1.

a. **Benefits**:

i. Primitive instructions are easier to implement and give more control over the program to make it run more efficiently (Direct memory access).

ii. Reduces program maintainability.

Drawbacks:

iii. Primitive instructions are more difficult to read and comprehend than high-level languages.

b.

i. Impacts ISA as it defines the registers that are visible to the software.

ii. Impacts microarchitecture as it relates to the implementation of the registers.

iii. Impacts ISA as it defines instruction format.

iv. Impacts ABI as it defines the conventions needed for register usage.

v. Impacts ISA as it defines the architectural behavior of some registers.

vi. Impacts ABI as it defines specific calling conventions.

vii. Impacts ABI as it defines the conventions needed for register usage.

viii. Impacts ISA as it establishes the memory address space.

ix. Impacts microarchitecture as it concerns implementation-specific hardware design.

x. Impacts ISA as it defines instruction behavior and control flow.

xi. Impacts ISA because the binary code of an instruction is pre-set.

xii. Impacts microarchitecture because it is based on how the chip is built.

2.

a. **Broadcom** has a Half-length PCI evaluation board called BCM91125F which is designed for software development and developing PCI and Ethernet applications that require high-performance processors.

Nintendo 64 used the MIPS R4300i CPU and was the first major home console to use a 64-bit processor.

Velocity's Cruz Tablet from Ingenic was one of the first MIPS-based tablets.

b. $s0 is the register we are attempting to clear:

andi $s0, $s0, 0

or $s0, $zero, $zero

sub $s0, $s0, $s0

c. The first instruction performs XOR between $1 and $2 and stores the result in $1. The second instruction performs XOR between the new value in $1 and the unchanged $2 value from the first instruction and stores it in $2. The last instruction performs XOR between the values in $1 ($1 XOR $2) and $2 (first value of $1) and stores the result in $1. This indirectly swaps the values in $1 and $2 without using temporary registers.

d.
```
srl $t0, $s1, 4          # b / 16
and  $t1, $t0, $s2       # ( b / 16 ) & c
andi $t2, $s3, 3         #  d % 4
sub $s0, $t1, $t2        # ( ( b / 16 ) & c ) – ( d % 4 )
```

3.

a. MARS simulates the ABI and ISA but not the microarchitecture of MIPS. The assembly instructions defined by the MIPS ISA, which include arithmetic, memory access, control flow, and syscalls, are simulated by MARS. MARS also simulates certain aspects of the MIPS ABI, such as register usage conventions and management of the stack.

b.

i. Inputs: 17, 100 Output: 544

ii. Inputs: 9, 0 Output: 313

iii. Inputs: 100000, 911 Output: 3200000

The assembly code uses the array of values labeled as ‘vals’ to perform some calculations. The first input is added to 32 and multiplied by the value at the second number’s array index labeled as ‘vals.’

C Code:

```
int vals [ ] = { 25 1 4 10 381 42 100 60 0 12 25 };
int input1, input2, output;
printf(“Please enter a number: \n”);
scanf(“%d”, &input1);
```

```
printf(“Please enter a number: \n”);
scanf(“%d”, &input2);
output = (input1 + 32) * vals [input2];
printf(“%d”, output);
```

c.

i. x = 0, y = 0, pos = 1, **output**: 25 at index 1

ii. x = 6, y = 1, pos = 0, **output**: 1 at index 0

iii. x = 0, y = 8, pos = 8, **output**: 0 at index 8