1.)
a.)

```
cnt $t0, $t1
# The first operand is rt, the second operand is rd
# if (M[R[rt]](7:0) >= 0)
# R[rt] = R[rt]+1, R[rd] = R[rd] + 1, PC=PC
```

R format, it is only taking values of registers, no immediates or jump labels
(This operation wouldn't work as RISC load store arch doesn't have way to connect memory to ALU directly and wants to keep the system simple so wouldn't want to anyway.)

```
lb $at, 0($t0)              # load byte of rt
slt $at, $at, $zero         # is byte less than zero  $at = 1 if at < 0,
bne $at, $zero, CNT_EXIT    # exit if at is not 0 so byte was less than 0

addiu $t0, $t0, 1
addiu $t1, $t1, 1           # PC = PC so nothing needed to be done
CNT_EXIT:
```

b.)

Assembly	Format	Memory Location	Decimal For each field R: op, rs, rt, rd, shamt, funct I: op, rs, rt, immd J: op, addr
Begin:			
addi $t5, $zero, 1195	I	0x00400000	8, 0, 13, 1195
j Cond	J	0x00400004	2, 1048582 (0x0100006)
Loop:			
sra $t5, $t5, 1	R	0x00400008	0, 0, 13, 13, 1, 3
andi $t6, $t5, 0x003c	I	0x0040000c	12, 13, 14, 60 (0x3c)
add $t9, $a0, $t6	R	0x00400010	0, 4, 14, 25, 0, 32
sw $t5, 4($t9)	I	0x00400014	43, 25, 13, 4
Cond:			
bne $t5, $zero, loop	I	0x00400018	5, 13, 0, -5
jr $s7	R	0x0040001c	0, 23, 0, 0, 0, 8

Hex Encoding:

001000 00000 01101 0000 0100 1010 1011 >> **0x200d 04ab**
000010 00 0001 0000 0000 0000 0000 0110 >> **0x0810 0006**

000000 00000 01101 01101 00001 000011 >> **0x000d 6843**
001100 01101 01110 0000 0000 0011 1100 >> **0x31ae 003c**
000000 00100 01110 11001 00000 100000 >> **0x008e c820**
101011 11001 01101 0000 0000 0000 0100 >> **0xaf2d 0004**

000101 01101 00000 -5 (FFFB) >> **0x15a0 fffb**

000000 10111 00000 00000 00000 001000 >> **0x02e0 0008**

1.)
c.)

Assembly	Fmt	Decimal for each field R: op, rs, rt, rd, shamt, funct I: op, rs, rt, immd J: op, addr	Hex Encoding
lui $t0, 0xFEED addiu $t0, $t0, 0x3210	I I	48, 0, 8, 65261 (0xFEED) 9, 8, 8, 12816 (0x3210)	**0xc008 FEED** **0x2508 3210**

addiu $t0, $zero, 0xFEED
sll $t0, $t0, 16
addiu $t0, $t0, 0x3210
	I
	R
	I
		9, 0, 8, 65261 (0xFEED)
		0, 0, 8, 8, 16, 0
		9, 8, 8, 12816 (0x3210)
			0x2408 FEED
			0x0008 4400
			0x2508 3210

2.)
a.) Code submitted
b.) Code submitted
c.)
In my first program for the dynamic parts, ie the FIB calculation loop, the program executes **5+6*N instructions**. I'm not counting the instructions it takes to print or get the variables.
In my code it is everything between the FIB and PRINT labels.

In the second program, each time Fib is call if it less than 2 it executes 4 instructions. If it goes through the entire sequence of adding fib(n-1) and fib(n-2) then it calls 31 instructions w/o the recursion.

This is the data a got for as N grows like 0, 1, 2, 3, 4, 5

Number of calls to fib:
1, 1, 3, 5, 9, 15

grows as a(n) = a(n-1) + a(n-2) +1

End calls to fib: (These have 4 instructions)
1, 1, 2, 3, 5, 8

grows as **b(n)** = b(n-1) + b(n-2)

Remaining Intermediate calls: (These have 31 instructions)
0, 0, 1, 2, 4, 7

grows as **c(n)** = c(n-1) + c(n-2) +1

so we go through the multiple sequences to find out how long this program takes.

31 * c (N) + 4 * b (N) instructions

Probably should use the linear one