1 True or False? (40 points.) This question occupies 3 pages.

1. On 32-bit machines, the largest memory size is 128 gigabytes.

True False

2. The ”word size” of a computer architecture (e.g. a 64-bit machine) determines the size of C pointer and integer datatypes.

True False

3. An important difference between a stack, accumulator, and general register types of computer architecture is the number of operands specified in each instruction (0, 1, 2 or 3)

True False

4. Standard computer organization separates the CPU and the memory, using byte-addresses to specify memory locations. These addresses must be aligned with the word size.

True False

5. To calculate a negative number, we start with positive number, flip every bit (i.e., take the one's complement), and then add one.

True False

6. After running the following code snippet on a big-endian machine, the short variable p will hold the value “0000”.

```
1  char x = 0x82;
2  unsigned y = (unsigned)x;
3  short p = *(&y);
```

True False

7. Executing the following two instructions sequentially (without anything happening in between) will not change the content in memory.

```
1  movq (%rax), %rdx
2  movq %rdx, (%rax)
```

True False

8. If after the following instructions **%rcx** holds the value of variable x, then **%rdx** holds the value of &x.

```
1       movq (%rdx), %rcx
```

True False

9. To implement "for" loops, one must use both conditional jumps (e.g., jl, jne, etc.) and unconditional jumps (e.g., jmp).

True False

10. Conditional jump instructions (jl, jne, etc.) use values in the general-purpose registers, in addition to the condition flags.

True False

11. By convention, right before a callee function returns (before executing ret instruction), the stack pointer must always point to the top of the caller frame.

True False

12. By convention, when a function returns, the returned value is always in %rax, the callee function must explicitly let the caller function know which register holds the returned value.

True False

13. By convention, if a caller function stores the value "237" in %rsi (a caller-saved register) right before entering a callee function, it can safely assume that when the callee returns, %rsi still holds the value "237".

True False

14. One consequence of the register calling convention is that if procedure A calls B calls C calls D, then the intermediate values in procedure A can only be stored in A's stack frame.

True False

15. If a C struct and a C union have the same set of member fields, the size of the struct (in bytes) must be greater than or equal to the size of the union.

True False

16. On a 64-bit machine, if we define the following struct, then after the struct is initialized in memory, the address of the short field is exactly 8 bytes away from the address of the long field.

```
1   struct S {
2     long a;
3     short b;
4   } *p
```

True False

17. Comparing a 8-way set associative cache to a 2-way set associative cache with the same capacity and block size, the 8-way cache generally results in less conflict misses, but it is more difficult to implement.

True False

18. Miss rate + hit rate is always 100% with respect to a single cache and a single program.

True False

19. Consider a program that accesses every 8-byte element of an integer array in order, and then it accesses every element of the same array in order again for a second time. The machine implements a fully-associative cache with a block size of 16 bytes. The miss rate of the program on this machine is 25%.

True False

20. In general, for caches, the write back policy generates more traffic to memory than the write through policy.

True False

2 (8 points.)

Assume that the length of `int` and `unsigned int` is 32 bits, signed integers use two's complement, and right/left shifts on signed integers are arithmetic. The following code generates arbitrary values for `int x` and `int y`, then converts them to `unsigned int`.

```
1 int x = random();      /* Create some arbitrary values */
2 int y = random();
3 unsigned int ux = x;   /* Convert to unsigned */
4 unsigned int uy = y;
```

Each part below is either a `C` expression of equality ("==") or a logical implication between two expressions ($A \Longrightarrow B$). A non-zero value is true in `C`; zero means false.

Answer whether the expression or implication is *always* true for all possible `x` and `y` (type "Yes"), or not (type "No"). **If "No", give specific values** for the variables involved to make it false. Specific values must be in hex (starting with "0x"), except that you may also use TMin_{32} and TMax_{32} for the smallest and largest values (respectively) that can be stored in an `int`.

A. `(x > 0) && (y < 0)` $\Longrightarrow$ `(x-y > 0)`

Yes No

B. `x > 4` $\Longrightarrow$ `(x>>1) == (int)(ux>>1)`

Yes No

C. x<0 $\Longrightarrow$ `x+0u <= 0u`

Yes No

D. `(int)(ux + y) == x + y`

Yes No

3 (16 points)

Consider this pseudocode compiled on a machine for which the sizes (in bytes) of `char`, `short`, and `int` are 1, 2, and 4, respectively.

```
1 int main()
2 {
3     char c[4] = {0x12,0x34,0x56,0x78};
4     char a1 = *((char *)(&c[0]));
5     short a2 = *((short *)(&c[2]));
6     short a3 = *((short *)(&c[1]));
7     int a4 = *((int *)(&c[0]));
8 }
```

A. What will be the following values on a big-endian machine (hexadecimal format, starting with 0x)?

a1: _____

a2: _____

a3: _____

a4: _____

B. What will be the following values on a little-endian machine (hexadecimal format, starting with 0x)?

a1: _____

a2: _____

a3: _____

a4: _____

4 (8 points.)

The following values are stored at the indicated memory addresses and registers:

Address	Value	Register	Value
0x410	0xAB	%rax	0x410
0x418	0x04	%rcx	0x420
0x420	0x09	%rdx	0x1
0x428	0x72	%rsi	0x2
0x430	0x54	%rdi	0x4

Assume that the values in the "Value" column occupy eight bytes of storage. For each part (A-D) of this question, answer the value stored in %r8, in hexadecimal format (starting with 0x):

Part A:

```
1 movq (%rax), %r8
2 addq 0x6(%rax, %rsi), %r8
```

Value stored in **%r8** after the **1st** line is _____

Value stored in **%r8** after the **2nd** line is _____

Part B:

```
1 movq 0x8(%rax), %r8
2 movq 0x6(%rcx, %rdx, 2), %r9
3 subq %r8, %r9
```

Value stored in **%r8** after the **1st** line is _____

Value stored in **%r9** after the **3rd** line is _____

Part C:

```
1 movq 0x428, %r8
2 leaq (%r8, %r8, 1), %r8
3 shrq %rdx, %r8
4 addq $0x428, %r8
```

Value stored in **%r8** after the **2nd** line is _____

Value stored in **%r8** after the **4th** line is _____

Part D:

```
1 movq 0x430, %r8
2 movq $0x430, %r9
3 notq %r8
4 notq %r9
5 xorq %r9, %r8
```

Value stored in **%r8** after the **5th** line is _____

5 (14 points) This question occupies 2 pages.

5.1 The assembly code below was compiled from a C function whose prototype is given below.

```
long funcP(long r, s, t);
```

```
1  _funcQ: #long funcQ (long x, long y)
2     addq   $2, %rdi
3    imulq   $3, %rdi
4     movq %rsi, %rax
5     addq %rdi, %rax
6     ret
7
8  _funcP: #long funcP (long r, long s, long t)
9     movq %rdi, %r8
10    subq %rdx, %r8
11
12   testq  %r8, %r8
13     jle foo
14
15   callq _funcQ
16     jmp bar
17
18  foo:
19    movq %rsi, %rax
20    movq %rdx, %rsi
21    movq %rax, %rdx
22   callq _funcQ
23
24  bar:
25     ret
```

Consider the assembly code above, assume that for parts A-D, the inputs are `r=2, s=3, t=5`**.**

A. How many times is funcQ called?

FuncQ is called _____ times.

B. What are argument values, x and y, to funcQ when funcQ is called for the first time?

x = _____, y = _____

C. What is the return value of funcQ when it is called for the first time?

Return value of funcQ is _____

D. What is the return value of funcP?

Return value of funcP is: _____

5.2 The assembly code below was compiled from a C function whose prototype is given below.

```
int funcP(long[][] x);
```

```
1  _funcP: #int funcP (long[][] x)
2    movq $0, %rax
3    movq $0, %r8
4
5  .loop:
6    movq %r8, %rdx
7   imulq $0x10, %rdx
8    movq (%rdi, %rdx), %r9
9    movq 0x8(%rdi, %rdx), %r10
10
11   cmpq %r10, %r9
12    jle foo
13   addq %r9, %rax
14    jmp bar
15
16 .foo:
17   addq %r10, %rax
18
19 .bar:
20   incq %r8
21
22   cmpq $2, %r8
23   jle loop
24   ret
```

E. Consider the assembly code above, what will be the returned value, if the input is `long[3][2]x = {{1,8}, {5,3}, {6,4}}`?

Return value is: _____

6 (14 Points)

Suppose we have a system with the following properties:

- The memory is byte addressable.
- Memory accesses are to **1-byte words** (not to 4-byte words).
- Addresses are 12 bits wide.
- The cache has four sets ($S = 4$). Each set consists of two lines (two-way set associative, $E = 2$). Each line holds four bytes of data ($B = 4$).
- LRU replacement is used for the cache.

The contents of the cache are as follows, with all addresses, tags, and values given in hex:

Set index	Valid	Tag	Byte 0	Byte 1	Byte 2	Byte 3
0	1	00	40	41	42	43
	1	83	FE	97	CC	D0
1	1	00	44	45	46	47
	0	83	–	–	–	–
2	1	00	48	49	4A	48
	0	40	–	–	–	–
3	1	FF	9A	C0	03	FF
	0	54	–	–	–	–

Part A. For this configuration, which address bits are used for the tag, set index, and block offset? Indicate your answer by writing a string with "t" to indicate tag bits, "s" to indicate the index bits, and "b" to indicate block offset bits. One possible answer is 'ttttssssbbbb' where the first 't' describes the most significant bit of the address, and the last 'b' describes the least significant bit.

Answer: _____

Part B. For each of the following memory reads which are **carried out in sequence** as listed, indicate two things:

1. Will it be a cache hit or miss?

2. If it is a hit, give the value of the corresponding memory location (in hex, starting with 0x). If it is a miss, write "unknown" in the "Value" column.

In total, there are 12 blanks to fill. The cache and memory are affected by nothing except the operations listed.

Address	Hit/Miss	Value
0x008		
0x831		
0x54E		
0x006		
0xAAD		
0xFFF		