

COMPUTER ORGANIZATION AND ARCHITECTURE

Assignment 1

by

Tajamul Ashraf (2018BITE031)
Fuzayil-Bin-Afzal Mir(2018BITE057)

October 10, 2020

Contents

1	ARM cross compiler	3
1.1	Introduction	3
2	Ubuntu	3
2.1	How to install ubuntu	3
2.2	How to Install arm-linux-gnueabi cross compiler	4
3	Qemu tool	4
3.1	How to install Qemu tool on your system.	4
3.2	How to run the generated executable using qemu tool.	4
4	Program Hello World	5
4.1	How to generate the assembly code for the hello-world.c program using the crosscompiler.	6
4.2	How to generate the executable of the program using the cross-compiler.	7
4.3	Program Tracing	8
5	Program Add	9
5.1	How to generate the assembly code for the add.c program using the crosscompiler.	10
5.2	How to generate the executable of the program using the cross-compiler.	11
5.3	Program Tracing	12
6	Program Factorial	13
6.1	How to generate the assembly code for the Factorial.c program using the crosscompiler.	14
6.2	How to generate the executable of the program using the cross-compiler.	16
6.3	Program Tracing	17

1 ARM cross compiler

1.1 Introduction

ARM cross compiler Cross-compilation is the process of compiling code for one computer system (also known as the target) on a different system, called the host. A cross compiler is a type of compiler, that generates machine code targeted to run on a system different than the one generating it. Like we used arm cross compiler to convert from X86 processor to ARM. The Process of creating executable code for different machines is called retargeting. The cross compiler is also called as retargetable compiler. We have used GNU GCC cross compiler. A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is running. For example, a compiler that runs on a X86 processor but generates code that runs on ARM Processor is a cross compiler.

A cross compiler is necessary to compile code for multiple platforms from one development host. Direct compilation on the target platform might be infeasible, for example on X86 using .o file, because those systems contain no operating system. In generalization, one computer runs multiple operating systems and a cross compiler could generate an executable for each of them from one main source.

Cross compilers are distinct from source-to-source compilers. A cross compiler is for cross-platform software development of machine code, while a source-to-source compiler translates from one programming language to another in text code. Both are programming tools. The GNU Arm Embedded Toolchain is a ready-to-use, open-source suite of tools for C (As in our case), C++ and assembly programming. The GNU Arm Embedded Toolchain targets the 32-bit Arm Cortex-A, Arm Cortex-M, and Arm Cortex-R processor families.

2 Ubuntu

2.1 How to install ubuntu

1. Open the Ubuntu website. Go to

You can download the Ubuntu disk image (also known as an ISO file) here.

2. Install VirtualBox.

3. Select your Ubuntu ISO. Go to the folder into which the Ubuntu ISO file downloaded (e.g., Desktop), then click the ISO file to select it.

4. Check the "Erase disk and install Ubuntu" box.

5. Personalize the ubuntu.
6. Restart the virtual machine. Once you see the Restart Now button, do the following: click the Exit button in the upper-right corner of the window (Windows) or the upper-left corner of the window (Mac), check the "Power off the machine" box, click OK.
7. You are ready to use ubuntu.

2.2 How to Install arm-linux-gnueabi cross compiler

1. Visit the link : <https://www.acmesystems.it/arm9toolchain>
2. As I am using an Arietta, open the terminal of ubuntu (by ctrl+alt+t) write:
`sudo apt - get install gcc - arm - linux - gnueabi g++ - arm - linux - gnueabi`
3. After writing the above code, package will start downloading.
4. You are ready to use ARM cross compiler

3 Qemu tool

3.1 How to install Qemu tool on your system.

1. Open the Terminal in Ubuntu
2. Run update command to update package repositories and get latest package information.
`sudo apt - get update - y`
3. Run the install command with -y flag to quickly install the packages and dependencies.
`sudo apt - get install - y qemu - user - static`
4. Check the system logs to confirm that there are no related errors.

3.2 How to run the generated executable using qemu tool.

1. First form the .c file.
2. Create a .s file.
3. We have to run the binary with Qemu
write : `qemu - arm - L/usr/arm - linux - gnueabi/programname.o`

4 Program Hello World

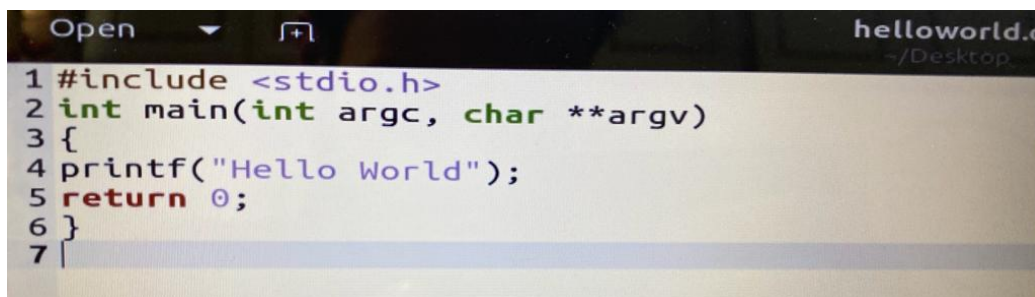
1. Open the Terminal.
2. Write the command: touch helloworld.c

A terminal window with a dark background. The prompt is 'Z8@tajamul@tajamul-VirtualBox:~/Desktop\$'. The command 'cat helloworld.c' has been entered. The output shows the C code for 'helloworld.c': '#include <stdio.h>', 'int main(int argc, char **argv)', '{', 'printf("Hello World");', 'return 0;', and '}'.

```
Z8@tajamul@tajamul-VirtualBox:~/Desktop$ cat helloworld.c
#include <stdio.h>
int main(int argc, char **argv)
{
printf("Hello World");
return 0;
}
tajamul@tajamul-VirtualBox:~/Desktop$
```

Figure 1: 4.1

3. Open the file and write the code for helloworld as shown in fig 4.1 and 4.2

A code editor window with a light background. The title bar shows 'helloworld.c' and the path '~/Desktop'. The code is written in a monospaced font with syntax highlighting: line 1 is '#include <stdio.h>', line 2 is 'int main(int argc, char **argv)', line 3 is '{', line 4 is 'printf("Hello World");', line 5 is 'return 0;', line 6 is '}', and line 7 is '|'.

```
1 #include <stdio.h>
2 int main(int argc, char **argv)
3 {
4 printf("Hello World");
5 return 0;
6 }
7 |
```

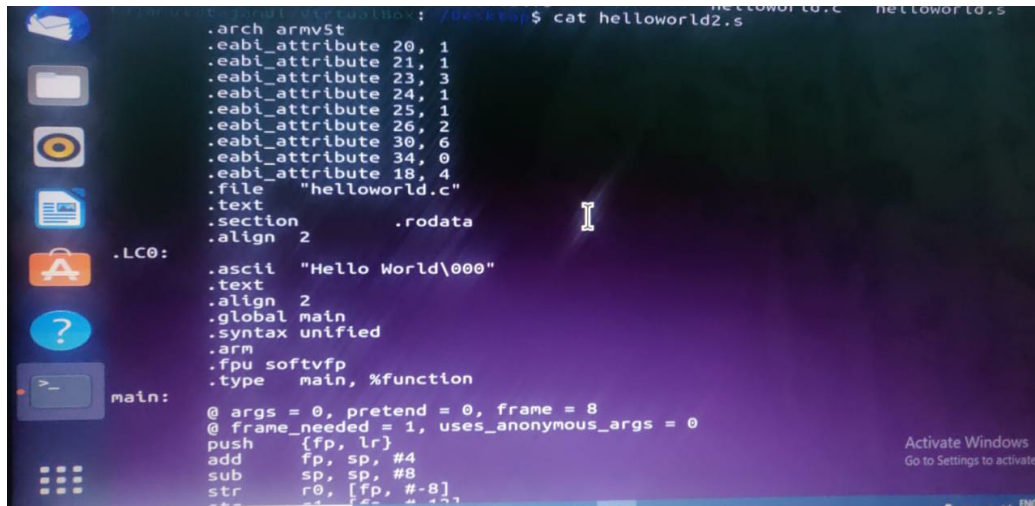
Figure 2: 4.2

4. Write the command: gcc helloworld.c -o helloworld
5. Your helloworld.c file is ready

4.1 How to generate the assembly code for the helloworld.c program using the crosscompiler.

Now for the assembly code using cross compiler

1. Write the command `gcc -S helloworld.c`



```
.arch armv5t
.eabi_attribute 20, 1
.eabi_attribute 21, 1
.eabi_attribute 23, 3
.eabi_attribute 24, 1
.eabi_attribute 25, 1
.eabi_attribute 26, 2
.eabi_attribute 30, 6
.eabi_attribute 34, 0
.eabi_attribute 18, 4
.file "helloworld.c"
.text
.section .rodata
.align 2
.LC0:
.ascii "Hello World\000"
.text
.align 2
.global main
.syntax unified
.arm
.fpu softvfp
.type main, %function

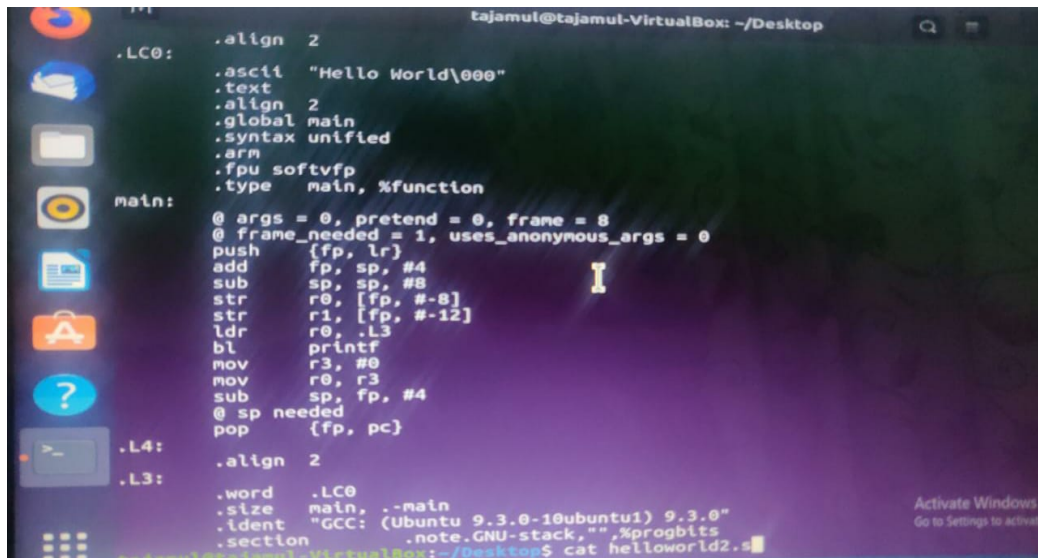
main:
@ args = 0, pretend = 0, frame = 8
@ frame_needed = 1, uses_anonymous_args = 0
push {fp, lr}
add fp, sp, #4
sub sp, sp, #8
str r0, [fp, #-8]
```

Figure 3: 4.3

Ls

cat helloworld.s

2. Your helloworld.s file (assembly code) is ready



```
tajamul@tajamul-VirtualBox: ~/Desktop
.LC0:
    .align 2
    .ascii "Hello World\000"
    .text
    .align 2
    .global main
    .syntax unified
    .arm
    .fpu softvfp
    .type main, %function

main:
    @ args = 0, pretend = 0, frame = 8
    @ frame_needed = 1, uses_anonymous_args = 0
    push    {fp, lr}
    add     fp, sp, #4
    sub     sp, sp, #8
    str     r0, [fp, #-8]
    str     r1, [fp, #-12]
    ldr     r0, .L3
    bl      printf
    mov     r3, #0
    mov     r0, r3
    sub     sp, fp, #4
    @ sp needed
    pop     {fp, pc}

.L4:
    .align 2
.L3:
    .word   .LC0
    .size   main, .-main
    .ident  "GCC: (Ubuntu 9.3.0-10ubuntu1) 9.3.0"
    .section .note.GNU-stack,"",%progbits

tajamul@tajamul-VirtualBox:~/Desktop$ cat helloworld2.s
```

Figure 4: 4.4

4.2 How to generate the executable of the program using the cross-compiler.

In the Terminal use the following commands:

```
arm-linux-gnueabi-gcc helloworld.c -o helloworld.o
```



```
Try: sudo apt install <deb name>
tajamul@tajamul-VirtualBox:~/Desktop$ arm-linux-gnueabi-gcc helloworld.c -o helloworld.o
tajamul@tajamul-VirtualBox:~/Desktop$ qemu-arm -L /usr/arm-linux-gnueabi/ helloworld.o
Hello Worldtajamul@tajamul-VirtualBox:~/Desktop$
```

Figure 5: 4.5

```
qemu-arm -L /usr/arm-linux-gnueabi/ helloworld.o
```

4.3 Program Tracing

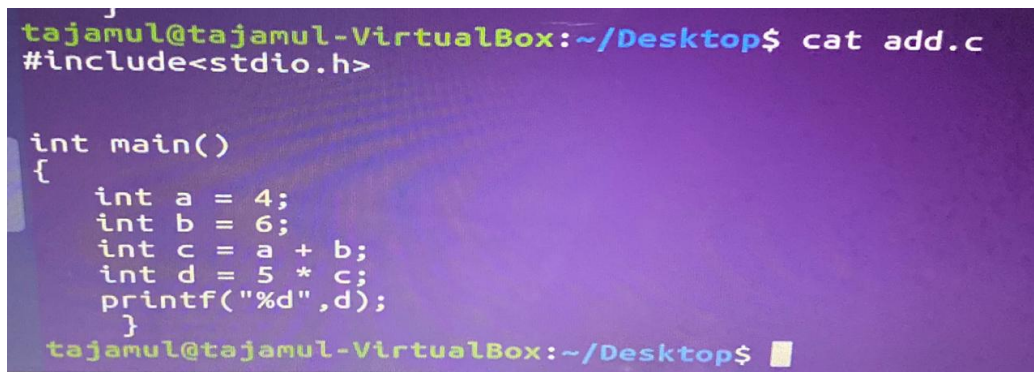
Instruction	DESTIN	ATION	SR	C1	SR	C2
	loc	value	loc	value	loc	value
push{fp,lr}	sp	64	fp	64	lr	
add fp,sp,#4	fp	64	sp	68		
sub sp,sp,#8	sp	56	sp	64	const	8
str r0,[fp,#-8]	addr(48)	60	r0	4		
str r1,[fp,#-12]	addr(64)	58	r1	8		
ldr r0,.L3	r0	L3	address	L3		
bl printf	lr	line31	func	printf		
mov r0,#0	r0	0	const	0		
mov r0,r3	r0	0	r3	r0		
sub sp,fp,#4	sp	44	fp	48	const	4
pop{fp,pc}	fp	46	pc	44		

For each line of the program, explaining each line of assembly code.

- 1.The push instruction at the beginning of this function: Pushes the return address, the value contained in the lr register, onto the stack,the value contained in the fp register, onto the stack. Updates the sp to show that two 32-bit values have been pushed onto the top of the stack.
- 2.This instruction doesn't set fp to 4, but rather to the current value of sp. fp serves as a reference to local information on the stack that remains constant during the execution of F. The old frame pointer, parameters and local variables are then stored at negative offsets from fp.
- 3.The parameters are passed in Registers r0 (a) and r1 (b); this is a convention defined in the Procedure Call Standard for the ARM Architecture.
- 3.Here, 8 Bytes of space are reserved on the stack, so that subsequent stack operations do not modify the local stack frame.
- 4.This copies r0 (Parameter a) into the local stack frame at address fp-8.
- 5.Loads r0 with .L3 contents
- 6.print string and pass parameters into r0, r1, and r2
7. Moves the constant value 0 in r0.
- 8.moves the value from r3 to r0.
- 9.pop the value out of pc

5 Program Add

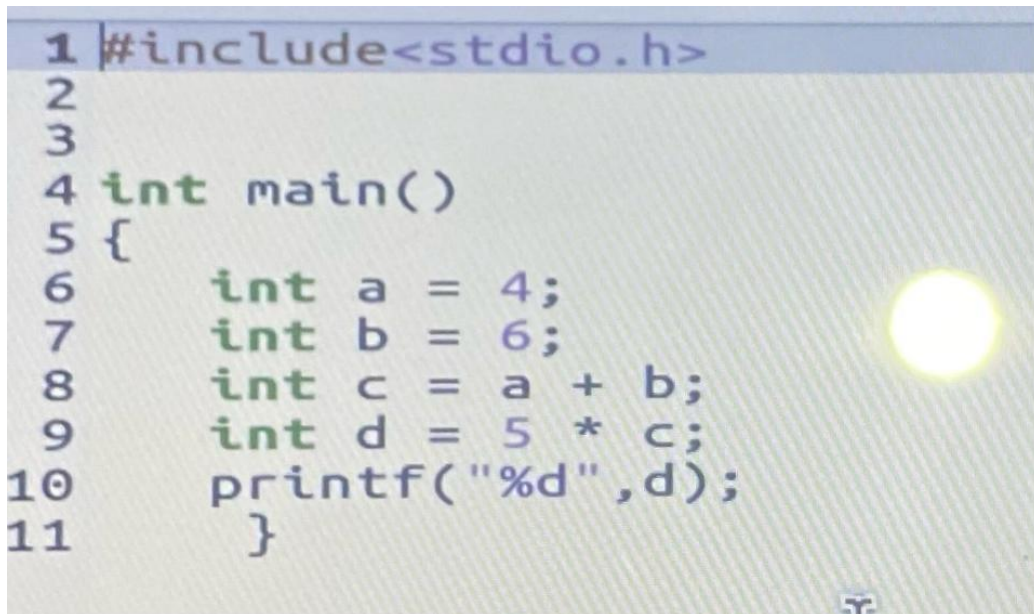
1. Open the Terminal.
2. Write the command: touch add.c

A terminal window with a dark purple background. The prompt is 'tajamul@tajamul-VirtualBox:~/Desktop\$'. The command 'cat add.c' has been entered, and the contents of the file are displayed. The code is a C program that defines two integers, a and b, calculates their sum and a product, and prints the product.

```
tajamul@tajamul-VirtualBox:~/Desktop$ cat add.c
#include<stdio.h>

int main()
{
    int a = 4;
    int b = 6;
    int c = a + b;
    int d = 5 * c;
    printf("%d",d);
}
tajamul@tajamul-VirtualBox:~/Desktop$
```

Figure 6: 5.6

A code editor window with a light blue background. The code is displayed with line numbers on the left. The code is a C program that defines two integers, a and b, calculates their sum and a product, and prints the product.

```
1 #include<stdio.h>
2
3
4 int main()
5 {
6     int a = 4;
7     int b = 6;
8     int c = a + b;
9     int d = 5 * c;
10    printf("%d",d);
11    }
```

Figure 7: 5.7

3. Open the file and write the code for program add as shown in fig 5.6 and 5.7
4. Write the command: gcc add.c -o add

5. Your `add.c` file is ready

5.1 How to generate the assembly code for the add.c program using the crosscompiler.

Now for the assembly code using cross compiler

1. Write the command `gcc -sS add.c`

```

tjajamul@tjajamul-VirtualBox: ~/Desktop$ cat add.s
.file      "add.c"
.text
.section   ,rodata
.LC0:
.string    "%d"
.text
.globl     main
.type      main, @function

main:
.LFB0:
.cfi_startproc
endbrq
pushq     %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq      %rsp, %rbp
.cfi_def_cfa_register 6
subq      $16, %rsp
movl      $4, -16(%rbp)
movl      $6, -12(%rbp)
movl      -16(%rbp), %edx
movl      -12(%rbp), %eax
addl      %edx, %eax
movl      %eax, -8(%rbp)
movl      -8(%rbp), %edx
movl      %edx, %eax
call      $2, %eax
movl      %edx, %eax
addl      %eax, -4(%rbp)
movl      -4(%rbp), %eax
movl      %eax, %esi
leaq      .LC0(%rip), %rdi

```

Figure 8: 5.8

```

tjbmout@tjbmout-VirtualBox: ~$dos2unix
.LC0:
    .align 2
    .ascii "Hello World!\000"
    .text
    .align 2
    .globl main
    .syntax unified
    .arm
    .cpu softfp
    .type main, @function

main:
    @ args = 0, pretend = 0, frame = 0
    @ frame_needed = 1, uses_anonymous_args = 0
    push    {fp, lr}
    add     fp, sp, #4
    sub     sp, sp, #8
    str     r0, [fp, #4]
    str     r1, [fp, #-12]
    ldr     r0, r13
    bl      printf
    mov     r3, #0
    mov     r0, r3
    sub     sp, sp, #4
    @ sp needed, fp, pc
    pop     {fp, pc}

.L4:
    .align 2

.L3:
    .word .LC0
    .ident 0cc1 (Ubuntu 9.3.0-10ubuntu1) 9.3.0
    .section .note.GNU-stack,"",%progbits

```

Figure 9: 5.9

Ls

cat add.s

2. Your add.s file (assembly code) is ready

5.2 How to generate the executable of the program using the cross-compiler.

In the Terminal use the following commands:

```
arm-linux-gnueabi-gcc add.c -o add.o
```

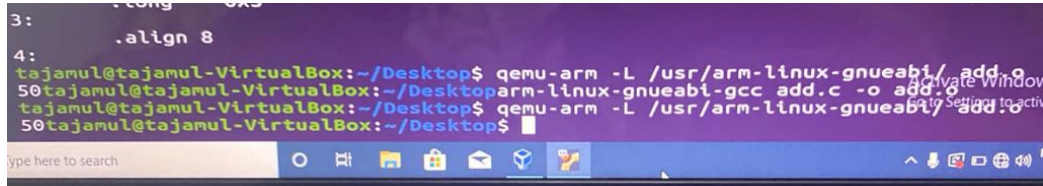


Figure 10: 5.5

```
qemu-arm -L /usr/arm-linux-gnueabi/ add.o
```

5.3 Program Tracing

Instruction	DESTIN	ATION	SR	C1	SR	C2
	loc	value	loc	value	loc	value
mov ip,sp	ip	100	sp	100		
stmfd sp!,fp,ip,lr,pc	below	is	the	Breakdown		
sub fp,ip,#4	fp	96	ip	100	const	4
sub sp,sp,#16	sp	68	sp	84	const	16
mov r3,#4	r3	4	const	4		
str r3,[fp,#-16]	addr(80)	4	r3	4		
mov r3,#6	r3	6	const	6		
str r3,[fp,#-20]	addr(76)	6	r3	6		
ldr r2,[fp,#-16]	r2	4	addr(80)	4		
ldr r3,[fp,#-20]	addr(76)	6	r3	4		
add r3,r2,r3	r3	10	r2	4	r3	6
str r3,[fp,#-24]	addr(72)	10	r3	10		
idr r2,[fp,#-24]	r2	10	addr(72)	10		
move r3,r2	r3	10	r2	10		
move r3,r3, asl #2	r3	10	r3;2	10;2		
add r3,r3,r2	r3	50	r3	40	r2	10
str r3,[fp,#24]	add(68)	50	r3	50		
idr r0,.L2	r0	L2	address	L2		
idr r1,[fp,28]	r1	50	addr(68)	50		
bl printf	lr	line33	func	printf		
move r0,r3	r0	*r3	r3	*r3		
sub sp, fp,#12	sp	84	fp	94	const	12
ldmfd sp,fp,sp,pc	below	is	the	breakdown		

stmfd sp!,fp,ip,lr,pc breakdown into loads

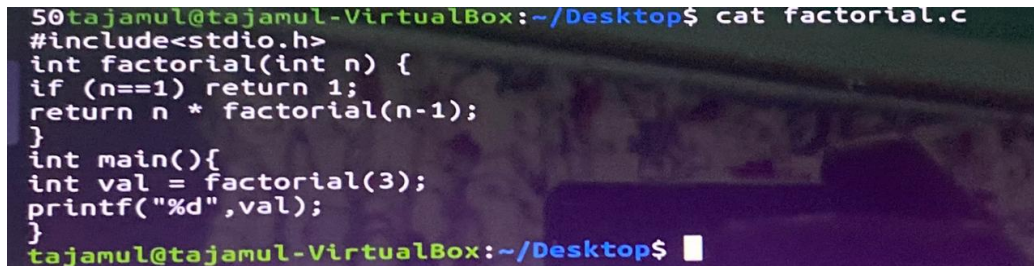
sub sp, sp, #4	sp	96	sp	100	constant	4
str pc,[sp]	add(96)	line 14	pc	line 14		
sub sp, sp, #4	sp	92	sp	96	const	4
str lr,[sp]	addr(92)	line 0	lr	line 0		
sub sp, sp, #4	sp	88	sp	92		
str ip,[sp]	add(88)	100	ip	100		
sub sp, sp, #4	sp	84	sp	88	constant	4
str fp, [sp] #4	addr(84)	100	fp	100		

ldmfd sp,fp,sp,pc breakdown into loads

ldr fp, [sp], #4	fp	100	addr(84)	100		
add sp, sp, #8	sp	92	sp	84	const	8
ldr pc, [sp],	pc	line 0	addr(92)	line0		
sub sp, sp, #4	sp	88	sp	92	const	4
ldr sp,[sp]	sp	100	addr(88)	100		

6 Program Factorial

1. Open the Terminal.
2. Write the command: touch factorial.c



```

50tajamul@tajamul-VirtualBox:~/Desktop$ cat factorial.c
#include<stdio.h>
int factorial(int n) {
    if (n==1) return 1;
    return n * factorial(n-1);
}
int main(){
    int val = factorial(3);
    printf("%d",val);
}
tajamul@tajamul-VirtualBox:~/Desktop$

```

Figure 11: 6.11

3. Open the file and write the code for program factorial as shown in fig 6.11 and 6.12
4. Write the command: gcc factorial.c -o factorial
5. Your factorial.c file is ready

```

1 #include<stdio.h>
2 int factorial(int n) {
3     if (n==1) return 1;
4     return n * factorial(n-1);
5 }
6 int main(){
7     int val = factorial(3);
8     printf("%d",val);
9 }

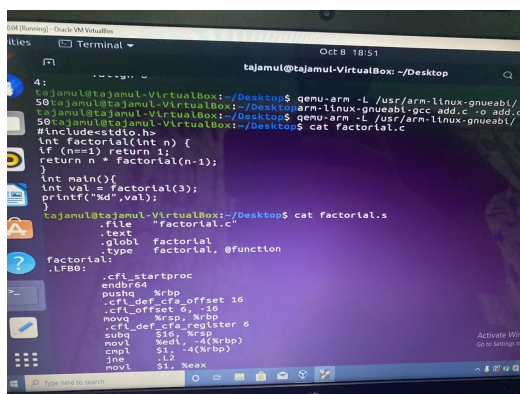
```

Figure 12: 6.12

6.1 How to generate the assembly code for the Factorial.c program using the crosscompiler.

Now for the assembly code using cross compiler

1. Write the command `gcc -sS factorial.c`



```

4:
tajanul@tajanul-VirtualBox: ~/Desktop
$ gcc -sS factorial.c
50:
tajanul@tajanul-VirtualBox: ~/Desktop$ gcc -sS factorial.c
tajanul@tajanul-VirtualBox: ~/Desktop$ cat factorial.s
#include<stdio.h>
int factorial(int n) {
    if (n==1) return 1;
    return n * factorial(n-1);
}
int main(){
    int val = factorial(3);
    printf("%d",val);
}
tajanul@tajanul-VirtualBox: ~/Desktop$ cat factorial.s
.file "factorial.c"
.text
.global factorial
.type factorial, @function
factorial:
.LFB0:
.cfi_startproc
endbr64
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 0, 16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $16, %rsp
movl $0, 4(%rbp)
cmpl $1, 4(%rbp)
jne .L1
movl $1, %eax
movl %eax, 4(%rbp)

```

Figure 13: 6.13

Ls

```

tajamul@tajamul-Virt
movl    $1, %eax
jmp     .L3
.L2:
movl    -4(%rbp), %eax
subl    $1, %eax
movl    %eax, %edi
call    factorial
.L3:
leave
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE0:
.size   factorial, .-factorial
.section .rodata
.LC0:
.string "%d"
.text
.globl  main
.type   main, @function
main:
.LFB1:
.cfi_startproc
endbr64
pushq   %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq    %rsp, %rbp
.cfi_def_cfa_register 6
subq    $16, %rsp
movl    $3, %edi
call    factorial

```

Figure 14: 6.14

- cat factorial.s
2. Your factorial.s file (assembly code) is ready


```

    call    factorial
    movl    %eax, -4(%rbp)
    movl    -4(%rbp), %eax
    movl    %eax, %esi
    leaq    .LC0(%rip), %rdi
    movl    $0, %eax
    call    printf@PLT
    movl    $0, %eax
    leave   0, %eax
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE1:
.size      main, .-main
.ident     "GCC: (Ubuntu 9.3.0-10ubuntu2) 9.3.0"
.section   .note.GNU-stack,"",@progbits
.section   .note.gnu.property,"a"
.align     8
.long      1f - 0f
.long      4f - 1f
.long      5
0:
.string    "GNU"
1:
.align     8
.long      0xc0000002
.long      3f - 2f
2:
.long      0x3
3:
.align     8
4:
tjajamul@tjajamul-VirtualBox:~/Desktop$

```

Figure 15: 6.15

6.2 How to generate the executable of the program using the cross-compiler.

In the Terminal use the following commands:

```
arm-linux-gnueabi-gcc factorial.c -o factorial.o
```

```

tjajamul@tjajamul-VirtualBox:~/Desktop$ arm-linux-gnueabi-gcc factorial.c -o factorial.o
tjajamul@tjajamul-VirtualBox:~/Desktop$ qemu-arm -L /usr/arm-linux-gnueabi/ factorial.o
6tjajamul@tjajamul-VirtualBox:~/Desktop$

```

Figure 16: 6.16

```
qemu-arm -L /usr/arm-linux-gnueabi/ factorial.o
```


6.3 Program Tracing

Instruction	DESTIN	ATION	SR	C1	SR	C2
	loc	value	loc	value	loc	value
mov ip, sp	ip	100	sp	100		
stmfd sp!, {fp, ip, lr, pc}	BELOW	IS	THE	BREAK	DOWN	
sub fp, ip, #4	fp	96	ip	100	const	4
sub sp, sp, #4	sp	84	sp	84	constant	4
mov r0, #3	r0	3	const	3		
bl factorial	lr	line 51	func	factorial		
mov r3, r0	r3	6	r0	6		
str r3, [fp, #-16]	addr(80)	6	r3	6		
ldr r0, .L4,	r0	L4	label	L4		
ldr r1, [fp, #-16]	r1	6	addr(84)	6		
bl printf	lr	line 58	func	printf		
mov r0, r3	r0	*r3	r3	*r3		
ldmfd sp, {r3, fp, sp, pc}	BELOW	IS	THE	BREAK	DOWN	

stmfd sp!, {fp, ip, lr, pc} breakDown into loads

mov ip,sp, #4	ip	100	sp	100		
sub sp, sp, #4	sp	96	sp	100	const	4
str sp,sp, #4	addr(96)	line 46	pc	line 46		
sub sp, sp, #4	sp	92	sp	96	const	4
str lr, [sp]	addr(92)	line 0	lr	line 0		
sub sp, sp, #4	sp	88	sp	92		
str ip,[sp]	addr(88)	100	ip	100		
sub sp, sp, #4	sp	84	sp	88	const	4
str fp,[sp], #4	addr(84)	100	fp	100		
sub fp, sp, #4	fp	96	ip	100	const	4

ldmfd sp, {r3, fp, sp, pc} breakdown into loads

ldr r3,[sp]	r3	6	addr(80)	6	
add sp,sp, #4	sp	84	fp	96	const
ldr fp,[sp]	fp	100	addr(84)	100	
add sp,sp, #8	sp	92	sp	84	const
ldr pc,[sp]	pc	line 0	addr(92)	line 0	
sub sp,sp, #4	sp	88	sp	92	const
ldr sp,[sp]	sp	100	addr(88)	100	

End of the Document