

Computer Organisation COMP2008


Lab Sheet 6 (starts session week 7, [Week 8: intra-session break] due in week 9)

Student Name and Number	
Date, Grade and Tutor signature, max mark 4	

Keep this cover sheet marked and signed by the tutor.

1. Preparation

The main goal of today's lab is to understand the assembly language techniques for procedure calls, and stack usage conventions. References: the lecture notes, the textbook paragraphs: 3.6 from Ed2, or 2.7 from Ed3, or 2.8 from Ed4, 2.8 from Ed5, 2.8 from Ed6, and A.6 from **HP_AppA.pdf** (a downloadable PDF file on the website).

General Data	UnitOutline LearningGuide Teaching Schedule Aligning Assessments 
Extra Materials	ascii_chart.pdf bias_representation.pdf HP_AppA.pdf instruction_decoding.pdf masking_help.pdf PCSpim.pdf PCSpim Portable Version Library materials

Self study task: Use any materials of your choice to study iterative and recursive programming techniques (HP_AppA.pdf is a good starting point). Illustrate your explanation with a hand drawing. Using any example explain the difference between iterative and recursive method of solving problems. You can use fragments of the code provided with this lab, but you can also use any other code or a simple mathematical formula (for example the Fibonacci string).

2. Workshop Tasks [Total max. mark: 4]

Get assembly language files *leafsample.s*, *iterativesample.s* and *simplerecursive.s* for this lab. Open and run in PC SPIM programs *leafsample.s* and *iterativesample.s*. Experiment with these programs to **understand how they work**, specifically how the procedures are called from the main program. Observe activity on the stack when single stepping. Be sure that you understand how the stack is used.

TASKS SUMMARY: Write MIPS programs which consist of sub-routine *fib* to calculate the value of the *n*-th member of the Fibonacci series, and a main program which calls *fib* with the parameter *n*.

Hint: Fibonacci string of numbers is defined as follows:

- The first element is: $F_0=0$
- The second element is: $F_1=1$
- All other elements are: $F_n = F_{n-1} + F_{n-2}$.

For example: $F_3 = F_2 + F_1 = (F_1 + F_0) + F_1 = (1+0) + 1 = 2$. The string goes like that: 0,1,1,2,3,5,8,13,...

DETAILED TASKS DESCRIPTION:

- Task I (1 mark):** Write procedure *fib* based on iterative approach to calculate F_n where value *n* is read from the keyboard and stored in your main program that is calling *fib*.
 - The main program reads number 'n' -- restrict range of 'n' to 2-45, and display an error message if this condition is not satisfied.
 - The main program calls *fib* with parameter passing. Procedure *fib* should return the calculated value of the *n*-th Fibonacci member to the main program. Note: Data transfer between the main program and procedure *fib* should be managed gracefully via procedure calling interface, namely, **via parameter passing (\$a registers to be used) and result return (\$v registers to be used)**. Avoid data sharing via registers globally.
 - The main program outputs the calculated value of the *n*-th member of the Fibonacci string. For example if 'n' read from the keyboard was 7, the output should be:

```
Element [7] of Fibonacci string is: 13.
```

Other Fibonacci members for your testing purpose:
Element [8] of Fibonacci string is: 21.
Element [21] of Fibonacci string is: 10946.
Element [35] of Fibonacci string is: 9227465.
Element [41] of Fibonacci string is: 165580141.
 - Make sure to follow the conventions for registers usage in your program.

Hint: The best approach is to use *iterativesample.s* as a base for your new program. Rename it, remove (or comment out) blocks not required in your program, and add one by one new blocks. Check proper operation of each added block before you add another one.

- Task II (1.5 marks):** Modify the code from **Task I** with the following consideration:
 - Restrict range of 'n' to 2-50, and display an error message if this condition is not satisfied.
 - The main program calls *fib* with parameter passing.
 - You would find that, after F_{47} , the Fibonacci numbers will be greater than `INT_MAX` (2147483647 or `0x7FFFFFFF`) and as a consequence an 'arithmetic overflow' occurs.

```
Fib(49)=7778742049; Fib(50)=12586269025
```
 - Rewrite the code by taking the power of double-precision floating point (FP) calculation so as to get around the 'arithmetic overflow' problem.
 - Data transfer between the main program and procedure *fib* should be managed gracefully via procedure calling interface, namely, **via parameter passing (\$a registers to be used) and result return (\$v registers to be used)**; data sharing via FP registers isn't considered a graceful way for this exercise. Work out the most elegant way (parameter passing by ref) for returning result (double-precision FP format) from the procedure *fib* via calling interface; otherwise, don't expect marks at a distinctive level (or above) for this task.

- Follow the conventions for registers usage where possible in your program.
- This task is specifically dealing with larger input of value n from 47 to 50, failed to do this is regarded as a failed implementation and will result in a zero mark.

Note 1: Task I and Task II should be developed and demonstrated separately; Task II doesn't automatically cover Task I.

Note 2: To implement this lab task, you need to use *fp* operations; please refer to **Lect. 05 supplement** lecture slides. Some *fp* instructions are attached below:

MIPS Floating Point Architecture

- **Instructions:** Single Precision, Double Precision versions of add, subtract, multiply, divide, compare
 - Single `add.s`, `sub.s`, `mul.s`, `div.s`, `c.lt.s`
 - Double `add.d`, `sub.d`, `mul.d`, `div.d`, `c.lt.d`
- **Registers:** MIPS provides 32 32-bit FP reg: `$f0`, `$f1`, `$f2` ...
 - FP data transfers:
 - `lwc1` [Load word coprocessor 1], `swc1`
 - Double Precision?
 - Even-odd pair of registers (`$f0#$f1`) act as 64-bit register: `$f0`, `$f2`, `$f4`, ...

MIPS FP arithmetic, branch instructions

```
add.s $f0,$f1,$f2 # $f0=$f1+$f2 FP Add (single)
add.d $f0,$f2,$f4 # $f0=$f2+$f4 FP Add (double)
sub.s $f0,$f1,$f2 # $f0=$f1-$f2 FP Subtract (single)
sub.d $f0,$f2,$f4 # $f0=$f2-$f4 FP Subtract (double)
mul.s $f0,$f1,$f2 # $f0=$f1x$f2 FP Multiply (single)
mul.d $f0,$f2,$f4 # $f0=$f2x$f4 FP Multiply (double)
div.s $f0,$f1,$f2 # $f0=$f1/$f2 FP Divide (single)
div.d $f0,$f2,$f4 # $f0=$f2/$f4 FP Divide (double)

c.X.s $f0,$f1 # flag1= $f0 X $f1 FP Compare (single)
c.X.d $f0,$f2 # flag1= $f0 X $f2 FP Compare (double)
# where X is: eq (equal), lt (less than), le (less than
# equal) to set flag value, which is used by:
bc1t # floating-point branch true [it's digital '1' in
# bc1t, not letter 'l' in bc1t; 't' means true]
bc1f # floating-point branch false
```

- Task III (advanced, 1.5 marks):** Analyse the sample program *simplerecursive.s* It shows example of recursive programming technique (recursion is a process in which an algorithm calls itself). Write a recursive version of the procedure *fib*.
 - Restrict range of 'n' to 2-45, and display an error message if this condition is not satisfied.
 - Unsophisticated/inefficient implementation (e.g. $O(2^n)$ time) may not be working with a larger input (e.g. 45 taking too long time and/or causing stack overflow). If this is the case, don't expect marks at a distinctive level (or above) from this part; you really need an improvement on your implementation of the advanced task.
 - For your information: There are a number of extra insights into this exercise. As an intellectual exercise, clearly you would want to concern yourself with efficiency and accomplish optimization. When developing Embedded Systems, programmers who have an intimate understanding of the assembly language will most likely develop the "best" code.

Be sure to document your program following the style of the programs discussed in lectures, and demonstrated in the lab examples. Insufficient documentation will detract from your mark.

Demonstration: Demonstrate to the tutor the program you wrote. You must be able to explain the code.

3. Assessment notice

When you ready, present to the tutor a printed copy of your program source code, with your name and student number included in the comments (`#...`), and typed or neatly written answers to all questions listed in the lab sheet. Your tutor may decide to keep the source code printout, but you should keep marked and signed cover sheet.

Warning: Any source code duplicated amongst students will result in a zero mark, and possible further action according to the WSU policy on plagiarism.