CS/ECE 552 Introduction to Computer Architecture

Midterm Exam

Tuesday, March 11, 2003

7:15-9:15 p.m

Name:_____

Limit your answers to the space provided. Unnecessarily long answers will be penalized. If you use more space than is provided, you are probably doing something wrong. Use the back of each page for any scratch work.

Problem 1.   _____ (out of 24 points)

Problem 2.   _____ (out of 14 points)

Problem 3.   _____ (out of 12 points)

Problem 4.   _____ (out of 14 points)

Problem 5.   _____ (out of 22 points)

Problem 6.   _____ (out of 14 points)

 Total   _____ (out of 100 points)

(1)   **Lookahead Adder Design (24 points)**

The figure below shows a 64-bit carry lookahead adder with three levels of lookahead. Some signals are shown in the figure to give signals are shown (for example the carries from the first level lookahead to the single-bit adders).

Single Bit Adders
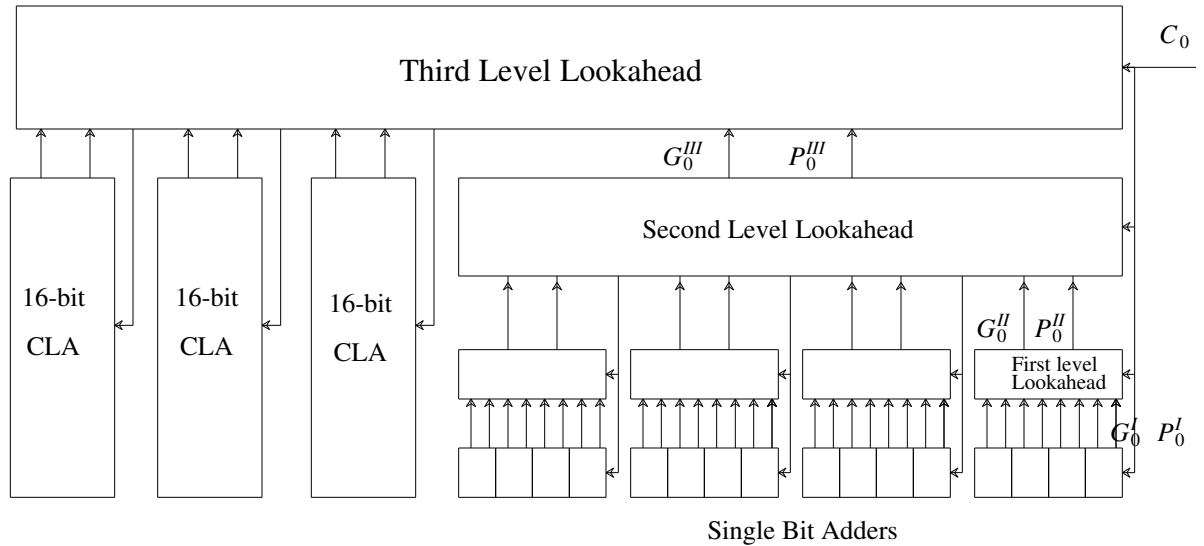
(a)   Let X_i, Y_i, C_i and S_i, with $i = 0, 1, ..., 63$, represent the individual bits of the two operands, the carries, and the sums, respectively. Let G_i^I and P_i^I, $i = 0,....,63$ represent the first level generates and propagates. Let G_j^{II} and P_j^{II}, $j = 0,...,15$ represent the second level generates and propagates, and let G_k^{III} and P_k^{III}, $k = 0,..,3$ represent the third level generates and propagates.

Write the logic equations for the following signals, in the space provided: **(10 points)**

$G_3^{II} =$	
$P_2^{II} =$	
$C_{20} =$	
$C_{32} =$	
$C_3 =$	

(b)   Now suppose that the same structure is to be used as an *incrementer*, i.e., to add 1 to an input number. How would the lookahead logic and single-bit addition units change? Show the design for the lookahead logic and single-bit addition units in case of the incrementer. **(8 points)**

(c)   Assume that each gate delay is τ time units, and all gates are available with up to 4 inputs. How much faster than the adder would the incrementer be? Show your work. **(6 points)**

(2) **System Performance (14 points)**

Recall that the speedup resulting from an enhancement is the time to complete a task using the enhancement divided by the time to complete the task without using the enhancement. Use this relationship to answer the following questions. Be sure to write out your calculations clearly.

The base system spends 82% of its time computing and 18% of its time waiting for the disk. During the time that it is computing, the instruction mix and the average cycles per instruction (CPI) for each type is:

Type	Percent of all instructions executed	CPI
Integer	40%	1
Floating point	30%	5
Other	30%	2

(i) There are three modifications that can be done to the system. For each modification, compute the speedup resulting from the modification.

a). The processor is replaced with a new one that reduces the total computation time by 35%. **(3 points)**

b). The disk is replaced with a solid state device that reduces the disk waiting time by 85%. **(3 points)**

c). The processor is replaced with a new one that has improved floating point performance. The average floating point CPI is reduced to 3; all other aspects are unchanged. **(3 points)**

(ii) Which modification results in the best speedup? **(1 point)**

(iii) For the two modifications in part (i) that did not result in the best speedup, is it possible for them to achieve the speedup achieved by the modification in part (ii)? Show your work and explain your answer. **(4 points)**

(3)   **MIPS ISA (12 points)**

(i)   In the MIPS instruction set, the offset in a branch instruction is from the PC of the instruction following the branch, not from the PC of the branch. What is the reason for this choice? (**3 points**)

(ii)   What are SET instructions in the MIPS architecture? How are they useful? (**3 points**)

(iii)   Why are the offsets for branch instructions and displacements for load and store instructions in the MIPS ISA (I-format) limited to 16 bits? (**3 points**)

(iv)   Why is the branch offset shifted left by 2 bits while the displacement for loads and stores not shifted? (**3 points**)

(4)   **Short Questions (14 points)**

(i)   What is *system balance*? How does the notion of system balance influence the design of a computer system? (**3 points**)

(ii)   Many processors have 5 or 6 stage pipelines. A typical value for the CPI (cycles per instruction) in such processors is in the range of 1.0 to 1.5. Does it mean that the *latency* of execution of most instruction 1 or 2 clock cycles? Why, or why not? (**3 points**)

(iii)   Why do conditional branches impact the performance of a pipelined implementation? (**2 points**)

(iv)   Briefly describe 3 solutions to reduce the performance impact of conditional branch instructions in a pipelined implementation. (**6 points**)

(5) **Pipelining (22 points)**

The following figure shows a simple pipeline. This pipeline has a register file that WRITES during the FIRST half of the clock cycle and READS during the SECOND half of the clock cycle. Instructions can be stalled only in the instruction-fetch and instruction-decode stages. There is a bypass from the output of the EX stage (output of the EX/MEM latch) to the input of the logic in the EX stage.
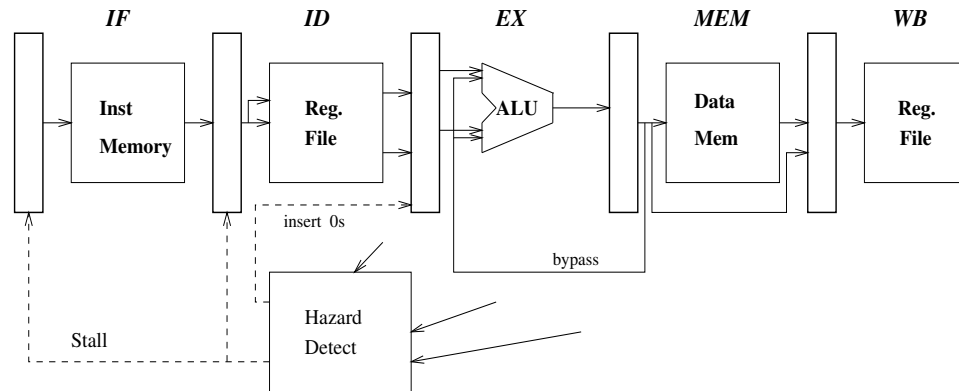
(a) For the following instruction sequence, show the pipeline timing in the table below. Label pipeline bubbles caused by stalls "bubble". (**Note:** you may not need all 16 cycles provided in the table.) **(10 points)**

```
r3 <- r2 + r1                        ; ADD1
load r2 <- mem(r3 + r1)              ; LOAD
r2 <- r2 + r1                        ; ADD2
r4 <- r3 - r1                        ; SUB
r1 <- r2 + r1                        ; ADD3
```

Cycles	IF	ID	EX	MEM	WB
1	ADD1				
2	LOAD	ADD1			
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

THIS QUESTION CONTINUES ON THE NEXT PAGE.

Pipelining question continued

(b)   In part (a) it was assumed that the register file writes in the first half of the clock cycle and reads in the second half of the cycle. Now assume that the register file is changed so that READS occur in the first half of the clock cycle and WRITES occur in the second half of the clock cycle. Draw an additional bypass and any additional hardware on the pipeline diagram so that the execution is the same as in part (a). Label the bypass as "bypass 1". Also describe the bypass you added in the space below. **(6 points)**

(c)   Eliminate ALL of the bubbles caused by the instruction sequence in part (a) by reordering the instructions (without changing the semantics of course) and by adding an additional bypass path. Assume the original pipeline configuration exists, WITHOUT any of the modifications made in part (b). Show the reordered instruction sequence in the space below and draw the additional bypass on the pipeline diagram labeled "bypass 2". For your convenience, the original code sequence is repeated below. **(6 points)**

```
r3 <- r2 + r1               ; ADD1
load r2 <- mem(r3 + r1)     ; LOAD
r2 <- r2 + r1               ; ADD2
r4 <- r3 - r1               ; SUB
r1 <- r2 + r1               ; ADD3
```

(6) **Datapath (14 points)**

In this problem you will extend the multi-cycle datapath as presented in the course text in Figures 5.33 and 5.34. These figures have been provided on the last page of this exam.

It is possible to decrease the execution time of a program by combining two or more instructions (that would normally take two or more instruction times to execute) into a single instruction that takes only one instruction time (or a fraction more than one) to execute. These hybrid instructions are chosen because the instructions they replace appear often as a group in programs, and because they can be implemented without extensive modification to the datapath.

Modify the datapath as necessary to allow the implementation of each instruction. Be sure to include complete information about any new signals that appear in the modified datapath.

After the datapath is modified, fill in the list of control signals that must appears at each clock cycle in order to implement the instruction. To reduce the number of signals that you have to write, assume that the initial value of all signals is zero. Remember that if you assert a signal, you must de-assert it when necessary. The number of clock cycles needed for each instruction may vary. Use as many as you need, but make the implementation as efficient as possible. The signals for the first clock cycle already appear in the table. You should not have to modify them.

ALL OF YOUR WORK FOR EACH INSTRUCTION SHOULD APPEAR ONLY ON THE PAGE FOR THAT INSTRUCTION.

Note: In figure 5.34, ALU stands for arithmetic logic unit, MDR for memory data register, PC for program counter, and IR for instruction register.

(i) On the next page, modify the datapath(and control signals, as necessary), and fill in the list of control signal to implement a `LDINC Rs, offset(Rt)` instruction. The LDINC Rs, offset(Rt) instruction loads the word addressed by the sum of the sign-extended offset (bits 15-0) and the value in Rt (bits 25-21) into register Rs (bits 20-16). The value in register Rt is also auto-incremented by 4 so that the next execution of the instruction will load the next word in memory. (**Note: you may not need all the clock cycles provided in the table on the next page.**). **Points will be taken off for solutions that require more hardware, or take more time steps, than necessary. (14 points)**
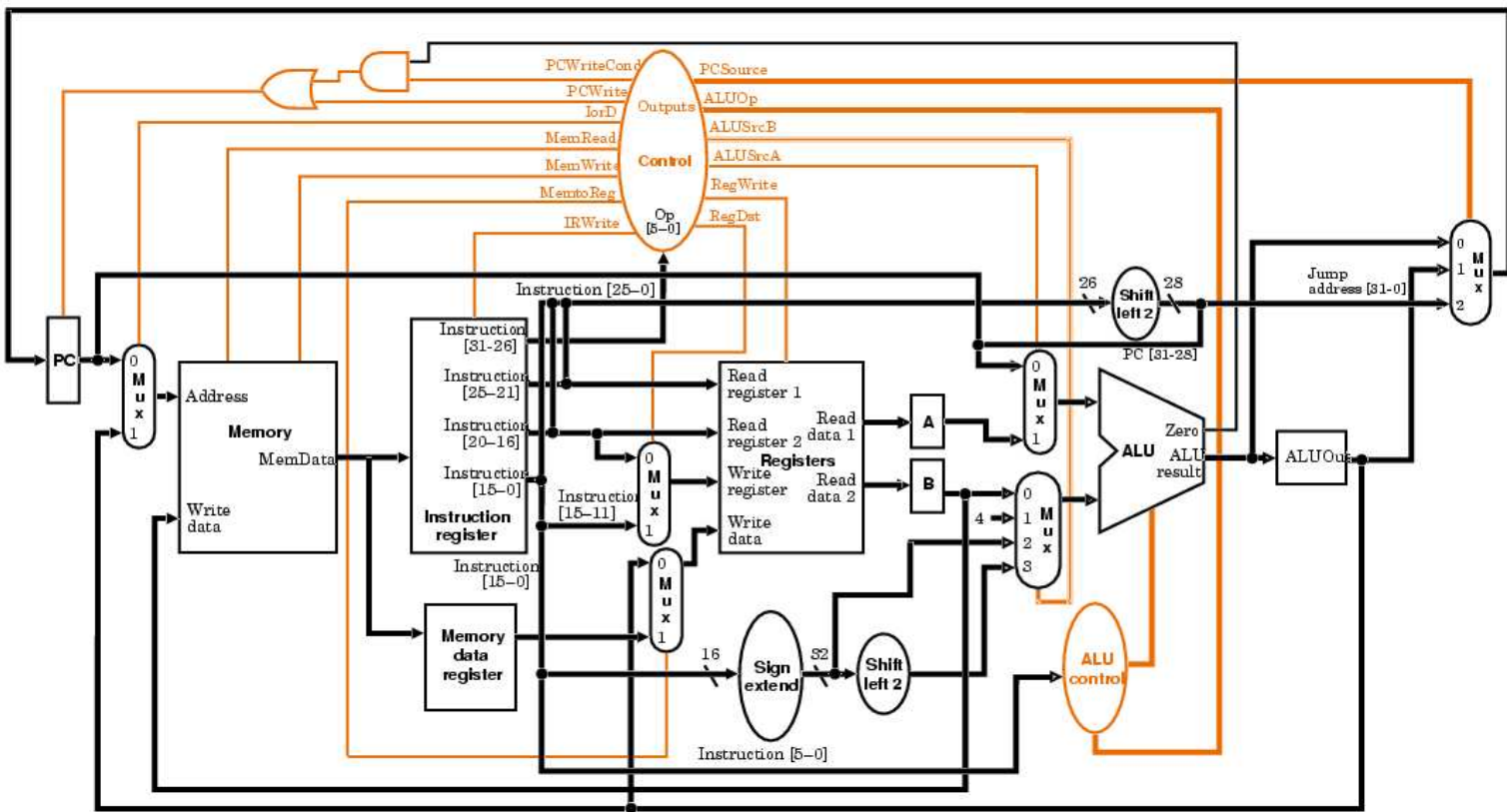
Clock Cycle	Functional Description	Signals
1	Instruction -> IR and PC=PC+4	MemRd=1, ALUSrcA=0, IorD=0, IRWrite=1, ALUSrcB=01 ALUOp=00, PCWrite=1, PCSource=00
2		
3		
4		
5		
6		
7		
8		