Problem Set One Due on Thu 2/22/96	**Advanced Computer Architecture** CLA CS-550 / Spring 1996	Prof. Azer Bestavros Email: best@cs.bu.edu

State your assumptions clearly and explain if necessary
Your grade will depend on the clarity and soundness of your assumptions

1

In this problem we will get acquainted with two software tools designed to experiment with the DLX architecture: the `dlxgcc` (a GNU C compiler for the DLX architecture) and the `dlxsim` (a simulator for the DLX architecture). The software we will be using is installed under the directory "/home/course/cs550/dlx/bin".

- ◇ The function `fact()` shown below is a recursive program to compute the Factorial of an integer `n`.

```
int fact(n)
  int n ;
{
  if (n <= 1)
    return(1) ;
  else
    return(n*fact(n-1)) ;
}
```

```
void main()
{
  int i, j ;

  i = 4 ;
  j = fact(i) ;
}
```

- ◇ Type in the above program and compile it using `dlxgcc`. To do so, invoke `dlxgcc` with the name of the file where your factorial program resides as a parameter. In the absence of any errors, `dlxgcc` will produce a DLX assembly language program (with a ".s" extension) in the same directory where the source (C) code resides. Show a listing of the compiled code (produced by `dlxgcc`).
- ◇ Identify the function of each section of the compiled code (e.g. initializations, function call sequence, function return sequence, ... etc.) Also, identify the assembly code segments for each one of the instructions in the program.
- ◇ Show a memory map (basically the stack and the activation record) for each invocation of `fact()`.
- ◇ Run the compiled program using `dlxsim`. Refer to the document explaining the use of `dlxsim` for a full description of the capabilities of the simulator. Compare the dynamic instruction mix (you can obtain that using the `stat` command) of the above program with the instruction mixes for the benchmarks we discussed in class. Comment.

2

In this problem (and the next) we will experiment with a stack architecture: the SLX machine. The SLX has an instruction set similar (to some extent) to the DLX instruction set. The following is a description of the SLX ISA.

The SLX is stack machine, which means that it is a zero-address machine. The SLX is a 32-bit machine. All of the ALU operations are performed on operands stored on top of the SLX stack. The two topmost elements of the stack are processor registers and are called S0 and S1. The remainder of the stack is stored in a *stack memory*, which is separate from *main memory*. Access to the main memory is through the use of PUSH (load) and POP (store) primitives.

The instructions of the SLX have two formats. The first is 1-byte long and the second is 4-byte long.

0	7-bit opcode

1	7-bit opcode	24-bit constant

The following is a partial specification of the ISA of the SLX.

`ADD`	`ST0` ← `ST0 + ST1`
`ADD nnn`	`ST0` ← `ST0 + nnn`
`NEG`	`ST0` ← `-ST0`
`SHR`	`ST0 >> 1`
`AND`	`ST0` ← `ST0 & ST1`
`AND nnn`	`ST0` ← `ST0 & nnn`
`NOT`	`ST0` ← `!ST0`
`NOT nnn`	`ST0` ← `!nnn`
`GT`	`ST0` ← `(ST0 > ST1) ? 1 : 0`
`GT nnn`	`ST0` ← `(ST0 > nnn) ? 1 : 0`
`EQ`	`ST0` ← `(ST0 == ST1) ? 1 : 0`
`EQ nnn`	`ST0` ← `(ST0 == nnn) ? 1 : 0`
`BR`	`IP` ← `IP + ST0`
`BR nnn`	`IP` ← `IP + nnn`
`BRZ`	`IP` ← `IP + (ST0 == 0) ? ST1 : 1`
`BRZ nnn`	`IP` ← `IP + (ST0 == 0) ? nnn : 1`
`PUSH`	`ST0` ← `M[ST0]`
`PUSH nnn`	`ST0` ← `M[nnn]`
`POP`	`M[ST1]` ← `ST0`
`POP nnn`	`M[nnn]` ← `ST0`
`PCPUSH`	`ST0` ← `PC`
`PCPOP`	`PC` ← `ST0`

◇ Hand-compile the following assignment statement into code for <u>both</u> the DLX and SLX architectures.[1]

$$A = ((K == I)\&\&(J > 7))||(K > J)$$

Assume that A, I, J, and K are all integers. An integer value of "1" denotes TRUE, while a value of "0" denotes FALSE. Assume that the values of the variables A, I, J, and K, have been stored in memory prior to the execution of this statement. Use &A, &I, &J, and &K to denote the address of these variables.

◇ For each of the two machines, indicate statements where the compiler would have to make register-allocation decisions. What is the status of the decision ? For the DLX machine you should try to use as few registers as you can.

◇ How many statements are there in the two programs (count in bytes) ? Identify the principal architectural characteristics responsible for the difference.

◇ Hand-compile the above statement for the DLX and SLX machines by taking into consideration the following optimization: If one input sub-expression for an "AND" operator is FALSE then the second sub-expression need not be evaluated. When would that optimization be worth it ?

◇ Most of the SLX operations have ST0 as both an input operand and an output operand. Does this constitute a problem in the design of a pipelined datapath for this architecture? Why ?

3

In this problem, we will design a datapath for the SLX architecture.

◇ We will use the tailor-made stack memory described below:

- The stack memory has one input and two-output 32-bit-wide buses for data input/output. The stack memory has a 2-bit control input which works as follows:

c0	c1	meaning
0	0	No operation
0	1	Pop top of the stack to output bus #0
1	0	Pop top of the stack to output bus #1
1	1	Push data from the input bus on top of the stack

[1] You might want to use `dlxgcc` to find out how it compiles it.

– Moreover, the stack memory has 2 status bits, which work as follows:

c0	c1	meaning
0	0	Normal operation
0	1	Stack underflow
1	0	Stack overflow
1	1	Impossible!

- ◇ The SLC's main memory can be accessed using two registers a memory-address register (MAR) and a memory-data register (MDR). It has two input control bits (Read/Write & Select/noselect) and one output status bit (Ready/Busy).
- ◇ The SLC's ALU is identical to the ALU of the DLX architecture.

Design a datapath for the SLC architecture. Your design should show (at least) the following features:

- ◇ All the buses labeled with their width and direction of dataflow,
- ◇ All the control points that can be used by the controller as outputs,
- ◇ All the status bits that can be used by the controller as inputs,
- ◇ All the registers in the datapath.

Discuss the potentials for pipelining the SLX architecture.[2] What are the aspects of the SLX ISA that make the pipelined design more (or less) complicated or straightforward. How many pipeline stages would you suggest for the SLX? Why?

4

Solve problems 1.1, 1.6, 1.7, 1.12, 1.13, 2.2 from the textbook (Hennessy and Patterson, 2nd Edition). Note this is from the *new* edition.

Good Luck!

[2] You might want to think about this problem without necessarily sticking to your datapath design.