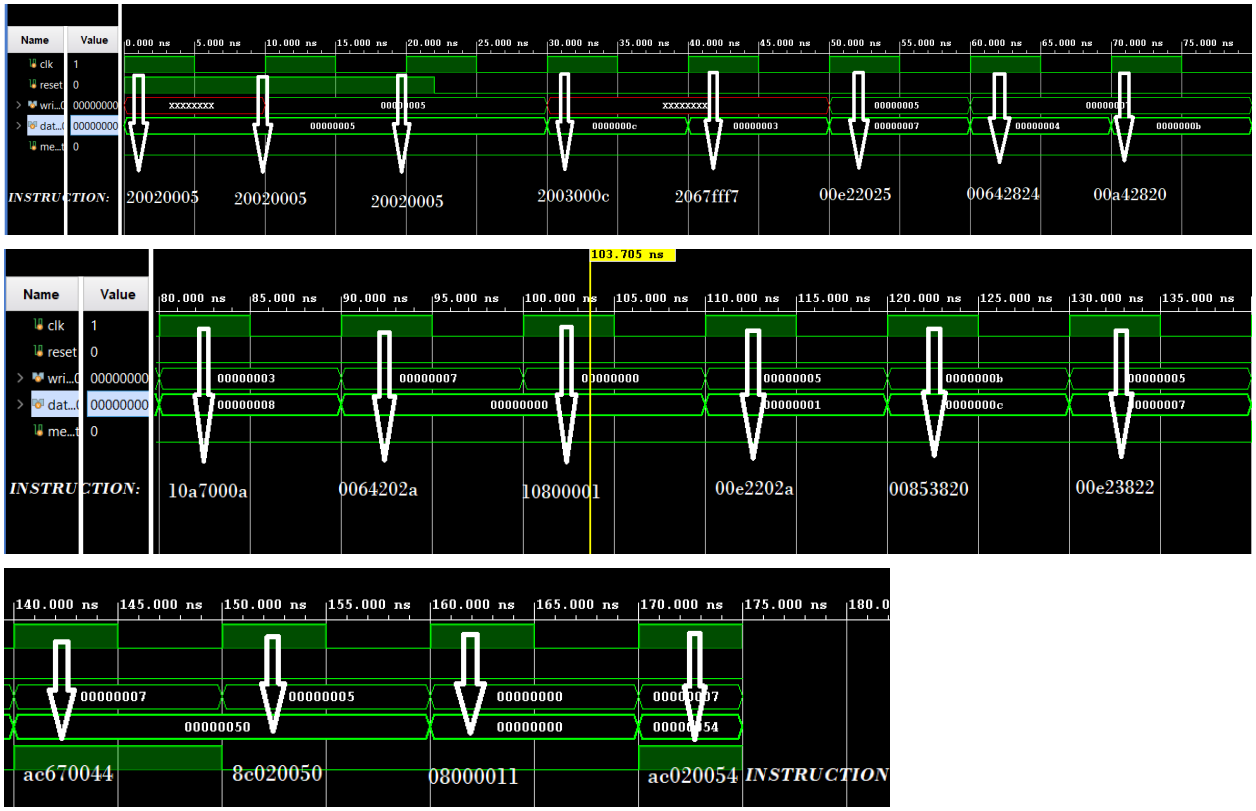
1.

Using this images we can see which instruction (in machine code) is being executed per clock cycle. Further explanation is provided below on some unique portions of the graph.



It is important to note that for instruction 1, it took 3 clock cycles to run since it had reset set to 1 so it cannot move to the next instruction. Hence only on the 4th clock cycle will the 2nd instruction come through. On the 9th clock cycle where a beq was not taken the output was 0x8 since beq requires the subtraction of the 2 register values. It would only branch out if their difference is 0. On the 15th clock cycle we now have a store word and since we want to store into memory we activate the memwrite. Then we disable it on the next cycle since we only want to load what was on the memory address. On the last clock cycle we set to 1 memwrite since we want to write 7 to memory address 84. Moreover, based on the testbench code if we reached the desired output, we must stop the program.

2.

A.

```
addi $0, $0, 0
addi $1, $0, 1
addi $2, $0, 2
addi $3, $0, 3
addi $4, $0, 4
addi $5, $0, 5
addi $6, $0, 6
```

addi $7, $0, 7
addi $8, $0, 8
addi $9, $0, 9
addi $10, $0, 10
addi $11, $0, 11
addi $12, $0, 12
addi $13, $0, 13
addi $14, $0, 14
addi $15, $0, 15
addi $16, $0, 16
addi $17, $0, 17
addi $18, $0, 18
addi $19, $0, 19
addi $20, $0, 20
addi $21, $0, 21
addi $22, $0, 22
addi $23, $0, 23
addi $24, $0, 24
addi $25, $0, 25
addi $26, $0, 26
addi $27, $0, 27
addi $28, $0, 28
addi $29, $0, 29
addi $30, $0, 30
addi $31, $0, 31

B. add $1, $2, $8

C. sub $2, $10, $1

D. sw $1, 12($0)

- since we want to store a register value we can use store word with a specific offset instead of increasing the value of the rs register

E. sw $2, 16($0)

- since we want to store a register value we can use store word with a specific offset instead of increasing the value of the rs register

3.

20000000
20010001
20020002
20030003
20040004
20050005
20060006
20070007
20080008

20090009
200a000a
200b000b
200c000c
200d000d
200e000e
200f000f
20100010
20110011
20120012
20130013
20140014
20150015
20160016
20170017
20180018
20190019
201a001a
201b001b
201c001c
201d001d
201e001e
201f001f
00480820
002a1022
ac01000c
ac020010

4.