CSE 30321 – Computer Architecture I – Fall 2010
Homework 05 – Datapath Design – 75 points
Assigned: September 28, 2010 **– Due:** October 7, 2010

Problem 1: (20 points)
Background:

For this question, you'll need to add some instructions/additional functionality to the MIPS datapath we discussed in class. Note: when you decide to add or support a new instruction, this will most often require more hardware (i.e. multiplexors, wires, etc.) and more control signals. Ideally, you'd like an instruction to improve execution time (by lowering CPI, clock rate, etc.) with as little additional hardware and control signal overhead as possible.

Part A: (10 points)

We wish to add the instruction lui (load upper immediate) to the MIPS multi-cycle datapath. For the lui instruction, the immediate value is shifted left 16 bits and stored in a register. The lower 16 bits are zeroes.

- The instruction syntax is: lui $t, imm
- Thus, $t ← (imm << 16)
- The instruction encoding is: 0011 – 11xx – xxxt – tttt – iiii – iiii – iiii – iiii
    - (where the first 0 is the most significant bit and the last i is the least significant bit)
    - (the x's stand for "don't care")

For the MIPS multi-cycle datapath, discuss/show the necessary modifications to the datapath and finite state machine. Note that you **do not** have to update all other states in the finite state machine if you add new control signals. (See the end of this handout for schematics of both the datapath and finite state machine.) The instruction should take 4 clock cycles.

Part B: (10 points)

Now, modify the datapath so that the lui instruction takes just 3 clock cycles. Again, add any necessary data paths and control signals to the multi-cycle datapath. You have to maintain the assumption that you don't know what the instruction is before the end of state 1 (the end of the second cycle). Show any necessary changes to the finite state machine. Note that you **do not** have to update all other states in the finite state machine if you add new control signals.

<u>Problem</u> <u>2</u>: (10 points)
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
```

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.

Problem 3: (45 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.

Part A: (10 points)

Two important parameters control the performance of a processor: cycle time and cycles per instruction. There is something of an enduring trade-off between these two parameters in the design process of microprocessors:

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.

Part B: (25 points)

Below, I've included representative instruction counts to do an MP3 encoding and decoding from 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:

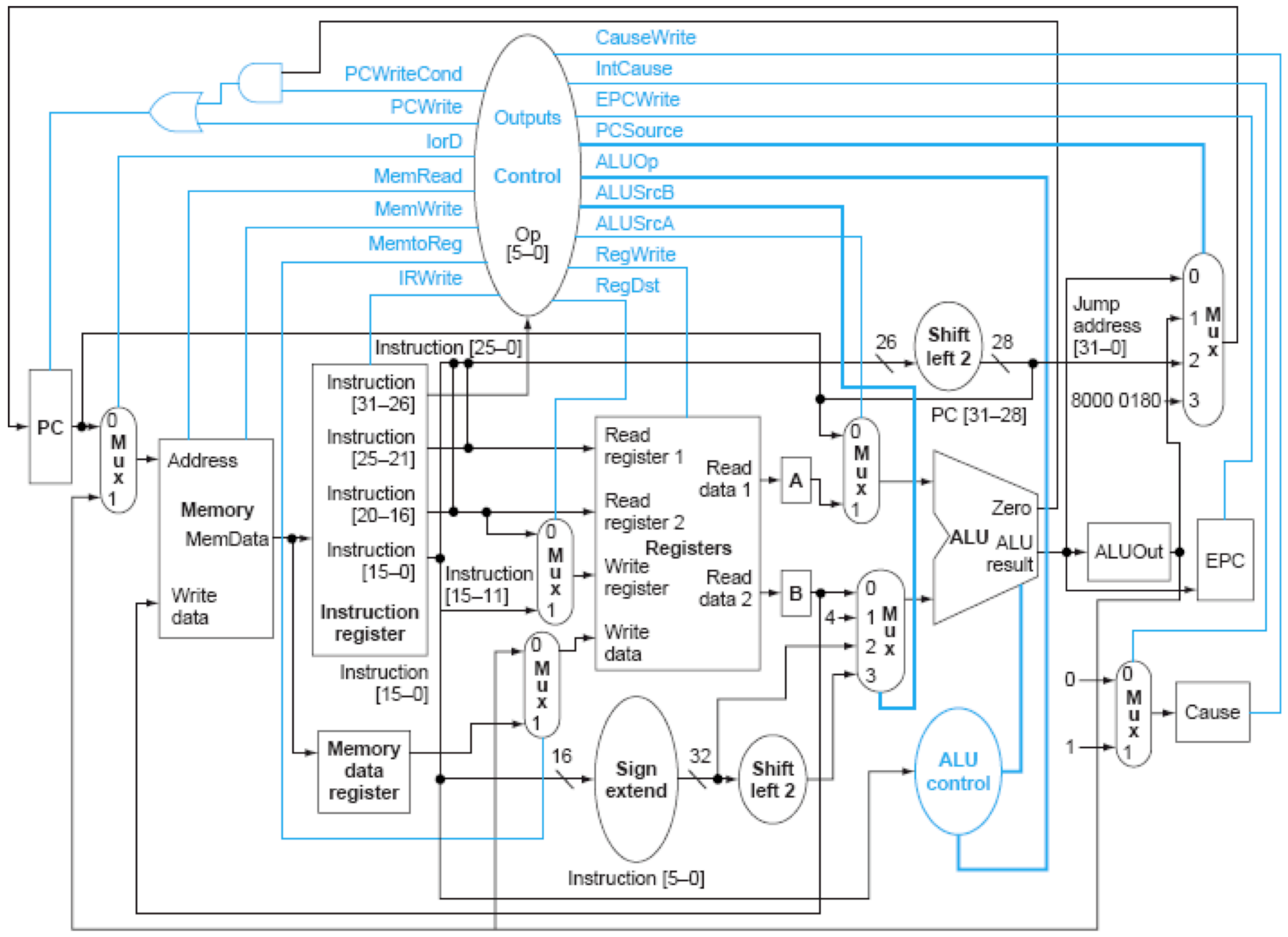
D1: The multicycle datapath that we have worked with in class (see handout) with a 4 GHz clock.

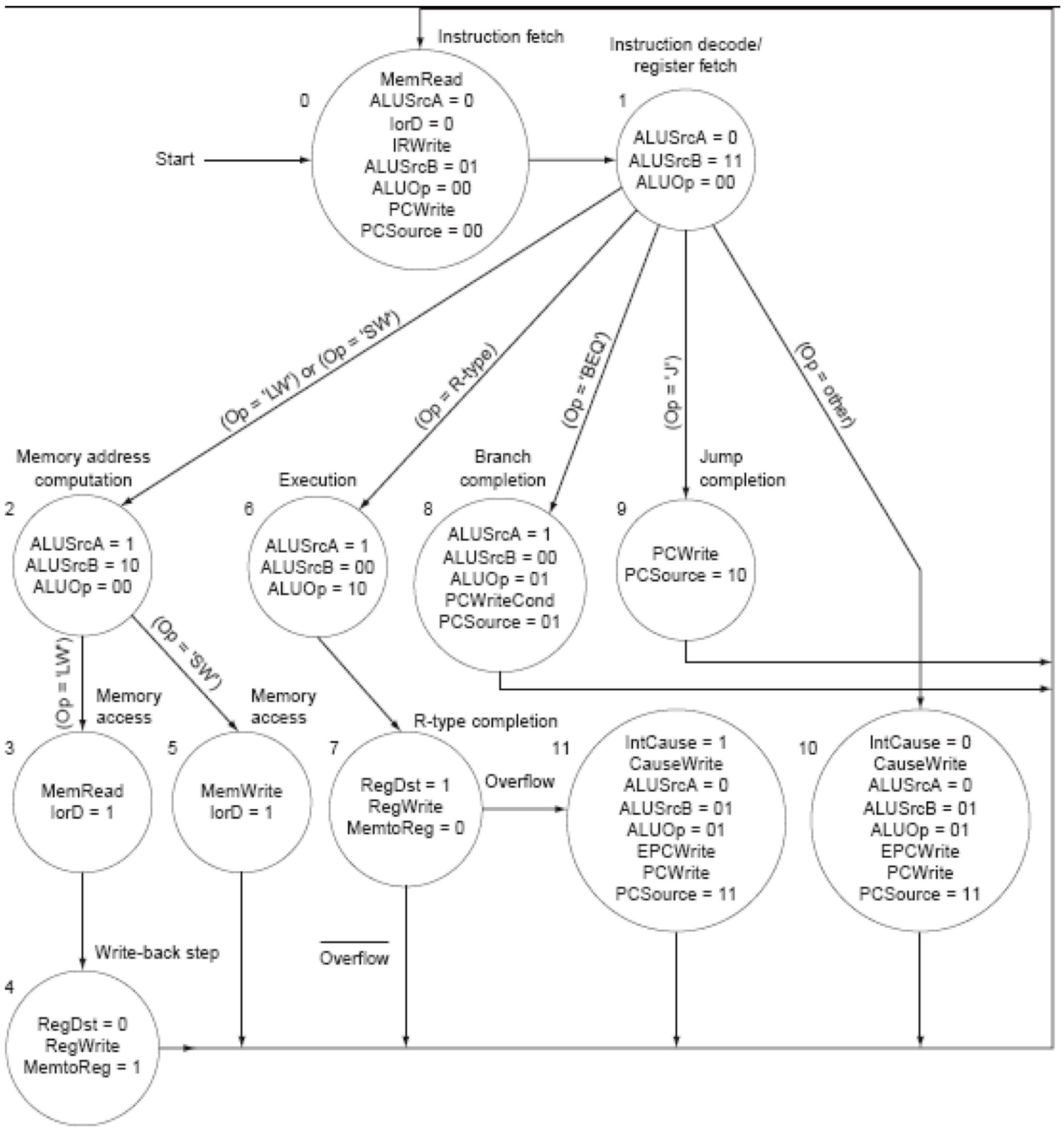
D2: A machine with a multicycle datapath like D1, except that register updates are done in the same clock cycle as a memory read or ALU operation. (Thus, in the FSM of Problem 1, states 6 and 7 and states 3 and 4 are combined.) This machine has a 3.1 GHz clock, as the register update increases the length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


PC
Mux
Address
Memory
MemData
Write data
Instruction [31–26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–0]
Instruction register
Instruction [15–0]
Memory data register
Instruction [25–0]
Instruction [15–11]
PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op [5–0]
CauseWrite
IntCause
EPCWrite
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
A
B
16
Sign extend
32
Shift left 2
Instruction [5–0]
26
Shift left 2
28
PC [31–28]
4
Zero
ALU
ALU result
ALU control
Jump address [31–0]
8000 0180
ALUOut
EPC
Cause
0
1
2
3



Instruction fetch
Instruction decode/ register fetch
Start
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
(Op = other)
Memory address computation
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
Execution
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
Branch completion
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
Jump completion
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory access
Memory access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Overflow
11
IntCause = 1
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
10
IntCause = 0
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
Write-back step
$\overline{\text{Overflow}}$
4
RegDst = 0
RegWrite
MemtoReg = 1