**Question 4:** Numbers (1 point per answer, 5 points total)

Given the indicated number representation, fill in the blank with exactly one of LESS THAN (<), GREATER THAN (>), or EQUAL TO (=). E.g., 1 < 2 and 1 > 0!

a) Unsigned 32-bit numbers

A. 0111 0000 1111 0101 0111 0000 1111 0101_{two}
B. 0111 0000 1011 0101 0111 0000 1111 0101_{two}

A ___ > ___ B

d) IEEE Standard Single-Precision Floating-Point Numbers

A. 1111 1111 0111 0101 0111 0000 1111 0101_{two}
B. 1111 1110 0111 0101 0111 0000 1111 0101_{two}

A ___ < ___ B

e) IEEE Standard Single-Precision Floating-Point Numbers

A. 1000 0000 1111 0101 0111 0000 1111 0101_{two}
B. 1000 0001 0111 0101 0111 0000 1111 0101_{two}

A ___ > ___ B
**Question 2:** Pointers *(1 point per answer, 5 points total)*

Given below is a main function that prints five statements. What are the five things that are printed? If it is undefined what would be printed, please write UNDEFINED for the print. **Assume that the numbers are stored in littleendian format!**

Assume that ints are 32 bits and chars are 8 bits!

```c
int main(int argc, char** argv) {
    unsigned int data[] = {0x01234567, 0x89ABCDEF};
    unsigned char* charFront = (char *) data;
    unsigned char* charMid = (char *) &(data[1]);
    unsigned char* charEnd = (char *) &(*(data+2));

    // %x means print the variable in hex.
    printf("The first print: %x\n", (int) *(charFront+1));
    printf("The second print: %x\n", (int) *(charFront+4));
    printf("The third print: %x\n", (int) charMid[1]);
    printf("The fourth print: %x\n", (int) *(charMid-4));
    printf("The fifth print: %x\n", (int) charEnd[-1]);
}
```

The printed statements are:

a) The first print: **45**

b) The second print: **ef**

c) The third print: **cd**

d) The fourth print: **67**

e) The fifth print: **89**

Half credit: Solving the correct 8 bits, then extending those in some way.

*Ex: FFFFFFEEF EF000000*

Half credit: Writing correct answer OR wrong answer

(As in explicitly had two answers and used “or”)

Half credit: Writing correct answer in binary
Question 3: You won't mind this question one bit! (15 pts, 36 min)

We wish to implement a bit array, where we can read and write a particular bit. Normally for read/write array access, we would just use bracket notation (e.g., x=A[5]; A[5]=y;), but since a bit is smaller than the smallest datatype in C, we have to design our own `getBit()` and `setBit()` functions. We'll use the following typedefs to make our job easier:

```c
typedef uint8_t bit_t; // If it's a single bit, value is in least significant bit.
typed uint32_t index_t; // The index into a bit_t array to select which bit is used
```

E.g., imagine a 16-bit bit array: `Bit_t A[2]; A[1]=0x82; A[0]=0x1F;` Internally, A would look like this:

<table>
<thead>
<tr>
<th>Array A:</th>
<th>8</th>
<th>2</th>
<th>1</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

GetBit(A,0) would return 1, as would GetBit(A,1), GetBit(A,2), GetBit(A,3), and GetBit(A,4).
GetBit(A,5) would return 0, as would GetBit(A,6), GetBit(A,7), and GetBit(A,8). Etc.

a) How much space would the largest usable bit array take up? "Usable" means we could read and write every bit in the array. Express your answer in IEC format. E.g., 128 KIB, 32TIB, etc. 512 MiB

b) Write `setBit` in C. You may not need to use all the lines.

```c
void setBit(bit_t A[], index_t n, bit_t b) {
    // b is either 0 or 1
    A[n/8] = (A[n/8] & ~((b) << (n%8))); // The one liner. Or ...

    uint32_t byte_index = n/8; // (1) Find out which byte we want
    uint8_t bit_index = n%8; // (2) Find where within that byte is the bit

    A[byte_index] &= ~(1 << bit_index); // (3) Reset that bit in the byte

    A[byte_index] |= b << bit_index; // (4) Assign that bit in the byte
}
```

c) Write `GetBit(bit_t A[], index_t n)` in MAL; `sv0` should be 1 if the bit is on, and 0 if it's off. Hint: It might help if you start from the `srlv` and work backwards.

```c
GetBit:

| srlv | $t0, $a1, 3 | # $t0 = byte_index = n/8
| addu | $t1, $a0, $t0 | # $t1 = A + byte_index
| lbu  | $t2, 0($t1) | # $t2 = byte = *(A+byte_index) = A[byte_index]
| andi | $t3, $a1, 7 | # $t3 = bit_index = n%8
| srlv | $v0, $t2, $t3 | # $v0 = byte >> bit_index (slide bit to lsb slot)
| srlv | $v0, $t2, $t3 | # "srlv rd,rt,rs" means (in C): rd = rt >> rs
| andi | $v0, $v0, 1 | # $v0 &= 1 (mask out the lsb bit)
| jr   | $ra | # $v0 better be either a 0 or 1
```

4/6
Question 5: C->Assembly->MIPS (1 point per instruction fill-in, 10 points total)

Given below is a C code fragment from a binary search tree routine (don’t worry if you don’t know exactly what that is). It returns the array index of an integer that matches the given search element, and for simplicity we assume the element is always present. Convert the C code into the minimum number of MIPS instructions necessary to implement the C functionality. Answer by filling in the blanks with MIPS assembly language in the code template below. Note that the minimum code necessary may use fewer blanks than the number shown.

C:
int helperPBST (int *arr, int pos, int elem) {
    int temp = pos - 1;
    if (elem == *(arr + temp)) {
        return pos;
    } else if (elem > *(arr + temp)) {
        return helperPBST(arr, pos*2+1, elem);
    } else {
        return helperPBST(arr, pos*2, elem);
    }
}

MIPS:
# $a0 is arr
# $a1 is pos
# $a2 is elem
pbst:
addi $t0, $a1, -1
sll $t0, $t0, 2  #$t0 is being used as a pointer offset
add $t0, $t0, $a0  #calculate arr + temp
lw $t1, 0($t0)  #dereference (arr + temp)
beq $a2, $t1, success
slt $t3, $a2, $t1
beq $t3, $0, right
    sll $a1, $a1, 1  #pos = pos*2
    j pbst  #recurse. Use j instead of jal because #this is tail recursive
right:
    sll $a1, $a1, 1  #pos = pos*2
    addiu $a1, $a1, 1  #pos = pos + 1
    j pbst  #recurse.
success:
    addiu $v0, $a1, 0  #return pos
jr $ra
This problem was graded in blocks. After the first mistake in a block the remainder of a block was marked as incorrect. For example, many people failed to shift $t0$ to the left by 2, and so all of the first three blanks were marked as wrong. Similar logic was applied to blanks 4-6, 7-9, and 10.

There were two situations in which we showed leniency. If a student mixed up which part of the MIPS code corresponded to the else if and which corresponded to the else then 3 points were deducted, as opposed to 6. Additionally, if a student performed the pos*2 multiplication in the first sll slot, but managed to turn in a solution that would otherwise be correct they were only docked for the first three blanks.
A MATter of Performance (Su 11 Question 7)

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 AMAT?

\[
\text{AMAT} = \text{HT} + \text{MR} \times \text{MP} \\
= 1 + .1 \times 100 = 11 \text{ cycles}
\]

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

\[
\text{CPI} = \text{CPI}_{\text{ideal}} + (\text{mem access/instr}) \times (\text{MR} \times \text{MP}) \\
= 1 + 1.5 \times .1 \times 100 = 16 \text{ cycles}
\]

Disgusted at his slow computer, he requests 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?

\[
\text{AMAT} = \text{HT}_{L1} + \text{MR}_{L1} \times (\text{HT}_{L2} + \text{MR}_{L2} \times \text{MP}_{L2}) \\
6 = 1 + .1 \times (x + .35 \times 100) \\
6 = 1 \times x + 3.5 \\
x = 15
\]

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?

\[
\text{CPI} = \text{CPI}_{\text{ideal}} + (\text{mem access/instr}) \times (\text{MR}_{L1} \times (\text{HT}_{L2} + \text{MR}_{L2} \times \text{MP}_{L2})) \\
\text{But MR}_{L2} = \text{LOCAL miss rate, so MR}_{L2} = \text{GMR}_{L2} / \text{MR}_{L1} = .06 / .1 = .6 \\
= 1 + 1.5 \times (.1 \times (10 + .6 \times 100)) \\
= 1 + 1.5 \times 7 = 11.5
\]

OR

\[
\text{CPI} = \text{CPI}_{\text{ideal}} + (\text{mem access/instr}) \times (\text{MR}_{L1} \times \text{HT}_{L2} + \text{GMR}_{L2} \times \text{MP}_{L2}) \\
= 1 + 1.5 \times (.1 \times 10 + .06 \times 100) \\
= 1 + 1.5 \times 7 = 11.5
\]

e) What is the relative performance of his upgraded computer versus his old one? You may leave the value as a ratio.

\[
\text{Rel perf} = \frac{\text{CPI}_{\text{old}}}{\text{CPI}_{\text{new}}} = \frac{16}{11.5}
\]

Note order! \( \text{Insts}_{\text{new}} / \text{Insts}_{\text{old}} = (1 / \text{cycles/insts}_{\text{new}}) / (1 / \text{cycles/insts}_{\text{old}}) = (1 / \text{CPI}_{\text{new}}) / (1 / \text{CPI}_{\text{old}}) \)

If performance increase, rel performance is BIGGER
Question 2: Did somebody say “Free Lunch”?! .......................... (20 pts, 20 min)

Consider the following 10-bit floating-point format. It contains the same fields (sign, exponent, significand) and follows the same general rules as the 32-bit IEEE standard (denorms, biased exponent, non-numeric values, etc.). It simply allocates its bits differently. Please answer the following questions, and show all your work in the space provided. We went ahead and got you started.

Number represented by 0x00: 0

# Bits in the Mantissa: 6

a) Exponent Bias: 3

b) Implicit exponent for denormalized #’s: -2

c) # of Numbers between (2 ≤ n < 8): 128

d) Largest number x such that x + .5 = .5: $2^{-8} = \frac{1}{256}$
3. Caches

a. The Average Memory Access Time equation (AMAT) has three components: hit time, miss rate, and miss penalty. For each of the following cache optimizations, indicate which component of the AMAT equation may be improved. Circle one.

- Using a second-level cache
- Using smaller blocks
- Using larger blocks
- Using a smaller first-level cache
- Using a larger first-level cache

b. Given a direct-mapped cache, initially empty, and the following memory access pattern (all byte addresses and 32-bit word accesses, 32-bit addresses)

<table>
<thead>
<tr>
<th>8</th>
<th>0</th>
<th>4</th>
<th>32</th>
<th>36</th>
<th>8</th>
<th>0</th>
<th>4</th>
<th>16</th>
<th>0</th>
</tr>
</thead>
</table>

What is the hit rate, miss rate, and what blocks are in the cache after these accesses if

i. The cache has 8 32-bit blocks?

hit rate: 0.2 miss rate: 0.8

blocks at end (write the appropriate full byte addresses (NOT the tag) in the appropriate blocks, or "EMPTY" if the block is empty):

```
0  32  0
4  36  4
8
16
```

-1: minor mistake, easily seen
-2: fairly major mistake
-3: had no idea what was happening

Login: cs61c-__
The memory access pattern is repeated here for your convenience:

| 8 | 0 | 4 | 32 | 36 | 8 | 0 | 4 | 16 | 0 |

ii. the cache has 4 32-bit blocks?

hit rate: 0.1  miss rate: 0.9  

blocks at end (write the appropriate full byte addresses (NOT the tag) in the appropriate blocks, or "EMPTY" if the block is empty):

| 0 |
| 4 |
| 8 |
| EMPTY |

same as before

---

c. Consider a write-allocate, write-back, direct-mapped cache with 16 byte blocks and $64 \times 2^{10}$ bytes of data bits. Assume a byte-addressed machine with 32-bit addresses.

i. Partition the following address and label each field with its name and size in bits.

```
31 16 15 4 3
```

TAG - 16 b  INDEX - 12 b  OFFSET - 4 b

ii. Given the address DEADBEEF$_{hex}$, what is the value of the index, offset, and tag? (Write your answers in hexadecimal.)

index = 0x BEE
offset = 0x F

tag = 0x DEAD

1 pt, correctness based on above breakdown

---

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

```
TAG 16  VALID  2 pts
DIRTY 18
```

-1 per mistake

---

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

$2^{12}$ blocks  
18 management bits per block  
128 data bits per block  

$2^{12} (128 + 18)$ bits = 584 Kbits