NOTICE: Past Exams

Past exams are provided as-is for reference only.

- The content of this exam reflects course content and learning outcomes as they were at the time this exam was administered.
- This course's content, structure, and/or focus may have changed since this past exam was administered.
- As such, **the content, structure and types of questions** contained herein **may differ from those in your final exam**.
- If you plan on using this as a study resource, be sure to do so in conjuction with the current course's syllabus and resources to ensure that your study covers the correct content.

THE UNIVERSITY of ADELAIDE

Primary Examination, Semester 1, 2017

Computer Systems
COMP SCI 2000, 7081

Official Reading Time:	10 mins
Writing Time:	120 mins
Total Duration:	130 mins

Questions	**Time**	**Marks**
Answer all 12 questions	120 mins	120 marks
		120 Total

Instructions for Candidates

- This is a Closed-book examination.
- Begin each answer on a new page
- Examination material must not be removed from the examination room.

Materials

- Foreign Language Dictionaries are Permitted

DO NOT COMMENCE WRITING UNTIL INSTRUCTED TO DO SO

Basic Gates and Boolean Logic

Question 1

(a) Consider the expression:

$$x \neq y$$

which is *true* if the boolean values x and y are not equal. The expression is *false* otherwise. Answer the following two questions:

i. Draw the truth table for the $\neq$ operator in terms of x and y.

[3 marks]

ii. Draw an implementaton of the $\neq$ operator solely in terms of And, Or and Not gates.

[6 marks]

[Total for Question 1: 9 marks]

Boolean Arithmetic and ALU design

Question 2

For the following questions you may find the information in Figure 2 useful.

(a) Look at the Hack ALU truth-table shown in Figure 2 in the Appendix. Note that it is possible to design the functionality of the ALU a few operations at a time. For this question, we will consider the last line of the table, which describes the Or operation. Answer the following:

i. In the ALU the Or operation is implemented using an And chip and some other chips and wires. Give the logical expression for the Or operation implemented by the ALU. Your answer must include an And operation.

[4 marks]

ii. Using a truth table, show that the expression you gave in part i above is equivalent to the Or operation.

[3 marks]

iii. Draw an implementation of a 16-bit Or chip that uses a 16-bit And chip.

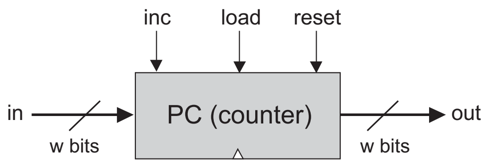
[3 marks]

[Total for Question 2: 10 marks]

Sequential Logic

Question 3

(a) Look at the following diagram and text description for a program counter (PC) from figure 3.5 of the textbook:




```
Chip name: PC  // 16-bit counter
Inputs:    in[16], inc, load, reset
Outputs:   out[16]
Function:  If reset(t-1) then out(t)=0
              else if load(t-1) then out(t)=in(t-1)
                   else if inc(t-1) then out(t)=out(t-1)+1
                        else out(t)=out(t-1)
Comment:   "=" is 16-bit assignment.
           "+" is 16-bit arithmetic addition.
```


Draw an implementation of the PC chip *without the reset wire*. That is, draw an implementation of the PC that can handle signals from the *inc* and the *load* wires but *doesn't* provide a reset function.

Note, you do not have to express your solution in terms of primitive gates such as Nand. You can use large scale chips such as Inc16.

[10 marks]

[Total for Question 3: 10 marks]

Hack Assembler and Machine Code

Question 4

For the following questions you may find the information in Figures 3, 4, 5, 6 and 7 in the appendix of this paper useful.

(a) Look at the following Hack machine code:

```
0000000000010000
1111110000010000
0000000000010001
1111000111001000
1111110000010000
0000000000000000
1110001100000001
```

Answer the following:

i. Using the instruction formats in Figures 3, 4, 5, 6, and 7 as a guide, write down the Hack assember instructions that are equivalent to this code.

[7 marks]

ii. Describe what the machine code above does.

[3 marks]

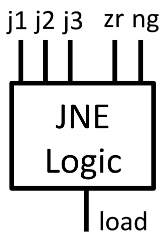
[Total for Question 4: 10 marks]

Computer Architecture

Question 5

For the following questions you may find the information in Figures 1, 2, 3, 4, 5, 6, 7 and 8 in the appendix of this paper useful.

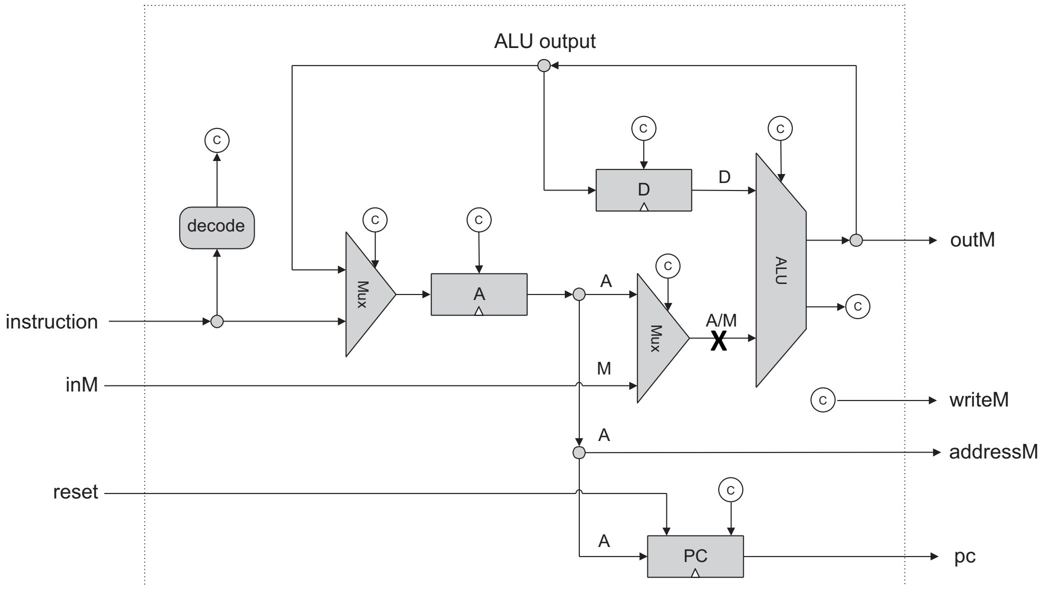
(a) Draw an implementation of the logic that implements the JNE (jump not-equal-to) part of the C-instruction. The interface for the JNE is:



where the inputs are the jump wires from the C instruction and the zr and the ng wires from the ALU and the ouput is the load wire for the PC register. Note that you do not have to implement the logic for every type of jump – just JNE. Hint, in answering your question you may find the information in Figure 7 useful.

[6 marks]

(b) Consider the following diagram of the Hack CPU.



The A/M wire, marked with an **X**, selects either the A-register or RAM as one of the input values for a C-instruction. In the Hack machine a C-instruction cannot access *both* the A-register and RAM as input in the same instruction. So, for example, the following instruction:

```
D=M+A
```

Is not a valid Hack instruction since it has both M and A as input.

Briefly describe why such access is unlikely to be useful even if it were permitted by the Hack machine.

[3 marks]

[Total for Question 5: 9 marks]

Assembler

Question 6

(a) Look at the following Hack assembler code:

```
@x
M=0
@y
M=0
(LOOP)
@x
MD=M+1
@y
M=M+D
@3
D=D-A
@LOOP
D;JLE
```

Hand-assemble this code by writing out the binary machine code the assembler would produce. For this question you may find the information in Figures 3, 4, 5, 6, and 7 useful.

[12 marks]

[Total for Question 6: 12 marks]

Virtual Machine - Expressions

Question 7

(a) Translate the following Jack let statement into Hack Virtual Machine language:


```
let d = (2 - x) * (y + 5)
```

The variables d, x and y are in memory segment *local* at indexes 2,5 and 7 respectively. Assume there is a function named *multiply* that will take two arguments and return the result of multiplying the two numbers together.

[8 marks]

[Total for Question 7: 8 marks]

Virtual Machine - Subroutines

Question 8

(a) The Hack Virtual Machine language provides three function related commands:

- call f m
- function f n
- return

i. Briefly describe the arguments to the `function` command.

[3 marks]

ii. If the `function` command did not have the second argument, what alternate virtual machine code would need to be generated to implement: `function c.x 2` ?

[4 marks]

iii. Why does the second argument to the `call` command need to be provided?

[3 marks]

[Total for Question 8: 10 marks]

Jack

Question 9

(a) How does the **Jack** compiler provided with the nand2Tetris tools ensure that a constructor, function or method from another class is being called correctly? Why does it do this?

[3 marks]

(b) List the syntax errors in the following Jack class definition:

```
class x
{
    function int xx(var int n)
    {
        if ( n <= 2 ) return 17 ;
        return y.xxx(n--) ;
    }
}
```

[5 marks]

[Total for Question 9: 8 marks]

Parsing

Question 10

(a) Turn the following **Jack** code fragment into XML with one node for each non-terminal in the grammar.

```
let x[ix] = y ;
```

You should start with a node for a let statement and you may omit nodes for any keywords, identifiers or symbols. The grammar can be found in Figure 9 in the appendix.

[8 marks]

[Total for Question 10: 8 marks]

Code Generation

Question 11

(a) Consider the following **Jack** method:

```
// class Complex contains 4 instance variables
// declared in this order: aa, bb, cc and dd, aa is an Array
method Complex useful(Complex a, Complex b)
{
}
```

What Hack Virtual Machine language code would implement the following Jack program fragments if they were in the body of the method `useful`?

i. `let b = Complex.new(3,2) ;`

[4 marks]

ii. `aa[7] = a ;`

[6 marks]

iii. `return b ;`

[2 marks]

iv. `let bb = dd ;`

[3 marks]

(b) Show the two symbol tables for the following code just after the variable declaration in the method getSerial has been parsed.


```
class SerialNums
{
        static int id ;
        field int myid ;

        constructor SerialsNums new(int key)
        {
                let myid = id ;
                return this ;
        }
        method int getSerial(int password)
        {
                var int ignore_me ;
                return myid ;
        }
}
```

[4 marks]

[Total for Question 11: 19 marks]

Jack OS, Optimisation

Question 12

(a) Why might implementing a 16-bit multiply operation in the ALU of the Hack machine significantly increase the time it takes to execute an instruction that sets the A and D registers to the value 0?

[3 marks]

(b) What three aspects of a processor's physical implementation determine the power consumption and how is this calculated?

[4 marks]

[Total for Question 12: 7 marks]

APPENDICES

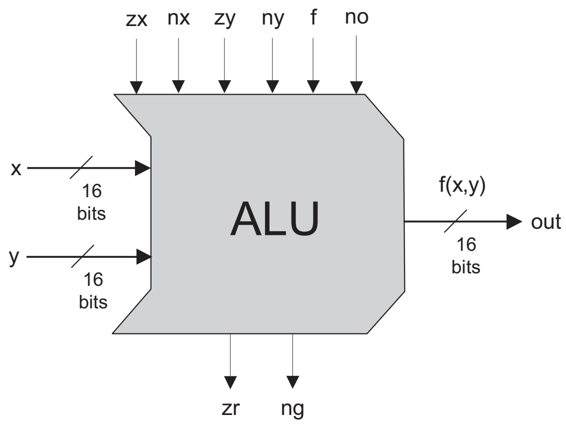


Figure 1: An interface diagram for the ALU. From figure 2.5 of the textbook.

These bits instruct how to preset the x input		These bits instruct how to preset the y input		This bit selects between + / And	This bit instructs how to postset out	Resulting ALU output
zx	nx	zy	ny	f	no	out=
if zx then x=0	if nx then x=!x	if zy then y=0	if ny then y=!y	if f then out=x+y else out=x&y	if no then out=!out	f(x,y)=
1	0	1	0	1	0	0
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	1	1	0	0	x
1	1	0	0	0	0	y
0	0	1	1	0	1	!x
1	1	0	0	0	1	!y
0	0	1	1	1	1	-x
1	1	0	0	1	1	-y
0	1	1	1	1	1	x+1
1	1	0	1	1	1	y+1
0	0	1	1	1	0	x-1
1	1	0	0	1	0	y-1
0	0	0	0	1	0	x+y
0	1	0	0	1	1	x-y
0	0	0	1	1	1	y-x
0	0	0	0	0	0	x&y
0	1	0	1	0	1	x|y

Figure 2: The Hack ALU truth table. From figure 2.6 of the textbook.

A-instruction: *@value* // Where *value* is either a non-negative decimal number
// or a symbol referring to such number.

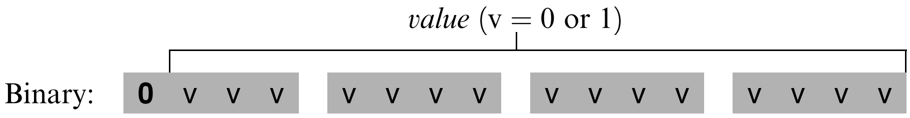


Figure 3: The format of an A-instruction. From page 64 of the text book.

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.

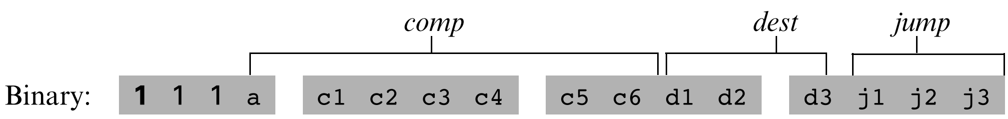


Figure 4: The format of an C-instruction. From page 66 of the text book.

(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

Figure 5: The meaning of C-instruction Fields. From figure 4.3 of the textbook.

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

Figure 6: The meaning of the destination bits of the C-instruction From figure 4.4 of the textbook.

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

Figure 4.5 The *jump* field of the *C*-instruction. *Out* refers to the ALU output (resulting from the instruction's *comp* part), and *jump* implies "continue execution with the instruction addressed by the A register."

Figure 7: The meaning of the jump bits of the C-instruction From figure 4.5 of the textbook.

Label	***RAM address***
`SP`	0
`LCL`	1
`ARG`	2
`THIS`	3
`THAT`	4
`R0-R15`	0-15
`SCREEN`	16384
`KBD`	24576

Figure 8: The predefined symbols in Hack Assembly language. From page 110 of the text book.

Lexical Elements

```
keyword         ::= 'class' | 'constructor' | 'function' | 'method' | \
                    'field' | 'static' | 'var' | 'int' | 'char' | \
                    'boolean' | 'void' | 'true' | 'false' | 'null' | \
                    'this' | 'let' | 'do' | 'if' | 'else' | 'while' | \
                    'return'
symbol          ::= '{' | '}' | '(' | ')' | '[' | ']' | '.' | \
                    ',' | ';' | '+' | '-' | '*' | '/' | '&' | \
                    '|' | '<' | '>' | '=' | '~' | '
integerConstant ::= A decimal number in the range 0 .. 32767
stringConstant  ::= '"' A sequence of Unicode characters not including
                        double quote or newline '"'
identifier      ::= A sequence of letters, digits and underscore ('_')
                    not starting with a digit.
```

Statements

```
statements      ::= statement*
statement       ::= letStatement | ifStatement | whileStatement | \
                    doStatement | returnStatement
letStatement    ::= 'let' varName ('[' expression ']')? '=' expression ';'
ifStatement     ::= 'if' '(' expression ')' '{' statements '}' \
                    ('else' '{' statements '}')?
whileStatement  ::= 'while' '(' expression ')' '{' statements '}'
doStatement     ::= 'do' subroutineCall ';'
returnStatement ::= 'return' expression? ';'
```


Expressions

```
expression      ::= term (op term)*
term            ::= integerConstant | stringConstant | \
                    keywordConstant | varName | \
                    varName '[' expression ']' | subroutineCall | \
                    '(' expression ')' |  unaryOp term
subroutineCall  ::= subroutineName '(' expressionList ')' |  \
                    (className | varName) '.' subroutineName '(' expressionList ')'
expressionList  ::= (expression (',' expression)*)?
op              ::= '+' | '-' | '*' | '/' | '&' | '|' | '<' | '>' | '='
unaryOp         ::= '-' | '~'
keywordConstant ::= 'true' | 'false' | 'null' | 'this'
varName         ::= identifier
```

Figure 9: The Jack grammar. From figure 10.5 of the textbook.

END OF EXAMINATION PAPER