

NAME \_\_\_\_\_  
First Last

Student ID# \_\_\_\_\_

**STONY BROOK UNIVERSITY**  
**COMPUTER SCIENCE DEPARTMENT**  
**MIDTERM #1 EXAMINATION**

CSE 220  
Spring Semester 2008  
February 28, 2008

This is a closed-book exam. (80 minutes)  
Use this form for your work and return it.

The exam has 12 problems.

**It is crucial to show all work done in order to obtain full/partial credit**

# 1 _____ (8 pts)	# 7 _____ (7 pts)
# 2 _____ (3 pts)	# 8 _____ (2 pts)
# 3 _____ (4 pts)	# 9 _____ (5 pts)
# 4 _____ (3 pts)	#10 _____ (4 pts)
# 5 _____ (1 pt)	#11 _____ (3 pts)
# 6 _____ (6 pts)	#12 _____ (4 pts)

TOTAL \_\_\_\_\_ (50 max)

- 1 (i) (3 pts) Convert C5.B3 in hexadecimal to binary. Normalize for single precision IEEE format.  
(Write answer in the form: sign \* normalized fraction \*  $2^{\text{exponent}}$ )

**11000101.10110011**

$$(-1)^0 * 1.100010110110011 * 2^7$$

- (ii) (3 pts) State the sign bit, exponent, and fraction (significand) for the normalized number in (i), the way these would appear in IEEE single-precision floating point representation.

**sign bit = 0**

**exponent = 7; convert to excess 127:  $7 + 127 = 134$ ; 134 in binary = 10000110**

**fraction is 100010110110011000000000**

**IEEE Format: 0 10000110 100010110110011000000000**

- (iii) (2 pts) Now convert C5B.3 in hexadecimal to base 4. Show all steps.

**11000101.10110011**

**Group by 2's for base 4: 11 00 01 01 . 10 11 00 11**

**Covert to base 4 values (0,1,2,3): 3011.2303**

- 2 (3 pts) Convert 19.7 in decimal to base 3. Show all steps. You must have at least two digits after the decimal point.

**First convert the integer part 19 to base 3:**

**19/3 → remainder 1**

**6/3 → remainder 0**

**2/3 → remainder 2**

**Reading upwards, 19 → 201**

**Now convert the decimal part .7 to base 3:**

**.7 \* 3 = 2.1 → 2**

**.1 \* 3 = 0.3 → 0**

**.3 \* 3 = 0.9 → 0**

**.9 \* 3 = 2.7 → 2**

**.7 \* 3 = repeating**

**Reading downwards .7 → .2002**

**Answer:  $19.7_{10} = 201.\overline{2002}_3$**

3 (4 pts) Consider the decimal number -23. We want to store it in an 8-bit long word that includes the sign bit. What are all eight bits for the following representations of -23?

(i) Signed Magnitude: **10010111**

(ii) 1's complement: **00010111; invert** → **11101000**

(iii) 2's complement: **add 1 to 1's complement** → **11101001**

4 (3 pts) What is the largest 2-digit number in base 12? (Give your answer in base 12). Convert this base 12 number to decimal.

**Base 12 has 12 symbols 0 - 9, A, B**

**Therefore the largest number is : BB**

$$\mathbf{BB}_{12} = 11 \cdot 12^1 + 11 \cdot 12^0 = 143_{10}$$

5 (1 pt) Which of the following number formats has the number of positive ( $> 0$ ) values and the number of negative values that can be represented **not** equal to each other? Circle your answer.

- Sign Magnitude
- 1's complement
- **2's complement** [ with 2's complement in  $n$  bits we can represent the number  $-2^{n-1}$  but not  $+2^{n-1}$ . The largest positive value we can have is  $(2^{n-1} - 1)$  ]
- IEEE single precision floating point
- IEEE double precision floating point

6 Consider two signed numbers A and B. Numbers are six bits long including the sign bit. Negative numbers are stored in their 2's complement form.  $A = 110110$  and  $B = 101011$ .

(i) (1 pt) What is the decimal value of A?

**$A = 110110$  ; invert  $\rightarrow 001001$  ; add 1  $\rightarrow 001010 = 10_{10}$  ;  
therefore  $A = -10_{10}$**

(ii) (2 pts) What is the result of  $A - B$  in binary form? For full credit you must do binary subtraction and fully show your work.

**$A - B = 110110 - 101011 = 001011$**

(iii) (1 pt) State your answer (for  $A - B$ ) in decimal form.

**$001011 = +11_{10}$  [ $A = -10_{10}$  and  $B = -21_{10} \rightarrow A - B = (-10) - (-21) = 11$ ]**

(iv) (2 pts) Did an overflow/underflow occur? If No, for what values of B would it occur? If Yes, for what values of B would it not occur?

**No.** Overflow/underflow cannot occur when we subtract two numbers of the same sign, which is the same as adding two numbers of opposite signs.

**$A - B$  would underflow if  $B \geq +23_{10}$ . For no value of B could we have an overflow.**

- 7 Consider the 4-byte number expressed in HEX shown in the figure below. It is stored in the memory of a 32-bit architecture machine starting at word address 25 (which starts at byte address 100).

byte 100	byte 101	byte 102	byte 103
0x4D	0xE1	0x15	0x9A

- (i) (1 pt) Suppose the machine is big endian and the number is a 2's complement integer. Write the number in HEX.

**0x4DE1159A**

- (ii) (2 pts) Suppose the machine is big endian and the number is a single-precision floating point value. Is the number positive or negative?

**0x4 = 0100 → the first bit of the number is therefore 0, so the number is positive**

- (iii) (1 pt) Suppose the machine is little endian and the number is a 2's complement integer. Write the number in HEX.

**0x9A15E14D**

- (iv) (3 pts) Suppose the machine is little endian and the number is a single-precision floating point value. What is the value of the exponent in decimal, after removing the excess?

**The first 12 bits of the number are 0x9A1 = 1001 1010 0001**

**The exponent is bits 1 - 8 (where we start the counting at the leftmost bit, the sign bit, as bit 0) → 00110100 = 52<sub>10</sub>**

**Removing the excess 127 → 52 - 127 = -75<sub>10</sub>**

- 8 (2 pts) Consider the following Fetch-Decode-Execute cycle of a machine that is similar to the John von Neumann machine.
- (1) Fetch the instruction.
  - (2) Decode the instruction.
  - (3) Execute the instruction.
  - (4) Update the Program Counter ( $PC = PC+1$ ).
- Will this work properly? If so, explain why. If not, explain how it might fail.

**No. If we are executing an instruction which alters the value of the PC (a Jump instruction, for example), the Execution step (3) above will cause the PC to be correctly updated to the address of the next instruction to be executed. Step (4) will now go and increment that address by 1. The instruction that will be executed in the next Fetch-Decode-Execute cycle will therefore be the one after the targeted instruction, which is wrong.**

**The PC should be updated immediately after the Fetch step, and before the Decode and Execute steps.**

- 9 Consider a machine that is similar to the von Neumann machine, except that one word contains only one instruction. Also the size or length of each word is the same as the size of an instruction. Each instruction has a 6-bit opcode. Remaining bits in an instruction are for an address. The machine has 1,000 words of memory.

- (i) (2 pts) What would be the size of MAR?

**Answer =  $10_{10}$ . We need 10 bits to address 1,000 words. The MAR has to be big enough to hold an address.**

- (ii) (2 pts) What would be the size of the MDR be?

**The size of a word is  $6 + 10 = 16$  bits. The MDR has to be big enough to hold a word.**

- (iii) (1 pt) How many different instructions can this machine have?

**$2^6 = 64$  instructions.**

- 10 (i) (2 pts) Briefly explain what is meant by a cache *hit*.

A cache hit is when we find the next instruction we want to fetch for execution, or the data referred to by the current instruction during its execution phase, is already in the cache.

- (ii) (2 pts) Briefly explain what is meant by *locality* and how it relates to the effectiveness of using a cache memory.

Locality refers to the notion that when we access an instruction for execution, the next few instructions to be executed after that are likely to be in the same vicinity of main memory as this instruction. This is due to the nature of program code: it is straight-line code that is executed serially with relatively few jumps (due to method invocations, for example). Furthermore, because of the prevalence of loops in programs, a considerable amount of execution time is spent repeatedly re-executing a given block of code.

Thus, when we fetch a block of code from memory into the cache in response to a cache miss, there is a good chance that this block will also contain the next few instructions to be executed, so that a cache miss is likely to be followed by many cache hits. Without this, cache would not be an effective solution to the processor-memory speed mismatch.

A similar, though weaker, dynamic of memory locality also applies to the relationship between the data items the current instruction being executed refers to, and the data items that are likely to be referred to by the next few instructions.

- 11 (i) (1 pt) For a register machine, we refer to a register in an instruction using its number. Suppose we have 128 registers in our processor. How many bits do we need in an instruction to refer to a register?

$$2^7 = 128 \rightarrow \text{so we need 7 bits}$$

- (ii) (2 pts) Briefly explain what the difference is between a *register-memory* and a *memory-memory* architecture.

In register-memory architecture, one of the operands referred to in a machine instruction can be in memory (i.e., one operand in the instruction can be a main memory address). The other operand(s) must be in registers. In memory-memory, (up to) two operands can be in memory.

- 12 (4 pts) Consider an array A of 10 integers in MIPS memory. Assume that the base address of A (the address of A[0]) is in register \$s0. Suppose we want to carry out the calculation  $RESULT = (A[3] + A[7]) - (A[1] + A[5])$ . The address of variable RESULT is in register \$s1. Write the corresponding MIPS instructions to perform the operation. You may use temporary registers.

```
lw $t0, 12($s0)    # load A[3]
lw $t1, 28($s0)    # load A[7]
add $t0, $t0, $t1  # $t0 gets A[3] + A[7]
lw $t1, 4($s0)     # load A[1]
lw $t2, 20($s0)    # load A[5]
add $t1, $t1, $t2  # $t1 gets A[1] + A[5]
sub $t0, $t0, $t1  # calculate RESULT
sw $t0, $s1        # store RESULT
```

Other patterns of register use are possible. The solution above uses the minimum number of registers possible.