We acknowledge and pay our respects to the Kaurna people, the traditional custodians whose ancestral lands we gather on.

We acknowledge the deep feelings of attachment and relationship of the Kaurna people to country and we respect and value their past, present and ongoing connection to the land and cultural beliefs.

Computer Systems

Lecture 14: Virtual Machine Review and Exercises

THE UNIVERSITY *of* ADELAIDE

Question 1

1 pts

Which are in the 9 VM operations?

Which of the following operators are provided by the Hack Virtual Machine?

- [ ] le
- [ ] gt
- [ ] lt
- [ ] ge
- [ ] add
- [ ] sub
- [ ] multiply
- [ ] neg
- [ ] divide
- [ ] and
- [ ] xor

Arithmetic / Boolean commands

```
add
sub
neg
eq
gt
lt
and
or
not
```

Memory access commands

`pop x` (pop into x, which is a variable)

`push y` (y being a variable or a constant)

THE UNIVERSITY of ADELAIDE

Question 2

1 pts

What are each of the following memory segments used for in the Hack Virtual machine?

static	[Choose]	holds values of class static variables
argument	[Choose]	holds the arguments for the current function
local	[Choose]	holds the local variables for the current function
constant	[Choose]	represents all the constants in the range 0 ... 32767
pointer	[Choose]	used to change the start addresses of the this and that segments
temp	[Choose]	fixed 8-entry segment that holds temporary variables for general use
field	[Choose]	this is not a segment
var	[Choose]	this is not a segment
result	[Choose]	this is not a segment

Textbook 7.4.1

Memory Segments Mapping

Local, argument, this, that: In the next chapter we discuss how the VM implementation maps these segments dynamically on the host RAM. For now, all we have to know is that the base addresses of these segments are stored in the registers LCL, ARG, THIS, and THAT, respectively. Therefore any access to the ith entry of a virtual segment (in the context of a VM "push / pop *segmentName i*" command) should be translated into assembly code that accesses address (*base* + *i*) in the RAM, where *base* is one of the pointers LCL, ARG, THIS, or THAT.

Pointer: Unlike the virtual segments described above, the pointer segment contains exactly two values and is mapped directly onto RAM locations 3 and 4. Recall that these RAM locations are also called, respectively, THIS and THAT. Thus, the semantics of the pointer segment is as follows. Any access to pointer 0 should result in accessing the THIS pointer, and any access to pointer 1 should result in accessing the THAT pointer. For example, pop pointer 0 should set THIS to the popped value, and push pointer 1 should push onto the stack the current value of THAT. These peculiar semantics will make perfect sense when we write the compiler in chapters 10–11, so stay tuned.

Temp: This 8-word segment is also fixed and mapped directly on RAM locations 5 – 12. With that in mind, any access to temp *i*, where *i* varies from 0 to 7, should be translated into assembly code that accesses RAM location 5 + *i*.

Constant: This virtual memory segment is truly virtual, as it does not occupy any physical RAM space. Instead, the VM implementation handles any access to constant *i* by simply supplying the constant *i*. For example, the command push constant 17 should be translated into assembly code that pushes the value 17 onto the stack.

Static: Static variables are mapped on addresses 16 to 255 of the host RAM. The VM translator can realize this mapping automatically, as follows. Each reference to static *i* in a VM program stored in file Foo.vm can be translated to the assembly symbol Foo.*i*. According to the *Hack machine language specification* (chapter 6), the Hack assembler will map these symbolic variables on the host RAM, starting at address 16. This convention will cause the static variables that appear in a VM program to be mapped on addresses 16 and onward, *in the order in which they appear in the VM code*.

Question 3

1 pts

Consider the following **Jack** class:

```
class bob
{
    function int foo(int a)
    {
            var int x;
            let x = a + x ;
            return x ;
    }
}
```

What are the first three virtual machine commands that implement function foo?

1: function header (initialise nVar)
function bob.foo 1
2: push a (push argument 0)
3: push x (push local 0)
4: sum

Question 4

1 pts

Consider the following **Jack** function:

```
function int foo()
{
    var a,b,sum ;

    let b = 10 ;
    while ( a < b )
    {
        let sum = sum + a ;
        let a = a + 1 ;
    }
    return sum ;
}
```

What sequence of virtual machine commands will implement the two let statements in the body of the while loop?

1: push local 2
2: push local 0
3: add
4: pop local 2
5: push local 0
6: push constant 1
7: add
8: pop local 0

THE UNIVERSITY of ADELAIDE

Question 5

1 pts

Consider the following **Jack** class:

```
class bob
{
    function int foo(int a)
    {
            var int x;
            let x = a + x ;
            return x ;
    }
}
```

What sequence of virtual machine commands is used to initialise the local variable x?

Local variables are initialised by the function declaration command (function header).

function bob.foo **1**

THE UNIVERSITY of ADELAIDE

C-instruction: *dest=comp;jump*   // Either the *dest* or *jump* fields may be empty.
// If *dest* is empty, the "=" is omitted;
// If *jump* is empty, the ";" is omitted.

	comp			*dest*		*jump*	
Binary:	1 1 1 a	c1 c2 c3 c4	c5 c6	d1 d2	d3	j1 j2 j3	

(when a=0) *comp* mnemonic	c1	c2	c3	c4	c5	c6	(when a=1) *comp* mnemonic
0	1	0	1	0	1	0	
1	1	1	1	1	1	1	
-1	1	1	1	0	1	0	
D	0	0	1	1	0	0	
A	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
-D	0	0	1	1	1	1	
-A	1	1	0	0	1	1	-M
D+1	0	1	1	1	1	1	
A+1	1	1	0	1	1	1	M+1
D-1	0	0	1	1	1	0	
A-1	1	1	0	0	1	0	M-1
D+A	0	0	0	0	1	0	D+M
D-A	0	1	0	0	1	1	D-M
A-D	0	0	0	1	1	1	M-D
D&A	0	0	0	0	0	0	D&M
D|A	0	1	0	1	0	1	D|M

d1	d2	d3	*Mnemonic*	*Destination (where to store the computed value)*
0	0	0	null	The value is not stored anywhere
0	0	1	M	Memory[A] (memory register addressed by A)
0	1	0	D	D register
0	1	1	MD	Memory[A] and D register
1	0	0	A	A register
1	0	1	AM	A register and Memory[A]
1	1	0	AD	A register and D register
1	1	1	AMD	A register, Memory[A], and D register

j1 ($out < 0$)	j2 ($out = 0$)	j3 ($out > 0$)	**Mnemonic**	**Effect**
0	0	0	null	No jump
0	0	1	JGT	If $out > 0$ jump
0	1	0	JEQ	If $out = 0$ jump
0	1	1	JGE	If $out \geq 0$ jump
1	0	0	JLT	If $out < 0$ jump
1	0	1	JNE	If $out \neq 0$ jump
1	1	0	JLE	If $out \leq 0$ jump
1	1	1	JMP	Jump

Question 1

1 pts

Category: Assembly to Hack

For each of the following Hack Assembly Language instructions, write to their 16-bit binary representation.

Write your answer as a single 16 bit binary number with spaces every 4 bits e.g. 0000 1111 0000 1111

@7	A	0000 0000 0000 0111
1	C	1110 1111 1100 0000
D; JGT	C	1110 0011 0000 0001
M=D	C	1110 0011 0000 1000

1: A or C?
2: A translate to 0 followed by 15-bit unsigned binary
3: C identify comp, dest, jump
4: look up the table

THE UNIVERSITY *of* ADELAIDE

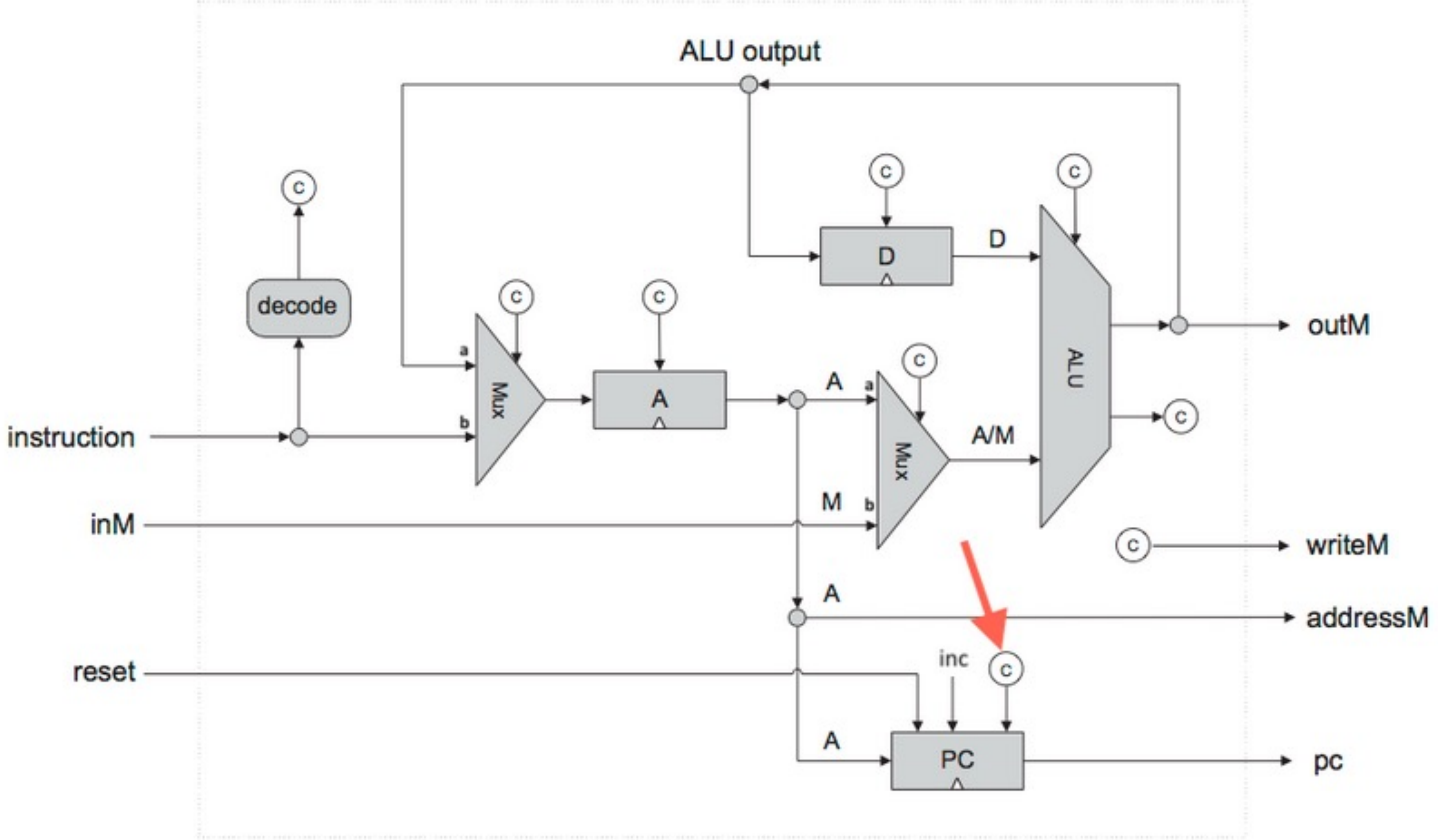
Question 2

1 pts

Category: CPU Wiring

Look at the following (incomplete) diagram of the Hack CPU. Look at the wire (and it is a single wire) pointed to by the large red arrow.

Where does the signal on this wire come from and what action does this signal trigger?



The wire is load wire of the PC register.

This tells the PC whether or not to load the value of the A-register into the PC which effectively performs a jump.

This wire comes from some logic gates that combines some status signals from the ALU with the three right-most (jump) wires of the C-instruction.

Jump Unit (Assignment 4)

The Jump Unit provides a method for us to tell the CPU when to Jump to a different part of the running program.

This will allow us to have loops and conditional statements. In the Hack architecture, this is achieved using the Jump bits of the C-instruction, and comparing these to the ALU's output using its status bits.

- **The j1, j2 and j3 bits of the C-instruction tell the CPU whether to perform a Jump if a condition is met.**
    - If the j1 bit is set true, a jump should occur if the ALU's output is less than 0
    - If the j2 bit is set true, a jump should occur if the ALU's output is equal to 0
    - If the j3 bit is set true, a jump should occur if the ALU's output is greater than 0
- We can combine these bits:
    - If j1 and j2 are both true, a jump occurs if either the ALU's output is less than 0, or if the ALU's output is equal to 0
    - If all 3 bits are true, a jump always occurs (because the ALU's output is either < 0 or > 0 or 0)
    - If none of 3 bits are true, a jump never occurs
- **We can determine whether the ALU's output is < 0 or > 0 or 0 by checking the ALU's status bits:**
    - the zr bit will be true if the ALU's output is 0.
    - the ng bit will be true if the ALU's output is < 0.

THE UNIVERSITY of ADELAIDE

Question 3

1 pts

Category: VM Expressions ((a+b)+c)

Select 5 lines of Hack VM Language, in the order they must be executed, that will implement the Jack expression:

((a+b)+c)

Notes:

- The expression must be evaluated left to right.
- The variables **a**, **b** and **c** are in the local segment at offsets 4, 5 and 6.

ab+c+

1: push local 4
2: push local 5
3: add
4: push local 6
5: add

THE UNIVERSITY of ADELAIDE

Question 4

1 pts

Category: Assembler Symbol Tables

Consider the following Hack assembly code:

```
        @R0
        D=M
        @END
        D;JLE
        @counter
        M=D
        @x
        M=D
(LOOP)
        D=D+A
        @LOOP
        D;JGT
(END)
        @END
        0;JMP
```

Fill in the appropriate symbol table entries below as they would be after the assembler's **first pass.**

Your answer should be a positive integer or `-`

Answer `-` for **predefined symbols** or **other symbols that should not be present** in the symbol table following the **first pass.**

R0	-
D	-
M	-
END	11
JLE	-
counter	-
x	-
LOOP	8
A	-
JGT	-
JMP	-

THE UNIVERSITY of ADELAIDE

Question 5

1 pts

Category: VM to Assembler Pop Arg 0

Consider the Hack Virtual Machine command:

`pop argument 0`

Complete the lines of Hack Assembly Language below, so that they implement this Hack Virtual Machine command.

@SP

```
AM=M-1
D=M
```

@ARG

```
A=M
M=D
```

THE UNIVERSITY of ADELAIDE

VM Translator Parsing

- **`push constant 1`**

```
@SP
AM=M+1
A=A-1
M=1
```

- **`pop static 7 (in a VM file named Bob.vm)`**

```
@SP
AM=M-1
D=M
@Bob.7
M=D
```

VM Translator Parsing

- **push constant 5**

```
@5
D=A
@SP
AM=M+1
A=A-1
M=D
```

- **add**

```
@SP
AM=M-1
D=M
A=A-1
M=D+M
```

- **pop local 2**

```
@SP
AM=M-1
D=M
@LCL
A=M+1
A=A+1
M=D
```

Exercise: Function call and stack structure

The Hack Virtual Machine allocates an area of the stack for each active function call.

Assume that a function, foo, was passed <u>three</u> parameters and the virtual machine code for the function starts with function foo 3.

Show the structure of the function's stack frame immediately after the execution of the virtual machine command function foo 3.

Clearly indicate where the values ARG, LCL and SP are pointing.

THE UNIVERSITY of ADELAIDE

function foo 3

	...
ARG ->	arg 0
	arg 1
	arg 2
	return address
	saved LCL
	saved ARG
	saved THIS
	saved THAT
LCL ->	local 0
	local 1
	local 2
SP ->	

THE UNIVERSITY of ADELAIDE

This week

- Review Chapters 7 & 8 of the textbook (if you haven’t already)
- Assignment 5 due Sunday week (Sept 17)
- Week 8 Supervised Practical Exam (Please come to the workshop you enrolled).
- Week 8 Practice Questions available.
- Review Chapter 8 & 9 of the textbook before Week 8.

THE UNIVERSITY of ADELAIDE

make history.

THE UNIVERSITY of ADELAIDE