David Hatcher

74842093

Homework 3

Short Answer:

1. Main memory is slower to access than CPU Registers. In RISC-V to get data from or place data into main memory you have to run load/store operations which requires extra instructions to complete while with registers you can access them directly, less instructions.
2. A load/store architecture has two categories of instructions, memory access and operations that can be used between CPU registers. Meaning that if you want to add two numbers together and store it in memory you must first load them into the registers then add them together then store into a memory location. This relates to RISC-V as it is a load/store architecture so it must follow the same rules.
3. Single byte: 8-bits, Halfword: 16-bits, Word: 32-bits, Doubleword: 64-bits
4. Endianness is a way to describe which byte is the most significant. It is important because it tells the process which byte to start reading from first. RISC-V uses Little Endian
5. Byte addressability allows you to access memory locations by the specific byte, while word addressability allows you to access memory locations by the word location, so you would be accessing 4 bytes at a time. Word alignment and byte alignment are different ways to represent addresses or instructions, word alignment says that these will be in 32-bit words while byte alignment is similar but uses byte to represent the length of the addresses or instructions. RISC-V uses byte alignment.

Assembly Programming 1

1. Swap rs, rt
    a. Available register that may be destroyed
        i. sd t0, rs
        ii. sd rs, rt
        iii. sd rt, t0
    b. No available register for temporary storage
        i. sub rs, rs, rt
        ii. add rt, rt, rs
        iii. sub rs, rt, rs
2.

	add t0, zero, zero	Adding 0 and 0 and assigning it to t0
loop:	beq a1, zero, finish	If a1 and zero are equal branch to finish
	add t0, t0, a0	add t0 and a0 and assign it to t0
	addi a1, a1, -1	add a1 and -1 and assign to a1
	j loop	jump to loop
finish:	addi t0, t0, 42	add t0 and 42 and assign it to t0
	add a0, t0, zero	add t0 to 0 and assign it to a0

    a. It returns (a0*a1) + 42
    b. Shortened

i. mul a0, a0, a1
ii. addi a0, a0, 42