CSE 30321 – Computer Architecture I – Fall 2011
Homework 05 – Datapath Design – 65 points
Assigned: September 27, 2011 **– Due:** October 4, 2011

Problem 1: (10 points)

Background:

Here, you'll need to augment the MIPS datapath discussed in class to support a new instruction. When you decide to add or support a new instruction, this will most often require changes to datapath hardware, as well as to the finite state machine (FSM) / control logic that controls the datapath components in a given clock cycle. Ideally, you'd like an instruction to (i) improve execution time (by lowering CPI, improving clock rate, or reducing instruction count), (ii) improve functionality with respect to the baseline datapath, etc. with as little additional hardware (multiplexors, wires, etc.) and control signal overhead as possible.

In this problem, we want to augment the existing, multi-cycle MIPS datapath and FSM that we have discussed in class (see link on course website) to support a *load upper immediate* (lui) instruction.

For lui, the immediate value is shifted left 16 bits and stored in a register. The lower 16 bits are 0s.

- The instruction syntax is: lui $t, imm
- Thus, $t ← (imm << 16)
- The instruction encoding is:

Bits 31-26	Bits 25-21	Bits 20-16	Bits 15-0
Opcode	Unused	Destination	Immediate value
001111	XXXXX	TTTTT	iiii iiii iiii iiii

This instruction is not currently supported by the single cycle or multi-cycle MIPS datapath that was derived and discussed in class – which instead focused on a subset of commonly used instructions. In actuality, the MIPS datapath would contain hardware to facilitate the execution of 100s of instructions.

The lui instruction allows you to place constant values that are greater than 16 bits into a register.

- For example, assume that you want to load the number shown below into $4:
  - 0000 – 1000 – 0000 – 0000 – 0000 – 0000 – 0000 – 0001 (binary)
  - $134,217,729_{10}$ (base 10)
- In this instance, you could NOT do something like:
  - addi, $4, $0, $134,217,729_{10}$
  - (as seen above, $134,217,729_{10}$ cannot be represented with just 16 bits)
- However, this could be accomplished with the following sequence of instructions:

  lui $4, 2048_{10} # 2048 is the decimal representation of 0000 – 1000 – 0000 – 0000

  # Thus, $4 = 0000 – 1000 – 0000 – 0000 – 0000 – 0000 – 0000 – 0000

- ori $4, $4, 1_{10} # The or-immediate instruction will be used to set the lower order bits

Question:

- For the MIPS multi-cycle datapath, discuss/show the necessary modifications to the datapath and finite state machine to perform a lui in 3 clock cycles
- Note that for your answer, you **do not** have to update all other states in the finite state machine if you add new control signals
- Your answer should be supported by augmented FSM and datapath diagrams (available online)

Problem 2: (10 points)

Background:

In lecture, we did an in-class example where you were asked to implement a branch-if-less-than instruction – e.g. blt $s1, $s2, Label. There is no explicit branch-if-less-than or branch-if-greater than instruction in MIPS, and two separate instructions were needed to implement branch-if-less-than functionality.

The solution presented in class was to use the "set-on-less-than" instruction:

```
slt $t0, $s1, $s2      # $t0 ← 0 if $s1 ≥ $s2
                       # $t0 ← 1 if $s1 < $s2
```

Then, branch-if-less-than functionality could be implemented as follows:

```
slt $1, $s1, $s2
bne $1, $0, label
```

Question:

In this question, you are asked to explain how the MIPS multi-cycle datapath and finite state machine would need to be modified in order to implement an explicit blt (or bgt) instruction. Whether or not the data in one register is less than or greater than the data in another will be determined by examining the most significant bit of the ALUOut register after a comparison calculation is performed. (Thus, if the most significant bit of ALUOut is a 1, the result of the comparison is negative. If the bit is a 0, the result of the comparison is positive.)

Note – explicit modifications of the datapath and finite state machine diagrams are **not** required for this problem (although you can use them if you would like). However, please explain why you made the design decisions that you made – and your rationale for making them.

Remember that a branch address is calculated in clock cycle #2 per Lecture 11!

Problem 3: (10 points)

Background:

The purpose of this question is to quantify performance improvements to the MIPS multi-cycle datapath to determine what design improvement makes the most sense from the standpoint of execution time.

Question:

2 important parameters control the performance of a processor: cycle time and cycles per instruction.

1. Some designers prefer to increase the processor frequency at the expense of larger CPIs.
2. Other designers follow a different school of thought and try to reduce CPI at the expense of lower clock frequencies.

Qualitatively, explain how the processor datapaths for Options 1 and 2 might differ assuming the same ISA – i.e. both datapaths would execute the same ADD instruction, the ADD instruction would have the same 32-bit encoding given each datapath, etc.

Problem 4: (35 points)

Part A: (25 points)

Below, I've included representative instruction counts to do an MP3 encoding and decoding. (Data was obtained from the SimpleScalar tool used in Lab 01.) You will use this data to investigate possible changes to the MIPS datapath that may or may not improve benchmark performance. (Note that while this data was taken from simulations using an ARM core, we will assume that the instruction mixes would be relatively similar for the MIPS datapath too – especially as both the MIPS and ARM ISAs are RISC-like.)

MAD:

Instruction Type	Instruction Count	Base CPI
ALU	1,809,493,063	4
Store	177,074,647	4
Load	505,117,744	5
Branch	89,895,551	3

LAME:

Instruction Type	Instruction Count	Base CPI
ALU	1,074,420,032	4
Store	214,038,157	4
Load	631,988,109	5
Branch	47,775,211	3

The following design options are possible:

D1: The multicycle MIPS datapath that we have worked with in class (see handout) with a 4 GHz clock.

D2: A machine with a multicycle datapath like D1, except that register updates are done in the same clock cycle as a memory read or ALU operation. (Thus, in the FSM of Problem 1, states 6 and 7 and states 3 and 4 are combined.) This machine has a 3.1 GHz clock, as the register update increases the length of the critical path.

D3: A machine like D3 except that effective address calculations are done in the same clock cycle as a memory access. Thus, states 2, 3 and 4 can be combined, as can 2 and 5, as well as 6 and 7. This machine has a 2.8 GHz clock because of the long cycle created by combining the address calculation and memory access.

- What design is best if you only want to decode MP3s?
- What design is best if you only want to encode MP3s?
- What design is best if you want to do both (most likely)? Why do you think this is?

Part B: (10 points)

For Designs 1, 2, and 3, what is the maximum amount of time that you have to do the logic associated with a given clock cycle *before the data is latched?* (This question reviews the concept of a clock cycle.)