EEL 4768: Computer Architecture
Homework 2

Due date: Wednesday, February 4 Total points: 100

QUESTION 1. (10 points)

Translate the high-level language code below into assembly instructions. The variables A, B, C, D, E and F are located in the memory and can be accessed by their label (eg: LOAD R1, A will load A from the memory into R1). Minimize the number of instructions in the assembly code that you write.

$$F = (A+B) * C / (D / E)$$

Part a) Write the code for an accumulator architecture.

Part b) Write the code for a stack architecture. Assume that the division operation divides the topmost value in the stack by the second topmost value.

Part c) Write the code for a register-memory architecture.

Part d) Write the code for a load-store architecture.

QUESTION 2. (10 points)

Write a short essay on the history of early computers by summarizing the file posted on Webcourses. Mention the various computers, their use and their main features as discussed in the file. Discuss also how the performance evaluation of computers has changed over the years, as presented in the file. How was the computer performance evaluated back then? How is it done now?

The answer should be about 750 words.

QUESTION 3. (10 points)

Explain the concept of layered instruction set architecture. Where is it used? What is the motivation for using it? What problem does it solve?

QUESTION 4. (10 points)

In the early days of compilers, it was difficult to figure out how the compiler should map the variables to registers. For this reason, the stack architecture and the memory-memory architecture were preferred. Explain this idea and give an example to illustrate your point.

QUESTION 5. **(10 points)**

Below is an arithmetic expression and the corresponding codes in a register-memory architecture and in a load-store architecture. The variables A, B, C and D are in the memory and accessed through their labels, which are memory addresses.

Expression: (A+B) * C / D

Register-Memory Architecture	Load-Store Architecture
Load R1, A Add R1, B Mul R1, C Div R1, D	Load R1, A Load R2, B Add R3, R1, R2 Load R1, C Mul R3, R3, R1 Load R1, D Div R3, R3, R1

The register-memory architecture is based on the idea that reducing the code size allows the computer to run fast; that's why the corresponding code is smaller. The load-store architecture is based on the idea that simple operations make the program run fast.

Compare the two approaches and discuss the:

- Code size
- Number of clock cycles per instruction
- Suitability to build a pipeline
- Overall performance

QUESTION 6. **(10 points)**

Explain the concept of memory alignment and mention its benefit.

How do we determine if a 16-bit variable is memory aligned? How do we determine if a 32-bit variable is memory aligned? What about an 8-bit variable?

QUESTION 7. (10 points)

The slide #48 in Instruction Set Architecture (Part 1) shows various addressing modes. If we're designing a new computer architecture, we have to figure out which of these addressing modes to include. What experiment was shown in the lecture notes for this purpose? Comment on the experiment. What assumptions does it make about the compiler? What's the requirement on the program used?

QUESTION 8. (10 points)

Some architectures support the 'memory indirect' addressing mode. Below is an example. In this case, the register R3 contains a pointer to a pointer. Two memory accesses are required to load the data.

```
Add      R1, @(R3)
```

The MIPS CPU doesn't support this addressing mode. Write a MIPS code that's equivalent to the instruction above. The pointer-to-pointer is in register $t3. The other data is in register $t1.

QUESTION 9. (10 points)

Look at the figures in slides #96-97 in Instruction Set Architecture (Part 1). Can we implement the jump instruction that returns from the subroutine with the format below?

Opcode	Immediate

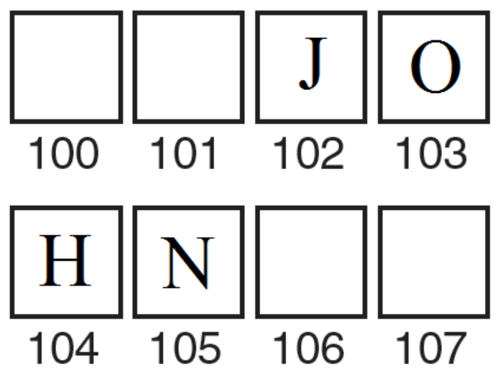
If yes, explain. Otherwise, explain what format would be suitable to return from the subroutine.

QUESTION 10. (10 points)

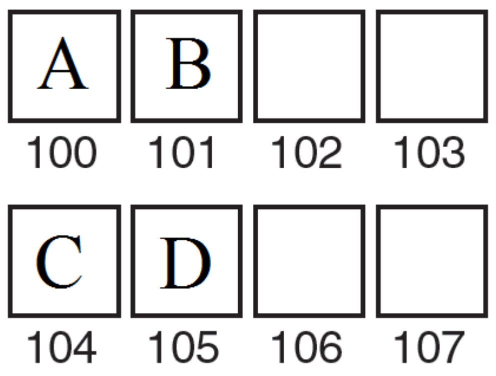
In general, the MIPS architecture stores data in the memory at aligned addresses. However, MIPS also supports misaligned data in the memory. Two MIPS instructions that are used to load misaligned data from the memory are 'Load Word Left' (LWL) and 'Load Word Right' (LWR). Refer to pages 24-26 in Appendix K and Figure K.31 to see the example on LWL and LWR.

Part a) Write a piece of code using 'LWL' and 'LWR' to extract the four-byte ASCII string "JOHN" and place it in register R1.

Part b) Write a piece of code using 'LWL' and 'LWR' to extract the four-byte ASCII string "ABCD" and place it in register R1.


J
O
H
N
100
101
102
103
104
105
106
107

Part (a)


A
B
C
D
100
101
102
103
104
105
106
107

Part (b)