George Mason University **Mid-term Exam**
CS 365 – Computer Architecture (Section 2) **March 26, 1996**

Important Notes:

1. You have 130 minutes for answering the following questions. There are 6 questions in all, some with multiple parts. Question 1 is worth 40 points, Questions 2 and 6 are worth 20 points, Question 3 is worth 10 points, and Question 4 and 5 are worth 5 points. The points for each question are shown in bold at the end of each question.

2. Be concise and accurate in your answers. If you don't know the answer to a question, the best strategy is to move on, and return to the question later. The estimated time to answer each question is shown in parentheses before each question.

3. Please answer the questions in the space provided below each question. If you need additional space, use the blank page at the end.

I. { **40 minutes**} Page 2 reproduces the data-path and control-unit for the single cycle implementation of a processor that supports a subset of the MIPS instruction set, specifically the arithmetic-logical instructions, `lw`, `sw`, and `beq`. Page 3 contains a table that explains the functions of the control signals, and a table that explains the settings of the **ALUOp** bits, and functions performed by the ALU for various ALU Control line settings. Finally, a figure with the 3 instruction formats found in the MIPS architecture is reproduced below.

Based on this information, answer the following questions:

ss

SS

A. We wish to add the instruction **lui** (load upper immediate) to our instruction set. Recall that the **lui** (load upper immediate) instruction is used to load large constants into a specified register. Specifically, the actions taken by this instruction are shown below:

Instruction	Format	Example	Meaning
lui	I	lui \$2,100	$\$2 = 100 \times 2^{16}$

Add any necessary data-paths and control signals to the Figure on Page 2 for this instructions. Add any new control signals to the table on Page 3 (explain what happens when the signal is asserted or deasserted). Briefly describe the additions you made in the space below (just so that I'm clear about the changes you made). **(10)**

B. Fill in the table below, which shows the setting of the control lines for these instructions.

Instruction	Reg Dest	ALU Src	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp2
lw									
sw									
add									
lui									

Note that the control line PCSrc (explained in the table on the previous page) is itself controlled by the Branch control line. Make a similar table below showing the settings of the control lines which you added for **lui.** **(15)**

C. Assume that the operation time for the major functional units in this implementation are:

- Memory units: 10 ns.
- ALU and adders: 8 ns,
- Register file: 5 ns,

Assume that PC access, wires, multiplexors, sign extension units, etc. have no delay.

Based on these numbers, how much time would needed to execute the **lui** instruction *if it was the only instruction you were implementing*? How much time would the **lui** instruction take *in the single cycle implementation on page 2*? Explain how you derived these times. **(5)**

D. What is the MIPS assembly code for loading the 32-bit constant below into register 18?
0000 0000 0000 0001 0000 0000 0000 0011 **(5)**

E. Give two reasons why the architects of MIPS processor did not provide a special instruction for loading a 32 bit constant into a specific register? **(5)**

II. { **10 minutes** } The following statistics were observed for program X on an implementation of the MIPS architecture.

Instruction category	Examples	CPI	Instruction Count
Arithmetic	add,sub,addi,addu,subu	1.0	400
Data Transfer	lw, sw	1.4	400
Conditional Branch	beq, bne	1.8	100
Jump	j, jr, jal	1.2	100

Given the information in the table below about the CPI of each instruction category and given that the cycle time for the MIPS implementation is 10 ns, find (i) the CPI for this program, and (ii) the execution time of this program. **(10)**

Suppose the MIPS instruction set architecture was extended to include the **push** and **pop** instructions, which have the following actions:
push $1 pushes the contents of register $1 on to the stack and decrements the stack pointer ($29) by one word. **pop** $1 stores the contents of the memory location pointed to by the stack pointer ($29) in register $1 and increments the stack pointer ($29) by one word.

Write the equivalent MIPS code for **push $1** and **pop $1** below: **(5)**

Assume that the new instructions **push** and **pop** have a CPI of 1.5, but their addition results in the clock cycle time going up to 11 ns. The following statistics were observed for the new program which uses **push** and **pop** wherever the MIPS instruction sequences you wrote above occur. Find the CPI for the program assuming it uses the new instructions and the execution time of this program on the "new and improved" machine. Show the steps in your derivation.

Instruction category	Examples	CPI	Instruction Count
Arithmetic	add,sub,addi,addu,subu	1.0	380
Data Transfer	lw, sw	1.4	380
"Arith-Data"	push,pop	1.5	20
Conditional Branch	beq, bne	1.8	100
Jump	j, jr, jal	1.2	100

Based on your calculations, do you think the new instruction should be added to this MIPS implementation? Are these measurements reported above and the calculations you did sufficient to justify a decision to add or not to add the **push** and **pop** instructions to your architecture? Explain your answer. **(5)**

III. { **10 minutes** } Why is it usually necessary to "save" registers while making a procedure call and to "restore" the registers during the return? Is it *always* necessary to save and restore *each and every* register during the execution of a procedure call? How can a smart compiler minimize the overhead of saving and restoring registers?

(10)

IV. { **10 minutes** } Why does the MIPS architecture have both an **add** and an **addu** (add unsigned) instructions? Couldn't the **add** instruction be used for both signed and unsigned numbers? If so, why is **addu** needed? **(5)**

V. { **10 minutes**} Show the IEEE 754 binary representation of the number -7.5_{ten} in single precision. Show how you derived the representation. **(5)**

VI. { **30 minutes** } The figures reproduced below show a 32-bit ALU that supports the **and**, **or**, **add**, **sub**, and **slt** (set on less than) instructions. Suppose we wanted to support the hypothetical instructions **sgt** (set on greater than) and **seq** (set on equal). These instructions are similar to **slt** except that the condition being tested is different. Modify the ALU shown below to support these instructions. Explain how the additions or changes you make will have the desired result. **(20)**