Number Representation

For this question, we are using 16-bit numerals. For floating point, use 1 sign bit, 5 exponent bits, and 10 mantissa bits. (Bias = 15)

Indicate in which representation, two's complement or floating point, the numeral is closest to zero:

	Value in TC:	Value in FP:	Closer to Zero:
1) 0x0000	0	0	Same
2) 0xFFFF	-1	NaN	Two's Complement
3) 0x0001	1	Denorm	Floating Point
4) 0xFFFE	-2	NaN	Two's Complement
5) 0x8000	INT_MIN	-0	Floating Point

We now wish to add the numerals from top to bottom first in two's complement, and then in floating point. For each, state which of the following occurs: overflow, underflow, NaN, or no error.

Two's Complement:

0 + (-1) = -1

-1 + 1 = 0

0 + -2 = -2

-2 + INT_MIN

Floating Point:

0 + NaN = NaN

C Programming

Given the following declarations:

```
char a[14] = "pointers in C";

char c = 'b';

char *p1 = &c, **p2 = &p1;
```

Which of the following are legal in C?

1) `p1 = a+5;`

The "a" without a subscript means &a[0], a constant of type pointer-to-char. Adding an integer to a pointer is legal, and returns a pointer-to-char, which matches the type of p1, so the assignment is LEGAL.

2) `&p1 = &a[0];`

Any expression starting with & is a constant, not a variable, so this is an attempt to assign a value to a constant, like saying 3 = 4; so it's ILLEGAL.

3) `p2 = a;`

Again, "a" means &a[0], a constant of type pointer-to-char. But p2 is of type pointer-to-pointer-to-char, so this is a type mismatch and is ILLEGAL. It would be legal, although weird, with an explicit cast: p2 = (char **)a;

4) `*(a+10) = 't'`

The "a+10" is a valid expression of type pointer-to-char, like a+5 in the first statement. So *(a+10) is a variable of type char, and we are assigning a char value to it, so this is LEGAL.

5) `*p2 = &c;`

p2 is of type pointer-to-pointer-to-char, so *p2 is a variable of type pointer-to-char. &c is a constant of type pointer-to-char. These match, so this assignment is LEGAL.

C-to-MIPS

We wish to convert the following code to MIPS without using pseudoinstructions:

```
struct Node { int n; struct Node *next; };

int sum (struct Node *head) {
   if (head == NULL)
       return 0
   else
       return head->n + sum(head->next);
}
```

Fill in the blanks (and where it says “maybe some lines here”) below:

```
sum:  li $v0 0
      beq $a0 $0 done        # check if head==NULL
      # Maybe some lines here?
      addiu $sp $sp -8       # save $ra and $a0 onto the stack
      sw $ra 4($sp)
      sw $a0 0($sp)
      lw $a0 4($a0)          # pass in head->next to sum()
      jal sum                # Make recursive call
      # Maybe some lines here?
      lw $a0 0($sp)          # load $ra and $a0 from the stack
      lw $ra 4($sp)
      addiu $sp $sp 8
      lw $t0 0($a0)          # get head->n
      addu $v0 $v0 $t0       # compute the sum
      # Note: we need to store the result into $v0
done: jr $ra
```

Reading MIPS

$a0 = the address of an array. $a1 = array length

```
Mystery:
        move $v0, $0
Label:
        slti $t0, $a1, 2        # exit if fewer than
        bne $t0, $0, Done       # 2 elements remaining
        lw $t0, 0($a0)          # loads the currently indexed element
        lw $t1, 4($a0)          # loads the next element
        slt $t2, $t1, $t0       # is next element less than current?
        add $v0, $v0, $t2       # if it is, increment $v0
        subi $a1, $a1, 1        # decrement
        addi $a0, $a0, 4        # increment index
        j Label                 # repeat
Done:
        beq $v0, $0, Return1    # branch if $v0 contains all zeros
        addi $v0, $0, 0         # if $v0 is nonzero, return 0
        jr $ra
Return1:
        addi $v0, $0, 1         # if $v0 is zero, return 1
        jr $ra
```

This code returns true if the array is ascending (non-decreasing) and false otherwise.

AMAT

Suppose that for 1000 memory references to a direct-mapped, three level cache, we have

- 40 misses in L1$, L1$ hits in 1 cycle — GMR_{L1} = 4%
- 10 misses in L2$, L2$ hits in 10 cycles — GMR_{L2} = 1%
- 10 misses in L3$, L3$ hits in 100 cycles — GMR_{L3} = 1%
- Main memory access costs 1000 cycles

a) What is the local miss rate for L3$?

10 / 10 = 100%

b) What is the AMAT?

1 + .04 * 10 + .01 * 100 + .01 * 1000 = 12.4 cycles

Or, we can compute the local miss rates first and use the recursive definition of AMAT.

LMR_{L1} = 4% , LMR_{L2} = 25%, LMR_{L3} = 100%

1 + .04 * (10 + .25 * (100 + 1 * 1000))) = 12.4 cycles

c) What is the AMAT if we didn't have L3$?

1 + .04 * 10 + .01 * 1000 = 11.4 cycles

Caches

Given a direct mapped cache, initially empty, and the following memory access pattern, what is the hit rate and miss rate if the cache has 8 32-bit blocks? Assume that memory is byte-addressed.

8	0	4	32	36	8	0	4	16	0
M	M	M	M	M	H	M	M	M	H

Hit rate = 0.2 Miss rate = 0.8

~~0~~ ~~32~~ 0
~~4~~ ~~36~~ 4
8
Empty
16
Empty
Empty
Empty

Consider a write-back, direct-mapped cache with 16 byte blocks and 64 KiB of data. Assume a byte-addressed machine with 32-bit addresses

a) Partition the following address and label each field with its name and size in bits:

Tag (16)	Index (12)	Offset (4)

b) Given the address 0xDEADBEEF, what is the value of the index, offset, and tag?

Index = 0xBEE
Offset = 0xF
Tag = 0xDEAD

c) How many cache management bits are there for each block? List them.

Tag = 16 bits
Valid = 1
Dirty = 1
Total 18 bits.

d) What is the total number of bits (data AND cache management) that comprise the cache?

2^{12} blocks * (128 data bits + 18 management bits) = 584 KiB

Miscellaneous Questions

Slide 1

Which of the following is *not* a job of the linker?

1) Relocation

This is the linker's main job: Each .o file think it starts at address zero, and the linker moves each one to its own actual starting address (code and data separately, etc).

2) **Compute branch offsets ← ANSWER**

The assembler does this. The relocation process doesn't change the offset, because both the branch instruction itself and the instruction to which it branches are moved, so the distance between them doesn't change.

3) Combine .o files

This is the reason relocation is necessary; that's what the "link" in "linker" means – to combine things.

4) Resolve external symbols

This is also part of the combining process, so that the .o files can refer to each other.

Fill in the blanks: The dominant form of parallelism in WSCs is _**MIMD**_.

Slide 2

True or False: The greatest part of monthly expenses for a datacenter are amortized capital expenses (CAPEX).

TRUE

True or False: MapReduce can give the wrong answer if a worker crashes.

FALSE

Slide 3

You have to finish your project using Amazon EC2 servers. You know the servers will finish the problem in 1hr using 10 machines, but you need to submit within 10 minutes or be late, so you booted 60 machines.

However, even though the cluster booted instantaneously (!!!) you were still late. This scenario indicates your solution lacked what kind of scaling?

Strong Scaling