1.
What are the differences between RISC and CISC? What are some of the advantages and disadvantages?

Reduced Instruction Set Computer (RISC) ISAs have fewer instructions overall. However, you end up using more instructions for a given task, as to replicate the same CISC instruction multiple single cycle instructions are required. Prioritizes efficiency in cycles per instruction.

Complex Instruction Set Computer (CISC) ISAs have more instructions overall. However, as your instructions are more complex, you end up using fewer instructions for a given task. The primary goal of CISC architecture is to complete a task in as few lines of assembly as possible. The emphasis is put on building complex instructions directly into the hardware.

RISC	CISC
Requires more instructions on average	Requires fewer instructions on average
Requires more working memory (RAM) to store the assembly level instructions	Requires less working memory (RAM), as the length of code is shorter
Fewer transistors are used for the instructions	More chip space required for the instructions
Compiler needs to perform more work to translate high-level language statement into assembly	Compiler has to do very little work to translate a high-level language statement into assembly
Executes one machine instruction per clock cycle	Takes multiple clock cycles per machine instruction

2.
Translate the following x86 instructions into MIPS:

a.
```
add 0x200(,%rdx,4),%rcx
```

Assuming $t0 corresponds to %rdx, and $t1 corresponds to %ecx,

```
add $t2, $t0, $t0
add $t2, $t2, $t2
addi $t2, $t2, 512
lw $t2, 0($t2)
add $t1, $t1, $t2
```

b.
```
lea 0xc(%rdi),%rax
```

Assuming $t0 corresponds to %rdi, and $t1 corresponds to %rax,

```
addi $t1, $t0, 12
```

c.
```
mov 0x30(%rsp,%rbx,4),%rax
```

Assuming $sp corresponds to %rsp, $t0 corresponds to %rbx, and $t1 corresponds to %rax,

```
add $t2, $t0, $t0
add $t2, $t2, $t2
add $t3, $sp, $t2
lw $t1, 48($t3)
```

loading = taking from memory and putting into register

d.
```
mov %rcx,-0x30(%rsp,%rdx,4)
```

Assuming $sp corresponds to %rsp, $t0 corresponds to %rdx, and $t1 corresponds to %rcx,

```
add $t2, $t0, $t0
add $t2, $t2, $t2
add $t3, $t2, $sp
sw $t1, -48($t3)
```

storing = taking from register and putting it in a register

3.
Translate the x86 code into MIPS. Assume variables a,b, and i are in register $s0, $s1, and $t0. Assume a, b, and i are in rdi, rsi, and rdx.

```
for(i = 0; i < 5; i++) {
        a+=b;
}
```

```
            mov $0, rdx
.loop:      cmp $4, rdx
            jg leaveloop
            add rsi, rdi
            add $1, rdx
            jmp .loop
```

```
            li $t0, 0
.loop:      slti $t2, $t0, 5
            beq $t2, 0, leaveloop
            add $s0, $s0, $s1
            addi $t0 $t0, 1
            j .loop
```

4.
What does the following MIPS code snippet do?

```
Loop:    lw $t0, 0($s0)
         lw $t1, 0($t0)
         add $t1, $s1, $t1
         sw $t1, 0($t0)
         addi $s0, $s0, 4
         bne $s0, $s2, Loop
```

The $s0 register appears to represent a pointer to a pointer, that is incremented by 4 each loop. This suggests that the $s0 represents an array of pointers.

The code snippet iterates through the array, incrementing each item pointed to by each pointer by $s1. The code snippet stops iterating once $s0 points to the same index as $s2.

5.
When does False Sharing occur, and how does it affect performance when parallelizing?

False sharing occurs during parallelization when the cache is “ping-pong”ed between different threads, due to separate threads modifying independent variables sharing the same cache line. This negatively affects performance, due to the overhead incurred when ping-pong-ing cache lines back and forth.

The ping-pong-ing occurs because a different thread writing to the same cache line would result in inconsistency between the two threads regarding the exact same cache line, even though different variables were accessed. The cache line is thus invalidated, to maintain cache coherency. This circumstance is called false sharing because each thread is not actually sharing access to the same variable

To avoid this, we must ensure that no two threads write to the cache line.

Further reading:
https://software.intel.com/en-us/articles/avoiding-and-identifying-false-sharing-among-threads