UNIVERSITY OF BRISTOL

January 11, 2024

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 **sixteen** questions.
All questions will be marked.
The maximum for this paper is **100 marks**.

Other Instructions

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

TURN OVER ONLY WHEN TOLD TO BEGIN WORK

Section 1 — Theory

<u>**You should answer this section on Blackboard**</u>, in the "In-Class Test 1 (Theory component)" test. We have provided a paper copy here only to allow you to make rough notes. No submissions are allowed from outside the exam room under any circumstances.

Question 1 (3 marks)
Which of the following memory segments in Hack VM is used, when compiling from Jack, to store the fields of the current object?

○ pointer ○ local ○ this ○ that ○ it depends on context

Question 2 (5 marks)
In an **assembler**, which of the following tasks would typically be carried out during lexing?

a) Converting the program from a string into a sequence of tokens.
b) Constructing a symbol table mapping labels to memory addresses.
c) Constructing a symbol table mapping variables to memory addresses.
d) Converting each line of assembly into machine code.
e) Optimising the resulting machine code.

Question 3 (4 marks)
What are the binary values of the *comp*, *dest*, and *jump* operands for the C-instruction AM=!M? Your answers should include any leading 0s, where appropriate.

Question 4 (5 marks)
In the context of memory allocation, which of the following are possible consequences of fragmentation? If relevant, you may assume the memory allocation algorithm is the version covered in lectures with coalescence of free segments but without bins ("attempt 3").

a) Existing data in memory becomes corrupted and is no longer accessible.
b) Allocating a large new memory segment fails, even though there is enough usable space for it in memory.
c) Allocating a two-word memory segment fails, even though half of memory is free.
d) Allocating a new memory segment takes much longer than normal.
e) Freeing a memory segment takes much longer than normal.

Question 5 (4 marks)

What **signed decimal** value will the following sequence of VM operations leave on top of the stack?

```
push 4
push 5
sub
not
push 0
eq
```

Question 6 (5 marks)

Complete the following sentences accurately. (Each blank contains either "RISC" or "CISC".)

a) Instructions are more likely to be of variable length in a ____________ architecture.

b) Compared to ARM, MIPS is more of a ____________ ISA and x86-64 is more of a ____________ ISA.

c) If the same C code is compiled to machine code in a CISC ISA and a RISC ISA, then one would expect the ____________ machine code to have more instructions.

d) ____________ ISAs typically use memory less efficiently.

Question 7 (4 marks)

Which of the following fragments of Hack assembly code could contain a data hazard for the purpose of pipelining?

○ M=!M	○ D=M+1	○ M=A-D	○ A=A+1
A=!A	A=M+1	A=A+D	M=M+1
D=!D		D=0	D=D+A

○ More than one of the other options. ○ None of the other options.

Question 8 (5 marks)

Below is part of the Jack grammar:

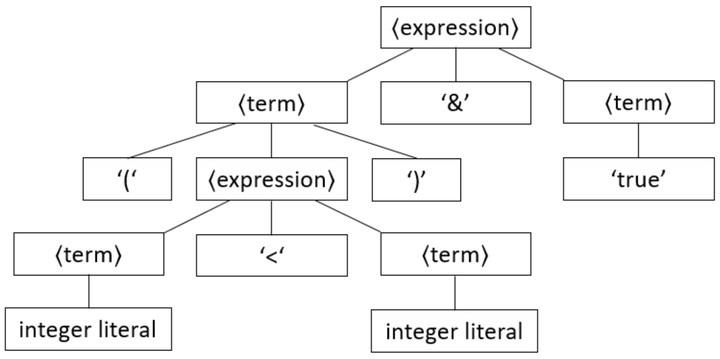
$$\langle\text{expression}\rangle ::= \langle\text{term}\rangle,\ \{(\text{`+'} \mid \text{`-'} \mid \text{`*'} \mid \text{`/'} \mid \text{`\&'} \mid \text{`|'} \mid \text{`<'} \mid \text{`>'} \mid \text{`='}),\ \langle\text{term}\rangle\}$$

$$\langle\text{term}\rangle ::= \text{integer literal} \mid \text{string literal} \mid \text{`true'} \mid \text{`false'} \mid \text{`null'} \mid \text{`this'} \mid \text{identifier},\ [\text{`['},\ \langle\text{expression}\rangle,\ \text{`]'}] \mid \text{`('},\ \langle\text{expression}\rangle,\ \text{`)'} \mid ((\text{`-'} \mid \text{`\textasciitilde'}),\ \langle\text{term}\rangle) \mid \langle\text{subroutineCall}\rangle;$$

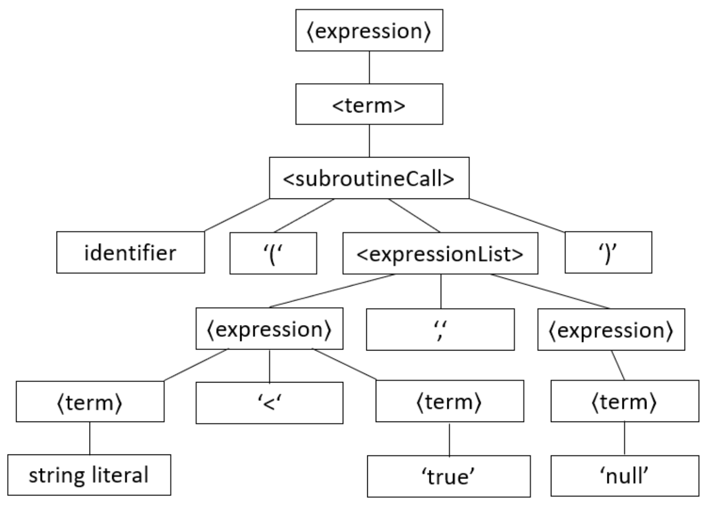
$$\langle\text{subroutineCall}\rangle ::= \text{identifier},\ [\text{`.'},\ \text{identifier}],\ \text{`('},\ \langle\text{expressionList}\rangle, \text{`)'};$$

$$\langle\text{expressionList}\rangle ::= [\langle\text{expression}\rangle, \{\text{`,'},\ \langle\text{expression}\rangle\}];$$

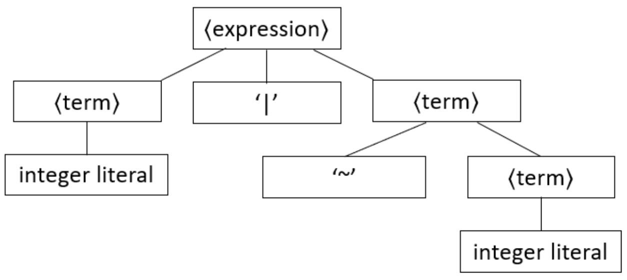
Which of the below is not a valid CST for a valid ⟨expression⟩?

○ 

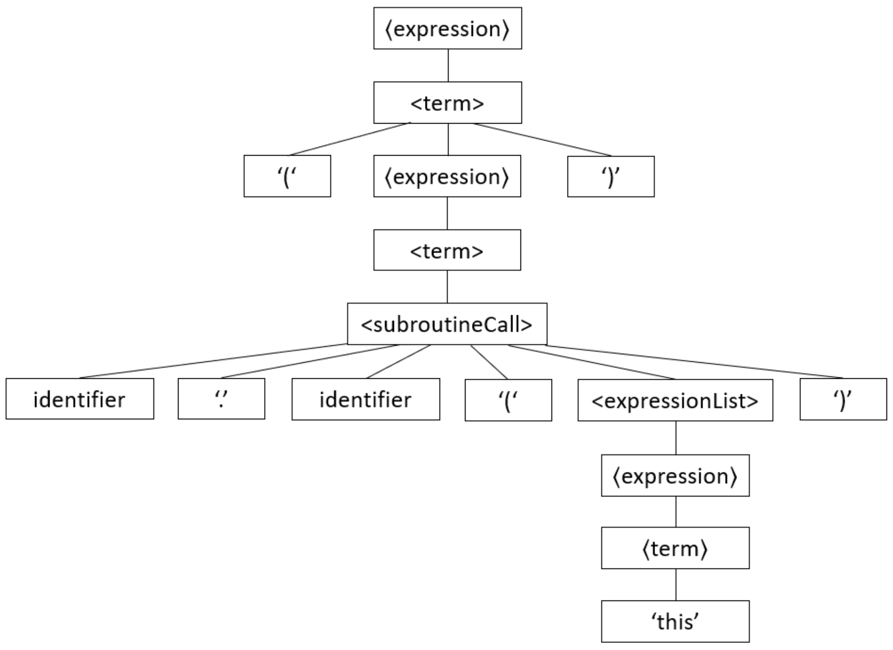
(cont.)



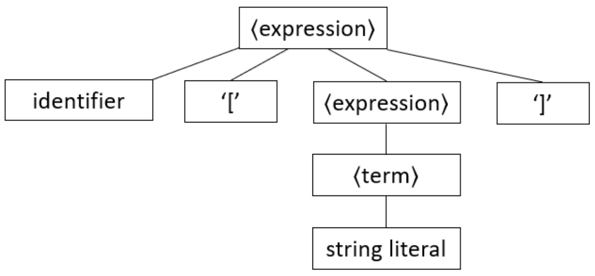
○



○



○



○

○ More than one of the other options.

○ None of the other options.

Question 9 (5 marks)
You are reading the documentation for a new programming language called Crust. The documentation tells you that "In Crust, all strings are stored as pointers to heap memory". From this, what can you tell about the following statements about strings in Crust? (Each answer is either "true", "false", or "impossible to tell".)

a) If you define a string variable inside a function, and then return that string, your code will break.

b) If you define a string variable, it will use the ASCII character set.

c) If you define a string variable and pass it to a function, and that function modifies the string, that modification will persist after the function returns.

d) If you define two string variables myString and myOtherString, then set myOtherString = myString, then myOtherString and myString will be independent copies of each other. You can modify myString without modifying myOtherString, and vice versa.

e) If you define a string variable, then at some point the memory associated with it will need to be freed with a call to something like C's free function or Jack's Memory.deAlloc function. This call might be manual or automatic, depending on how Crust works.

Question 10 (4 marks)
Consider the following proposed grammar for baby noises, given below in EBNF form with tokens 'A' and 'B'.

$$\langle \text{a} \rangle ::= \langle \text{b} \rangle, \text{‘}\texttt{A}\text{’} \mid [\langle \text{a} \rangle], \text{‘}\texttt{A}\text{’};$$
$$\langle \text{b} \rangle ::= \langle \text{b} \rangle, \langle \text{b} \rangle, \langle \text{b} \rangle, \{\langle \text{b} \rangle\} \mid \langle \text{a} \rangle, \langle \text{a} \rangle \mid \text{‘}\texttt{B}\text{’};$$

Which of the following strings is **not** a valid <a> under this grammar?

- ○ A
- ○ BBBBBAAAAAAA
- ○ AABBA
- ○ BABABABABABAA
- ○ BAABAABAABAABAAA
- ○ More than one of the other options.
- ○ None of the other options.

Question 11 (6 marks)
Note: In this question, RAM addresses are shown with round brackets rather than square ones (e.g. "RAM(0)") due to a technical limitation of Blackboard. The meaning is the same.

You are trying to write Hack VM code that will dump each of the user's keypresses into RAM(0x1000)–RAM(0x1FFF). The first keypress should be entered into RAM(0x1000), the second into RAM(0x1001), and so on. On entering a keypress into RAM(0x1FFF), you want your code to wrap around and start entering keypresses into RAM(0x1000) again. You also want your code to register each distinct keypress only once - if the user holds a key down, it should be recorded in memory once only.

Fill in the blanks in the following Hack VM program to accomplish this goal. This question is implemented on Blackboard with three drop-downs for each blank line. Each Hack VM instruction contains 1-3 words. If you want to fill a line with an instruction containing only one word (like "add"), then choose "add", "N/A", and "N/A" from the drop-downs in that order. Likewise for instructions containing only two words. The available options (as listed on Blackboard) are all Hack VM keywords together with 0, 1, 1000, 4096, 6000, 8192, 24576, "main_loop_start", and "N/A".

```
// initialise values
push constant 0
pop local 0
push constant 4096
pop pointer 0
____________
pop pointer 1
____________
// test if the keypress is new
push that 0
push local 0
eq
if-goto main_loop_start
// store keypress in local 0 and next location in RAM(0x1000)–RAM(0x1FFF)
push that 0
pop local 0
push local 0
____________
// increment location to store next new keypress
____________
push constant 1
add
pop pointer 0
// check if 0x1FFF has been reached
push constant 8192
push pointer 0
eq
____________
____________
// wrap around location to store next keypress
push constant 4096
pop pointer 0
goto main_loop_start
```

Question 12 (0 marks)

Are there any questions in the test (in either the theory or practical component) where you weren't sure what the question meant, or where you think there might be a mistake in the question? If so, please give details and explain how you interpreted the question.

This question has no marks associated with it, and you should just leave it blank if the answer is "no".

Section 2 — Practical

You should create one .asm file for each question you attempt, with the question number indicated in the filename (e.g. "Q1.asm" through "Q4.asm"). You should attach all of these files to a single submission to the submission point on Blackboard. Multiple submissions are allowed, but only the last one will be marked (in particular, do **not** create one submission for each .asm file). No submissions are allowed from outside the exam room under any circumstances.

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.

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].

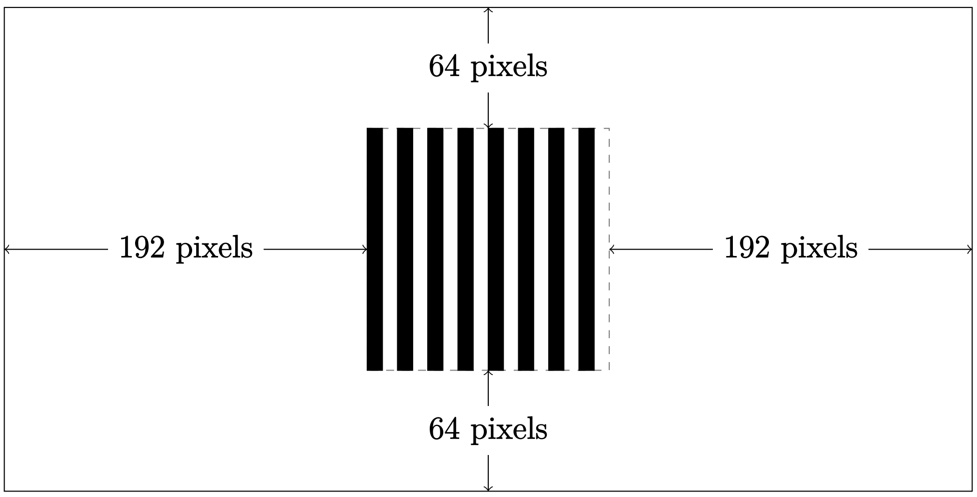
For example, given these input values, your code should leave the following values in RAM[2]:

RAM[0]	RAM[1]	RAM[2]
-2	0	6
51	3	1
30	15	14

You may assume that RAM[0] is between -10000 and 10000 (so that no integer overflows can occur). For full credit your code will need to pass several test cases.

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 8 pixels wide, and (as depicted) the square itself is centred on the screen and is 128 × 128 pixels. 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 100 lines long, or you will receive very limited credit.

Question 3 (15 marks)
Your code should be a direct translation of the Hack VM instruction `return` 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 as the `return` statement would. You may assume the standard memory map from Hack VM to assembly (given in the reference materials). For full credit your code will need to pass several test cases.

Question 4 (10 marks)
RAM contains a doubly-linked list similar to those 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. Each linked list node contains no further information beyond the forward and backward pointers. 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 n. Your code should delete the element of the list with index n, replacing its entries with zeroes and updating pointers accordingly. If n 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:

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

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

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

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.