LSU EE 4720 **Homework 2** **Due: 7 March 2003**

Design a stack ISA with the following characteristics:

- Memory has a 64-bit address space and consists of 8-bit characters.
- The stack consists of 64-bit registers.
- The ISA uses 2's complement signed integers.
- Only add other data types as necessary.

The stack ISA must realize these goals:

- Small program size.
- Low energy consumption. (For here, assume energy consumption is proportional to dynamic instruction count.)
- Relatively simple implementation. Instructions should be no more complex than RISC instructions.

Design the instruction set based on the sample programs in the problems below and the following:

Arithmetic and Logical Instructions

They should read their source operands from the top of stack (top one or two items) and push their result on the top of stack. Arithmetic instructions cannot read memory and they cannot read beyond the top two stack elements. (That is, you can't add an element five registers down to one ten registers down. Instead use rearrangement instructions before the add.) Specify whether the arithmetic and logical instructions pop their source operands. One can have both versions of an instruction. For example, `add` might pop its two source operands off the stack while `addkeep` might leave the two operands:

```
# Stack:  26 3 2003
add
# Stack:  29 2003
addkeep
# Stack:   2032 29 2003
```

Remember that arithmetic and logical instructions cannot rearrange the stack and cannot access memory.

Immediates

Decide how immediates will be handled. There can be immediate versions of arithmetic instructions or one can have push immediate instructions. See the example below. Keep in mind that the register size is 64 bits.

```
# Stack: 123
addi 3   # An immediate add.
# Stack: 126
pushi 3
# Stack: 3 126
add
# Stack: 129
```

Load and Store Instructions

Memory is read only by load instructions which push the loaded item on the stack. Memory is written only by store instructions which get the data to store from the top of the stack. Determine which addressing modes are needed for loads and stores, and design instructions with those modes.

Stack Rearrangement Instructions

The stack rearrangement instructions change the order of items on the stack. Consider adding the following: `exch`, swaps the top two stack elements. `roll n j`, remove the top `j` stack elements and insert them starting after what was the `n`th stack element. (See the example.) Other stack rearrangement instructions are possible.

```
# stack 11 22 33 44 55 66
exch
# stack 22 11 33 44 55 66
roll 5 2
# stack 33 44 55 22 11 66
```

Control Transfer Instructions

Your ISA must have instructions to perform conditional branches, unconditional jumps, indirect jumps, and procedure calls. It must be possible to jump or make a procedure call to anywhere in the address space. (The only thing special the instruction used for a procedure call has to do is save a return address.) The branch instructions can (but do not have to) use a condition code register. No other registers can be used (other than those in the stack). Don't forget about the target address.

Problem 1: As specified below, describe your ISA and the design decisions used. (Don't completely solve this part until you have solved the other problems.)

(*a*) For each instruction used to solve the problems below or requested above, show the assembler syntax and the instruction's coding. The coding should show the opcode, immediate, and any other fields that are present. Don't forget the design goals. Also don't forget about control transfer targets.

There is no need to list a complete set of instructions, but for coding purposes assume their existence. (There must be a way of coding a complete set of instructions that realize the goals of this stack ISA.)

(*b*) Determine the size of the stack. Specify instruction coding and implementation issues used to determine the size.

(*c*) Explain your decision on whether there are immediate versions of arithmetic instructions. (The alternative is instructions like `pushi`.)

(*d*) Explain your selection of memory addressing modes. Also, pick an addressing mode that you did not use and explain why not.

(*e*) Explain how other design decisions you have made help realize the goals of small program size, low energy, or simple implementation.

(*f*) Describe any design decision you made that involved a tradeoff between code size, energy, or implementation simplicity. (Pick any pair.) *The original question asked only about code size and energy.* If you didn't make such a decision make one up.

Problem 2: Re-write the following MIPS code in your stack ISA.

```
lui $a0, %hi(array)       # High 16 bits of symbol array.
ori $a0, $a0, %lo(array)  # Low 16 bits of symbol array.
jal lookup                # The name of a routine.
nop
```

Problem 3: Re-write the solution to Homework 1 in your stack ISA, use the template below. (Use your own solution or the one posted.)

```
lookup:
        # Call Arguments  (TOS is the top of the stack.)
        #
        # TOS:     Return address
        # TOS + 1: ADDR of first element of array.  Array holds 64-bit integers.
        # TOS + 2: Number of elements in array.
        # TOS + 3: TARGET, element to count.
        #
        # Return Value
        #
        # TOS: Number of times TARGET appears in the array starting at ADDR.

        # Solution Here
        #
        # [ ] Don't forget the return.
```