

NAME _____
First Last

Student ID# _____

UNIVERSITY AT STONY BROOK
COMPUTER SCIENCE DEPARTMENT
MIDTERM #1 EXAMINATION

CSE 220
Spring Semester 2009
February 26, 2009

This is a closed-book exam (80 minutes). It has 10 problems.

You are NOT allowed to use calculators

- | | |
|-------------------|-------------------|
| # 1 _____ (5 pts) | # 6 _____ (2 pts) |
| # 2 _____ (7 pts) | # 7 _____ (3 pts) |
| # 3 _____ (9 pts) | # 8 _____ (4 pts) |
| # 4 _____ (9 pts) | # 9 _____ (3 pts) |
| # 5 _____ (6 pts) | #10 _____ (2 pts) |

TOTAL _____ (50 max)

1 (5 points) Mark Yes or No.

- (a) A MIPS instruction carries/stores the full 32-bit operand address in it. N
- (b) An I-cache is used for fast access to data items (variables) of a program. N
- (c) Register 'zero' of MIPS reserved for use by the operating system. N
- (d) The Floating Point representation that we studied, uses a special representation (other than normalized) for decimal 1.0 . N
- (e) According to the memory/storage pyramid (hierarchy) that we studied, the cost per byte goes down as we move up the pyramid. N

2 (7 points)

- (a) (2 points) Consider binary numbers that are 6 bit wide including the sign bit. Let $X = 8$ (in decimal). Represent $-X$ (negative X) in signed magnitude (SM), and in 1's complement form.
- (i) (1 point) SM **101000**
- (ii) (1 point) 1's complement **110111**
- (b) (5 points) Let numbers be six (6) bits wide including the sign bit. Negative numbers are stored in 2's complement form.
- (i) (2 points) What is the range for negative numbers that we can store? Give answer in decimal, such as "-1 to -7" or something to that effect.

The range is -1 to -32

- (ii) (3 points) Let $A = 010010$ and $B = 011011$. Perform $A-B$ by first obtaining the 2's complement of B . **Also, clearly state if you get an overflow or underflow.** Show all steps and explain your answer.

010010	We have no overflow or underflow here. Sign bits
+ 100101	of A and 2's comp of B are different. We will never
-----	have overflow/underflow in such a case.
110111	

3 (9 points)

(a) (3 points) Convert 27.61 in base 8 to a number in base 4. Show all steps.

First convert to binary: 27.61 is 010111.110001_2
Next, group every 2 bits together: 01 01 11 . 11 00 01
Now convert back to base 4 $\rightarrow 113.301_4$

(b) (1 point) The 7-bit ASCII code for A is 1000001. What is the 7-bit ASCII code for E?

The 7-bit ASCII code for E is 1000101

(c) (1 point) What would be an 8-bit ASCII code for E be?

The 8-bit ASCII code for E is 01000101

(d) (1 point) What is the largest 2-digit number in base 5? (Give your answer in base 5.)

44_5

(e) (1 point) Convert base-5 answer in part (d) above to decimal.

24_{10}

(f) (2 points) Convert 0.9 in decimal to a binary number. Give your answer with up to 3 bits after binary point.

Multiply 0.9 by 2 repeatedly, and collect the whole integers to the left of the decimal point:

$0.9 \times 2 = 1.8$, collect the '1' $\rightarrow 0.1$
 $0.8 \times 2 = 1.6$, collect the '1' $\rightarrow 0.11$
 $0.6 \times 2 = 1.2$, collect the '1' $\rightarrow 0.111$

4 (9 points) Perform the addition $(0.0111 + 0.111)$ after normalizing these two binary numbers. **You must follow the single-precision IEEE standard and its procedures throughout.** Align the exponents and perform the addition. Next, normalize the result and convert it to IEEE format.

(a) (2 points) First do the normalization only, giving the normalized binary numbers in a $x \times 2^b$ format.

First number: **1.11×2^{-2}**

Second number: **1.11×2^{-1}**

(b) (2 points) Align the exponents and add.

We align the smaller number (the first number) to the larger (the second number) and add:

$$\begin{array}{r}
 0.111 \times 2^{-1} \quad \text{First number after 1 left shift of the binary point} \\
 1.110 \times 2^{-1} \quad \text{Second number} \\
 \hline
 10.101 \times 2^{-1} \quad \text{Sum}
 \end{array}$$

(c) (3 points) Normalize the result and convert it to IEEE format. Show all steps. Clearly indentify the sign, biased exponent and fraction bits.

Normalize the sum: 1.0101×2^0

Convert this to 32-bit IEEE single precision format:

- Sign bit is 0
- Exponent is $0 + 127 = 127$. In 8 bits, the excess 127 exponent is 01111111
- The fraction is 01010000000000000000

Thus, the 32-bit single precision binary number is:

00111111 10101000 00000000 00000000

(d) (1 point) Write your answer from part (c) above as an 8-digit hexadecimal value.

0x3FA80000

(e) (1 point) Suppose we load the result into a little endian memory. Show how this would be done by filling the boxes below with the hexadecimal digits of your answer to part (d) above.

byte 100	byte 101	byte 102	byte 103
0x00	0x00	0xA8	0x3F

5 (6 points)

- (a) (2 points) When do we increment the PC (Program Counter) during the instruction (“Fetch-Decode-Execute”) cycle of the John von Neumann machine? Clearly explain why for full credit.

After Fetch, before the Execute phase, to allow for ‘Jump’ instructions. Because a Jump instruction alters the value of the PC, we do not want to then go and increment that altered value, which is what would happen if PC were incremented after the Execute phase.

- (b) (2 points) When computer architects decided to add several registers to a processor chip, what part of the Fetch-Decode-Execute cycle’s performance was improved or? Clearly explain why for full credit.

The Execute phase of the instruction cycle was improved after registers were added to the processor. This increased the chance of operands being found already in registers, thus saving slow memory access to fetch them, and improving the overall performance of the computer.

- (c) (2 points)

- (i) (1 point) Explain what a *cache miss* is?

When processor/CPU tries to read an item from cache and the item is not there, we call it cache miss.

- (ii) (1 point) Suppose we access cache 300 times during which there are 30 misses. What is the percentage (%) hit rate?

**30 misses out of 300 access attempts, so the miss rate is 10%
Therefore, hit rate is (100 - miss rate) = 90%**

6 (2 points) Consider a machine similar to the John Von Neumann machine, but with 16K words of memory and 7 bits for the opcode. An instruction in this machine stores an opcode and **two** memory addresses.

- (a) (1 point) What is the size of its MAR (Memory Address Register) in bits?

$$\log_2(16K) = 14 \text{ bits}$$

- (b) (1 point) What would the size of its instruction be, in bits?

$$\text{opcode} + 2 \text{ addresses} = 7 + 14 + 14 = 35 \text{ bits}$$

- 7** (3 points) Consider the following declarations in some ‘generic’ high level language. Assume that memory is byte addressable, and the byte-order is big endian. The word size is 32 bits. Memory is allocated to the variables in the order they are declared. Numerical variables (*int*, *real*, *double*) must be properly aligned on word boundaries in memory.

```
double z;
char c1, c2; //these are 1-byte character variables
int x;
```

After memory is allocated to the variables, it is found that the address of *c1* is 20025972 (in decimal). What would be the memory addresses (in decimal) of:

- (a) (1 point) *z*? **20025964**
- (b) (1 point) *c2*? **20025973**
- (c) (1 point) *x*? **20025976**

8 (4 points)

- (a) (1 point) A memory-memory architecture has an add instruction, ADD A, B. Here, A and B are locations (variables) in memory. This instruction reads A, followed by B, adds them, and writes the result back to B. How many times is memory accessed for read/write during the execution of this instruction?

Two reads and one write, for a total of 3 memory accesses

- (b) (2 points) According to the MIPS assembler, ‘lw \$s1, A’ is a valid instruction, where A is a location (variable) in memory, and \$s1 is a register of MIPS. Similarly, ‘sw \$s1, A’ is also a valid instruction. Suppose we write MIPS code for the high-level statement $X = A+B$, as follows:

MIPS code	Comment
lw \$s1, A	load A into \$s1.
add \$s1, B	add B to \$s1.
sw \$s1, X	store \$s1 into X.

- (i) (1 point) Is this a valid MIPS program? **N**

(ii) (1 point) Explain your answer in (i) above.

The instruction ‘add \$s1, B’ is not allowed. MIPS is a RISC load/store architecture: arithmetic operations cannot access memory.

(c) (1 point) Suppose we have 9 single precision variables stored in floating point registers of MIPS. How many additional double precision variables can be stored in the floating point registers of MIPS? You must explain your answer for full credit.

Number of single precision floating point registers available = 32. 9 are used, so 23 are left. We need two registers each for double precision variables. Thus 11 more double variables can be stored.

9 (3 points) Consider the following program for a machine similar to the John von Neumann machine. It divides A by B by repeated subtraction, and stores the quotient and remainder obtained in Q and R, respectively. After each subtraction, it increments Q to Q+1. When the result of subtraction is negative it adds back B to the accumulator and restores the remainder R. For convenience assume that one location in memory holds only one instruction. Also assume that A and B have positive values, and that the initial values for both Q and R is 0 (zero). The program is stored starting at location 100.

Location	Instruction	Comment
100	LOAD A	Load A to accumulator (acc)
101	SUB B	Subtract B from acc
102	JLTZ 107	Jump to location 107, if acc < 0
103	LOAD Q	Load Q to acc
104	INC	Increment acc by 1
105	STORE Q	Store acc to Q
106	JMP 100	Jump to location 100
107	ADD B	Add B to acc, restore remainder
108	STORE R	Store acc to R
109	STOP	Stop

(a) (1 point) This program does not work (in the sense that it does not stop for some values of A and B). State what is the error here.

After the subtraction A–B (instruction 101), we must store the result back into the memory location for A, otherwise we will lose it because the value in the accumulator is subsequently overwritten by ‘LOAD Q’ (instruction 103). The error here is that this ‘STORE A’ is missing. The code stores Q and stores R, but it fails to store A and so does not perform $A = A - B$ properly.

- (b) (2 points) Fix the error and rewrite the program, also including the memory location addresses of the instructions (as is done above). You do not need to write comments.

Location	Instruction	
100	LOAD A	
101	SUB B	
102	STORE A	<----- Insert this
103	JLTZ 108	<----- Changed from 107. Update addresses of
104	LOAD Q	instructions that
105	INC	were originally in
106	STORE Q	locations 102 to 109
107	JMP 100	
108	ADD B	<----- This is now in 108.
109	STORE R	
110	STOP	

- 10 (2 pts) Consider the following piece of MIPS assembly code which is doing some calculation on elements of an integer array A. The base address of A (the address of A[0]) is in register \$s0. The last line of the code stores the result of the calculation in variable RESULT whose address is in register \$s1.

```
lw $t0, 12($s0)
lw $t1, 20($s0)
sub $t0, $t1, $t0
lw $t1, 36($s0)
lw $t2, $s0
add $t1, $t1, $t2
sub $t0, $t1, $t0
sw $t0, $s1
```

Write what the calculation is as a high-level language instruction (e.g., "RESULT = A[3] - A[5] . . .").

RESULT = A[0] + A[9] - (A[5] - A[3])