EP1200 Introduktion till datorsystemteknik

Övningstentamen. Problemen i den ordnarie tentamen kan fördelas annorlunda över kursmaterialet.

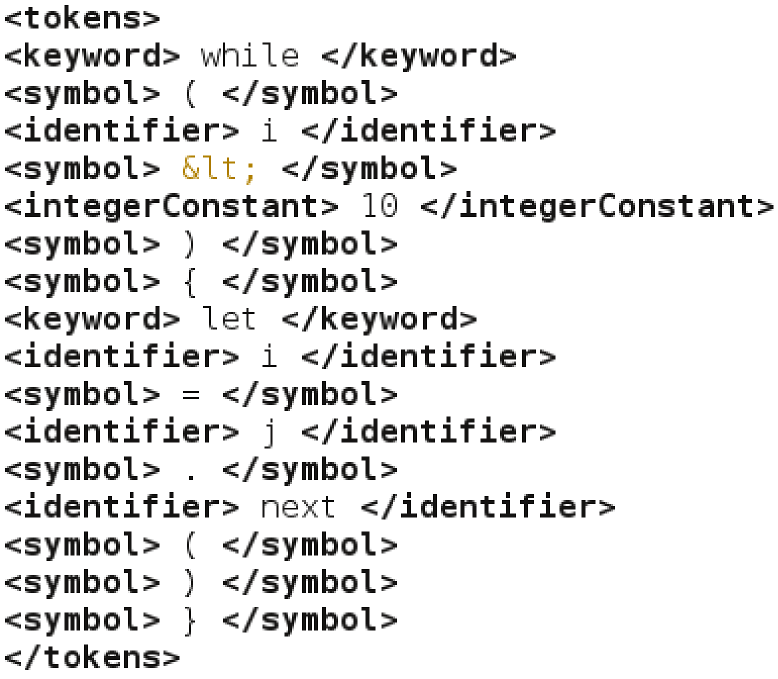
- Inga hjälpmedel är tillåtna utom de som följer med tentamenstexten.
- Skriv kurskod, namn och personnummer på alla ark som lämnas in.
- Svara på svenska eller engelska. Oläsliga och oförståeliga svar ger ingen poäng. Svara med välstrukturerade, korta och koncisa svar. Alla svar måste vara motiverade och all kod väl kommenterad.
- Poängtal anges för varje uppgift. Totalt kan tentamen ge 35 poäng mot slutbetyget i kursen.
- Lösningsförslag anslås på kursens hemsida på KTH Social.

1. Give a short answer for each question. (Each correct answer in addition to the first 5 correct ones gives one point.)

   a. What is the result of running the Jack tokenizer on the following code (use the <xxx> token <\xxx> syntax)?

```
while( i < 10 ){ let i = j.next()}
```

Solution:



```
<tokens>
<keyword> while </keyword>
<symbol> ( </symbol>
<identifier> i </identifier>
<symbol> &lt; </symbol>
<integerConstant> 10 </integerConstant>
<symbol> ) </symbol>
<symbol> { </symbol>
<keyword> let </keyword>
<identifier> i </identifier>
<symbol> = </symbol>
<identifier> j </identifier>
<symbol> . </symbol>
<identifier> next </identifier>
<symbol> ( </symbol>
<symbol> ) </symbol>
<symbol> } </symbol>
</tokens>
```

   b. Give the value in decimal notation for each of these four integers, given in 2's complement representation: 1001, 1000, 110001, 11 (all four must be correct)
   Leave least significant '1' and all zeroes to its right; flip remaining bits:
   hence, 1001 -> "-(0111)" = -7, "-(1000)" = -8, "-(001111)" = -15, "-(01)" = -1

   c. Give the definition of a variable and of a label in Hack assembly. Write a few lines of code to illustrate.

Variables are symbolic names for RAM locations. Labels are symbolic names for instruction memory locations, that is, they refer to a given command line. E.g. *var* is variable and *END* is label in the following code:

```
  @var
  M=5
(END)
  @END
  0,JMP
```

d. What is the difference between an object and a class?
A class is the definition of a complex data type including code to be executed (methods and/or functions). An object is an instance of a class. An object has memory allocated to it.

e. What is the role of the parsing step and what is the role of code generation step in the Assembly to binary Machine Language translation?
Parsing cuts the command into underlying fields (A command: value, C command: comp, dest, jump). Code generation puts together the machine language code by replacing the assembly fields with binary values. (Code generation in turn needs two passes, in the first pass it finds labels and inserts the label information in the symbol table, in the second pass translates the code.)

f. Why do we need a linked list to properly perform memory management; what cannot be achieved by using a single pointer?
With a single pointer we cannot keep track of the memory fragmentation.

g. Consider the expression (2/x)*(3-y)+1 . Rewrite the expression in prefix notation.
+*/2x-3y1

h. What is combinational and sequential logic and how do they differ?
Combinational logic computes Boolean expressions without any timing: the inputs eventually lead to the output (when the signals settle); sequential logic is clocked and output is a function of the input in previous clock cycles.

i. Which of the following gates can be used on its own to write Boolean expressions for all logic functions of one and two variables: AND, OR, XOR, NOT, NAND, NOR?
Only NAND and NOR can be used as a single gate; you need the possibility to invert the input, and De Morgan's law gives the relationship between AND and OR so that one or the other is sufficient on its own.

j. How is operator precedence handled by the Jack compiler?
It is *not* handled by the compiler.

2. Build a switch with 3-bit input and output words that *permutes* the input symbols according to a 3-bit *selector* input (*out[j]= in[i]* for *j=0, …2* and *i=0, …2*; any input symbol, *in[i]*, may only appear once in an output word).
   a. List and number all permutations and assign each permutation to a value of the selector. (1p)
   b. Propose a chip implementation by providing the block diagram of the chip. Use the chip set on the provided sheet. (3p)
   c. Specify the chip in HDL. (1p)

```
CHIP Switch3by3 {
   IN in[3], sel[3];
   OUT out[3];

   PARTS:
   // Put your code here:
}
```

See the hand written solution at the end of the document.

3. Write a program in Hack assembly that takes three integers in two's complement representation stored in R0, R1 and R2, and sorts them in increasing order so that the largest value is in R2 and the smallest in R0 when the program ends.
   a. Describe the algorithm. (2p)
   b. Write the assembly code. (2p)
   c. Provide the complete symbol table for the assembler. (1p)

   See one of the possible solutions below.

   a) Sort3

   ```
   compare R0 and R1
   if R0 > R1 switch them
   compare R1 and R2
   if R1 > R2 switch them
   compare R0 and R1
   if R0 > R1 switch them
   ```

   b)

```
          //  Sorts R0, R1,R2, uses R4 for additional memory
          //  location for the switch

          @R0   // compare and switch R0 and R1
          D=M
          @R1
          D=D-M
          @NOSW1
          D;JLT
          @R0
          D=M
          @R4
          M=D
          @R1
          D=M
          @R0
          M=D
          @R4
          D=M
          @R1
          M=D
(NOSW1)
          @R1     // compare and switch R1 and R2
          D=M
```

```
          @R2
          D=D-M
          @NOSW2
          D;JLT
          @R1
          D=M
          @R4
          M=D
          @R2
          D=M
          @R1
          M=D
          @R4
          D=M
          @R2
          M=D
(NOSW2)
          @R0    // compare and switch R0 and R1 again
          D=M
          @R1
          D=D-M
          @END
          D;JLT
          @R0
          D=M
          @R4
          M=D
          @R1
          D=M
          @R0
          M=D
          @R4
          D=M
          @R1
          M=D
(END)
          @END
          0;JMP   // Infinite loop
```

c) This version does not use user defined variables, so in addition to the pre-defined symbols, in the symbol table it has:

NOSW1	18
NOSW2	36
END	54

4. Implement a Hack VM function that solves the same sorting problem as in problem 3, assuming that the three integers are stored in the static segment at indices 0, 1 and 2. (5p)

```
push static 0
push static 1
lt
if-goto no_swap_01
push static 0
push static 1
pop static 1
pop static 0
label no_swap_01
push static 1
push static 2
lt
if-goto no_swap_12
push static 1
push static 2
pop static 2
pop static 1
label no_swap_12
push static 0
push static 1
lt
if-goto code_end
push static 0
push static 1
pop static 1
pop static 0
label code_end
```

5. Modify the Jack grammar to support the `switch` statement. Provide the definition of any new non-terminal that you need to introduce, as well as the new definition of the existing non-terminals that you need to modify. The following code shows an example of a `switch` statement. Note: at least one `case` statement is compulsory. The `default` statement is optional. Only one integer constant can follow the '**case**' keyword. (5p)

```
var int month;
var String monthString;
let month = 4;
switch (month) {
     case 1:      let monthString = "January";
                  break;
     case 2:      let monthString = "February";
                  break;
     default:     let monthString = "Invalid month";
                  break;
}
```

```
statement: switchStatement | letStatement …
switchStatement: ‘switch’ ‘(’ expression ‘)’ ‘{’ switchBlock ’}’
switchBlock: (caseStatement)+ (defaultStatement)?
caseStatement: ‘case’ integerConstant ‘:’ statements (‘break’ ‘;’)?
defaultStatement: ‘default’ ‘:’ statements (‘break’ ‘;’)?
keyword: ‘switch’ | ‘case’ | ‘default’ | ‘break’ | ‘class’ …
```

6. Compile the line of Jack code marked in bold into Hack assembly.
   a. Compile it first into VM code and then from VM code into assembly. (3p)
   b. Compile it directly from Jack into assembly. (2p)

```
class dummy {
   field int A;
   field int B;

…
   method void incrementA() {
       A=A+1;   //this is the line of code
       return;
    }
}
```

a) The VM code will be

```
push this 0
push constant 1
add
pop this 0
```

A corresponding Hack Assembly code is as follows.

```
//push this 0
@THIS   //load address of this pointer into A
A=M     //load value of this pointer into A
D=M      //load value at this[0] into D
@SP       //load address of stack pointer
A=M        //load top-of-stack into A
M=D        //store value on stack
@SP
M=M+1      //increment stack pointer
//push constant 1
D=1
@SP       //load address of stack pointer
A=M        //load top-of-stack into A
M=D        //store value on stack
@SP
M=M+1      //increment stack pointer
//add
@SP
M=M-1      //decrement stack pointer
A=M
D=M         //store value in D register
@SP         //load address of stack pointer
A=M-1       //load top-of-stack – 1 into A
M=M+D       //perform addition
//pop this 0
@SP
M=M-1      //decrement stack pointer
A=M
D=M         //store value in D register
@THIS      //load address of this pointer into A
A=M     //load value of this pointer into A
M=D      //set value at this[0] to value of D
```

Note that this is not the only valid solution. Anything that implements the VM code's stack arithmetic in assembly is valid.

b) The direct compilation results in the following code.

```
@THIS    //load address of this pointer into A
A=M     //load value of this pointer into A
M=M+1      //increment value at this[0]
```

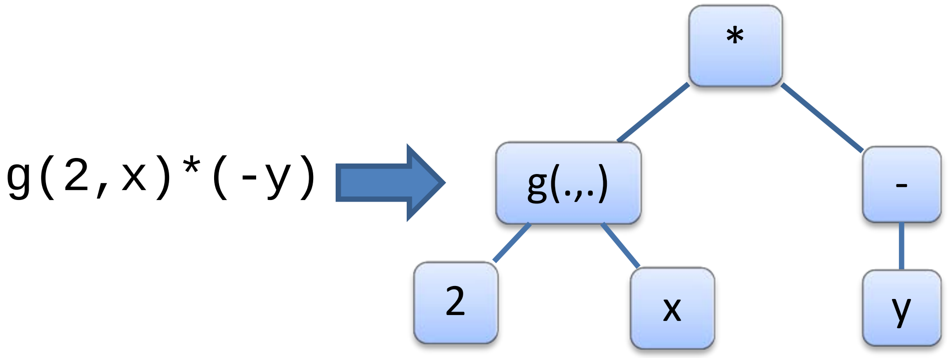
7. Complete the recursive post-order traversal algorithm in the following table.

CodeWrite(exp):	
`if exp is a number or a variable`	`write “push exp”`
`if exp = ( exp1 op exp 2)`	

`if exp = op ( exp1 )`	`call codeWrite(exp1)`
	`write “op”`

`if exp = funct( exp1, … ,expN )`	

Use the algorithm above to generate the pseudo VM code that corresponds to the following parse tree. (5p)



CodeWrite(exp):	
`if exp is a number or a variable`	`write “push exp”`

`if exp = ( exp1 op exp2)`	`codeWrite( exp1 )`
	`codeWrite( exp2 )`
	`write “op”`

`if exp = op ( exp1 )`	`call codeWrite(exp1)`
	`write “op”`

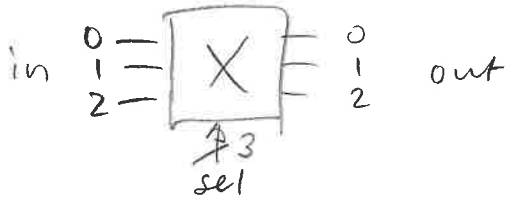
`if exp = funct( exp1, … ,expN )`	`codeWrite( exp1 )`
	`...`
	`codeWrite( expN )`
	`write “call funct N”`

The pseudo VM code is

```
push 2
push x
call g 2
push y
neg
mult
```

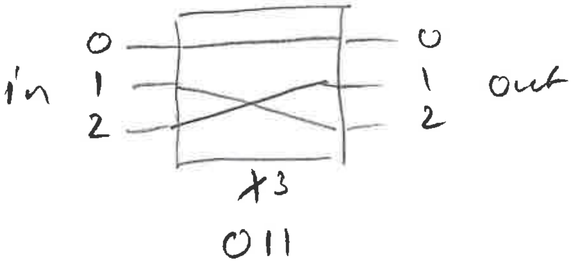
2 Solution of 3x3 switch

a)



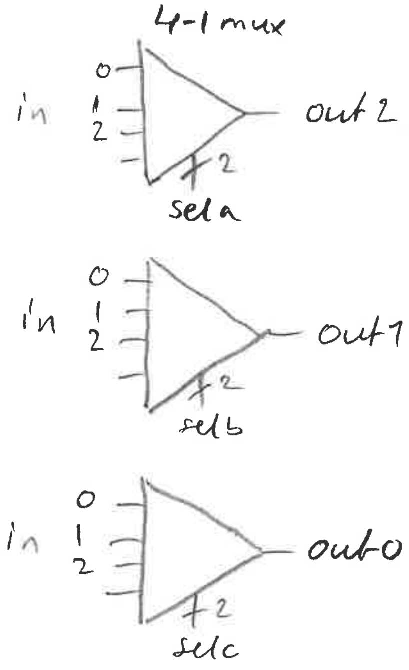
	Out			
	2	1	0	sel
in	0	1	2	000
	0	2	1	001
	1	0	2	010
	1	2	0	011
	2	0	1	100
	2	1	0	101

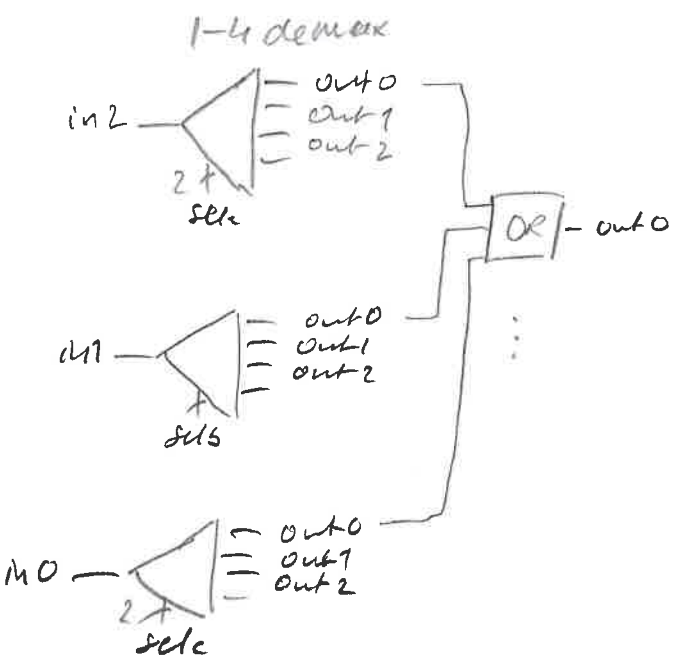
Ex.

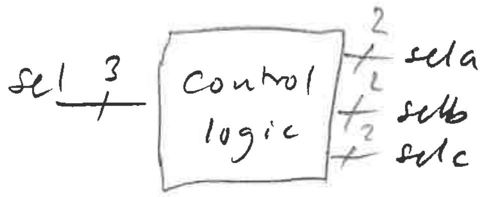


b) Design A

Design B







For design A:

sel	sela	selb	selc
000	00	01	00
001	00	10	01
010	01	00	10
011	01	10	00
100	10	00	01
101	10	01	00

2b (cont'd)

Control logic for design A

sela = sel [2..1]

selb[1]	sel[0]	
sel[2..1]	0	1
00	0	1
01	0	1
10	0	0
11	X	X

X = don't care

selb[1] = Not(sel[2]) AND sel[0]

selb[0]	sel[0]	
sel[2..1]	0	1
00	1	0
01	0	0
10	0	1
11	X	X

selb[0] = Not(sel[0]) OR sel[1
OR sel[2]) OR sel[2] AND
sel[0]

selc[1]	sel[0]	
sel[2..1]	0	1
00	0	0
01	1	0
10	0	0
11	X	X

selc[1] = Not(sel[2]) AND sel[1] AND Not(sel[0])

selc[0]	sel[0]	
sel[2..1]	0	1
00	0	1
01	0	0
10	1	0
11	X	X

selc[0] = sel[2] AND Not(sel[0])
OR sel[0] AND
Not(sel[2] OR sel[1])

c) Follows from block diagram and expressions above.