

Template Week 4 – Software

Student number: 582567

Assignment 4.1: ARM assembly

Screenshot of working assembly code of factorial calculation:

The screenshot shows a debugger interface with two main windows. On the left is the 'Registers' window, which lists various CPU registers with their initial values. The right window is the 'Assembly' window, which displays the ARM assembly code for a factorial program. A red box highlights the main loop and the end label in the assembly code.

Register	Value
r0	00000000
r1	00000078
r2	00000001
r3	00000000
r4	00000000
r5	00000000
r6	00000000
r7	00000000
r8	00000000
r9	00000000
r10	00000000
r11	00000000
r12	00000000
sp	00000000
lr	00000000
pc	0000001c
cpsr	600001d3 N Z C V I SVC
psr	00000000 NZCVI ?

```
1 .global _start
2 _start:
3 Main:
4 _start:
5   mov    r2, #5 ; 0x5
6   mov    r1, #1 ; 0x1
7   Loop:
8     cmp   r2, #1 ; 0x1
9     beq   End
10    beq   0x1c (0x1c: End)
11    mul   r1, r2
12    sub   r2, r2, #1 ; 0x1
13
14    b     Loop
15    b     0x8 (0x8: Loop)
16 End:
17 b   End
18 b   0x1c (0x1c: End)
19 _end:
```

(OakSim was heel laggy, dus ik heb deze applicatie gebruikt, zelfde principe.)

Assignment 4.2: Programming languages

Take screenshots that the following commands work:

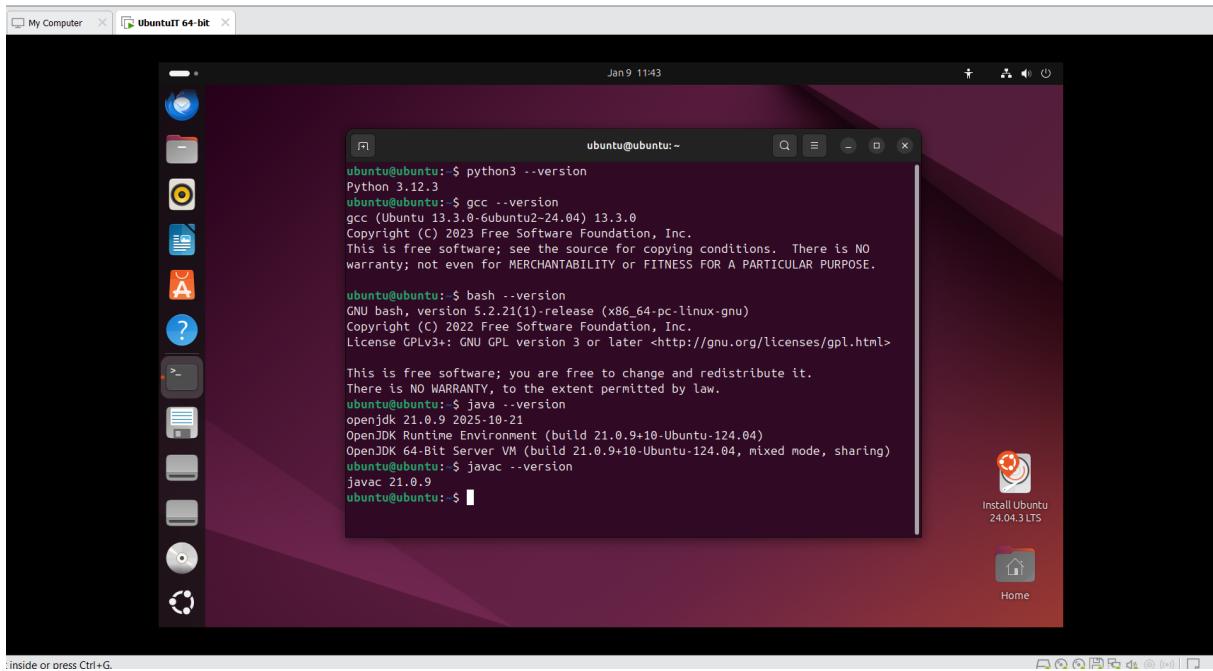
javac --version

java --version

gcc --version

python3 --version

bash --version



Assignment 4.3: Compile

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

fib.c en Fibonacci.java

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

fib.c

Which source code files are compiled to byte code?

Fibonacci.java

Which source code files are interpreted by an interpreter?

fib.py en fib.sh

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

fib.c

How do I run a Java program?

Volgens mij moet je hem eerst omzetten naar Bytecode (een taal dat de computer wel kan verstaan), dan kan je hem wel runnen via Java Virtual Machine (JVM).

How do I run a Python program?

Deze hoeft je niet te compilieren, de interpreter kan verwezen worden naar de sourcecode, en dan kan je deze runnen.

How do I run a C program?

Fib.c wordt omgezet naar een leesbare file voor gcc's unieke interpreter (fib_c), daarna kan je weer hetzelfde doen als Python.

How do I run a Bash script?

Mode veranderen, executable permisies geven, en dan kan je deze ook weer runnen.

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

Python en Bash creeeren geen nieuwe file, maar Java wordt gecompileerd naar een .class file en fib.c naar een fib_c.

Take relevant screenshots of the following commands:

- Compile the source files where necessary
- Make them executable
- Run them
- Which (compiled) source code file performs the calculation the fastest?

```
wbacon@wbacon-VMware-Virtual-Platform:~/Documents/code$ gcc fib.c -o fib_c
wbacon@wbacon-VMware-Virtual-Platform:~/Documents/code$ javac Fibonacci.java
wbacon@wbacon-VMware-Virtual-Platform:~/Documents/code$ python3 fib.py
Fibonacci(18) = 2584
Execution time: 0.49 milliseconds
wbacon@wbacon-VMware-Virtual-Platform:~/Documents/code$ sudo chmod a+x fib.sh
[sudo] password for wbacon:
wbacon@wbacon-VMware-Virtual-Platform:~/Documents/code$ ./fib.sh
Fibonacci(18) = 2584
Execution time 10379 milliseconds
wbacon@wbacon-VMware-Virtual-Platform:~/Documents/code$ java Fibonacci
Fibonacci(18) = 2584
Execution time: 0.81 milliseconds
wbacon@wbacon-VMware-Virtual-Platform:~/Documents/code$ ./fib_c
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds
```

Gcc is uiteraard het snelste.

Assignment 4.4: Optimize

Take relevant screenshots of the following commands:

- 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.
- Compile **fib.c** again with the optimization parameters

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

```
wbacon@wbacon-VMware-Virtual-Platform:~/Documents/code$ gcc -Ofast fib.c -o fib_ofast
wbacon@wbacon-VMware-Virtual-Platform:~/Documents/code$ ./fib_Ofast
bash: ./fib_Ofast: No such file or directory
wbacon@wbacon-VMware-Virtual-Platform:~/Documents/code$ ./fib_Ofast
bash: ./fib_Ofast: No such file or directory
wbacon@wbacon-VMware-Virtual-Platform:~/Documents/code$ time ./fib_ofast
Fibonacci(18) = 2584
Execution time: 0.01 milliseconds

real    0m0.004s
user    0m0.002s
sys     0m0.002s
wbacon@wbacon-VMware-Virtual-Platform:~/Documents/code$ time ./fib_c
Fibonacci(18) = 2584
Execution time: 0.02 milliseconds

real    0m0.006s
user    0m0.003s
sys     0m0.002s
```

- 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.

```
Running C program:
Fibonacci(19) = 4181
Execution time: 0.01 milliseconds

Running Java program:
Fibonacci(19) = 4181
Execution time: 0.54 milliseconds

Running Python program:
Fibonacci(19) = 4181
Execution time: 0.79 milliseconds

Running BASH Script
Fibonacci(19) = 4181
Excution time 16638 milliseconds
```

Assignment 4.5: More ARM Assembly

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.

Mai n:

```
mov r1, #2  
mov r2, #4
```

Loop:

End:

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

Screenshot of the completed code here.

The screenshot shows a debugger interface with two main panes: Registers and Disassembly.

Registers Pane: Displays the state of various registers. Notable values include r0 (00000010), r1 (00000002), r2 (00000000), r4 (00000000), r5 (00000000), r6 (00000000), r7 (00000000), r8 (00000000), r9 (00000000), r10 (00000000), r11 (00000000), r12 (00000000), sp (00000000), lr (00000000), pc (00000020), cpsr (600001d3 NZCVI SVC), and sprs (00000000 NZCVT ?).

Disassembly Pane: Shows the assembly code with addresses, opcodes, and disassembly. The code starts at address 00000000:

```
.global _start  
.start:  
Main:  
_start:  
Main:  
    mov r1, #2 ; 0x2  
    mov r2, #4 ; 0x4  
    mov r0, #1 ; 0x1  
Loop:  
Loop:  
    cmp r2, #0 ; 0x0  
    beq End  
    bge 0x20 (0x20: End)  
    mul r0, r0, r1  
    sub r2, r2, #1 ; 0x1  

```

Ready? Save this file and export it as a pdf file with the name: [week4.pdf](#)