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`
Sub	Performs reg1 = reg2 - reg3	`sub reg1 reg2 reg3`
Mov Imm	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`

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”`

<u>Caller-callee conventions:</u>

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()    // Use only callee saved registers
{
      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();
}
```

Solution:

```
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

      mov r8 #10              // r8 is a
      mov r9 #100             // r9 is b
      mov 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

      mov 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                 // Call bar function
      pop r1                  // Pop caller saved register
      mov r6 r2               // r6 is c

      mov r3 $10              // r3 is a, and the first argument to baz
      mov r4 $100             // r4 is b, and 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
```

Q2. Below is the high level code for myfunc. Write the assembly code for the same.

```
int myfunc (int n)
{
    if (n==0)
    {
        return 0;
    }
    return n + myfunc(n-1);
}
```

Solution:

```
fact:
     mov r5 $1
     mov r4 $0
     bneq r3 r4 Continue   // Jump to continue if r3 != r4
     Return:               // Base case
          mov r2 $0        // Return 0
          mov r15 r11
     Continue:             // Recursive case
          mov r6 r3        // Store n in r6
          sub r3 r3 r5     // Prepare n-1 for next call
          push r1          // Caller saved
          push r6          // Caller saved
          brl fact         // Recurse
          pop r6           // Restore n
          pop r1           // Restore return address
          add r2 r2 r6     // Perform n + myfunc(n-1) and return it
          mov r15 r1
```