With the changes in lab 5 it takes 4 cycles for a new address to come through, so if you changed a register in the last cycle it will be updated by the next cycle for use

Cycle: operation

1: load 0 into register 5

2: load 2 into register 6

3: load 2 into register 7

4: load 2 into register 8

5: load 2 into register 9

6: load 2 into register 18

7: load 2 into register 19

8: load 2 into register 20

9: load 2 into register 21

10: load 2 into register 22

11: load 2 into register 23

12: load 2 into register 24

13: load 2 into register 25

14: load 2 into register 26

15: load 2 into register 27

16: load 2 into register 28

17: load 2 into register 29

18: load 2 into register 30

19: load 2 into register 31

20: add register 24 with immediate store in register8

10+ 10101 = 10111

21: logical left shift register 5 by immediate store in register31

Shift 10 by 11101 = 1000000000000000000000000000000

22: xor register 20 with immediate store in register20

00010

11011 =

11001

23: shift logical right register 18 by immediate store in register5

10 by 110 = 0

24: shift arithmetic right register24 by immediate store in register19

10 by 10000001000 = 0

25: or register20 with immediate store in register26

11001

11100 =

11101

26: branch if register7 equal to register31

Sub 10 and 000000000000000000000000000000001000000000000000000000000000000

1111111111111111111111111111111111000000000000000000000000000010

Not equal NO BRANCH

27: Brach if register21 equal to register28

Sub 10 and 10 = 0

Equal BRANCH -> jump 2 addresses

28: and register30 with immediate store in register21

00010

11001 =

00000

28: add register 22 with register 20 store in register24

10 +11001 = 11011

29: subtract register25 with register 8 store in register21

10 – 10111 = 1111111111111111111111111111111111111111111111111111111111101011

30: logical left shift register 9 by register 28 store in register18

10 by 10 = 1000

31. Branch if register18 not equal to register9

Sub 1000 and 10 = 110

With carry

Not equal BRANCH -> jump 4 addresses

33: branch if register5 not equal to register6

Sub 0 and 10 1111111111111111111111111111111111111111111111111111111111111110

34: xor register 7 with register7 store in register8

10

10 =

0

35: shift logical right register25 by register5 store in register21

10 shift by 0 = 10

32: shift arithmetic right register22 by register30 store in register19

10 by 10 = 0

33: or register20 with register6 store in register5

11001

10 =

11011

34: and register30 with register29 store in register21

10

10 =

10

35: branch if register18 is less than register9

1000 sub 10 = 0110

Carry

Not less than NO BRANCH

36: branch if register 7 is less than register31

Sub 10 and 000000000000000000000000000000001000000000000000000000000000000

1111111111111111111111111111111111000000000000000000000000000010

No carry

Less than BRANCH -> jump 6 addresses

39: Store what is in register23 in data\_mem address 6

Store 10

40: store what is in register9 in data\_mem address7

Store 10

43: store what is in register 25 in data\_ mem address 30

Store 10

44: load from data\_mem address6 to register8

Load 10

45: load from data\_mem address7 to register24

Load 111

37: load from data\_mem addres30 to register31

Load 11110

STARTING THE C CODE

38. add register0 with immediate store in register5

01

39. add register0 with immediate store in register6

11

40. add register0 with register0 store in register7

0

41. add register0with register6 store in register8

11

42. branch if register 8 equal to register 7

11 – 0 = 11

NO BRANCH

43. branch if register 8 less than register 7

11- 0 = 11

NO BRANCH carry = 1

44. add immediate to register7 store in 7

0 + 1 = 1

45. branch if register 6 not equal to register 5

10-11 = 1111111111111111111111111111111111111111111111111111111111111111

BRANCH TO IF offset = 100

46. add immediate to register 6, store in register6

11-01 (+1111111111111111111111111111111111111111111111111111111111111111)= 10

47.branch if register 7 is less than register8

1 – 11 = 1111111111111111111111111111111111111111111111111111111111111110

No carry

BRANCH TO BEGINNING OF LOOP at (address) offset = 1010 (-6)

48. add register 7 and immediate, store in register 7

01+01 = 10

49. branch if register 6 not equal to register 5

10-10 = 0

NO BRANCH

50. add register5 with immediate store in 5

10 – 1 (+1111111111111111111111111111111111111111111111111111111111111111)= 1

51.branch if register 7 is less than register8

10 – 11 = 1111111111111111111111111111111111111111111111111111111111111111

No carry

BRANCH TO BEGINNING OF LOOP at (address) offset = 100 (-4)

52. branch if register 8 less than register 7

11- 10 = 1

NO BRANCH carry = 1

53. add register 7 and immediate, store in register 7

10+1 = 11

54. branch if register 6 not equal to register 5

1-10 = 1111111111111111111111111111111111111111111111111111111111111111

BRANCH TO IF offset = 100

55. add immediate to register 6, store in register6

10 – 1 (+1111111111111111111111111111111111111111111111111111111111111111) = 1

56.branch if register 7 is less than register8

11-11 = 0

NO BRANCH

57.ranch if not equal register 0 register 0

0 – 0 = 0

NO BRANCH

CONTINUE TO OTHER INSTURCTIONS IN THE MIF

**THE CODE**

C:

int a = 2; int b = 3;

for(int i = 0; i < 3; i++){

if(a ==b)

a--;

else

b--;

}

Assembly:

# use temp reg x5 and x6, x7, x8 to store a, b and k as well as the number 3 for comparison.

#First store what is in x5 and x6, x7, x8 into the stack to be resorted after

# load 2 and 3 into the temp registers by adding x0 and immediate

addi x5,x0, 2

addi x6,x0,3

#set i to 0 by adding x0 to itself

add x7, x0,x0

# set x8 to hold 3 for loop comparison, use b

add x8, x0,x6

#check if equal, go to the always exit to keep even

beq x7,x8 exit (forward 10)

#check if greater, this is check in non if branch to keep even branches

blt x8, x7, exit (forward 8)

loop:

#increment i

addi x7, x7, 1

#if a = b

bne x6,x5, else (forward 4)

#since we have no subi use addi -1

addi x5,x5,-1

#if under three loop again, to keep even check always check this again after branch

blt x7, x8, loop (back 4)

#branch out of the loop with always true since passed exit condition

beq x0,x0 exit (forward 4)

else:

#since we have no subi add -1

addi x6, x6, -1

#if under three loop again

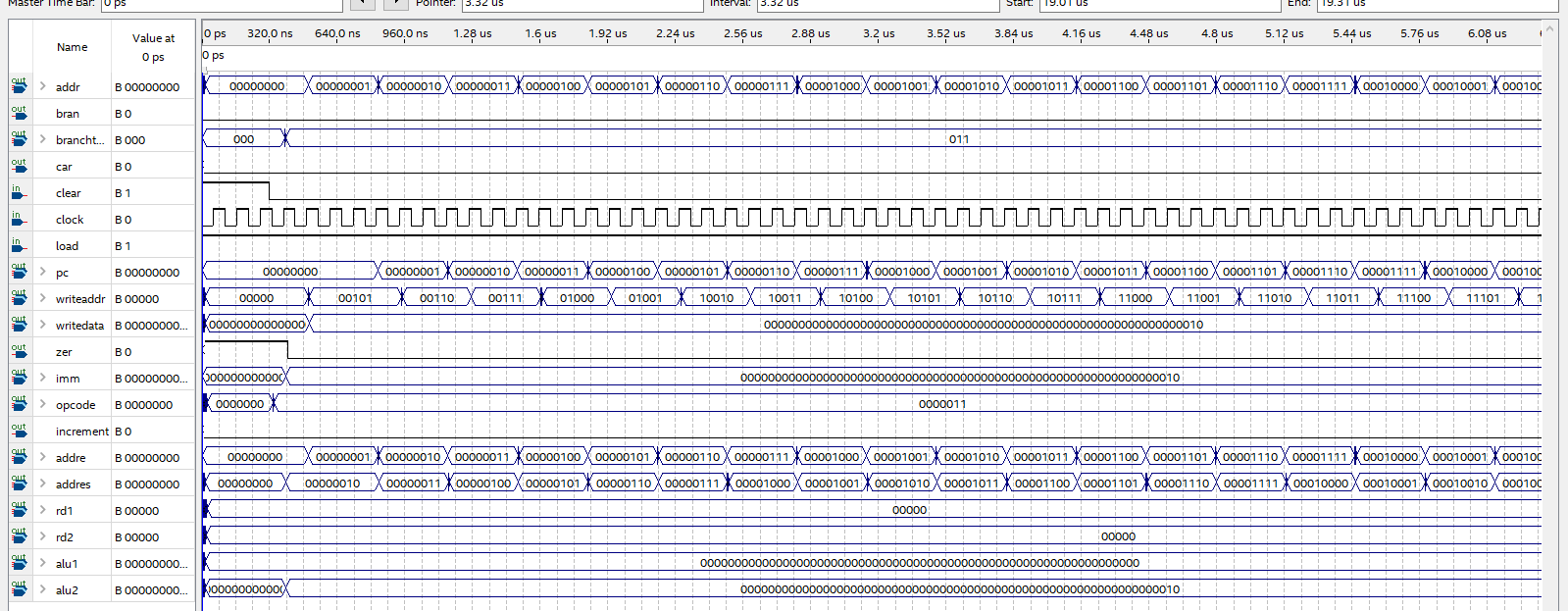
blt x7, x8, loop (back 6)

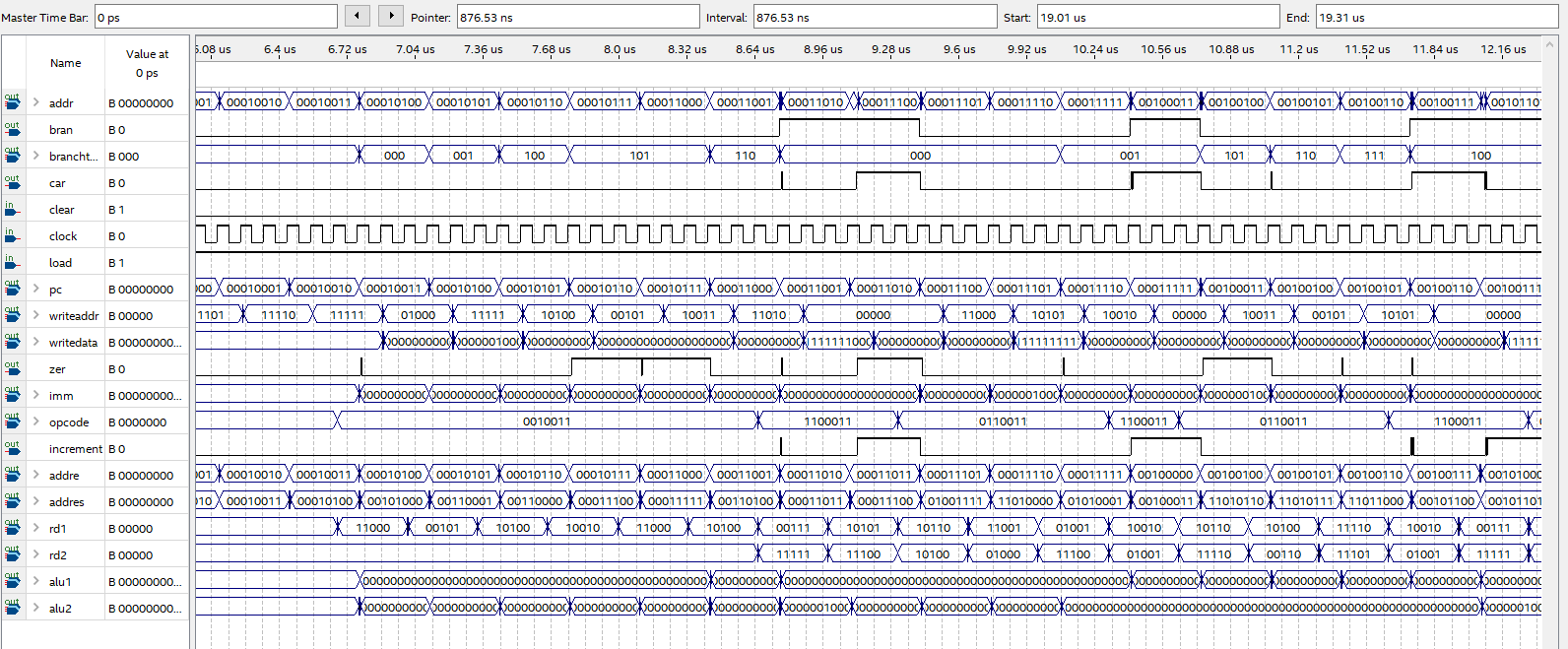
exit:

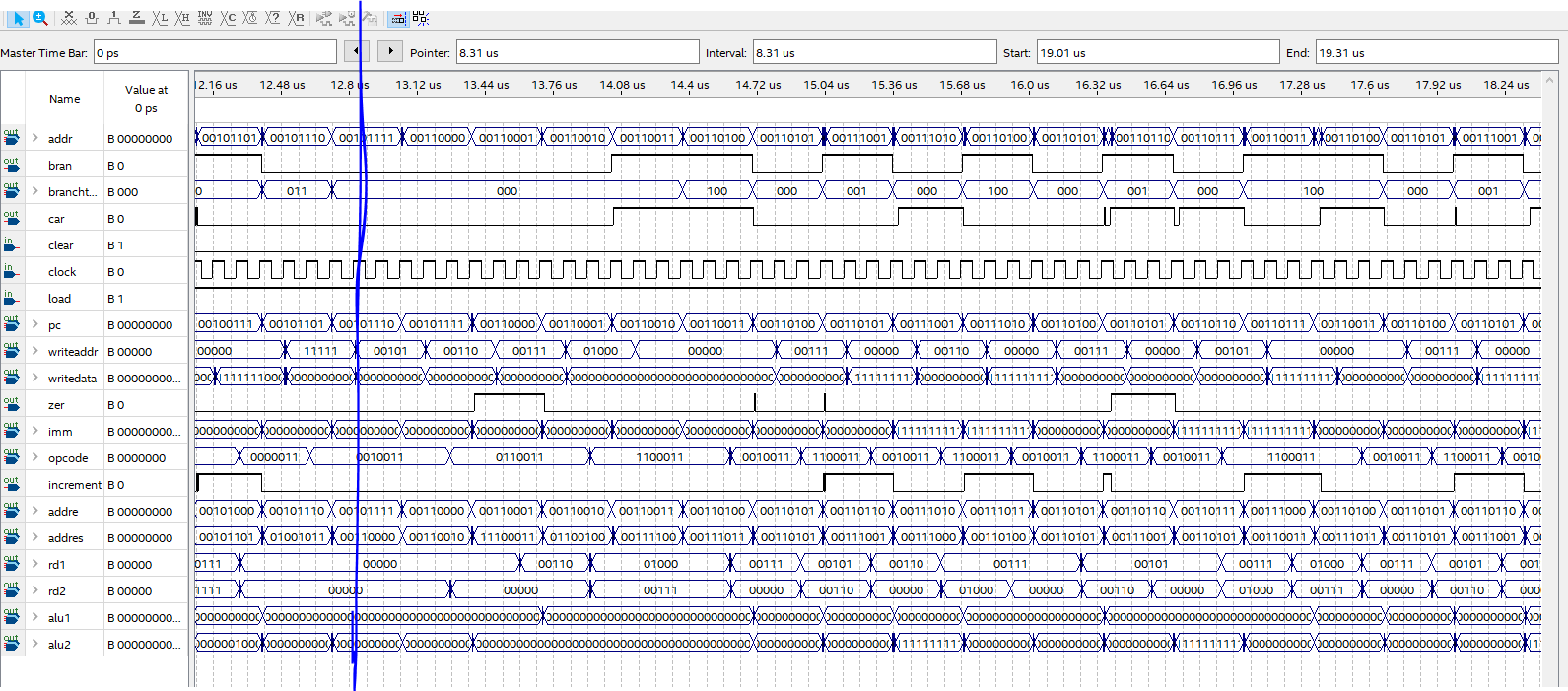
#noop always false helps earlier branch not skip first instruction

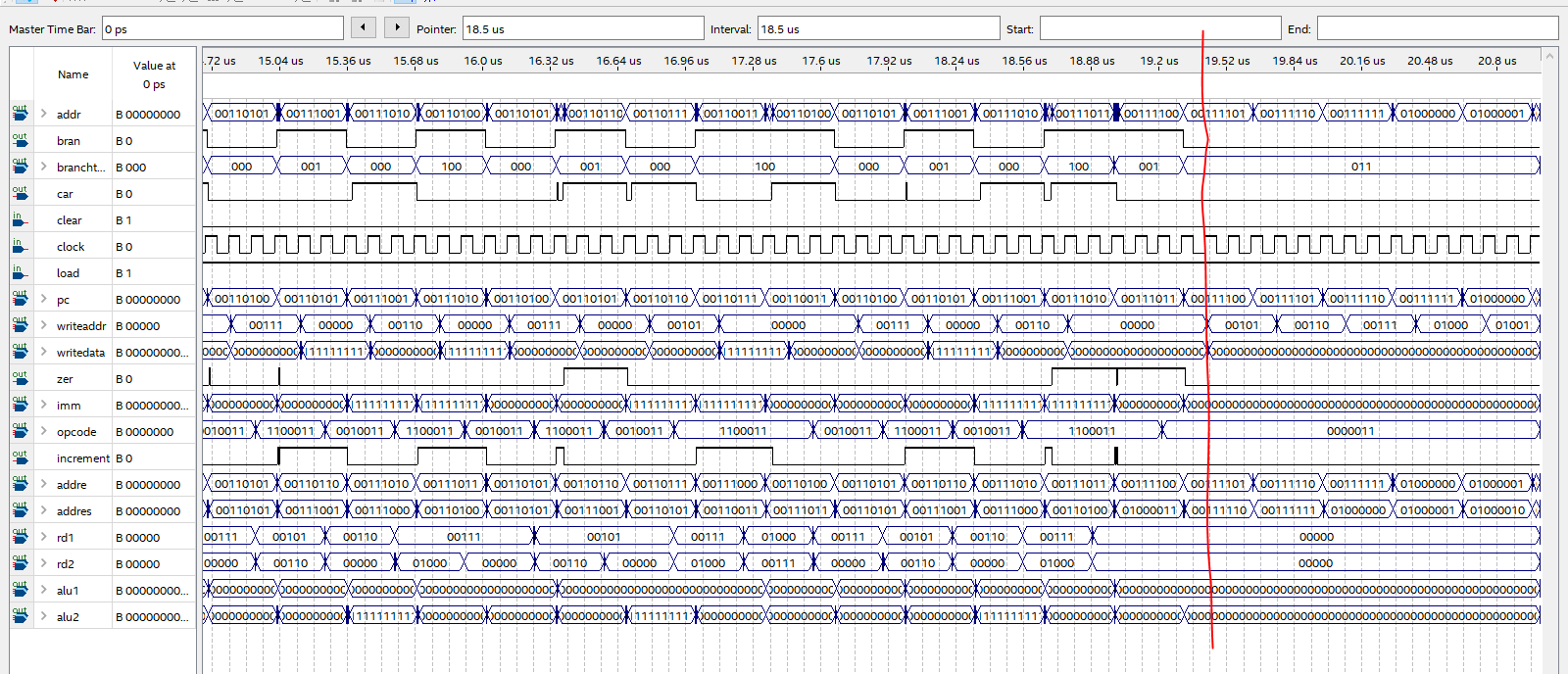
bne x0,x0

Testing Waveform for datapath: (blue line is start of C code red is end of C code)









Full datapath waveform:

