Question 1

Consider this C function:

```
void sumitup() {
 int i, j, k;
 for (i=0; i<DIM; i++) {
  for (j=0; j<DIM; j++) {
   summation+=bigarray[i][j];
   bigarray[i][j]++;
  }
 }
}
```

Below are four parallel implementations of the above function. One of these does **NOT** work as intended - the other three all work correctly (but of course differ in terms of efficiency). Which one of the three is **NOT** a correct parallelization of the above function?

Select one:

a.
```
void sumitup_par4() {
 int i, j, k;

 for (i=0; i<DIM; i++) {
#pragma omp parallel for reduction(+:summation) num_threads(4)
  for (j=0; j<DIM; j++) {
   summation+=bigarray[i][j];
   bigarray[i][j]++;
  }
 }
}
```

b.
```
void sumitup_par3() {
 int i, j, k;

#pragma omp parallel num_threads(4)
  for (i=0; i<DIM; i++) {
#pragma omp for reduction(+:summation)
   for (j=0; j<DIM; j++) {
    summation+=bigarray[i][j];
    bigarray[i][j]++;
   }
```

```
  }
}
```

○ c.

```
void sumitup_par() {
 int i, j, k;
 int tid;

 #pragma omp parallel num_threads(4) private(i,j,tid) reduction(+:summation)
 {
  tid = omp_get_thread_num();
  for (i=0; i<DIM; i++) {
   for (j=0; j<DIM; j++) {
    if ((j&3)==tid)
     {
      summation+=bigarray[i][j];
      bigarray[i][j]++;
     }
   }
  }
 }
}
```

○ d.

```
void sumitup_par2() {
 int i, j, k;

#pragma omp parallel for num_threads(4) reduction(+:summation) private(j)
  for (i=0; i<DIM; i++) {
   for (j=0; j<DIM; j++) {
    summation+=bigarray[i][j];
    bigarray[i][j]++;
   }
  }
}
```

Question 2

The following code is missing the condition inside the while clause of the do/while loop (see the /*MISSING*/ comment):

```
int main( int argc, const char* argv[] ) {
  int x;
  do {
    x+=rand();
  } while (/*MISSING*/);
    printf("x=%d\n",x);
}
```

Here is the compiled assembly for the code:

```
0000000000400510 <main>:
  400510: 53                  push   %rbx
  400511: 0f 1f 80 00 00 00 00 nopl   0x0(%rax)
  400518: e8 f3 fe ff ff       callq  400410 <rand@plt>
  40051d: 01 c3               add    %eax,%ebx
  40051f: 89 d8               mov    %ebx,%eax
  400521: 83 e0 03            and    $0x3,%eax
  400524: 83 f8 03            cmp    $0x3,%eax
  400527: 74 ef               je     400518 <main+0x8>
  400529: 89 de               mov    %ebx,%esi
  40052b: bf 38 06 40 00       mov    $0x400638,%edi
  400530: 31 c0               xor    %eax,%eax
  400532: 5b                  pop    %rbx
  400533: e9 b8 fe ff ff       jmpq   4003f0 <printf@plt>
```

Reverse engineer the assembly and choose the correct condition that would have been in the /*MISSING*/ part of the C code based on the assembly.

Select one:

- a. (x+3)>2
- b. (x&3)>1
- c. (x&3)>2
- d. (x>>1)>2

e. (x>>1)>1

f. (x+3)>1

Question 3

Consider the following MIPS code:

```
lw $t0, 0x0 ($gp)
addi $t1, $gp, 0x10
xor $t2, $t2, $t2
here:
sltu $t3, $t2, $t0
beq $t3, $zero, there
sll $t4, $t2, 2
addu $t4, $t4, $t1
sw $t2, -4 ($t4)
addi $t2, $t2, 1
j here
there:
lw $t9, 0x10 ($gp)
```

At the start of the code segment above, register $gp points to address 0x401200. This program is run on a big endian machine. The memory dump below is also taken at the start of the code segment above:

0x401200:	0x00	0x00	0x00	0x04	0x10	0x0a	0x00	0x00
0x401208:	0xaa	0xbb	0xaa	0xbb	0xcc	0xdd	0xcc	0xdd
0x401210:	0xaa	0xbb	0xaa	0xbb	0xcc	0xdd	0xcc	0xdd
0x401218:	0xaa	0xbb	0xaa	0xbb	0xcc	0xdd	0xcc	0xdd
0x401220:	0xaa	0xbb	0xaa	0xbb	0xcc	0xdd	0xcc	0xdd
0x401228:	0xaa	0xbb	0xaa	0xbb	0xcc	0xdd	0xcc	0xdd
0x401230:	0xaa	0xbb	0xaa	0xbb	0xcc	0xdd	0xcc	0xdd
0x401238:	0xaa	0xbb	0xaa	0xbb	0xcc	0xdd	0xcc	0xdd

Using the information above, what is the value of $t0 after the execution of the first lw instruction?

ANSWER:

Using the information above, what is the value of $t9 after the execution of the last lw instruction?

ANSWER: