EP1200 Introduktion till datorsystemteknik

Omtenta fredagen den 22 augusti 2014, 9.00 till 13.00

Possible solutions added with red. Other solutions may of course exist and worth full point.

- Inga hjälpmedel är tillåtna utom de som följer med tentamenstexten och engelskt-svenskt ordbok.
- 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; all kod väl kommenterad.
- Poängen anges för varje uppgift.
- Lösningsförslag anslås på kursens hemsida på KTH Social efter tentamens slut.

1. Give a short answer for each question. (Each correct answer gives 0.5 point.)
   a. Provide the implementation in pseudo-code of the following conversion function:

```
//converts a string to a non-negative number
string2int(string s){

        }
```

```
//converts a string to a non-negative number
string2int(string s){
    v = 0
    i = 0
    while i < s.length
         d = integer value of s[i]
         v = v * 10 + d
return v
}
```

   b. What determines the speed of an adder of two n-bit binary numbers (explain why)?
   A: The number of bits determine the speed since the carry bit needs to ripple through from the least significant bit to the most significant bit.
   c. What is in R2 when the program completes (that is, reaches the infinite END loop)?
   A: The program performs R2=R0+R1, the answer is 8.

Assembly code	RAM	
@R0	0	3
D=M	1	5
@R2	2	
M=D	3	
@R1	4	
D=M	5	
@R2	6	

M=M+D	7	
(END)	8	
@END	9	
0;JMP	10	

d. What determines the speed of sequential logic?
A: The clock frequency which in turn depends on how quickly the internal states of the circuits settle to their final values.

e. What are the differences between the Hack machine language and Hack assembly?
A: The machine code is expressed in binary values while assembly is in ASCII characters; assembly contains variable names and labels, while machine code instead refers to RAM and ROM memory location..

f. Rewrite this expression in postfix notation: 7*(y-1)/(x+3).
A: 7, y, 1, sub, mult, x, 3, add, div

g. What is the difference between tokenizing and parsing, when compiling from Jack to VM language?
A: Tokenizing is the process of grouping the input characters of a program into language atoms. Parsing is to attempt to match the atom stream resulting from the tokenizer to the syntax rules (grammar rules) of the underlying language.

h. Design a circuit for testing that two binary inputs are identical using NAND gates only.
A: A possible solution is like this. For inputs x and y: NAND(NAND(x, y), NAND (Not x, Not y)); where Not(x)=NAND(x, x)

i. What is the difference between token, terminal and non-terminal?
A: A token is the language atom , identified by the tokenizer. A token is a terminal. Terminals can be combined using grammar rules to form non-terminals

j. Show a possible translation of the following pseudocode into a pseudo VM code.

```
while( condition ){
        statement1;
        statement2;
}
statement3;
```

```
label L1
  VM code for computing ~(condition)
  If-goto L2
  VM code for statement1
  VM code for statement2
  goto L1
label L2
  statement3
```

2. Build a 4-bit shifter: `out[(i+shift) modulo 4] = in[i]`.
   a. Draw a block diagram. (2p)
   b. Give the chip in HDL. (3p)

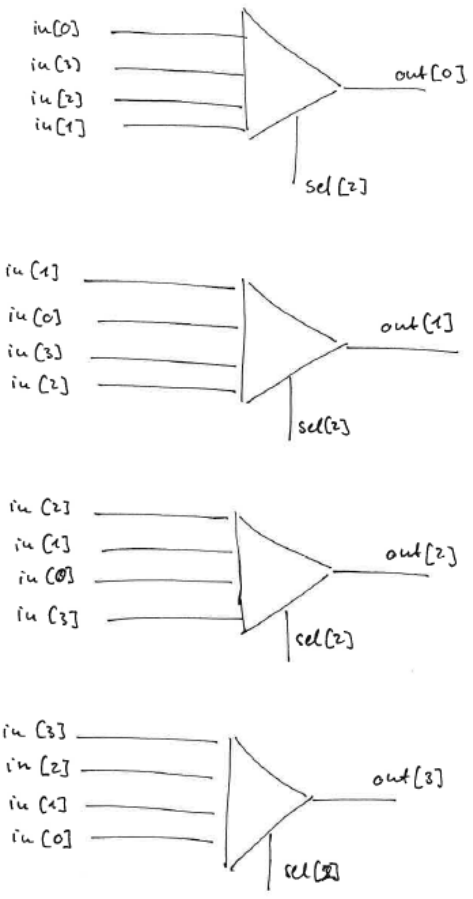
```
CHIP Shift4Way {
    IN in[4], shift[2];
    OUT out[4];

    PARTS:
    // Put your code here:

}
```

A: according to the definition `out[(i+shift) modulo 4] = in[i]` the shifter does this:
If shift=0 (binary:00), then out[i]=in[i] (i denotes the bits from 0 to 3)
If shift=1 (binary:01), then out[0]=in[3], out[1]=in[0], out[2]=in[1], out[3]=in[2]
If shift=2 (binary:10), then out[0]=in[2], out[1]=in[3], out[2]=in[0], out[3]=in[1]
If shift=3 (binary:11), then out[0]=in[1], out[1]=in[2], out[2]=in[3], out[3]=in[0]

This can be implemented with 4 way multiplexers, with `shift` as `sel`.



```
CHIP Shift4Way {
    IN in[4], shift[2];
    OUT out[4];
PARTS:
  Mux4Way(a=in[0], b=in[3], c=in[2], d=in[1], sel=shift, out=out[0];
  Mux4Way(a=in[1], b=in[0], c=in[3], d=in[2], sel=shift, out=out[1];
  Mux4Way(a=in[2], b=in[1], c=in[0], d=in[3], sel=shift, out=out[2];
  Mux4Way(a=in[3], b=in[2], c=in[1], d=in[0], sel=shift, out=out[3];

}
Mux4Way  is given, or can be implemented as:
CHIP Mux4Way {
    IN a, b, c, d, sel[2];
    OUT out;
PARTS:
  Mux(a=a, b=b, sel=sel[0] , out=out1);
  Mux(a=c, b=d, sel=sel[0] , out=out2);
  Mux(a=out1, b=out2, sel=sel[1], out=out);
}
```

3.

a. Write a HACK Assembly program that stays in a loop if no key is pressed on the keyboard. Once a key is pressed, the program leaves the loop, and sets key_pressed=1, where key_pressed is a user defined variable. (3p)

```
//  stays in LOOP until key is pressed, then writes key_pressed=1
0         @key_pressed  // here we initialize key_pressed=0
1         M=0
(LOOP)
2         @KBD      // here we check whether key is pressed
3         D=M
4         @LOOP
5         D; JLE
6         @key_pressed  // key pressed, we set key_pressed=1
7         M=1
(END)
8         @END
9         0;JMP   // Infinite loop
```

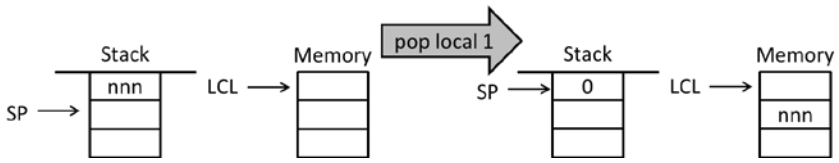
b. List all labels and variables you had to use in addition to the ones predefined in HACK, and assign a correct memory location for them. (1p)
If no other variables were defined before this piece of code, then
`key_pressed: 16 (RAM)`
`LOOP: 2 (ROM)`
`END: 8 (ROM)`

c. Mark all the C-instructions in your code. (1p)
Shown with bold in the code

4. Consider the following VM command: `pop local 1`

a. Draw the stack and the relevant part of the memory before and after this operation. Show clearly the stack pointer. (2p)

b. Give the Assembly code that implements this VM command. (3p)



```
// pop local 1: stack -> local 1
@LCL      // find the local segment address in the memory
D=M
D=D+1     // address of local 1
@R13
M=D       // address stored in virtual register R13
@SP       // find address of stack pointer
AM=M-1    // decrease with one, store address in A
D=M       // store value from top of stack in D
M=0       // clean the now empty stack position
@R13      // go to local 1
A=M
M=D       // write value removed from stack to local 1
```

5. Implement a Hack VM function that takes two numbers as arguments, and returns true if the sum of these two numbers exceeds 100. Otherwise it returns false. (5p)

```
function sumcompare 0
push arg 0
push arg 1
add
push constant 100
gt
if-goto ret_true
push 0
return
label ret_true
push -1
return
```

6. Why does `3 + 7*8` not equal to `3 + (7*8)` in Jack? In order to answer the question, identify the Jack grammar rules matched by the two expressions above. List all the non-terminals involved in the matching.

   A:

   `3 + 7*8 : expression term integerConstant op term integerConstant op term integerConstant`

   `3 + (7*8): expression term integerConstant op term` **`expression`** `term integerConstant op term integerConstant`

7.

a. Complete the implementation of the following function which is supposed to draw a line on screen. `(sx,sy)` are the coordinates of the beginning of the line and `(ex,ey)` are the coordinates of its end. Assume that `sx<ex` and `sy<ey`. The functions `Screen.drawPixel(int x,int y)` and `Math.divide(int x, int y)` are already implemented and available. (2p)

```
function drawLine(int sx, int sy, int ex, int ey){
      var int current_x,current_y,dx,dy;
      let current_x = 0;
      let current_y = 0;
      let dx = ex - sx;
      let dy = ey - sy;
      while (current_x <= dx && current_y<=dy){
           Screen.drawPixel(sx + current_x,sy + current_y);

           // write here the code to increment current_x
            // or to increment current_y
           if (Math.divide(current_x,current_y)
                      < Math.divide(dx,dy)){
                    let current_x = current_x + 1;
            }
            else {
                    let current_y = current_y + 1;
            }

      }
}
```

b. Explain how the function `Screen.drawPixel(int x,int y)` is implemented in the HACK OS. (1p)
A: The screen is represented with the memory map. Pixel is drawn by setting a bit with `poke`. Requires some care as the pixel is one bit, while `poke` sets an entire 16 bit.

c. Provide the pseudo code of the function `Screen.drawHorisontalLine(int x, int y, int length)` that draws an horizontal line on screen from position `(x,y)` ending at **`(x+lengh, y)`**. (2p)

```
function drawHorisontalLine(int x, int y, int length){
      var int current_x;
      let current_x = 0;
      while (current_x <= length){
           Screen.drawPixel(x + current_x, y);
           let current_x = current_x + 1;

      }
}
```

(These implementations do not care about running out of the screen, which could be added…)