1.)

a.) RISC vs CISC

i.) CISC
ii.) CISC
iii.) RISC
iv.) RISC
v.) CISC
vi.) RISC

b.) Impacts either ISA, ABI, and/or Micro-Architecture

i.) **ISA**, the ISA would need to modify the instruction length depending on the number of registers that there are to be used
ii.) **Micro-Arch**, the Micro-arch defines subcomponents of the processor (datapath) so the number of cycles an instruction can take impacts how the subcomponents for instructions can be designed to fit that time constraint
iii.) **ISA and Micro-Arch**, the ISA would need to define additional instructions for immediates (like addi) or ways to handle an immediate to be translated into binary for the processor. As well, the micro-arch will need to define a way to operate and read the immediate from the instruction. (Additional, datapath for immediates in binary instructions, which could be handled differently)
iv.) **ISA**, the ISA defines what named registers are physically and would be impacted by which one the SP must be in its design.
v.) **ISA**, the ISA define which registers are where and what they are for, and this would be impacted by the specifications of constant registers in the design
vi.) **ABI**, the ABI is impacted as when a program is compiled it needs to know where to move values to make the function calls
vii.) **ISA**, the ISA will need to know what values the temp/saved registers are in order to translate to in binary to be handled by the processor
viii.) **ISA**, the ISA will need to define the size of registers to hold memory addresses depending on the range of addresses
ix.) **Micro-Arch**, the Micro-Arch defines the subcomponents of the processor so the type of adder in the processor impacts/changes the Micro-Arch's design
x.) **ISA**, the ISA defines what instructions do so which instructions change the PC would change the ISAs design
xi.) **ISA**, the ISA is impacted as it defines how the processor will read binary instructions so which bits are opcodes/operands changes such definition
xii.) **Micro-Arch**, the Micro-Arch is impacted as it defines subcomponents in the processor and would need to implement the number of functional units

2.)
a.) Nintendo 64, LTE Modems, Smart Televeisions (Non-Android)

b.)

```
subi dst, src, 0
or dst, src, $zero
multi dst, src, 1
```

c.) The program calculates both $1 xor $2 xor $2 and stores it in $2 and calculates ($1 xor $2) xor ($1xor $2 xor $2) and stores it in $1. ($1 and $2 are their initial values.)

d.)

```
srl $t0, $s2, 3        # $t0 = c / 8
andi $t1, $s3, 3       # $t1 = d % 4
sll $t2, $s1, 2        # $t2 = b * 2

add $t2, $t2, $s1      # $t2 = b * 3
add $s0, $t2, $t1      # a = b * 3 + d % 4
add $s0, $s0, $t0      # a = b*3 + d % 4 + c / 8
```

3.)
a.) MARS simulates the ISA and Micro-Architechure as it creates an artifical system of the registers and memory addresses to be used and converts the assembly code into executable instructions that it interprets/processes. The system also has to create the datapath like how the system preforms arithmetic and how the system connects these together with registers/memory.

b.)

Inputs	Output
0, 0	25
1, 0	57
1, 1	33

The program takes in two integers. The first one is mulitplied by 32 and the second is used as the index of the vals array loading in the int at that index. These two numbers are then added together and printed out. (32 * in1 + vals[in2])

c.)

Inputs	Ouputs (in vals array)
0, 0, 0	25 still in index 0
0, 1, 0	25 changes to 13 in index 0
2, 6, 9	12 changes to 52 in index 9

Correctness:
I tested for correctness by doing the math of the expected internal operations and checked the vals array updating in memory using MARS' data segment view. For instance, the 0, 1, 0 outputs 13. I checked this by calculating the average of vals[0] and vals[1], which are 25 and 1 respectively. The average is 13 (26 / 2) and is put into vals[0]. This is expected operation.