Computer Organisation 300096

Lab Sheet 8 (starts session week 10, due in week 11)

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

Keep this cover sheet marked and signed by the tutor.

Preparation [Total max. mark: 1]

The main goal of today's lab is to understand the impact of hardware performance improvements on program execution, and to gain some experience with floating point arithmetic. Download the assembly file ***float loop.s*** and PDF file **Amdahls-Law.pdf** to your folder.

References: the lecture notes and the textbook:

- Ed2: §2.1, 2.2, 2.7, relevant parts of 4.8, Ed3: §4.1-4.3, relevant parts of 3.6, Ed4: §5.1-5.3, relevant parts of 3.5, Ed5: §5.1-5.5, relevant parts of 3.5

Self study task [1 mark]: study and solve Exercises 1, 2 and 3 from **Amdahls-Law.pdf** file. [It's optional to solve Exercises 4-6, which is non graded task].

Hint: the total execution time is the sum of the execution time for fp-calculations and the execution time for non-fp calculations. The improvement affects the execution time of fp-calculations only. It's helpful to work on the preparation questions together with the *Workshop Tasks* below to examine and understand Amdahls Law, which should be re-written to derive the appropriate formula for solving corresponding exercises (use expression to describe the relationship among relevant variables). This work is left to students.

Amdahl's Law (lab 8)

- Observation: The overall **Speedup** depends on the fraction of the time **affected** by the enhancement
  - **Speedup** = $\text{Time}_{\text{before enhancement}}$ **T** / $\text{Time}_{\text{after enhancement}}$ **T'** (1)
    Indicating how many times faster overall
  - $T = t_1 + t_2$ (2)
  - $T' = t_1 + t'_2$ (3)
  - For easy calculation, let **a** = t_2/T; **N** = t_2/t'_2 (**N** times faster on Task_2)

- Consider the following scenario
  - A program 'DEMO' runs for 100 sec, 80% being multiply operations
  - The aim is to make the program 'DEMO' 4 times faster (overall Speedup)
  - Question: how to achieve the above Speedup by improving only the speed of multiplication (calculate N)?

Let T, t_1, t_2 be the total time, non-fp time, and fp time before enhancement respectively.
Let T', t_1, t'_2 be the total time, non-fp time, and fp time after enhancement respectively.
Assume the improvement affects fp calculation only.

Then $T = t_1 + t_2$, $T' = t_1 + t'_2$ and **Speedup = T / T'**

For easy calculation, you can derive formulas in different forms:
Speedup = f(T, t_2, N) or
Speedup = f(a, N)

where ***a*** = t_2/T, ***N*** = t_2/t'_2
… …

Workshop Tasks [Total max. mark: 3]

1. Open and run the program *float loop.s* Experiment with this program and understand how it works. Specifically observe mixture of floating point and non-fp instructions: analyse how, where and why floating point instructions are used.

If you have not solved the Exercise 1 and Exercise 3 from Amdahls-Law.pdf in the preparation step above, solve them now. This will help you with the lab tasks.

2. **Task I (2 marks):** write a MIPS program which implements the calculation steps in Exercise 1 from Amdahls-Law.pdf for a wider range of values entered from the keyboard (it allows arbitrary input values for relevant parameters). **Use floating point instructions where appropriate.** Your program should perform the tasks as listed below:

- Read in from the keyboard decimal fraction value T, where T is the total execution time of some benchmark before the floating point enhancement (in Exercise 1 from Amdahls-Law.pdf, T is 10 sec);
- Read in from the keyboard decimal fraction value N, where N is the amount of improvement to floating point instruction running time only (in Exercise 1 from Amdahls-Law.pdf, N = 5). Restrict the range of 'N' to 0-20, and generate an error message asking for correct input if this condition is not satisfied;
- Read in from the keyboard decimal fraction value t2, where t2 is the time in the original program (before improvement) spent doing floating-point operations (in Exercise 1 from Amdahls-Law.pdf, t2 = 6);
  hint: T, N, and t2 are **decimal fraction numbers**, for example: T=12.35, N=1.25, t2=7.10
- Calculate the Speedup.

3. **Task II (1 mark):** write a program which implements the calculation steps to the Exercise 3 from Amdahls-Law.pdf (while you can reuse the code for Task I, please note that the formula in Task II is different from that in Task I).

- Assume that we enhance the floating-point unit which is running 5 times faster. The program of Task II should produce a table (a list) of overall Speedup for test programs changing with the proportion of fp-calculation time to the total execution time. The proportion of fp-calculation time to the total execution time changes from 0% (0.0) to 100% (1.0). For Task II implementation, use 10% intervals (not 25% intervals as in Exercise 3), i.e. the table will contain 11 rows.
- Illustrate the results with a graph (a hand drawing is sufficient).

To implement lab tasks this week, you may use *fp* operations. 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 bclt; 't' means true]
bc1f # floating-point branch false
```

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 questions, if there are any 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.