CS 351 Midterm 1 Solutions

Question 1 [10 points].

a. Name one advantage and one disadvantage of using a memory-to-memory instruction set vs. a load-store instruction set.

Advantage of memory-to-memory instruction sets:

- Fewer instructions/simpler syntax
- Doesn't require a register file

Disadvantage of memory-to-memory instruction sets:

- Requires 3 memory accesses per instruction => high data bandwidth
- Instructions will probably execute more slowly because of all the memory accesses

b. Name one advantage and one disadvantage of the increasing the number of stages of a pipelined datapath.

Advantage of deeply pipelined datapaths:

- Higher throughput: 1 instruction/cycle for shorter cycles
- Partial credit for saying that the clock speed increases or the cycle time decreases without explaining why this is good (performance-wise, it will likely be offset by the increased CPI)

Disadvantage of deeply pipelined datapaths:

- Increases power (since clock speed increases)
- Increases circuit complexity
- Results in more speculation (and thus inefficiencies of failed speculation)
- More hazards/more complicated hazard logic

Question 2 [10 points].

A proposed hardware optimization for a given processor would eliminate 10% of instructions outright and decrease the CPI of the remaining instructions by 10%. Unfortunately, this optimization would also result in decreasing the clock rate by 14%.

a) Is this optimization worth implementing? Show your calculations.

IC(new) = 0.9 * IC(old) (10% of instructions eliminated)
CPI(new) = 0.9 * CPI (old) (CPI decreased by 10%)
Clock rate(new) = 0.86 * clock rate(old)

Execution time(new)
= 0.9*IC(old) * 0.9*CPI(old) * 1/(0.86 * clk rate(old))
= 0.94 * IC(old) * CPI(old) / clk rate(old)
= 0.94 * Execution time(old)
Since the new execution time is lower, the optimization is worthwhile.

b) What is the speedup of the optimized machine over the original machine?

Speedup (new over old) = Execution time (old) / Execution time(new)
= Execution time (old) / (0.94 * Execution time(old))
= 1/0.94
= 1.06

Question 3 [10 points].

Using Amdahl's Law, show which is better: making 20% of the instructions in a program 80% faster, or making 80% of the instructions 20% faster.

Making 20% of instructions 80% faster…
Speedup(overall) = 1/(1-0.2 + 0.2/1.8) = 1.098

Making 80% of instructions 20% faster…
Speedup(overall) = 1/(1-0.8 + 0.8/1.2) = 1.153

Making 80% of instructions 20% faster is the better option.

Question 4 [10 points].

Translate the following snippet of C code to MIPS. Assume that the address of `A[0]` is in `$s0`, the address of `B[0]` is in $s1, and the variable `i` is in `$s2`. Your code should not modify these three registers. Also assume that `A` and `B` are arrays of integers.

```
// C code to translate
A[i] = B[i];
```

```
SLL $t2, $s2, 2        # Integers are 4 bytes, so the address of
                       # element i = Address of element 0 + 4i
                       # Shift i left by 2 to multiply by 4
                       # $t2 now contains 4i
ADD $t0, $s0, $t2      # $t0 now contains &(A[0]) + 4i = &(A[i])
ADD $t1, $s1, $t2      # $t1 now contains &(B[0]) + 4i = &(B[i])
LW $t3, 0($t1)         # $t3 now contains B[i]
SW $t3, 0($t0)         # …which we store at &(A[i])
```

Question 5 [15 points].

Translate the following snippet of C code to MIPS. Assume that the variables `a, b,` and `c` are in registers `$s0, $s1,` and `$s2` respectively.

```
// C code to translate
if (a == b) c = a;
else c = a-b;
```

```
      BNE $s0, $s1, ELSE     # Branch to ELSE condition if a != b
      ADD $s2, $s0, $zero    # c = a
      J EXIT                 # jump over ELSE condition
ELSE: SUB $s2, $s0, $s1      # c = a-b
EXIT:
```

Question 6 [15 points].

Translate the following snippet of C code to MIPS. Use the traditional MIPS conventions for argument passing, return values, and adjusting the stack pointer.

```
int saturate(int sum) {
     if (byte_overflow(sum)) return 0xff;
     else return sum;
}

int byte_overflow(int num) {
     if (num >= 0x0100) return 1;
     else return 0;
}
```

```
byte_overflow:
      ADDI $t0, $zero, 0xff
      SLT   $v0, $t0, $a0      # If 0xff < num, return 1; else 0
      JR    $ra

saturate:
      ADDI $sp, $sp, -4
      SW $ra, 0($sp)           # Save return address; otherwise it's lost
                               # when we jal to byte_overflow
      JAL byte_overflow
      LW $ra, 0($sp)
      ADDI $sp, $sp, 4

      BEQ $v0, $zero, ret_sum  # If byte_overflow returns 0, return sum
      ADDI $v0, $zero, 0xff    # Otherwise return 0xff
      JR $ra

ret_sum: ADD $v0, $zero, $a0
           JR $ra
```

Question 7 [10 points].

For each instruction type on the single-cycle MIPS datapath, state whether or not the instruction writes to the register file. For each instruction that writes to the register file, state what data it writes to the register file and what this data conceptually represents. Also state the value and name of any MUX select signals that allow this data to be written to the register file.

	Writes regfile?	**Which data gets written**	**Name of MUX select input**	**Value of MUX select input**
R-format	Y	ALU result	MemtoReg	0
LW	Y	Mem result	MemtoReg	1
SW	N			
BEQ	N			
J	N			

Question 8 [10 points].

For each of the following states of the multicycle datapath, state…

1) What the ALU computes (conceptually)
2) What the ALU operation is (ADD, SUB, etc.)
3) What the ALU's first input is
4) What the ALU's second input is

	Computation	Op	Input A	Input B
State 0: Inst fetch	PC+4 (next instruction)	ADD	PC	4
State 8: Branch completion	Compares $rs and $rt to see if branch is taken	SUB	RegA	RegB
State 11: Overflow exception handling	Computes PC-4 (the PC that caused the exception) to save in EPC	SUB	PC	4

Question 9 [10 points].

For each of the following sequences of instructions, state

1) Whether a data hazard exists in the pipelined MIPS architecture
2) If that data hazard necessarily results in a stall, and
3) Which forwarding paths (from the output of which stage to the input of which stage) are necessary to eliminate or minimize the stall.

a) LW $s0, 4($s1)
ADDI $s2, $s0, 10

1. Yes ($s0); 2. Yes; 3. MEM->EX minimizes the stall.

b) SLT $s1, $s2, $s3
SW $s1, 4($t0)

1. Yes ($s1); 2. No; 3. EX->EX or MEM->MEM eliminates the stall.