Week 4 – Software

Student number: 569091

Contents

Assignment 4.1: ARM assembly

Factorial definition:

A factorial is a mathematical function that multiplies a number by every positive integer below it. For example, the factorial of 5 (written as 5!) is 5 × 4 × 3 × 2 × 1, which equals 120.

Assignment

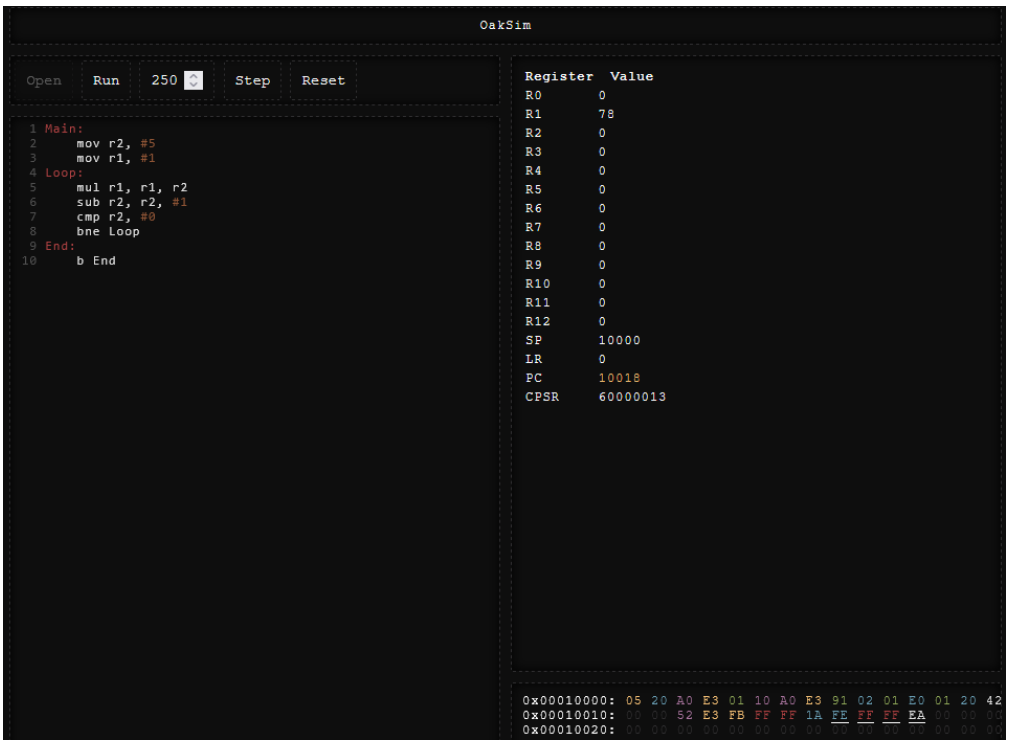
Create a loop that calculates the factorial of a number.

Answer

Code:

```
Main:                   ; Start of program
    mov r2, #5          ; Initialize counter r2 = 5 (n for n!)
    mov r1, #1          ; Initialize result r1 = 1 (1! = 1)
Loop:                   ; Loop label
    mul r1, r1, r2      ; Multiply: result = result * counter (r1 = r1 * r2)
    sub r2, r2, #1      ; Decrement counter: r2 = r2 - 1
    cmp r2, #0          ; Compare: r2 vs 0 (sets flags for branch)
    bne Loop            ; Branch if NOT equal: if r2 != 0, jump back to Loop
End:                    ; Loop exit point
    b End               ; Infinite loop: jump to End forever (prevents crash)
```

Screenshot of working assembly code of factorial calculation:



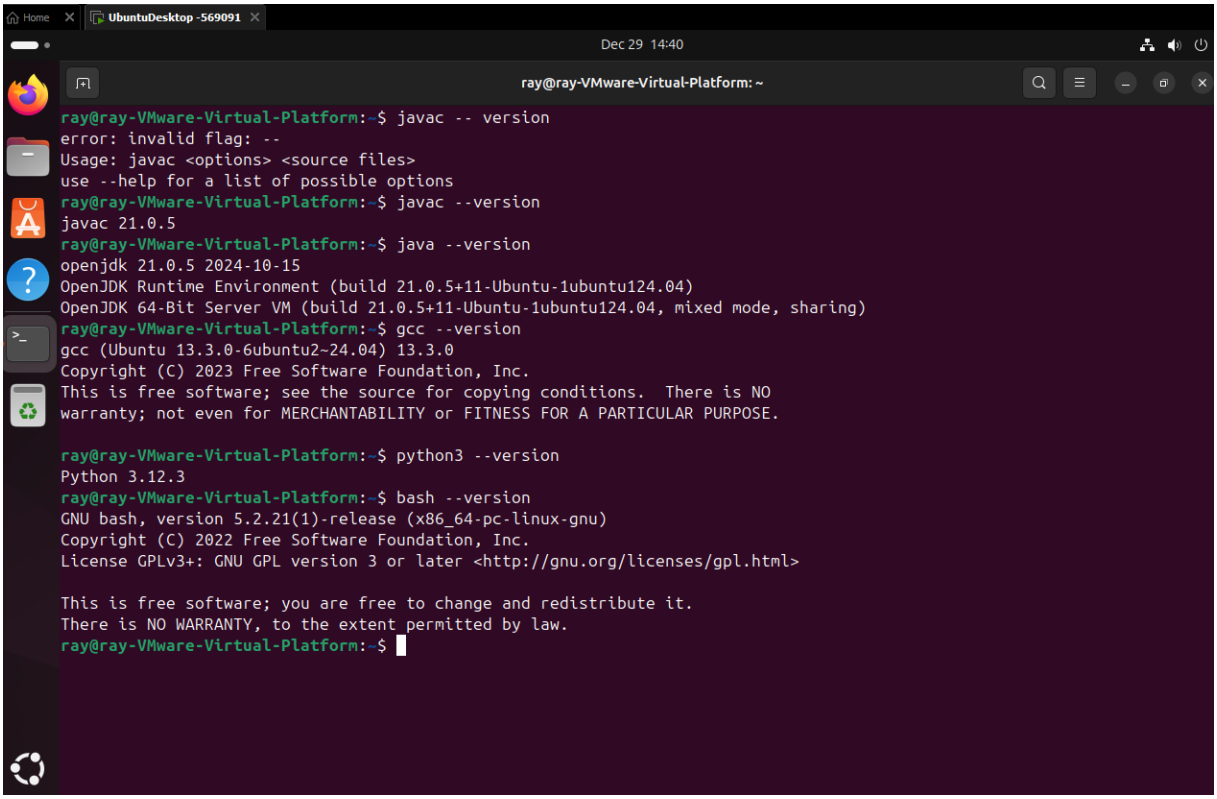
Decimal 120 to hexadecimal				
Dividend	Divisor (16)	Quotient	16 × Quotient	Remainder
120	16	7	112	8
7	16	0	0	7
				Hex: 78

Assignment 4.2: Programming languages

Assignment

Take screenshots that the following commands work:
javac --version
java --version
gcc --version
python3 --version
bash –version

Screenshot



Assignment 4.3: Compile

```
ray@ray-VMware-Virtual-Platform:~$ ls ~/Downloads/code
fib.c  Fibonacci.java  fib.py  fib.sh  runall.sh
ray@ray-VMware-Virtual-Platform:~$
```

Which of the above files need to be compiled before you can run them?

Files that require compiling:

- Fibonacci.java (Java source code).
- fib.c (C source code).

Files that do not need compilation:

- fib.py (Python source code is interpreted by an interpreter).
- fib.sh (Bash script executed by the shell interpreter).

Which source code files are compiled into machine code and then directly executable by a processor?

fib.c (compiled to a binary file, executable by the processor).

Which source code files are compiled to byte code?

Fibonacci.java (compiled to Java bytecode, typically in a .class file, executed by the Java Virtual Machine).

Which source code files are interpreted by an interpreter?

fib.py (interpreted by the Python interpreter).

fib.sh (interpreted by the Bash shell).

These source code files will perform the same calculation after compilation/interpretation. Which one is expected to do the calculation the fastest?

fib.c (C source code) is expected to perform the fastest because it is compiled into machine code and executed directly by the processor without additional layers like an interpreter or virtual machine.

How do I run a Java program?

Compile the Java file: **javac Fibonacci.java**

This creates a Fibonacci.class file.

Run the compiled Java program: **Java Fibonacci**

How do I run a Python program?

Run the Python program using the Python interpreter: **python3 fib.py**

How do I run a C program?

Compile the C file using a compiler like gcc: **gcc fib.c -o fib**

Run the compiled program: **./fib**

How do I run a Bash script?

Make the script executable: **chmod a+x fib.sh**

Run the script: **./fib.sh**

If I compile the above source code, will a new file be created? If so, which file?

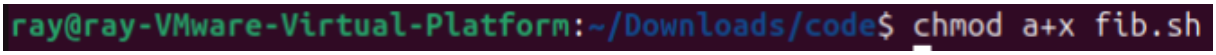
Yes, compiling the source code creates new files:

- **Fibonacci.java**: Creates Fibonacci.class.
- **fib.c**: Creates fib (or another name you specify with -o during compilation).
- **fib.py and fib.sh**: Do not create compiled files as they are interpreted.

Take relevant screenshots of the following commands:

- Compile the source files where necessary

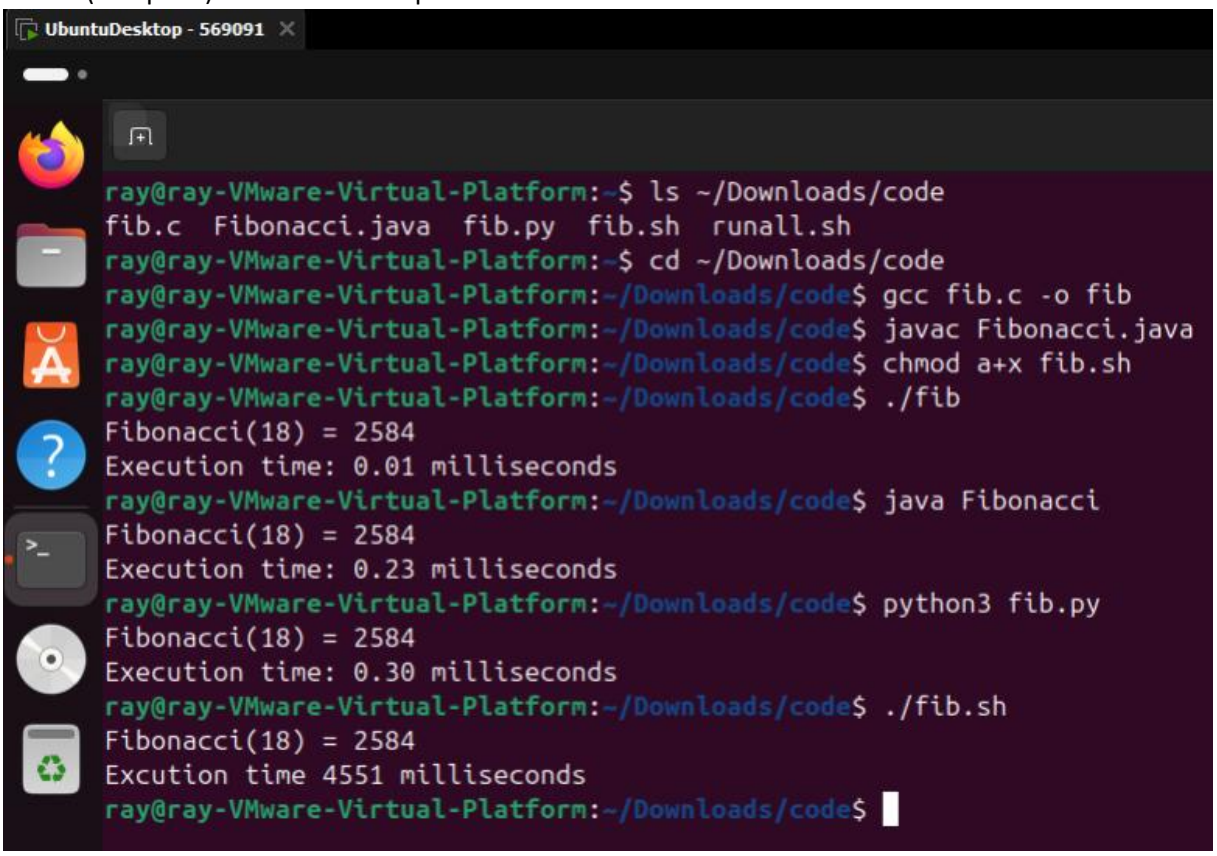
- Make them executable



- Run them

```
ray@ray-VMware-Virtual-Platform:~/Downloads/code$ ./fib
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds
ray@ray-VMware-Virtual-Platform:~/Downloads/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.23 milliseconds
ray@ray-VMware-Virtual-Platform:~/Downloads/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.30 milliseconds
ray@ray-VMware-Virtual-Platform:~/Downloads/code$ ./fib.sh
Fibonacci(18) = 2584
Excution time 4551 milliseconds
ray@ray-VMware-Virtual-Platform:~/Downloads/code$
```

- Which (compiled) source code file performs the calculation the fastest?



Fib.c performs the calculation the fastest

Assignment 4.4: Optimize

This assignment was not made/documented.

Take relevant screenshots of the following commands:

a) Figure out which parameters you need to pass to **the gcc** compiler so that the compiler performs a number of optimizations that will ensure that the compiled source code will run faster. **Tip!** The parameters are usually a letter followed by a number. Also read **page 191** of your book, but find a better optimization in the man pages. Please note that Linux is case sensitive.

b) Compile **fib.c** again with the optimization parameters

c) Run the newly compiled program. Is it true that it now performs the calculation faster?

d) Edit the file **runall.sh**, so you can perform all four calculations in a row using this Bash script. So the (compiled/interpreted) C, Java, Python and Bash versions of Fibonacci one after the other.

Assignment 4.5: More ARM Assembly

Assignment

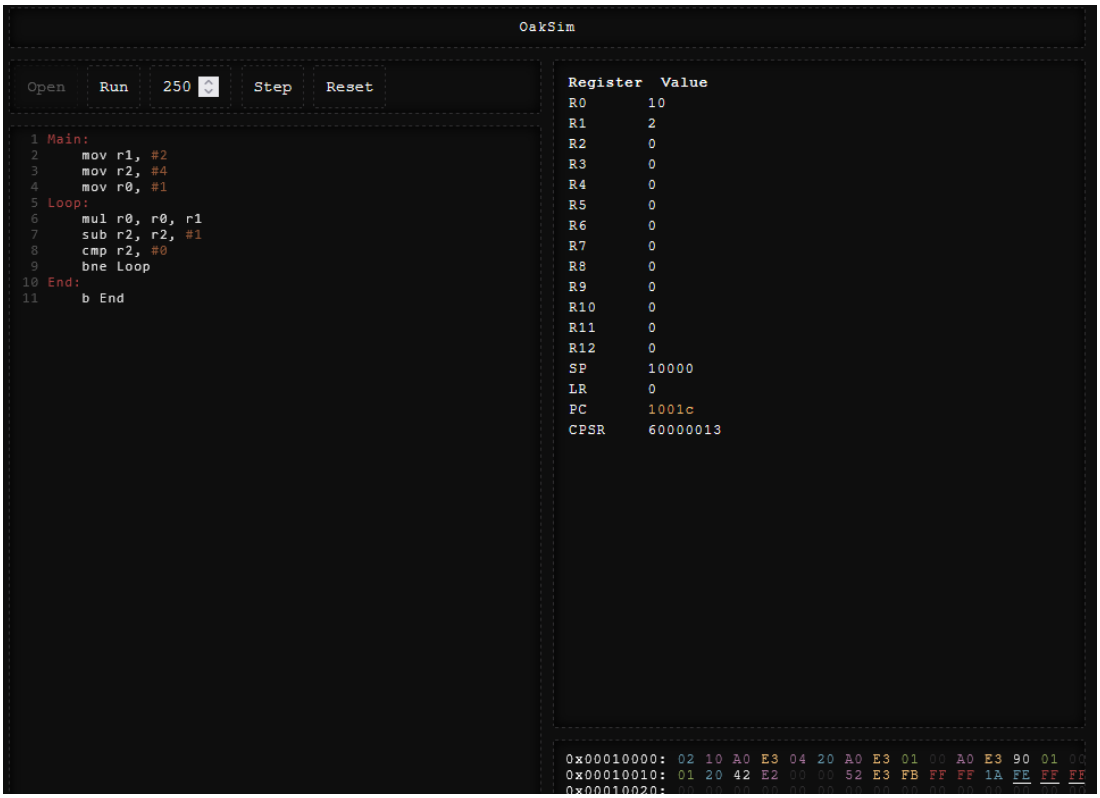
Like the factorial example, you can also implement the calculation of a power of 2 in assembly. For example you want to calculate $2^4 = 16$. Use iteration to calculate the result. Store the result in r0.

Complete the code. See the PowerPoint slides of week 4.

Code

```
Main:                   ; Program start
    mov r1, #2          ; Base = 2
    mov r2, #4          ; Exponent = 4 (calculate 2^4)
    mov r0, #1          ; Result = 1 (start with 2^0 = 1)
Loop:                   ; Power calculation loop
    mul r0, r0, r1      ; result = result * base (multiply by 2 eachtime)
    sub r2, r2, #1      ; exponent = exponent - 1
    cmp r2, #0          ; compare exponent with 0
    bne Loop            ; if exponent != 0, continue loop
End:                    ; Loop finished
    b End               ; infinite loop to halt (show r0 = 16)
```

Screenshot



Decimal 16 to hexadecimal				
Dividend	Divisor (16)	Quotient	16 × Quotient	Remainder
16	16	1	16	0
1	16	0	0	1
				Hex: 10