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()}
      ```
   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)
   c. Give the definition of a variable and of a label in Hack assembly. Write a few lines of code to illustrate.
   d. What is the difference between an object and a class?
   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?
   f. Why do we need a linked list to properly perform memory management; what cannot be achieved by using a single pointer?
   g. Consider the expression (2/x)*(3-y)+1 . Rewrite the expression in prefix notation.
   h. What is combinational and sequential logic and how do they differ?
   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?
   j. How is operator precedence handled by the Jack 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:
      }
      ```

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)

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

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;
}
```

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;
    }
}
```

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)