Durham University

Computer Science

Summative Assignment

Module code and title	COMP1071 Computer Systems
Academic year	2024-25
Coursework title	LMC programming assignment
Coursework credits	3.4 credits
% of module's final mark	17%
Lecturer	Ioannis Ivrissimtzis
Submission date*	Thursday, December 05, 2024 14:00
Estimated hours of work	6.8 hours
Submission method	Ultra

Additional coursework files	test1.txt, test2.txt, test3.txt, test4.txt, testAll.txt
Required submission items and formats	lmcCOMP1071.txt (LMC assembly code in plain text)

* This is the deadline for all submissions except where an approved extension is in place.

Late submissions received within 5 working days of the deadline will be capped at 40%.
Late submissions received later than 5 days after the deadline will receive a mark of 0.
It is your responsibility to check that your submission has uploaded successfully and obtain a submission receipt.

Your work must be carried out by you only and comply with the university rules about plagiarism and collusion. Students suspected of plagiarism, either of published or unpublished sources, including the work of other students and the use of AI tools such as ChatGPT or Gemini, or of collusion, will be dealt with according to University guidelines:

https://durhamuniversity.sharepoint.com/teams/LTH/SitePages/6.2.4.aspx

Computer Systems COMP1071

2024/2025

LMC programming assignment

Submit your work on **Ultra** before **05 December 2024, 14:00**. For any questions, contact the setter of the assignment Dr Ioannis Ivrissimtzis: ioannis.ivrissimtzis@durham.ac.uk

What to submit

You should submit LMC (Little Minion Computer) assembly code.

Submit your assembly program as a plaintext file (.txt), with comments to indicate how it works. The filename must be: lmcCOMP1071.txt

I should be able to open the file with the LMC Assembly Editor window and it should compile and run without any alteration.

Description of the task

Given an integer f_0, consider the sequence described iteratively by

$$f_{i+1} = \begin{cases} f_i \,/\, 2 & f_i \ even \\ (3 * f_i + 1)/2 & f_i \ odd \end{cases}$$

Create an LMC program with the following specifications:

i. The program accepts as input a positive integer f_0, in the range $1 \leq f_0 \leq 999$.

ii. The program outputs the elements $f_0, f_1, f_2, \ldots$ of the above sequence.

iii. If $f_i = 1$ for some $i \geq 0$, the program halts.

iv. If $f_i > 999$ for some $i \geq 1$, the program outputs 0 and halts.

Some examples of inputs with the corresponding outputs:

1	1
6	6, 3, 5, 8, 4, 2, 1
65	65, 98, 49, 74, 37, 56, 28, 14, 7, 11, 17, 26, 13, 20, 10, 5, 8, 4, 2, 1
71	71, 107, 161, 242, 121, 182, 91, 137, 206, 103, 155, 233, 350, 175, 263, 395, 593, 890, 445, 668, 334, 167, 251, 377, 566, 283, 425, 638, 319, 479, 719, 0
201	201, 302, 151, 227, 341, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1

Marking process

Your program will be mostly marked automatically.

Marks for *correctness* and *robustness* will be awarded by testing your program on a previously unseen test script, and according to the marking scheme.

If you program fails this unseen test, I will look at your assembly code and partial correctness and robustness marks may be awarded. I will only try to understand your approach from the structure of the code and the comments; no attempt to fix your code will be made.

For correctness, you can start testing your program on the 5 single test scripts (*test1* – *test4*). Your program should read the input triple, output the expected value, and halt. For robustness, you can start testing your program on the script *testAll*. On the *testAll* script, your program should run consecutively 10 times, without recompiling in between, and halt.

Note: you should NOT rely solely on these 5 scripts to test your program; they cover just a small subset of the cases you should be testing for. Instead, you should write your own tests, covering a diversity of cases.

Marks for *memory efficiency* will be awarded according to the marking scheme only to programs that pass the previously unseen test. If your program fails parts of that test, discretionary efficiency marks may be awarded depending on the nature of the errors. For example, a program that halts immediately without computing anything will not be awarded any efficiency marks. A program that only fails to handle a single special case, might be awarded half of the marks the corresponding correct and robust program would get.

Marking scheme

Correctness and robustness		
a	For any input $1 \leq f_0 \leq 999$, the program outputs the specified sequence of numbers.	40
b	For all inputs, the program halts as specified.	10
c	After halting and resetting the program counter to 0, the program should be able to correctly handle the next input without need to recompile.	10
Total		**60**

Memory efficiency		
a	Marks for memory efficiency will be awarded depending on how many mailboxes are used, the fewer the better.	30
Total		**30**

The mark for memory efficiency will be calculated from the number of mailboxes used by your program. Indicative range of the memory efficiency marks:

- low memory efficiency [0-10] (inefficient approach, minor attempt for optimisation)
- medium memory efficiency [11-20] (efficient approach, good attempt for optimisation)
- high memory efficiency [21-30] (efficient approach, excellent attempt for optimisation).

Comments		
	Marks will be awarded for the quality of the comments within the assembly code.	10
Total		**10**

By reading the comments within your code, one should be able to understand:

- what each of the main parts of the of the code is doing
- any salient points in the workings of the code

For examples of commented code, see the material for the lectures of Week 6 uploaded on Ultra.