Mikko Lipasti
Spring 2002

ECE/CS 552 : Introduction to Computer Architecture
IN-CLASS MIDTERM EXAM
March 14th, 2002

NAME:____

This exam is to be done individually in 75 minutes. Total 10 Questions, 75 points

1. **(10 Points)** A sequence of bits (binary digits) can mean various things depending on what it represents. Fill in the holes in the following table by identifying either the missing 8-bit pattern that represents the specified number, or computing the number or numbers that the 8-bit pattern represents.
NOTE: fill in all the empty slots in the table.

8-bit pattern	**Unsigned representation**	**Signed representation**
0101 1100		
1000 1000		
1111 0111		
		-101
	75	

Show your work here:

2. **(4 point) A fixed number of bits can only be used to represent a finite number of unique numbers. Fill in the least and greatest decimal numbers you can represent with a 7-bit pattern in the following table?**

7-bit representation	**Least decimal number**	**Greatest decimal number**
Unsigned		
Signed 2's complement		

Show your work here:

3. **(4 points) Explain how an 8-bit ALU can detect arithmetic overflow when adding both signed and unsigned numbers:**

Unsigned overflow:

Signed overflow:

4. **(2 points) Discuss if and how a processor should react when overflow is detected.**

5. **(10 points) The IBM study of pipelined processor performance assumed an instruction mix based on popular C programs in use in the 1980s. Since then, object-oriented languages like C++ and Java have become much more common. One of the effects of these languages is that object inheritance and polymorphism can be used to replace conditional branches with virtual function calls. Given the IBM instruction mix and CPI shown in the following table, perform the following transformations to reflect the use of C++/Java, and recompute the overall CPI and speedup or slowdown due to this change (fill in the table to reflect your work to receive partial credit):**
   - **Replace 50% of taken conditional branches with a load instruction followed by a jump register instruction (the load and jump register implement a virtual function call).**
   - **Replace 25% of not-taken branches with a load instruction followed by a jump register instruction.**

Instruction type	Old Mix %	Latency	Old CPI	Cycles	New Mix %	Instructions	Cycles	New CPI
Load	25.0%	2	0.50	500				
Store	15.0%	1	0.15	150				
Arithmetic	30.0%	1	0.30	300				
Logical	10.0%	1	0.10	100				
Branch - T	8.0%	3	0.24	240				
Branch - NT	6.0%	2	0.12	120				
Jump	5.0%	2	0.10	100				
Jump register	1.0%	3	0.03	30				
Total	100.0%		1.54	1540				

New overall CPI:

Speedup/slowdown:

Assume the standard 5-stage pipeline discussed in class. Also assume that rather than having a separate PC (program counter) register, as in the MIPS instruction set, consider an instruction set that uses the R31 general purpose register as the program counter. The pipeline is summarized in the following table.

Pipestage	**Action**
IF	**Fetch instruction from MEM[R31]**
RD	**Read source operands from R0..R31**
EX	**Execute ALU instructions;** **Generate address for load/store**
MEM	**Loads read from data memory;** **Stores write to data memory**
WB	**Results are written back to registers R0..R31**

6. **(5 points) Does this pipeline have any WAR or WAW hazards for register or memory operands? If so, describe what they are. If not, prove why not.**

7. **(5 points) Does this pipeline have any additional RAW hazards besides the ones discussed in lecture? If so, describe what they are. If not, prove why not.**

Mikko Lipasti
Spring 2002

Design of a Multicycle Datapath and Control Logic

Objective:

Given the following instruction set with four 8-bit instructions, design a multicycle data path and control logic. Assume 4 8-bit registers and an 8-bit ALU. Don't worry about initial values of PC, registers, etc. (do not show reset, clear, or preset signals).

Instruction set:

- ❑ **load $p1, ($p2)**
  - Contents of memory location ($p2) are loaded into register $p1.
- ❑ **store $p1,($p2)**
  - Contents of register $p1 are stored into memory location ($p2).
- ❑ **sub $p1,$p2**
  - $p1=$p1-$p2
- ❑ **branch_negative $p1,(4 bit PC-relative target)**
  - If $p1 is negative, then branch to newPC + the sign extended target, otherwise this instruction is a NOP.

Opcode table:

Instruction	Opcode
load	00
store	01
sub	10
branch_negative	11

Assembly level syntax: {MSB……LSB}

- ❑ **load $p1, ($p2)**
  00, aa, bb, XX

- ❑ **store $p1, ($p2)**
  01, aa, bb, XX

- ❑ **sub $p1, $p2**
  10, aa, bb, XX

- ❑ **branch_negative $p1, B B B B**
  11, aa, B B B B

Multicycle Execution Stages:

- ❑ IF – fetch instruction from memory, compute newPC = PC + 1
- ❑ RD – read instruction source operands from register file, compute branch target tPC = newPC + SE(BBBB)
- ❑ EX/MEM – load from memory, store to memory, execute sub, or check branch condition
- ❑ WB – write load or ALU result to register file

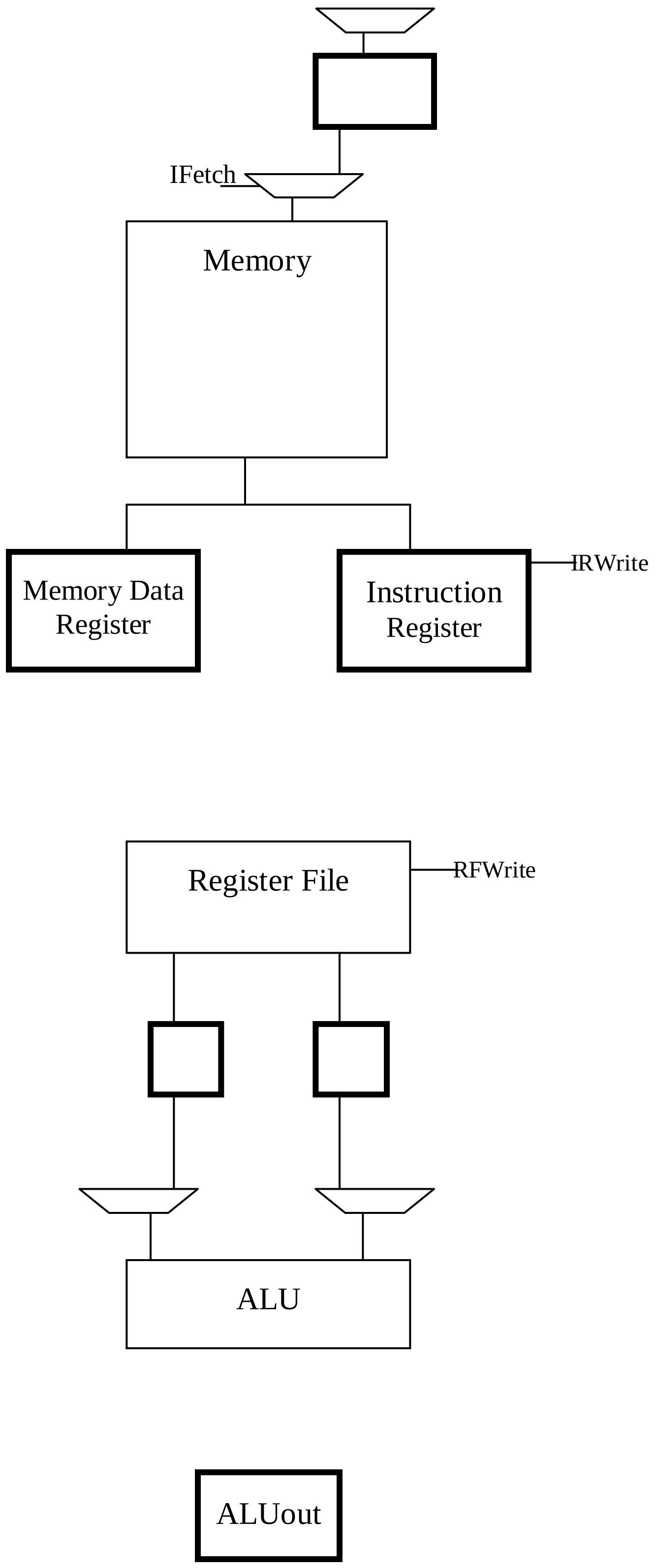
8. **(15 points) Design a shared 8-bit ALU for a multicycle data path for this instruction set. Assume that each logic gate can have no more than 4 inputs and has a delay of 1ns. Consider the combinational delay for both a ripple-carry and a carry-lookahead adder, choose the faster option, and show your design below. Include any control inputs needed in this ALU. Recall that this ALU is used for executing the sub instruction, computing newPC, and computing the branch target.**

   8-bit ripple-carry ALU combinational delay:

   8-bit carry-lookahead ALU combinational delay:

   Final ALU Design:

9. **(10 points) Given the multicycle datapath elements below, add all the data paths, control signals, and control inputs you will need to implement the 4 instructions in this instruction set. All data and control signal paths should indicate width. Some control signals and data paths are included as examples.**



10. (10 points) Complete the incomplete state transition and control signal table shown below. Add each control signal in your data path as a column in the table, and show what value each signal should have for each state in the table. Also show the next state given any relevant input signals (use 'x' as shown for don't care).

State	**Input/Next State**	**Control Signals (fill in additional ones in empty columns)**											
		IFetch	**IR Write**	**RF Write**									
IF	**X/RD**	**1**	**1**	**0**									
RD	**Op=00/MEM-LD** **Op=01/MEM-ST** **Op=10/EX-SUB** **Op=11/EX-BR**	**0**	**0**	**0**									
MEM-LD		**0**	**0**	**0**									
MEM-ST		**0**	**0**	**0**									
EX-SUB		**0**	**0**	**0**									
EX-BR		**0**	**0**	**0**									
WB	**X/IF**	**0**	**0**	**1**									