decrement.s**" to view the contents of the file.



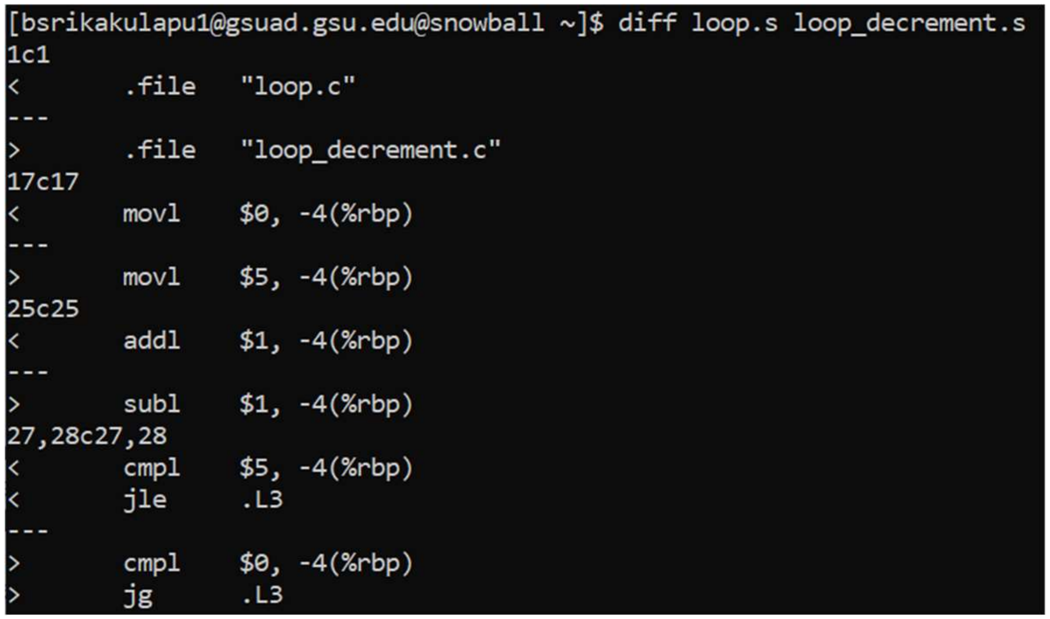
```
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ gcc loop_decrement.c -S
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ cat loop_decrement.s
        .file   "loop_decrement.c"
        .section        .rodata
.LC0:
        .string "%d "
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $5, -4(%rbp)
        jmp     .L2
```



```
.L3:
        movl    -4(%rbp), %eax
        movl    %eax, %esi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        subl    $1, -4(%rbp)
.L2:
        cmpl    $0, -4(%rbp)
        jg      .L3
        movl    $10, %edi
        call    putchar
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (GNU) 4.8.5 20150623 (Red Hat 4.8.5-44)"
        .section        .note.GNU-stack,"",@progbits
```

STEP-11

- We have assembly language program of "for" loop with decrementing test expression.
- Notice the difference in assembly language code for incrementing and decrementing test expressions.
- You can see this plainly using the command "**diff loop.s loop_decrement.s**".



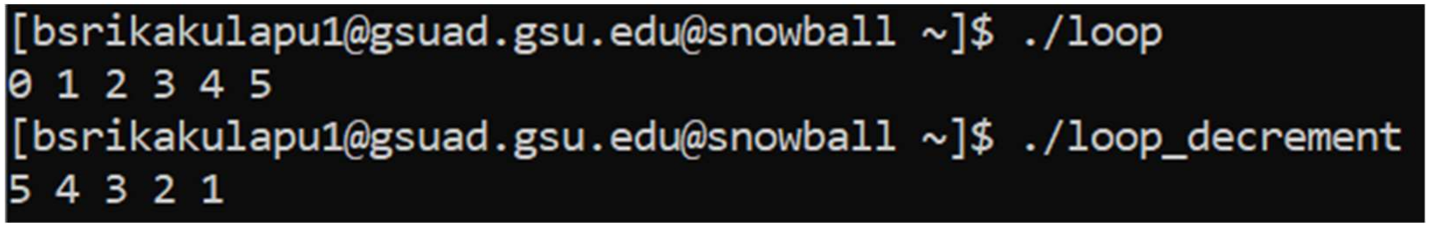
```
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ diff loop.s loop_decrement.s
1c1
<        .file   "loop.c"
---
>        .file   "loop_decrement.c"
17c17
<        movl    $0, -4(%rbp)
---
>        movl    $5, -4(%rbp)
25c25
<        addl    $1, -4(%rbp)
---
>        subl    $1, -4(%rbp)
27,28c27,28
<        cmpl    $5, -4(%rbp)
<        jle     .L3
---
>        cmpl    $0, -4(%rbp)
>        jg      .L3
```

STEP-12

- Notice what the "diff" command outputs. The line "1c1" means to change line 1.
- The line after has < meaning that it is how it looks in the first file. Then "---" separates the entries. The line after that starts with > meaning that it is how it looks in the second file.
- Taking the first 4 lines together, we can say that the text in file 1 is "loop" while the text in file 2 has "loop_decrement" instead, for that line.
- **Looking at the differences, explain the others. You can write your explanations as a text file.**
- You should recall that "movl" is a "long" move, that moves a 32 bit value to a 64 bit register, with zeros for the high bits.
- Similarly, "addl", "subl", and "cmpl" are the "long" versions of "add", "sub" (subtract) and "cmp" (compare).
- The dollar sign ("$") is used to indicate an immediate value, e.g. $7 simply means the number 7.
- ".L3" is a label, specifiing a point in the code to go to.
- "jle" means "jump less than or equal to" while "jg" is for "jump if greater".

STEP-13

- Now, run the original "loop" program.
- Then run the "loop_decrement" program.
- View the outputs



```
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ ./loop
0 1 2 3 4 5
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ ./loop_decrement
5 4 3 2 1
```

STEP-14

- As an exercise, create assembly language code for the tasks below.
- You are welcome to copy and change the .s file directly, especially for the first one. For the second and third, you may want to try changing the C file to get the assembly language code.

1. Write a program (lab3_1to20.s) to print numbers from 1 to 20.
2. Write a program (lab3_1to20even.s) to print the even numbers from 1 to 20.
3. Write a program (lab3_1to20odd.s) to print the odd numbers from 1 to 20.

- This requires changing the logic in the "for" loop body to print only even/odd numbers.
- To check if a number is even we can use expression <number>/2.
- If expression evaluates to 0 then it is an even number, otherwise it must be an odd number.
- Use in-line comments within the programs (starting with the # character) to say what lines you changed.
- Show the contents of the .s programs from above (use "cat").
- Also, compile them, and run them to show that they work.

STEP-15

ANSWER THESE QUESTIONS

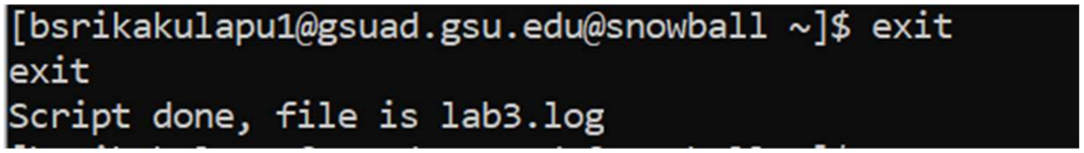
1) **Questions:** What did you change in the program to print numbers from 1 to 20? How difficult was it to figure out what to change? How difficult was it to make the change once you knew what to do, especially in relation to figuring out what to change?

2) **Questions:** If the loop variable is an integer, and you divide it by 2, is the result also an integer? Why or why not?

3) **Question:** Once you had it working for odd numbers, how easy was it to print the even numbers? (Use relative terms like "much more difficult", "somewhat more difficult", "about the same", "somewhat easier, and "much easier". Then state why.

STEP-16

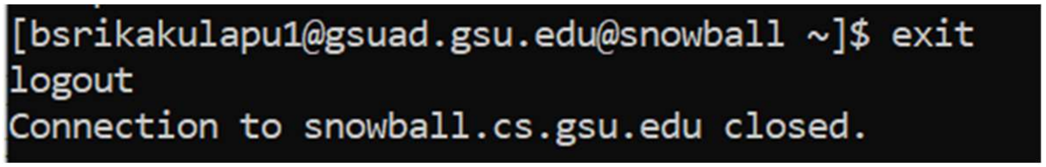
- Finally, we exit from the script command. This informs the "script" utility that we are done.



```
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ exit
exit
Script done, file is lab3.log
```

- You might want to next use "cat" on the "lab3.log" file, copy the output, and paste it into a text file called "lab3.txt". This should get rid of any extraneous (control) characters, though you should look it over to make sure. There are other ways to do this: one would be to make your own filtering program, that only allows characters with ASCII values 32 to 126, along with the newline (linefeed) character (10). Tab, character 9, might be useful too, maybe mapped to 4 spaces.

- Then we exit (log out) from the server. You can use "sftp" to get your log file from the server.



```
[bsrikakulapu1@gsuad.gsu.edu@snowball ~]$ exit
logout
Connection to snowball.cs.gsu.edu closed.
```

IMPORTANT NOTE:

When turning this in (as with all other labs), you should submit the cleaned-up .txt file, as well as the lab report (i.e. a .pdf file). We will grade the lab based on the report, and look at the .txt file if the report is not clear or if there is a problem.