1)

a) 0xAD490020 = 10*(16^7) + 13*(16^6) + 4*(16^5) + 9*(16^4) + 2*(16^1)

Answer: 2,907,242,528

b) 0xAD490020 = 1010 1101 0100 1001 0000 0000 0010 0000

2's Comp = 0101 0010 1011 0110 1111 1111 1101 1111

+1

= 0101 0010 1011 0110 1111 1111 1110 0000

Answer: -1387724768

c) MIPS command: 1010 1101 0100 1001 0000 0000 0010 0000

=> Bits 31-26: opcode = 101011 (which is the opcode for sw),

=> Bits 25-21: base = 01010 (code for $t2)

=> Bits 20-16: rt = 01001 (code for $t1)

=> Bits 15-0: offset = 10000 (32 bit offset)

Answer: sw $t1, 32($t2)

2)

a) Blocks to be used for this instructions:

Registers, ALU, Data Memory, and the Instruction Memory (Note: mux is also used but unsure if it must be stated)

b) New functional blocks needed:

a mux will be needed proceeding the ALU operation

c) New signals needed:

a control signal will be needed to operate the additional mux from 4.2.2

3)

a) **200ps** => since clock cycle is only dependent on I-Mem latency, which is the critical path given that the processor just

fetches.

b) Following the data path, we go through the I-Mem, Add, Mux, Sign-Extend, and Shift-Left-2 (not in order).

so: 200ps+70ps+20ps+15ps+10ps = **315ps**

c) This time, we pass I-Mem, Regs, Mux, ALU, and another Mux.

so: 200ps+90ps+20ps+90ps+20ps = **420ps**

d) **PC-relative branches and loads/stores** use this resource, since Shift-Left-2 is used to calculate offsets.

e) **Jump instructions** are on this critical path since they depend on the condition of the jump.

f) 420ps (computed from part c) is a critical path that utilizes Shift-Left-2 and is reflective of a beq instruction. When looking
at an add instruction, the cycle time would follow:
I-Mem, Regs, Mux, ALU, Mux, Regs
(200ps+90ps+20ps+90ps+20ps+90ps = 510ps)
=> Since the cycle time of an add instruction is greater than a beq, changes in the given latency will not change any
cycle time of the processor UNTIL the latency of Shift-Left-2 reaches more than 100ps.

4)

a) The results of the sign-extension would be: **00000000000000000000000000010100**
After executing the jump Shift-Left-2, the result is:
00011000100000000000000001010000

b) The opcode 101011 is sw. The values of ALU control unit's inputs for this instruction is **010100** from bits 5-0, and ALUOp is **00**.

c) **PC+4**. PC's value is moved into Add, where it adds 4 to PC. This value is given to the branch and jump mux-es which pass it.
The value is given to PC after this.

d) Since the opcode is 101011 (sw,) **the branch/jump mux = PC+4** (since no branch has to be made by sw.)
The write Mux input could be bits 20-16 OR bits 15-11. So for the write register mux, it is either **00010** or **00000**.
The ALU mux then selects the Sign-Extension of bits 15-0 (which is the offset) and the ALU then adds this value to the value of the register defined by bits 25-21, which would be **0…010100** (which is **20** in decimal, same as sign-extension for part a above.)

e) ALU = **-3 and 20**, Add(PC+4) = **PC and 4**, Add(branch) = **PC+4** and **80**

f) WriteReg = X (Don't care; it will be 0010 or 0000)
WriteData = X (Don't care)
ReadReg1 = bits 25-21 = **00011**
ReadReg2 = bits 20-16 = **00010**
RegWrite = **0**