

NAME _____
First Last

Student ID# _____

UNIVERSITY AT STONY BROOK
COMPUTER SCIENCE DEPARTMENT
MIDTERM #1 EXAMINATION

CSE 220
Fall Semester 2009
October 8, 2009

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

You are NOT allowed to use calculators

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

TOTAL _____ (50 max)

1 (5 points) Mark Yes or No.

- (a) The program counter must be updated before the execution phase of the instruction cycle. **Y**
- (b) If we provide a unified cache for a JVN-like machine, its performance may not improve that much. **N**
- (c) Register 1 of MIPS always stores 1. **N**
- (d) In the floating point system that we studied, zero is stored as a normalized number. **N**
- (e) Registers are high speed memory locations inside the processor (CPU). **Y**

2 (7 points)

- (a) (2 points) Consider binary numbers that are 7-bit wide including the sign bit. Let $X = 9_{10}$ (in decimal). Represent $-X$ (negative X) in signed-magnitude (SM), and in 1's complement form.

(i) (1 point) SM **1001001**

(ii) (1 point) 1's complement **1110110**

- (b) (5 points) Let numbers be 5-bit wide including the sign. Negative numbers are stored in 2's complement form. Suppose $A = 01011_2$ and $B = 11101_2$. Here, B is already negative.

(i) (2 points) Convert B to a decimal value.

-3_{10} in decimal

(ii) (3 points) 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.

01011	2's complement of $B = 00011$.
+ 00011	We have no overflow or underflow here: sign bits
-----	of A and 2's comp of B are the same value as the
01110	sign bit of the result.

3 (7 points)

(a) (2 points) Convert 23.12_4 (in base 4) to a number in base 8. Show all steps.

First convert to binary: 23.12 is 1011.0110_2

Next, group every 3 bits together: $001\ 011\ .\ 011\ 000$

Now convert back to base 8 $\rightarrow 13.3_8$

(b) (1 point) Suppose we want to code letters A through Z (only upper case) using binary numbers, starting with 0. What is the minimum number of bits that we need?

We need 5 bits to code A through Z (26 letters).

(c) (1 point) Suppose we want to code letters A through Z (only upper case) using base 3 numbers, starting with 0. What is the minimum number of base 3 digits that we need?

We need three base 3 digits (which will give us up to 27 values).

(d) (1 point) What would be the base-3 code for letter F in part (c) above?

012_3 (which is equal to 5_{10}).

(e) (2 points) Suppose we convert 2.2_{10} (in decimal) to a binary number with fraction. We stop the algorithm after we have 4 bits of binary fraction. Now suppose we convert this binary number with fraction, back to a decimal number, say $2.y_{10}$. (Note that you do not necessarily have to actually fully convert back and forth to be able to answer the question.)

(i) (1 point) Is $2.y_{10}$ greater than, less than or equal to 2.2_{10} ?

$2.y_{10}$ will be less than 2.2_{10} .

(ii) (1 point) Briefly explain your answer.

When we convert 2.2_{10} to binary, the process does not terminate. So whatever binary fraction we have accumulated so far (4 bits) does not exactly represent 0.2 in decimal. This process can go on for ever. We slowly approach the value 2.2_{10} from below, but it never becomes 2.2_{10} . So when we convert it back to the decimal value $2.y_{10}$, it has to be less than 2.2_{10} .

4 (3 points)

- (a) (1 point) RISC architectures are presumably better than CISC Intel processors. But they did not capture a major share of the market. Briefly explain why.

Because Intel provided backward compatibility with their new processors, customers did not change to a new architecture. Beginning with the 80486 and Pentium-1, Intel implemented a RISC-like core for its processors.

- (b) (1 point) MIPS instructions cannot store the full 32-bit memory address of an operand. Why?

Addresses in MIPS are 4-byte or 32-bit wide. All instructions are also 32-bit wide. Obviously, if an instruction stored a full address, there would be no room for storing an opcode, register numbers, etc. (That is why an operand address is first loaded into a register; this register is then specified or referred to in the instruction that wishes to make use of it, as in the instruction 'lw \$t0, 12(\$s0)', for example.)

- (c) (1 point) Why are RISC architecture machines are also known as *load-store* ?

RISC architectures are also know as load/store architectures because only load and store instructions are allowed to access memory. All other instructions work strictly with registers only.

5 (6 points) Assuming a John von Neumann architecture machine, answer the following:

- (a) (1 point) List the phases of an instruction cycle, in order.

Fetch – Decode – Execute.

- (b) (2 points) Explain the **execution** steps for an ADD instruction, using the MAR, MDR, IR (instruction register), ACC (accumulator), and memory read/write sub-steps.

For example, ADD B – read value in memory location B, and add it to accumulator.

**Steps: send address of B from IR to MAR
 issue memory read
 value of B is put in MDR
 $ACC \leftarrow ACC + MDR$**

(c) (3 points) Now suppose we change the machine slightly, so that memory access is using a shared bus.

(i) (1 point) Which one of the two operations {read} or {write} is affected by the shared bus?

The write operation will be affected by the shared bus.

(ii) (1 point) Is it made faster or slower?

It will be made slower.

(iii) (1 point) Briefly explain why.

For a write operation, the processor needs to send a memory address, and a value to be stored into memory at that address. On a shared bus this transfer takes two trips (two bus transfer cycles).

If we have a full width memory bus, this transfer would take only one bus cycle.

6 (9 points) Perform the addition $(11.0111 + 0.111)$ 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: 1.101111×2^1

Second number: 1.11×2^{-1}

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

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

1.10111×2^1	First number
0.01110×2^1	Second number after 2 left shifts of the binary point
10.00101×2^1	Sum

- (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: 1.000101×2^2

Convert this to 32-bit IEEE single precision format:

- Sign bit is 0
- Exponent is $2 + 127 = 129$. In 8 bits, the excess 127 exponent is 10000001
- The fraction is 0001010000000000000000

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

01000000 10001010 00000000 00000000

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

0x408A0000

- (e) (1 point) Suppose we load the result into a big 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
0x40	0x8A	0x00	0x00

7 (5 points)

- (a) (1 point) What is a cache hit?

When a processor tries to read a variable/value/location from the cache and finds it in the cache, we call that a cache hit.

- (b) (1 point) The idea of a cache memory is based on the principle of (fill in the missing word or phrase).

locality

- (c) (1 point) The block size for a cache is always more than just one word. Briefly explain why.

If the block were just one word, then on a cache miss we would bring in just that word and store it in the cache. No extra words would be brought in. These extra words are supposed to increase the number of cache hits, based on the principle of locality. So the net effect is just extra overhead in getting this single word from memory into the cache, with no significant improvement in performance.

- (d) (1 point) What is the total number of double-precision values we can store using all the floating point registers of MIPS?

16. There are 32 floating point registers, and we need to use two for each double-precision value.

- (e) (1 point) The Program Counter (PC) is one of the 32 ‘*general purpose*’ registers of MIPS. True or false?

False.

- 8 (2 points) Consider the following instruction in the `.data` portion of a MIPS assembly language program.

```
str: .asciiz "My-Pizza"
```

Suppose the string is loaded into a little endian memory starting at word address 100 (which is byte address 400: each word is four bytes). Show how this is done by filling the boxes below (you can just write the characters of the string into the boxes - you do not have to write the numerical ASCII values).

word 100				word 101				word 102			
'P'	'.'	'y'	'M'	'a'	'z'	'z'	'i'				'\0'

Split the string into groups of four bytes (one word). Each group is put into a memory word in little endian order; *i.e.*, we put the character in the least significant byte of the group into the low address byte of the memory word, and move ‘backwards’ along the characters of the group as we fill the next higher-address bytes of the memory word.

- 9 (4 points) Consider the following program for a machine similar to the John von Neumann machine. Assume for simplicity that one location in memory holds only one instruction. Variables A, B and C have positive (> 0) values. Variable S holds the result.

Location	Instruction	Comment
201	LOAD A	Program starts here
202	STORE S	
203	LOAD B	
204	SUB C	Subtract C from accumulator.
205	JMPZ 211	Jump to location 211, if accumulator contains 0.
206	STORE B	
207	LOAD S	
208	ADD A	
209	STORE S	
210	JMP 203	Jump to location 203
211	STOP	

- (a) (2 points) If initially A is 6, B is 4, and C is 2, what would the value of S be when program stops?

**Initially, we set $S = A$; so S is 6. The loop is executed once; so $S = S + A$.
At the end value in S is 12.**

- (b) (2 points) Now suppose A, B, C and S are re-assigned to locations (words) with addresses 2001, 2002, 2003, and 2004 respectively. Also, the real JVN machine stores **two** instructions in one word. Rewrite the program, with two instructions in each word, starting at word location 800. Change variable names to correct memory addresses. You also need to change the opcodes involving all jump instructions to JUMPL / JUMPR (jump to the instruction in the left / right half of the word), and/or JMPZL / JMPZR (jump left / right on zero accumulator).

Address	Left Instruction	Right Instruction
801	LOAD 2001	STORE 2004
802	LOAD 2002	SUB 2003
803	JMPLZ 806	STORE 2002
804	LOAD 2004	ADD 2001
805	STORE 2004	JMPL 802
806	STOP	

- 10 (2 points) 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 $A[9] = (A[6] - A[3]) - (A[2] + A[7])$. Write the corresponding MIPS instructions to perform the operation. You may use temporary registers.

```
lw $t0, 24($s0)    # load A[6]
lw $t1, 12($s0)    # load A[3]
sub $t0, $t0, $t1   # $t0 gets A[6] - A[3]
lw $t1, 8($s0)     # load A[2]
lw $t2, 28($s0)    # load A[7]
add $t1, $t1, $t2   # $t1 gets A[2] + A[7]
sub $t0, $t0, $t1   # calculate final result
sw $t0, 36($s0)    # store in A[9]
```

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