Practice Sheet 2

CSE 112 Computer Organization

The instructions supported by the ISA are mentioned in the table below. **The ISA has 16 General purpose registers: r0 to r15**

Name	Semantics	Syntax
Add	Performs reg1 = reg2 + reg3	`add reg1 reg2 reg3`
subi	Performs reg1 = reg2 - Imm	`sub reg1 reg2 $Imm`
mvi	Performs reg1 = Imm	`mov reg1 $Imm`
Mov	Performs reg1 = reg2	`mov reg1 reg2`
Branch not equal	Branch to addr if reg1!= reg2	`bneq reg1 reg2 addr`
Branch and link	Jumps to label after saving the return address to r1.	`brl label`
Push	Pushes the data stored in reg1 onto the stack	`push reg1`
Pop	Pops the data stored on the top of the stacks into reg1	`pop reg1`
mul	Performs reg1 = reg2 * reg3	`mul reg1 reg2 reg3`

Apart from the above instructions, the assembler and the operating system support the following subroutines:

Name	Semantics	Syntax
Input	Reads immediate data from user into reg	`in reg`
Output	Prints str on the console	`out "str"`

Caller-callee conventions:

The following are the caller callee convention:

- There are 15 registers r0 to r15.
- r15 - program counter.
- r0 - stack pointer.
- r1 - link register and return address
- r2 - return value.
- r3 and r4 holds the first and second argument to the callee
- The stack is automatically managed by push and pop.
- All the registers from r1-r7 are caller saved. On the other hand, registers r8-r14 are callee saved.
- Whenever the branch and link instruction is used, the return address is stored in r1 and the program counter jumps to the given label.

Q1: Convert the following high level code into assembly language. Follow the caller-callee conventions mentioned above.

You can only use callee saved registers for storing variables in bar functions and caller saved registers for foo function for storing variables.

```
int baz(int a,int b)

{

      return a+b;

}

int bar()

{

      int a = 10;

      int b = 100;

      int c = 1000;
```

```
    int d = baz(a,b);
    return a+b+c+d;
}
int foo()   // Use only caller saved registers
{
    int a = 10;
    int b = 100;
    int c = bar();
    int d = baz(a,b);
    return a+b+c+d;
}


int main()
{
    return foo();
}
```

Q2. Write a function myfunc which computes factorial of a number n passes as an argument in assembly language. Use ISA as provided in Q1.

a. Use iterative call

Assumptions for part b:

1. The number whose factorial is to be calculated (n) is present in r5.

b. Use recursive call

Assumptions for part b:

1. Return Address for myfunc is present in link register r1.
2. The number whose factorial is to be calculated (n) is present in register stack.

A factorial of a number n is:

n*(n-1)*(n-2)----*1

Solution Q1:

```
baz:
        add r2 r3 r4        // Add the arguments and return it
        mov r15 r1          // Return
        bar:
        push r8             // Push callee saved register
        push r9             // Push callee saved register
        push r10            // Push callee saved register
        push r11            // Push callee saved register

        mvi r8 #10          // r8 is a
        mvi r9 #100         // r9 is b
        mvi r10 #1000       // r10 is c

        mov r3 r8           // Prepare first argument of baz
        mov r4 r9           // Prepare second argument of baz
        push r1             // Push caller saved register
        brl baz             // call baz function
        pop r1              // Pop caller saved register
        mov r11 r2          // r11 is d

        mvi r2, #0          // Initialize sum with 0
        add r2 r2 r8        // Add a
        add r2 r2 r9        // Add b
        add r2 r2 r10       // Add c
        add r2 r2 r11       // Add d

        pop r11             // Pop callee saved register
        pop r10             // Pop callee saved register
        pop r9              // Pop callee saved register
```

```
      pop r8           // Pop callee saved register
      mov r15 r1       // Jump back to caller function

foo:  push r1          // Push caller saved register
      brl bar          // Push caller saved register
      pop r1           // Push caller saved register
      mov r6 r2        // r6 is c

      mvi r3 $10       // r3 is a, send the first argument to baz
      mvi r4 $100      // r4 is b, send the second argument to baz
      push r1          // Push caller saved register
      push r3          // Push caller saved register
      push r4          // Push caller saved register
      push r6          // Push caller saved register
      brl baz          // Call baz function
      pop r6           // Pop caller saved register
      pop r4           // Pop caller saved register
      pop r3           // Pop caller saved register
      pop r1           // Pop caller saved register


      add r2 r2 r6     // Prepare return value
      add r2 r2 r3     // Prepare return value
      add r2 r2 r4     // Prepare return value
      mov r15 r1       // Jump back to caller function

 main: push r1         // Push caller saved register
  brl foo        // Call foo function
  pop r1         // Pop caller saved register
  mov r15 r1     // Jump back to caller function
```

Solution Q2 (a)

```
myfunc:     mvi r6 $1
            mvi r7 $0
            Bneq r5 r7 MulLoop
            mvi r2 $0
            Pop r1
            Mov r15 r1
MulLoop:    Mul r6 r6 r5            // n is saved in r5
            subi r5 r5 $1
            Bneq r5 r7 MulLoop
            Pop r1
            Mov r15 r1
```

Solution Q2 (b):

```
myfunc:     pop r5
            Push r1
            Mvi r7 $0
            Bneq r5 r7 Done
myfunc2:    pop r5
            Push r1
            mvi r7 $1
            Bneq r5 r7 else
Done:       mvi r2 $1
            Pop r1
            Mov r15 r1
Else:       mov r8 r5
            subi r5 $1
            push r8
            push r5
```

brl myfunc2

pop r8

mul r2 r2 r8

Pop r1

Mov r15 r1