CPSC 201: Introduction to Computer Science
Carsten Schürmann
Date: February 28, 2002

Homework 4

Due: 11:30am
Wednesday, February 27, 2002.

Problem 1: Linear Search (50 points)

Linear search is the simplest algorithm to search for an element in an list. It makes no assumptions regarding the organization of the list over which search is to be performed. It simply traverses a list step by step and compares every element with the one we are looking for.

Write an assembly language program for linear search for a microprocessor described below. You don't have to implement your program, but it should be "in working condition". Note, that in machine programs we don't have the convenient argument passing machinery as in high level languages. Arguments are passed through registers, and so should your implementation of the linear search routine. Our microprocessor has five registers labeled R_0, R_1, R_2, R_3, R_4 and understands the machine instructions given below.

The input arguments to your program will be stored in registers as follows. The byte we would like to find in memory is contained in register R_1, the address where the search is to start from in register R_2, the address where to stop in register R_3. Your machine program is expected to traverse the memory in between addresses R_2 and R_3 print `1` on the screen if the search was successful, `0` if it was not.

As example consider the contents of memory as follows:

```
2000  54
2001  92
2002  45
2003  40
2004  54
2005  *
```

and the registers set as follows R_1 =`40`, R_2 =`2000`, and R_3 =`2005`. Your program should print a `1` on the screen. Here is a summary of the commands you are allowed to use. We use A for addresses.

`LOAD` A R_i: Load the contents of memory at address A into register R_i.

`LOADI` R_i R_j : Load the contents of memory at the address contained in register R_i into register R_j.

`MOVE` R_i R_j: Move the contents of register R_i to register R_j.

`STORE` R_i A: Store the contents of register R_i into memory at address A.

`ADD` R_i R_j: Add contents of registers R_i and R_j and store the result in register R_0.

`MULT` R_i R_j: Multiply contents of registers R_i and R_j and store the result in register R_0.

`CONST` C R_i: Moves constant C to register R_i.

`JMP` A: Jump to memory address A and continue program execution from that location.

`CJMP` A: Jump to memory address A iff register R_0 is zero.

`OUT` R_i: Output contents of register R_i.

`HALT`: Halt the program.

Hints and Assumptions:

- Assume that memory has 1000 locations from address 0 to address 999. You might want to divide the memory into two portions, one to store the program and other to store the data.
- Each instructions requires just a single memory address.

We shall assume that when the machine code for linear search is invoked, register $R1$ contains the value to be searched and registers $R2$ and $R3$ contain the starting and ending addresses of the data. The convention we follow is that the last element is at the memory location one less than the address contained in register $R3$. We shall also assume that memory addresses 0 to 499 form the data space and the memory addresses 500 to 999 form the instruction space. Note that our data need not take the entire data space, so starting and ending memory addresses need not be 0 and 499 respectively.

```
600 CONST -1,R4
601 LOADI R2,R0              # Load the value at address R2
602 MULT R0,R4
603 ADD R0,R1                # Perform the comparison
604 CJMP 612                 # Jump if value found
605 MULT R4,R4               # Get 1 in R0
606 ADD R0,R2                # Increment R2
607 MOV R0,R2
608 MULT R2,R4
609 ADD R3,R0                # Check if end of array reached
610 CJMP 615                 # Jump if end of data reached
611 JMP 601
612 MULT R4,R4               # Get 1 in R0
613 OUT R0                   # Value found
614 HALT
615 CONST 0,R0
616 OUT R0                   # Value not found
617 HALT
```

Problem 2: Largest Element (10 points)

Write an SML function `max` to find the largest element in a list of real numbers.

1. What is the type of this function?

   `max : real list` $\rightarrow$ `real`

2. What is the invariant of this function?

```
max : real list $\rightarrow$ real
max L = x

Invariant:
If x is the largest element of the list L then max L = x
```

3. Write the function itself.

```
fun max ((x::[]) : real list) : real = x
  | max ((x::xs) : real list) : real = let
                                         val m = max xs
                                        in
                                         if x < m then m else x
                                        end
```

Problem 3: Boolean Logic (40 Points)

A majority function $M_n(x_1, \ldots, x_n)$ is *true* if a majority of its inputs are *true* and $false$ otherwise (assume n odd). In this assignment we study majority functions with *three* inputs.

1. Write the truth table for a majority function $M_3(x_1, x_2, x_3)$ with just three inputs.

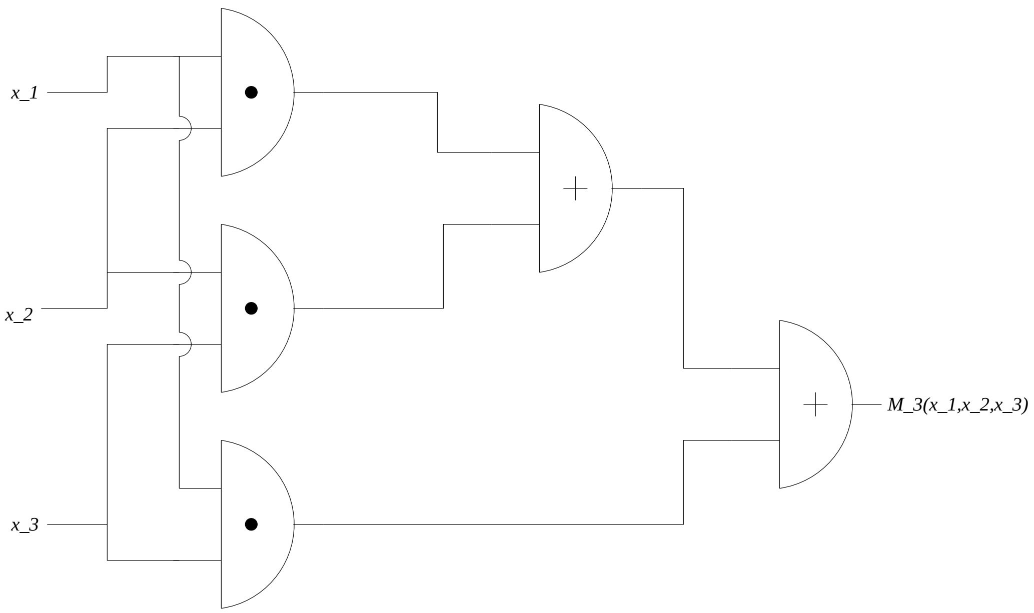
x_1	x_2	x_3	$M_3(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

2. Write the Boolean expression for this majority function.

$$M_3(x_1, x_2, x_3) = \overline{x}_1 x_2 x_3 + x_1 \overline{x}_2 x_3 + x_1 x_2 \overline{x}_3 + x_1 x_2 x_3$$

   simplifies to

$$M_3(x_1, x_2, x_3) = x_1 x_2 + x_2 x_3 + x_3 x_1$$



3. Construct the circuit for $M_3(x1, x2, x3)$ using only AND and OR gates.
   Hint: A typical implementation uses three AND gates and three OR gates. Reformulate your answer from 2. using distributivity laws.

4. Does $M_3(x_1, x_2, x_3)$ and NOT gate form a complete Boolean basis? Justify your answer.
   Hint: A NAND Gate alone, a NOR Gate alone, or the set of AND, OR and NOT gate form a complete Boolean basis. Try to implement one of these (or possibly other basis) using the $M_3(x_1, x_2, x_3)$ and a NOT gate.

   An implementation of a $NAND$ Gate is shown using $M_3(x_1, x_2, x_3)$ and a NOT Gate. We know from above that

$$M_3(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_1x_3$$

   Therefore,

$$M_3(x_1, x_2, 0) = x_1x_2 + x_20 + x_10 = x_1x_2$$

   Thus,

$$NOT(M_3(x_1, x_2, x_3)) = \overline{x_1x_2} = NAND(x_1, x_2)$$