nop is same as sll $0, $0, 0 for example

Question 1: Potpourri (19 points, 30 minutes)

a) Decide whether each of the following statements is True or False. 1 Pt each

i) T F The MIPS instruction addiu will sign-extend the immediate to 32 bits.

ii) T F Every different MAL instruction assembles to a distinct binary encoding. MAL includes pseudoinstrs.

iii) T F If a label is never jumped to, it is not needed in the linking stage to instr.

iv) T F Like Two's Complement, Floating Point has more negative values than positive Same amt (sign bit on or off)

v) T F It is the Caller's responsibility to save temporary registers before making a function call. Fxn call will destroy registers.

b) For each of the numbered statements (i-v) below, choose the letters of the cache parameter changes that definitely achieve the named outcome. There may be more than one letter for each statement. 2 Pts each

A) Adding a unified L2 cache, which is larger than L1 but smaller than memory
B) Increasing block size while keeping cache size constant
C) Increasing associativity while keeping cache size constant
D) Increasing cache size while keeping block size constant.

i) Definitely increases number of tag bits used (for L1 cache) C
ii) Definitely increases number of index bits used (for L1 cache) D
iii) Definitely increases number of offset bits used (for L1 cache) B
iv) Definitely decreases L1 miss penalty A
v) Definitely increases L1 hit time CD -1 missing one or extra one

c) We have extracted 4 bytes of data from 3 files of distinct file types (left column). By drawing arrows, match each data with the program in the right column that would take as input the file that the data is from.

j 0x0013F00, a resolved target addr., matches w/ Loader

4 Pts
2 Pts if one matches and other 2 are switched.

0x08013F00	Compiler
0x0c000000	Linker
0x6D61696E	Loader

jal 0x0
Target addr not yet resolved (0 is invalid addr), matches w/ Linker

ASCII for "main"
Compiler takes text input.

See few. file for more details on rubric.

Question 2: Sum Things Up With This Question (10 points, 15 minutes)

128 32 42

a) What is the value in decimal of 0b10100110 in each representation? 1 pt each

0b01011001

unsigned	166
one's complement	-89
two's complement	-90
sign and magnitude	38
floating point exponent	166-127 = 39 or 2^{39}

To negate: Flip bits

Flip bits, add one

Interpret bits as unsigned #

Subtract the bias

b) Circle the cases in which using unsigned addition will yield the correct summation value when the arguments and the result are interpreted in the following number representations. Ignore overflow cases.

3 pts for this col. 2 pts for this col

one's complement	adding positive integer arguments	adding negative integer arguments
two's complement	adding positive integer arguments	adding negative integer arguments
sign and magnitude	adding positive integer arguments	adding negative integer arguments

-1 pt for each difference.

Can use unsigned addition to perform 2's C. addition.

Representation of positive integers for each of those is same as for unsigned addition.

0b10000001 (-1)

+ 0b10000001 (-1)

0b00000010 (2)

0b1111 1110 (-1)

+ 0b1111 1110 (-1)

0b 1111 1100 (-3)

See test file for more details on rubric.

Question 3: "I'm Jack Bauer, and this is the coolest problem of my life" (15 points, 25 minutes)

After sitting through yet another heart-stopping marathon of the TV series *24*, we've decided to design a new 24-bit processor in honor of Jack Bauer. Naturally, it has the following characteristics:

- 24-bit words
- 24 24-bit registers
- 2^{24} locations of byte-addressed memory
- No floating point – the only point Jack Bauer needs is his .45
- The $zero register has been renamed $jb. You don't mess with Jack Bauer.

Otherwise it is similar to MIPS in that we will use many of the same instructions, branches use PC-relative addressing, and jumps use absolute addressing. Without floating point, the instruction set is smaller, so we will encode everything into a single opcode field and eliminate the funct field. The left column of the front side of your MIPS Green Card contains 31 instructions that we'd like to reproduce. We have decided on the field configurations for the three instruction types shown below:

R-format:	[opcode: 5 | rs: 5 | rt: 5 | rd: 5 | shamt: 4]
I-format:	[opcode: 5 | rs: 5 | rt: 5 | immediate: 9]
J-format:	[opcode: 5 | target address: 19]

Also accepted
- Make shifts I-fmt
- Use rs + shamt field for shift.

3pts
1pt limitation
2pts fix

a) What is one potential limitation with the shamt field for R-format instructions? How can we deal with this limitation to allow for full functionality?

4 bit shamt => can't shift more than 15. Can shift up to full 24 by adding pseudoinstr. that assembles to multiple shifts.

3pts

b) Based on the size of our immediate and our word size, it turns out that 'lui' and 'ori' are inadequate for our processing needs. Describe a new instruction that will fix the problem. Don't forget to give the instruction a name!

lmi (or whatever you called it). Immediate only 9 bits, need 3 different instrs. to load a 24 bit constant into a reg: lui, lmi, ori.

Hunting down terrorists is hard work. Jack needs to make sure he can move around effectively:

1pt.

c) What is the expression to update the $PC to go to the next instruction?

$PC = $PC + 3

2pts

d) What is the new expression for the resulting $PC address after a branch? Assume that the immediate counts by words.

$PC = $PC + 3 + (Immediate) * 3

2pts

e) What is the maximum jump distance (in bytes) of a branch statement? Assume the target address also counts by words.

$3 \cdot 2^8$ bytes

2pts

f) How much memory (in bytes) can we access using a jump statement?

$3 \cdot 2^{19}$ bytes (or 2^{19} since we never explicitly said the jump addr counted by words)

A developer is complaining that the word size is making it difficult/inefficient to get around. Jack's used to it, since he never does things the easy way, but we're willing to listen. He suggests that we align each 24-bit word into 32-bit slots so we can reuse some of the hardware from MIPS.

1pt each

g) Name one advantage of each scheme (be specific, you are not allowed to say "we can reuse some of the hardware from MIPS"):

An advantage to using sequential 24-bit addressed words is:

Save space in memory

An advantage to using aligned 32-bit slots for our words is:

Easier to compute target addresses (multiply by 4 instead of by 3, can just shift by 2)

Addressing for j instrs becomes much simpler.

Question 4: split (17 points, 40 minutes)

In this question, you will be implementing the function `split` in C. Given a string s and a char c, `split` should return an array of the strings that result when s is separated by c. In general, if c occurs n times in s, `split` should return an array of n+1 strings.

Examples:

```
split("My name is Michael",' ');
{"My","name","is","Michael"}

split("Howdy",'d');            // The character doesn't have to be a space
{"How","y"}

split("Hello World",'l');      // Note the empty string returned where 'l'
{"He","","o Wor","d"}          // appears twice.

split("Banana",'x');
{"Banana"}
```

a) First, complete the function countChar, which returns the number of occurrences of char c in string s.

```
int countChar(char* s, char c)   {
    int count = 0;
    while (*s) {
        if (*s == c) count ++;
        s++;
    }

    return count;

}
```

4 pts

-2 pts - using strlen or not checking null terminator (-1 pt for writing '\n' instead of '\0')

-1 pt minor error (missing dereference, return statement, etc)

b) Now, complete the function `split`. You may use `countChar` in your solution. A few comments and lines of code have been provided for you.

You may not modify the original string that s points to.

You may use any of the functions in the C library <string.h> except `strtok` (which is very similar to `split`). In particular, the function `strncpy(char* destination, char* source, size_t num)`, which copies `num` characters from `source` to `destination`, may be particularly useful.

Space for array of ptrs to result strings.

```
char** split(char* s, char c)    {
      char** result;
      int resultIndex, resultLength;
      //Put other local variables here
        char * temp;

      //Initialize variables, do other work to set up
        resultLength = countChar(s, c) +1;

        result = (char**) malloc(sizeof(char*) * resultLength);
        temp = s;

      //Process each result string
      for(resultIndex=0; resultIndex<resultLength; resultIndex++) {
          while (temp && temp != c) temp++;

          result[resultIndex] = malloc(sizeof(char) * (temp - s +1));
          strncpy(result[resultIndex], s, temp - s);
          result[resultIndex][temp-s] = '\0';
          s = temp +1;
      }
      return result;
}
```

Last iter ends at null rather than 'c' character

Rubric 13 pts

6pts for mallocs

3 pts each

-1 pt per missing detail (no null terminator, dereference error, etc.)

-2 pts for missing sizeof(char*) in top malloc.

7pts for algo and pointer use

-1 pt per minor error

about -.5 pts per error after 3 errors

-2pts No temp variable to handle start or end of current result string.

-2pts no inner while loop (or similar)

No pts. deducted for some minor mistakes.

<u>Question 5:</u> Cache Flow (10 points, 15 minutes)

Consider the following data structure that keeps track of employees at a company.

```
typedef struct {
	int salary;
	int bonus;
	int vacationTime;
} Employee;
```

Your co-worker (a Stanford graduate) writes a short routine to sum these pieces of information across a very large employee database:

```
//Returns a pointer to the sums of the three employee statistics
int* computeStatistics(Employee *database, int numEmployees) {
	int i;
	int *result = malloc(sizeof(int)*3);

	result[0] = result[1] = result[2] = 0; //initialize sums

	for(i=0;i<numEmployees;i++) {
		result[0] += database[i].salary;
	}

	for(i=0;i<numEmployees;i++) {
		result[1] += database[i].bonus;
	}

	for(i=0;i<numEmployees;i++) {
		result[2] += database[i].vacationTime;
	}

	return result;
}
```

Your co-worker complains that his routine runs too slowly.

a) Describe in one sentence what about your co-worker's routine causes it to run slowly?

Accesses memory w/ poor spatial locality.

b) Below, rewrite a version of `computeStatistics` that will achieve better performance:

```
int* computeStatistics(Employee *database, int numEmployees) {
    int i;
    int *result = malloc(sizeof(int)*3);
    result[0] = result[1] = result[2] = 0; //initialize sums
    for (i=0; i < numEmployees; i++) {
        result[0] += database[i].salary;
        result[1] += database[i].bonus;
        result[2] += database[i].vacationTime;
    }
    return result;
}
```

Rubric (Question considered as a whole) 10pts

4pts - Code in part b) correct, other parts wrong or too vague

6pts - Other parts mention cache or mem performance.

8pts - Other parts nearly correct.

10pts - Other parts completely correct, full explanation.

7pts - "Loop overhead" with correct explanation but wrong performance increase (would be relatively small)

c) By approximately what factor does your optimized code in part b) outperform your co-worker's code? Briefly justify your answer.

A factor approaching 3.

My code accesses the array at increments of 4 bytes instead of 12 bytes, so I get 3x as many cache hits per block that is fetched => 1/3 as many misses.

— or —

His code loads all of the blocks that span the array 3 times, mine loads each of these blocks once => 1/3 as many misses refreshing these blocks.

1/3 the miss rate => close to 3x speedup since miss time is typically larger.

```
# Function prototype, for reference:
# play(char* world, int* commands, int currentRow)

Play:
      addiu $sp $sp -4          #fill in the prologue
      sw   $ra 0($sp)
      lw    $t1, 0($a1)         #$t1 holds current command
      addu  $t2, $a0, $a2       #$t2 holds address of current row
      lbu   $t3, 0($t2)         #$t3 holds the current row's byte
Check1:
      addiu $t0, $0, 1          #$t0 used to check current move
      bne   $t0, $t1, Check2
      sb    $t3, -1 ($t2)       #move up
      sb    $0, 0($t2)          #clear old position
      addiu $a2, $a2, -1        #update currentRow
      j     NextMove
Check2:
      addiu $t0, $t0, 1
      bne   $t0, $t1, Finished
      srl $t3 $t3 1
      sb $t3 0($t2)             #move right and update world
NextMove:
      addiu $a1, $a1, 4         #set up arguments
      jal Play

Finished:
      lw $ra 0($sp)             #fill in the epilogue
      addiu $sp $sp 4
      jr $ra
```

Rubric

1 pt per correct full instruction

1/2 pt per correct fill-in

Similar mistakes indicated with ©
and only docked as one mistake

Question 7: A MATter of Performance (9 points, 15 minutes)

Bob's computer specs are currently:

Unified L1 cache
L1 cache hit rate of 90%
L1 cache hit time of 1 cycle
The miss penalty to main memory is 100 cycles
Ideal CPI of 1

a) What is his AMAT?

1pt

HT+MR·MP

$1 + .1(100) = 11$

$CPI_{stalls} = (AMAT - HT)\left(\frac{\text{mem access}}{\text{instr}}\right) = (MR \cdot MP)\left(\frac{\text{mem accesses}}{\text{instr}}\right)$

b) If he runs a program that has 50% loads/stores, what is his program's CPI?

50% loads/stores $= 1.5 \frac{\text{mem accesses}}{\text{instr}}$

$CPI_{ideal} + CPI_{stalls}$

2pt

-1 max if expression present

-1 per minor error

Instructions are data and therefore require a memory access!

$1 + 1.5 \frac{\text{mem accesses}}{\text{instr}}(.1 \cdot 100) = 16 \frac{\text{cycles}}{\text{instr}}$

Disgusted at his slow computer, he requests that you improve it by adding an L2 cache. He wants you to cut down his AMAT to 6.

c) The L2 cache you have in mind has a Local Miss Rate of 35%. What is the worst Hit Time it can have while still meeting Bob's request?

2pt

-1 max if expression present

-1 per new minor error

$HT_{L1} + MR_{L1}(HT_{L2} + MR_{L2} MP_{L2})$

$6 = 1 + .1(x + .35 \cdot 100)$

$.1x + 4.5 = 6$

$x = 15$

d) Bob doesn't like it, so you set Bob up with a different L2 cache with a Hit Time of 10 cycles. Now, his system has a Global Miss Rate of 6%. What is his new CPI?

2pt

-1 max if correct expanded expression present

-1 per new minor error

-1 if no new minor errors but 3 minor errors in b) (rare, but fair!)

Global $MR = MR_{L1} \cdot MR_{L2}$

Local $MR_{L2} = \frac{6\%}{10\%} = 60\%$

$(CPI_{ideal} + \frac{\text{mem}}{\text{instr}})(MR_{L1}(HT_{L2} + MR_{L2} MP_{L2}))$

$1 + 1.5(.1(10 + .6(100))) = 11.5$

e) What is the relative performance of his upgraded computer versus his old one?

$\frac{CPI_{old}}{CPI_{new}} = \frac{16}{11.5}$

2pt

-1 ratio flipped

-2 incorrect, no equation, and inconsistent

-0 correct term and consistent with b) and d) answers

Question 8: Mystery (7 points, 15 minutes)

Decipher the MIPS code below and explain in a sentence or two what it does (not instruction-by-instruction):

```
# $a0 -> array, $a1 -> length of array
Mystery:
        move $v0, $0
Label:
        slti $t0, $a1, 2      # exit if fewer than
        bne  $t0, $0,  Done   # 2 elements remaining
        lw   $t0, 0($a0)
        lw   $t1, 4($a0)
        slt  $t2, $t1, $t0
        add  $v0, $v0, $t2
        subi $a1, $a1, 1
        addi $a0, $a0, 4
        j    Label
Done:
        beq  $v0, $0,  Return1
        addi $v0, $0,  0
        jr   $ra
Return1:
        addi $v0, $0,  1
        jr   $ra
```

(increasing)
(non-decreasing)
(non-descending)

Returns 1 (true) if array ascending, 0 otherwise

7 pts

6 pts for "strictly ascending"

4 pts for "descending"

2 pts for "sorts the array"

4 pts for "1 if ascending, 0 if descending"