02155 - Assignment 1

Practical information

This is a deliverable assignment, meaning that you have to write a small report in English with the requirements listed below. The grade you will obtain for the report will be part of the final grade. There should be one report per group. Groups may contain at most two students. At this point, you do not need to communicate your group members, after we receive your reports we will register the groups and assign you a group number to be used for the final project (the second deliverable assignment is individual).

The report should have a front page similar to the one appended at the end of this document. There are no other requirements on the template to be used for the report.

The report should include the following:

- Detailed problem solutions for the exercises in the section Problems.
- Detailed answers to questions specified in each exercise in the section Lab Exercises.
- Assembly code of the designed procedures properly formatted and commented.
- For all problems, assume a 32-bit architecture

The report should be handed-in only in electronic format (PDF and assembly code as appendix in the PDF) using the Assignment utility in DTU Inside. Hand-in the report as a group.

Feel free to ask or send an e-mail or use Slack if you have questions regarding the practical information and the assignment in general.

Paper & Pencil Problems

Exercise A1.1

Simple RISC-V assembly problems.

a. Write the RISC-V segment of code to detect if a negative integer number, stored in register x4, is a multiple of 8. If this condition is verified, set register x1 to zero.

b. Extract the 5 least-significant bits from register x5 and put the 5 extracted bits in register x1 (other bits in x1 must be zero).

   Write the RISC-V code for two different ways of doing it. Use a different set of instructions in the two cases.

Exercise A1.2

Write a RISC-V assembly program that reads a vector of 100 signed integer numbers from memory and computes the minimum value and the maximum value. Assume the vector base address is $10\ 0000|_{\text{Hex}}$. Once finished, the minimum value should be stored in `a0` and the maximum value in `a1`.

Exercise A1.3

A processor has three types of instructions: Type A execute in 3 cycles, Type B in 4 cycles and Type C in 5 cycles. The clock rate is 2 GHz.

a. Determine the average instruction execution time of a benchmark program that has 35% Type A, 40% of Type B and 25% Type C instructions. Determine the execution time of this program if it has 10^6 instructions.

b. An architecture modification introduces a Type D instruction which executes in 6 cycles. With this new instruction the total number of instructions is reduced by 10% and the fraction is now 35% Type A, 35% Type B, 25% Type C and 5% Type D.

   Determine whether this modification is advantageous for this benchmark.

Exercise A1.4

Examine the following fragment of RISC-V code. The symbol "..." means one or more instructions.

```
PC   instr.            | PC     instr.           | PC     instr.
-----------------------+-------------------------+----------------------
                       | ... B: ...              | ... C: ...
...  ...               | ...    ...              | ...    ...
152  add  x24,x0,x0    | 240    add x24,x0,x23   | 360    add  x20,x0,x21
156  addi x23,x24,4    | ...    ...              | 364    addi x24,x20,4
160  addi x22,x24,16   | 260    jal x1, C        | ...    ...
164  addi x24,x24,2    | ...    ...              | 388    jalr x0,0(x1)
168  add  x20,x22,x24  | 280    lw  x22,0(x20)   |
172  jal  x1, B        | ...    ...              |
176  ...               | 300    jalr x0,0(x1)    |
```

Assume *callee save* strategy is used.
Based on the information provided in the code, show what is contained in the stack when the PC is in the following locations.:

168, 240, 260, 360, 364, 388, 280, 176.

Assume that lower-numbered registers are pushed onto the stack first, and that the stack is empty prior to execution. Assume that the stack pointer is initialized to $10\,000|_{\text{Hex}}$.

Lab Exercises

In this section of the assignment your task is to write an assembly program capable of calculating the product of two complex numbers. Remember that the product of the two complex numbers $z = a + \boldsymbol{i}b$ and $w = c + \boldsymbol{i}d$ can be expanded as follows

$$(a + \boldsymbol{i}b)(c + \boldsymbol{i}d) = (ac - bd) + \boldsymbol{i}(ad + bc)$$

The program should be able to handle 32-bit signed integers where the result fits in a 32-bit register. You do not have to handle overflow. All the procedures you write in this exercise should follow the caller/callee conventions outlined in Section 2.8 and the green sheet of the lecture textbook. You should avoid using the frame pointer by only changing the stack pointer on entry and exit of a routine.

You can use the following template for your solution.

```
        # Course 02155, Assignment 1, A1 template

        .data
aa:     .word a         # Re part of z
bb:     .word b         # Im part of z

cc:     .word c         # Re part of w
dd:     .word d         # Im part of w

        .text
        .globl main
main:
        lw a0, aa
        lw a1, bb
        lw a2, cc
        lw a3, dd
        jal complexMul # Multiply z and w
        nop
        j end          # Jump to end of program
        nop

complexmul:
        ##################################
        #                                #
        #         YOUR CODE HERE         #
        #                                #
        ##################################

end:
        nop
```

Exercise A1.5 - Complex Multiplication

Write an assembly routine `complexMul` that takes two complex numbers z and w and returns their product. This routine should make use of integer multiplications, additions and subtractions. Assume no overflows occur.
The routine should expect the real parts of z and w to be placed in `a0` and `a2`, and the imaginary parts in `a1` and `a3`. Likewise, the real part of the result should be returned in `a0` and the imaginary part in `a1`.

When you have written `complexMul` answer the following questions.

a. Is `complexMul` a leaf or non-leaf procedure?

b. Did you need to use the stack? Explain why.

c. How many clock cycles does it take to execute `complexMul`, assuming multiplications take 2 clock cycles and all other operations take 1 clock cycle? You do not need to count `nop` operations.

Exercise A1.6 - Alternative Complex Multiplication

An alternative algorithm to reduce the number of multiplications from four to three is the following:

$$(a+\boldsymbol{i}b)(c+\boldsymbol{i}d) = \left\{ \begin{array}{lcl} k_1 & = & c(a+b) \\ k_2 & = & a(d-c) \\ k_3 & = & b(c+d) \end{array} \right\} = (k_1 - k_3) + \boldsymbol{i}(k_1 + k_2)$$

Write an assembly routine `altComplexMul` to implement this alternative method. Use the same registers as in A1.5 to pass/return the parameters.
Answer the following questions.

a. Is `altComplexMul` a leaf or non-leaf procedure?

b. Did you need to use the stack? Explain why.

c. How many clock cycles does it take to execute `altComplexMul`, assuming multiplications take 2 clock cycles and all other operations take 1 clock cycle? You do not need to count `nop` operations.

Exercise A1.7 - Extra questions

Answer the following questions.

a. How would your program have differed if RISC-V only had 2 argument passing registers, `a0` and `a1`, and one result register `a0`?

b. Which registers are saved and not saved across a procedure call? What does this mean?

Report front page

The report should have a front page similar the one shown below.

02155 - Computer Architecture and Engineering
Fall <year>

Assignment 1

Group members:
<student number> – <student name>
<student number> – <student name>

This report contains <n> pages

<date>