UNIVERSITY OF BRISTOL

Practice Paper 2B (practical section)

School of Computer Science

Second In-class Test for the Degree of Master of Science in Computer Science (conversion)

COMSM1302
Overview of Computer Architecture

TIME ALLOWED:
2 Hours

This paper contains **four** questions.
All questions will be marked.
The maximum for this paper is **50 marks**.

Other Instructions

1. **You are only permitted to use the following software: Calculator, KCalc, Text editor, the Hack assembler, the Hack CPU emulator, and a web browser.**
2. **You are not permitted to visit any web sites other than those in the "In-class test 2" sidebar of the unit Blackboard page.**
3. **You are not permitted the use of physical calculators.**
4. **You are not permitted external reference materials.**
5. **This is not a real exam paper.**

TURN OVER ONLY WHEN TOLD TO BEGIN WORK

For each question, write Hack assembly within a .asm file. Each file should be named according to its question number, e.g. `Q4.asm`. You **are** permitted (and encouraged) to use the Hack assembler and Hack CPU emulator to test your code. You **are not** permitted to use the Hack VM emulator.

Full marks will be given to all programs that display the correct behaviour and obey any restrictions explicitly specified in the question. Partial marks will be available. We recommend that you include **brief** comments describing your code's intended behaviour to ensure that we understand your thought process correctly while marking; however, beyond this you will not be marked on e.g. code complexity or efficiency unless otherwise stated.

In the actual test, you should zip your completed .asm files and submit them to the "In-Class Test 2 (Practical component) submission point" on Blackboard. Multiple submissions are allowed, but only the last one will be marked. No submissions are allowed from outside the test room under any circumstances.

Question 1 (10 marks)

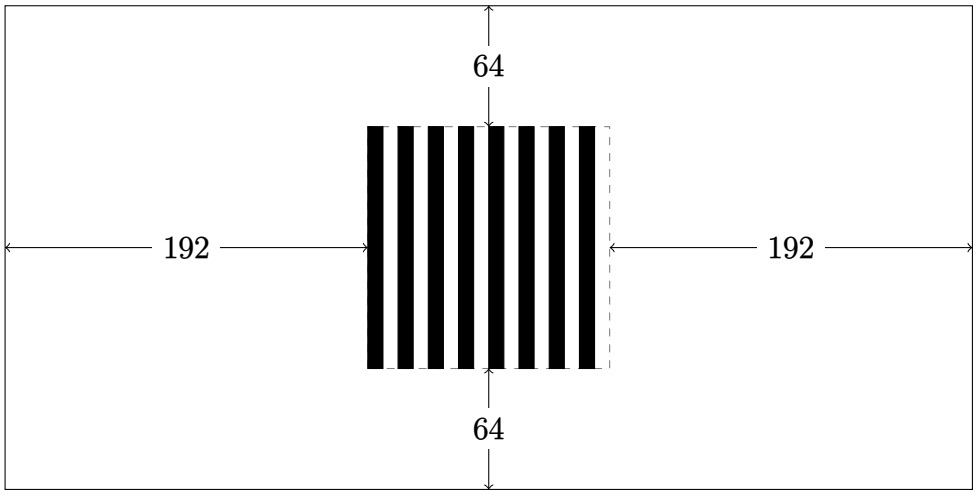
Your code should read the values in RAM[0] and RAM[1], then behave as follows.

- If RAM[1] = 0, multiply RAM[0] by −3 and store the result in RAM[2].
- If RAM[1] = 3, subtract 50 from RAM[0] and store the result in RAM[2].
- Otherwise, take a bitwise AND of RAM[0] with RAM[1] and store the result in RAM[2].

You may assume that RAM[0] is between −10000 and 10000 (so that no integer overflows can occur).

Question 2 (15 marks)

Your code should draw the square of thin alternating vertical stripes shown below on the screen. Each black and white stripe is eight pixels wide, and (as depicted) the square is centred on the screen with 128 pixels on each side. You do not need to draw the grey dashed borders of the square — these are shown purely for illustration.



Your code for Question 2 must be at most 200 lines long, or you will receive very limited credit.

Question 3 (15 marks)

For each part of this question, your code should be a direct translation of the given Hack VM instruction into assembly. In other words, your code should assume that the current contents of RAM are a state of the Hack VM, and then change RAM (and the program counter) in exactly the same way that the statement would. You may assume the standard memory map from Hack VM to assembly (given in the reference materials).

(a) `add`. **(5 marks)**

(b) `push that 3`. **(5 marks)**

(c) `if-goto summit`. You may assume that the label `summit` is mapped to ROM address 777. **(5 marks)**

Question 4 (10 marks)

RAM contains a doubly-linked list similar to the one used in memory allocation algorithms in lectures. Each list node is a two-word segment of memory. The base of the segment is a forward pointer in the list, and the second word of the segment is a backward pointer in the list. Both pointers use -1 to represent a null value. RAM[0] will contain a pointer to the base of the first list node, and RAM[1] will contain a non-negative integer. Your code should delete the RAM[1]'st element of the list, replacing its entries by zeroes and updating pointers accordingly. If RAM[1] is outside the bounds of the list, then your code should instead set RAM[1] to -1.

For example, suppose RAM is initialised to the following values:

$$\begin{array}{lll} \text{RAM}[0] = \text{0x255}, & \text{RAM}[\text{0x256}] = -1, & \text{RAM}[\text{0x1}AB] = \text{0x256}, \\ \text{RAM}[1] = 1, & \text{RAM}[\text{0x255}] = \text{0x1}AA, & \text{RAM}[\text{0x1}AA] = -1, \end{array}$$

with all other values set to zero. Then your code should leave memory set to the following values:

$$\begin{array}{ll} \text{RAM}[0] = \text{0x255}, & \text{RAM}[\text{0x256}] = -1. \\ \text{RAM}[1] = 1, & \text{RAM}[\text{0x255}] = -1, \end{array}$$

with all other values set to zero. If RAM[1] were instead initialised to 2, then your code should leave memory unchanged except for changing RAM[1] to -1. For full credit your code will need to pass several test cases, so you should make it robust. (However, you may always assume that the contents of RAM are as specified in the question.)