CPU_Processor_21363904

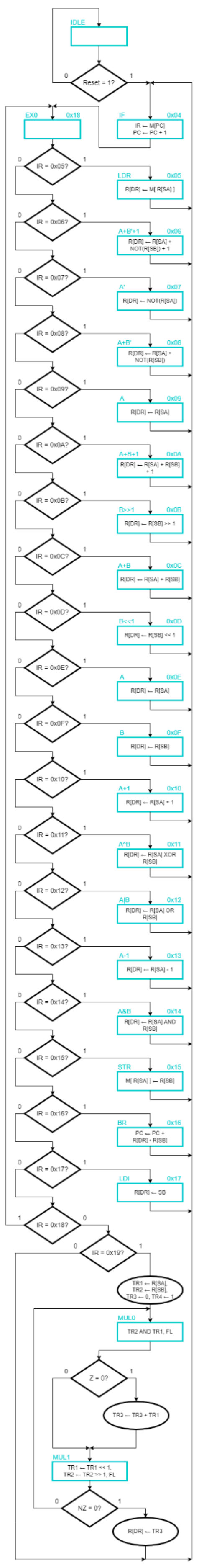
Introduction

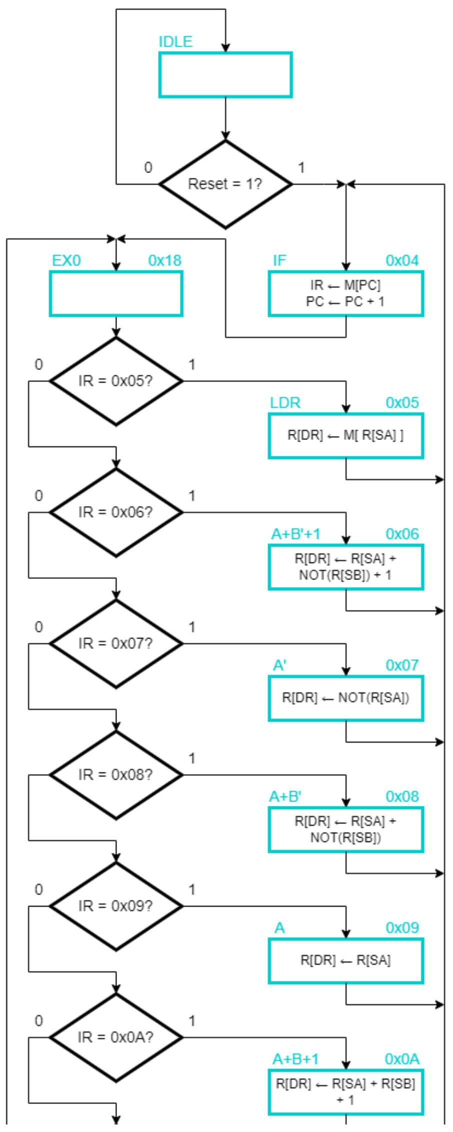
This is the document for the full Processor Assignment. It contains all the information regarding the simulation of this entity: Timing Diagrams, Instruction Set ASM, RAM Program utilizing the Instruction Set, and the order for my FS Code (retained from Function Unit, Datapath and Processor Tests 01 and 04). All numbers in the Timing Diagrams are in Hexadecimal notation. Each clock cycle takes 360ns, and each RAM instruction takes 3 cycles to execute: one to fetch the instruction, one to load it and one to execute it. The only exception to this is the Shift-Add Multiplication instruction, which can take place over many clock cycles, the exact number depending on the input.

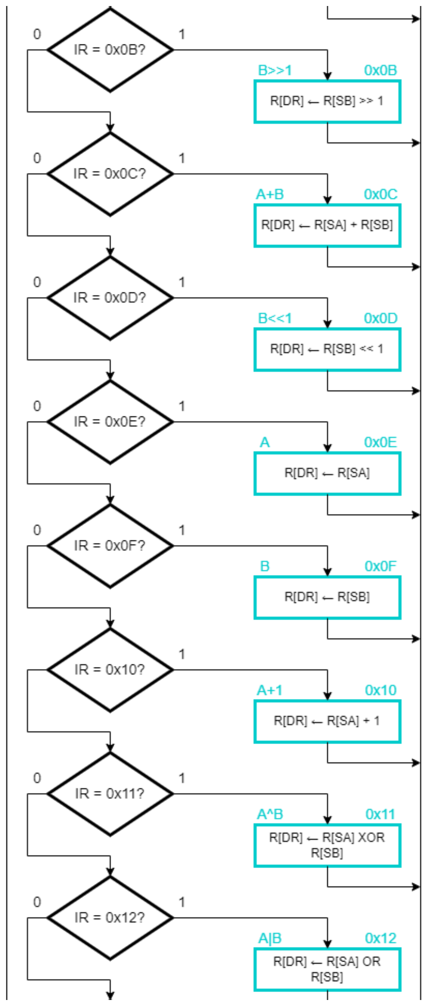
The Instruction Set

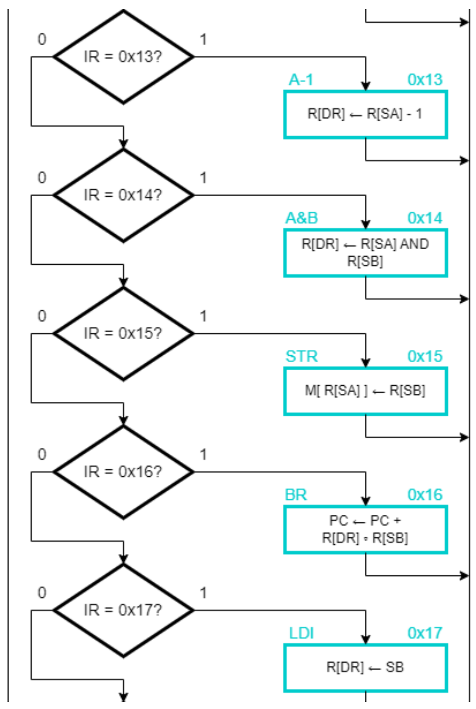
As requested in the Simulation Procedure, I have designed an instruction set in my Control Memory to allow for the execution of basic instructions by my processor. This instruction set consists of 32 total control words, though 11 of these are part of a multi-cycle instruction. The address of the first instruction is 0x04, the last two digits of my Student ID. Firstly, there is an **Instruction Fetch (IF)**, in order to fetch the next instruction in RAM. The Instruction Fetch then calls the **Execute Instruction (EX0)**, which allows the RAM instruction to be loaded into the Control ROM. There is an instruction for each of the **15 Datapath Microoperations**, and there are a further 3 instructions for register loading and storing: the **Load Instruction (LDR)**, which loads a value from a specified address in RAM into a specified register; the **Load Immediate Instruction (LDI)**, which loads the number SB (the number of the register itself, not its contents) into a specified register (useful when initializing a register for the first time); and the **Store Instruction (STR)**, which stores the value of a specified register into a specified address in RAM. Then there is a **Branch Instruction (BR)**, which displaces the PC by a specified amount, allowing for looping and accessing subroutines, and finally, a **Multiply Instruction (MUL)**, which is a multi-cycle instruction consisting of 11 control words which implement Shift-Add Multiplication. The Algorithmic State Machine chart (ASM) for this instruction set is shown overleaf.

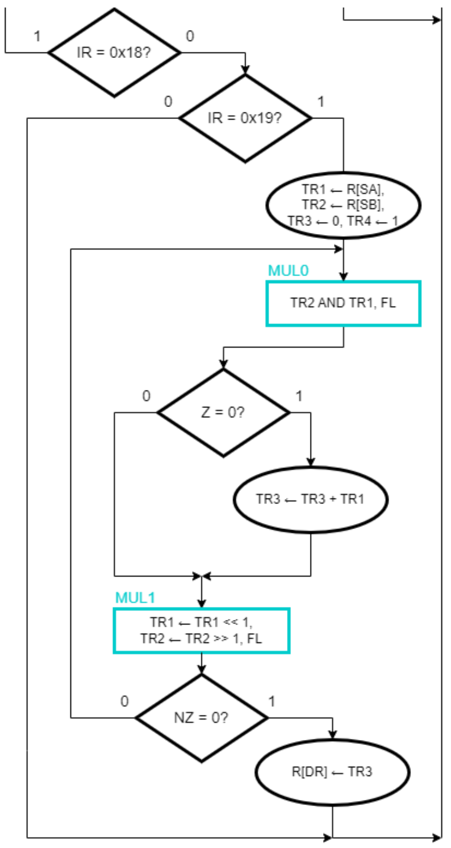
CPU_Processor_21363904











CPU_Processor_21363904

The Multiply Instruction (MUL)

The Multiply instruction consists of the following Control Words:

1. Load R[SA] into TR1 (this will be our Multiplicand, B)

2. Load R[SB] into TR2 (this will be our Multiplier, Q)

3. Load #0 into TR3 by XORing TR1 with itself (this will be our Product, A)

4. Load #1 into TR4 by incrementing TR3 (this will allow us to get the last digit of Q)

5. TR2 AND TR4 (Q AND #1), loading flags

6. If the Zero flag is set (i.e. $Q_0 = 0$), branch to 8., else continue

7. TR3 = TR3 + TR1 (A = A + B)

8. TR1 = TR1 shifted left by 1 (B << 1)

9. TR2 = TR2 shifted right by 1 (Q >> 1), loading flags

10. If the Not Zero Flag is set (i.e. $Q \neq 0$), branch to 5., else continue

11. Load TR3 (A) into R[DR]

This multi-cycle instruction implementation can be seen at address 0x19 in the Control Memory and in the ASM chart above.

CPU_Processor_21363904

The ProgRAM

In order to demonstrate that my Processor can carry out all the instructions detailed in the Control Memory, I have written a program in my RAM which utilizes every instruction at least once. The start address of this program is 0x03 in the RAM (digit 3 of my Student ID, like for Processor Test 03). The program also makes use of 3 shifting subroutines: sl4_R0, which shifts R0 left four times, sr2, which shifts R2 right twice, and sl4_R2, which shifts R2 left four times. They are implemented at addresses 0x3C, 0x2C and 0x41 respectively. Finally, there are 2 values (0x3 and 0x6) stored in the RAM at addresses 0x70 and 0x71 respectively, with space for a third at 0x72. These values are loaded into registers by the program, and the final values of registers R1, R2 and R3 are loaded into those addresses at the end of the program (the values being 0x10, 0x10 and 0x100 respectively).

This entire program including the subroutines is provided overleaf.

Here is the order of my FS Code as determined by the last digit of my Student ID for reference.

FS Code for ID_0 = 4:

```
0x05 - 00101 = A + NOT(B) + 1
0x0E - 01110 = NOT(A)
0x04 - 00100 = A + NOT(B)
0x07 - 00111 = A
0x03 - 00011 = A + B + 1
0x14 - 10100 = B>>
0x02 - 00010 = A + B
0x18 - 11000 = B<<
0x00 - 00000 = A
0x10 - 10000 = B
0x01 - 00001 = A + 1
0x0C - 01100 = A XOR B
0x0A - 01010 = A OR B
0x06 - 00110 = A – 1
0x08 - 01000 = A AND B
```

CPU_Processor_21363904

```
Main:

1.  Load R2 with 0x7 via Load Immediate using MuxB
2.  Branch to sl4_R0 to assign 0x70 to R0 - Displace to 0x3C
3.  Load Data from 0x70 into R1
4.  Increment R0
5.  Load Data from 0x71 into R2
6.  Increment R0
7.  R3 <= R1 + NOT(R2) + 1
8.  R3 <= NOT(R3)
9.  R3 <= R3 + NOT(R2)
10. R1 <= R3
11. R3 <= R1 + R2 + 1
12. Branch to sr2 - Displace to 0x2C
13. R3 <= R2 + R3
14. Branch to sl4_R2 - Displace to 0x41
15. R1 <= R3
16. R3 <= R2
17. R1 <= R1 + 1
18. R3 <= R1 XOR R2
19. R3 <= R1 OR R3
20. R3 <= R3 - 1
21. R1 <= R2 AND R3
22. R3 <= R1 * R2 (Multi-Cycle Shift-Add Multiplication)
23. Store Data from R3 into 0x72
24. Decrement R0
25. Store Data from R2 into 0x71
26. Decrement R0
27. Store Data from R1 into 0x70

End Main:


sr2:

1. R2 <= R2 >> 1
2. R2 <= R2 >> 1
3. Branch to Main - Displace to 0x0F

End sr2:
```

CPU_Processor_21363904

```
sl4_R0:

1. R0 <= R0 << 1
2. R0 <= R0 << 1
3. R0 <= R0 << 1
4. R0 <= R0 << 1
5. Branch to Main - Displace to 0x05

End sl4_R0:


sl4_R2:

1. R2 <= R2 << 1
2. R2 <= R2 << 1
3. R2 <= R2 << 1
4. R2 <= R2 << 1
5. Branch to Main - Displace to 0x11

End sl4_R2:
```

CPU_Processor_21363904

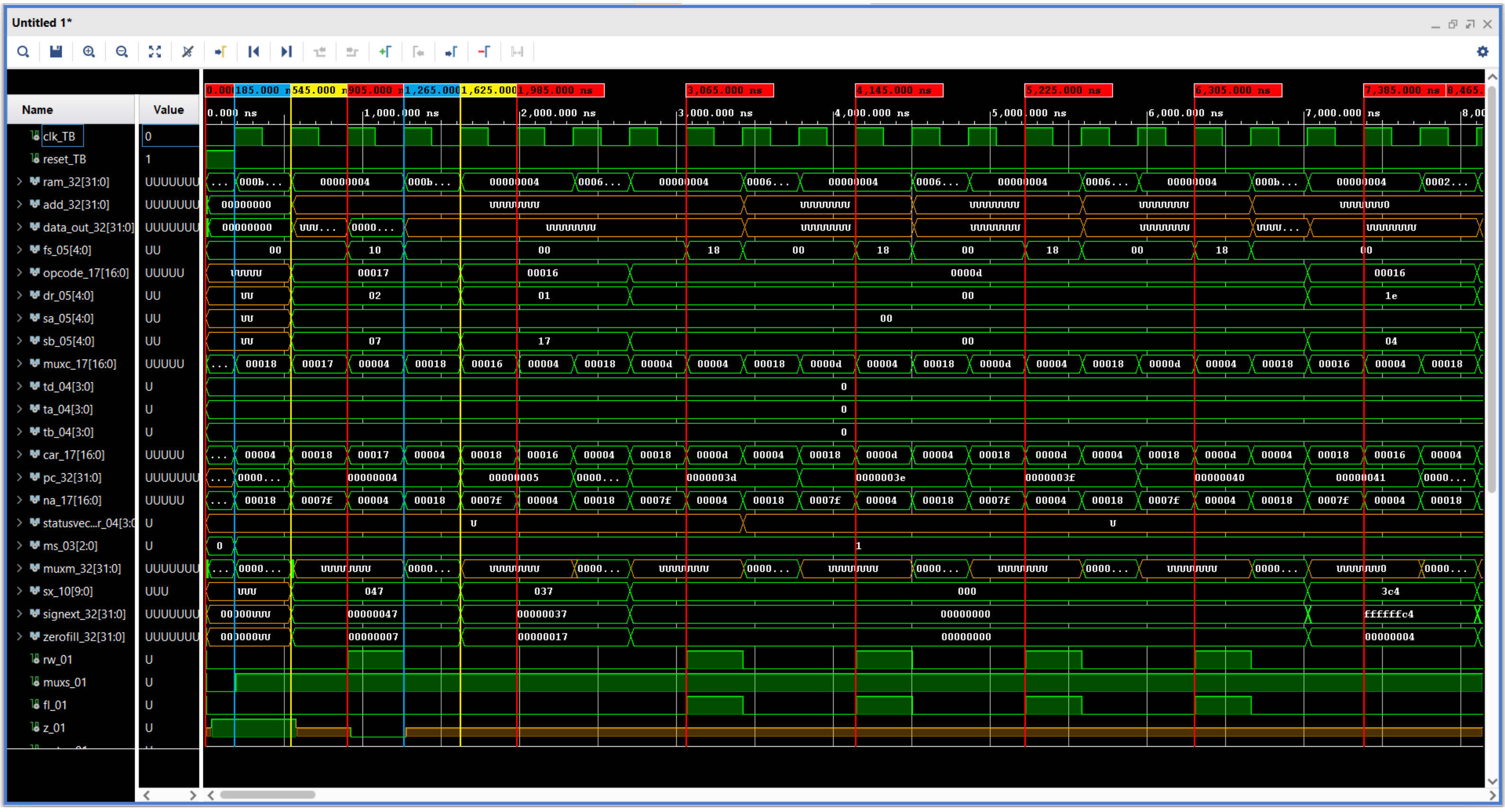
The Timing Diagrams

The timing diagrams for the Processor are shown overleaf. All numbers are in Hexadecimal notation. Each clock cycle takes 360ns, and each RAM instruction takes 3 cycles to execute: one to fetch the instruction, one to load it and one to execute it. The only exception to this is the Shift-Add Multiplication instruction, which takes 33 clock cycles to execute (including IF and EX0).

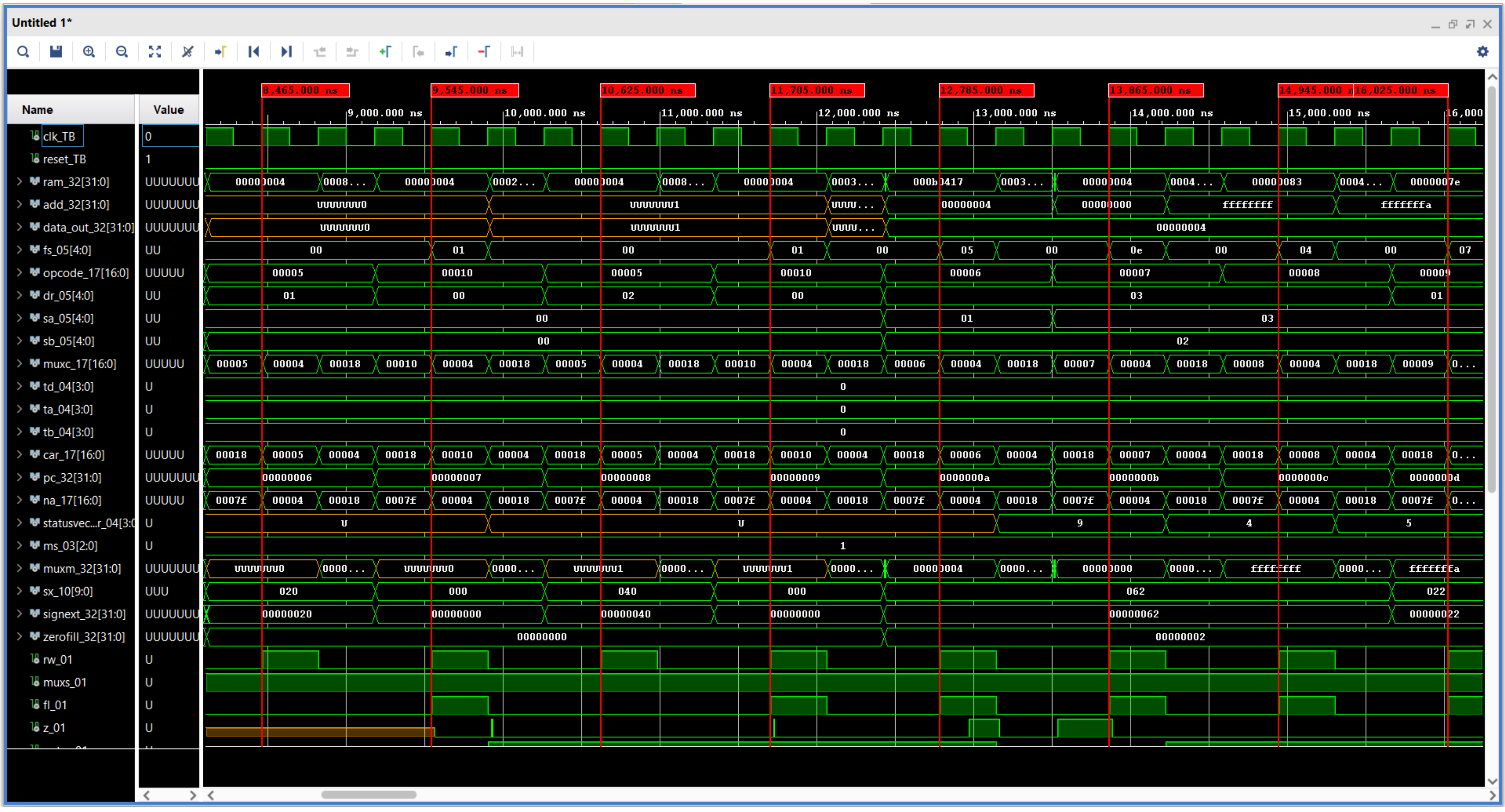
In the first diagram, the blue lines show when the RAM instruction is fetched, the yellow lines when the instruction is loaded, and the red lines when the instruction is executed. In all the remaining diagrams only the red lines are shown. The program is kickstarted in the testbench by setting the reset signal, which resets the CAR to the address of IF and the PC to the first RAM instruction. The first RAM instruction loads the value of SB (7) into R0 using LDI, in order to initialize the first register. This works by pushing the zero-extended SB from MuxB through the Function Unit and writing it into R0, giving us a value that we can now operate on. This value in R0 is shifted left 4 times so that it becomes 0x70, and then we can use that as our address to load values from RAM into registers 1 and 2. From there, the 15 microoperations are performed on registers 1, 2 and 3 (which includes branching to the shifting subroutines), followed by the multi-cycle MUL instruction, and the program finishes off by storing the contents of R1, R2 and R3 into RAM addresses 0x70, 0x71 and 0x72 respectively. The program then calls the EX0 instruction, entering an infinite loop to signify that the program has been completed.

The Timing Diagrams follow, with captions underneath to help illustrate which instructions are being carried out in each image.

CPU_Processor_21363904

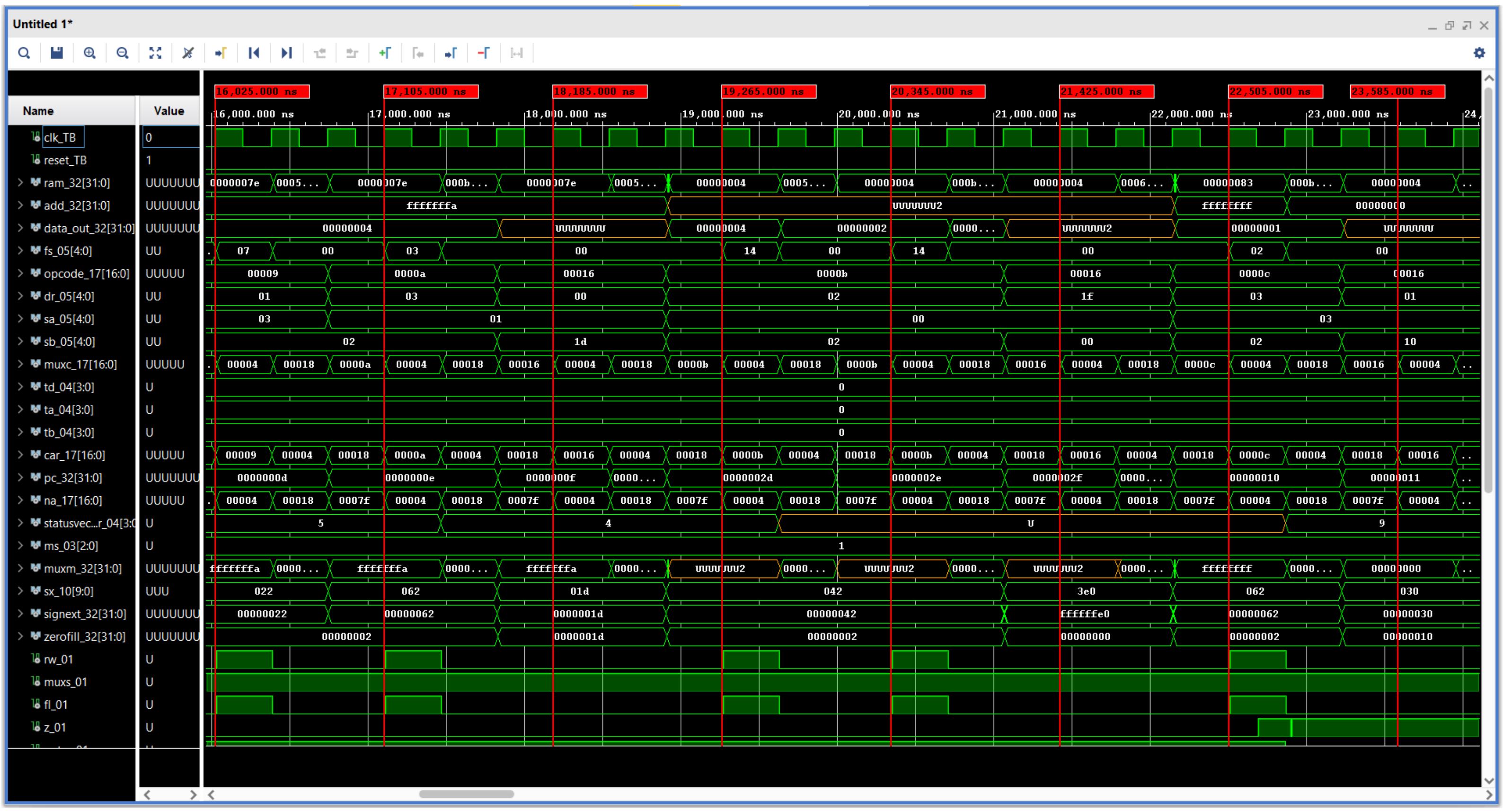


1. Program starts, R0 loaded with SB, Branch to sl4_R0, B<<1 x4

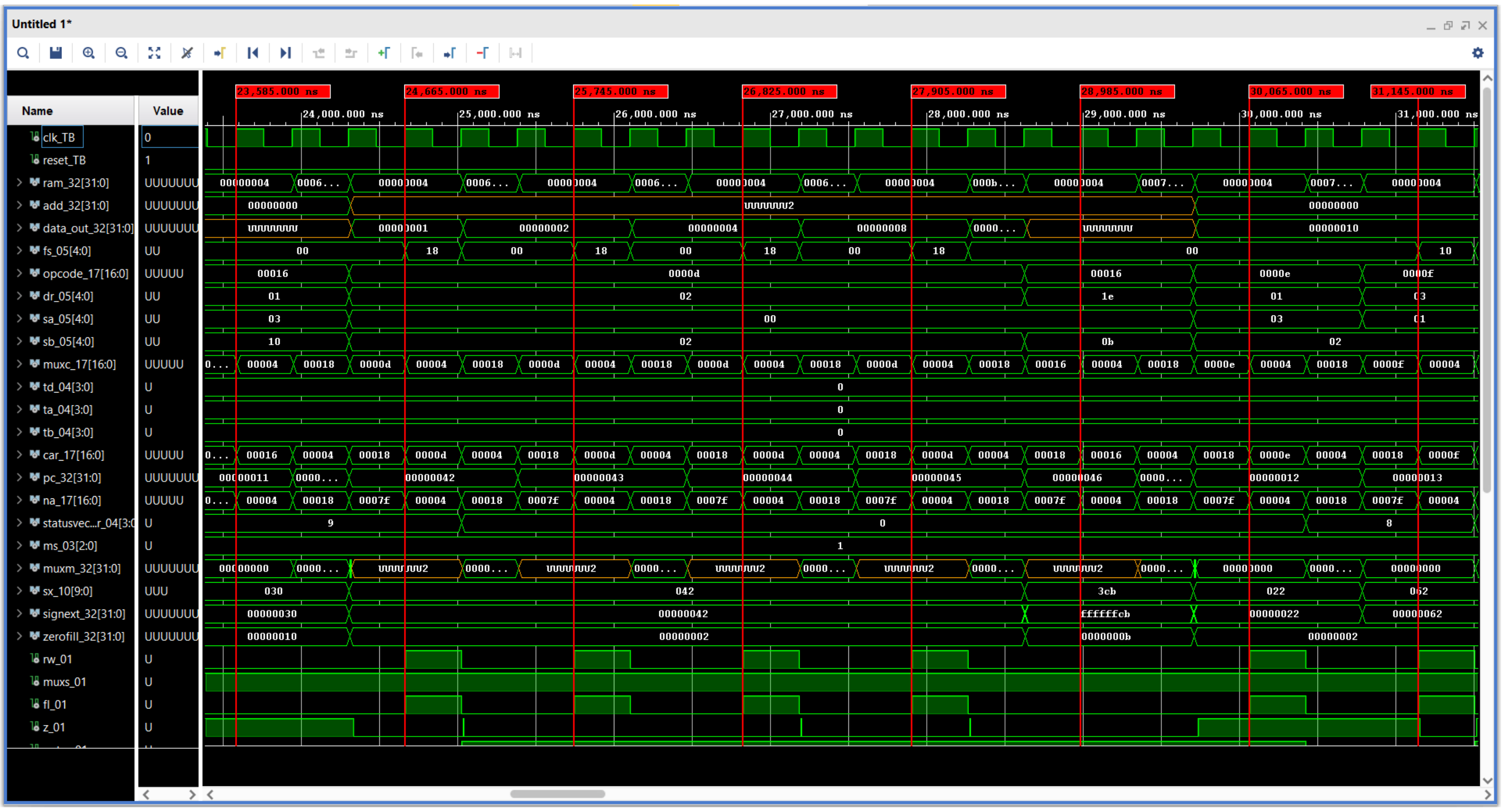


2. R1 loaded with 3, R0+1, R2 loaded with 6, R0+1, A+B’+1, A’, A+B’

CPU_Processor_21363904

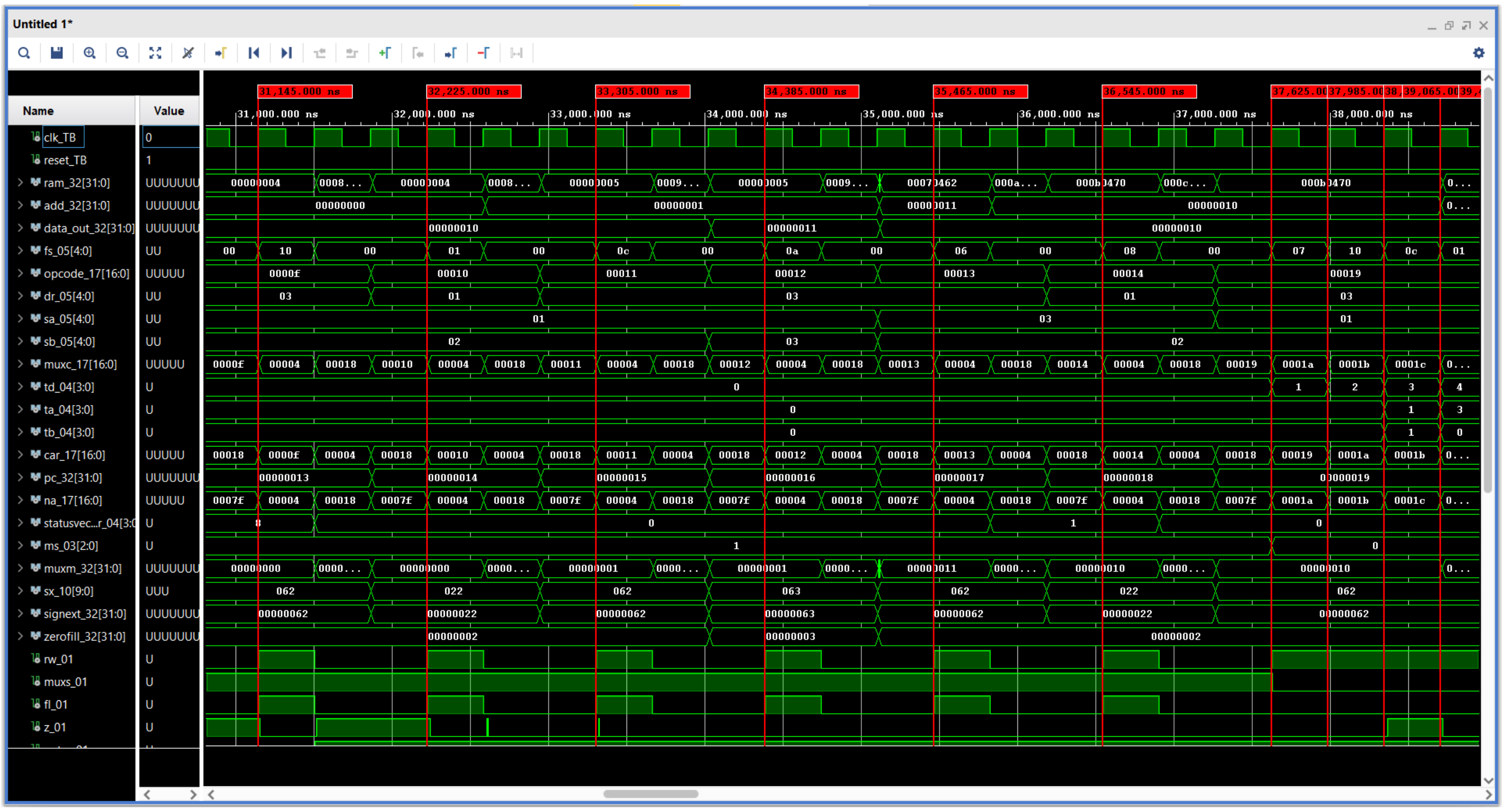


3. A, A+B+1, Branch to sr2, B>>1 x2, Branch to main, A+B

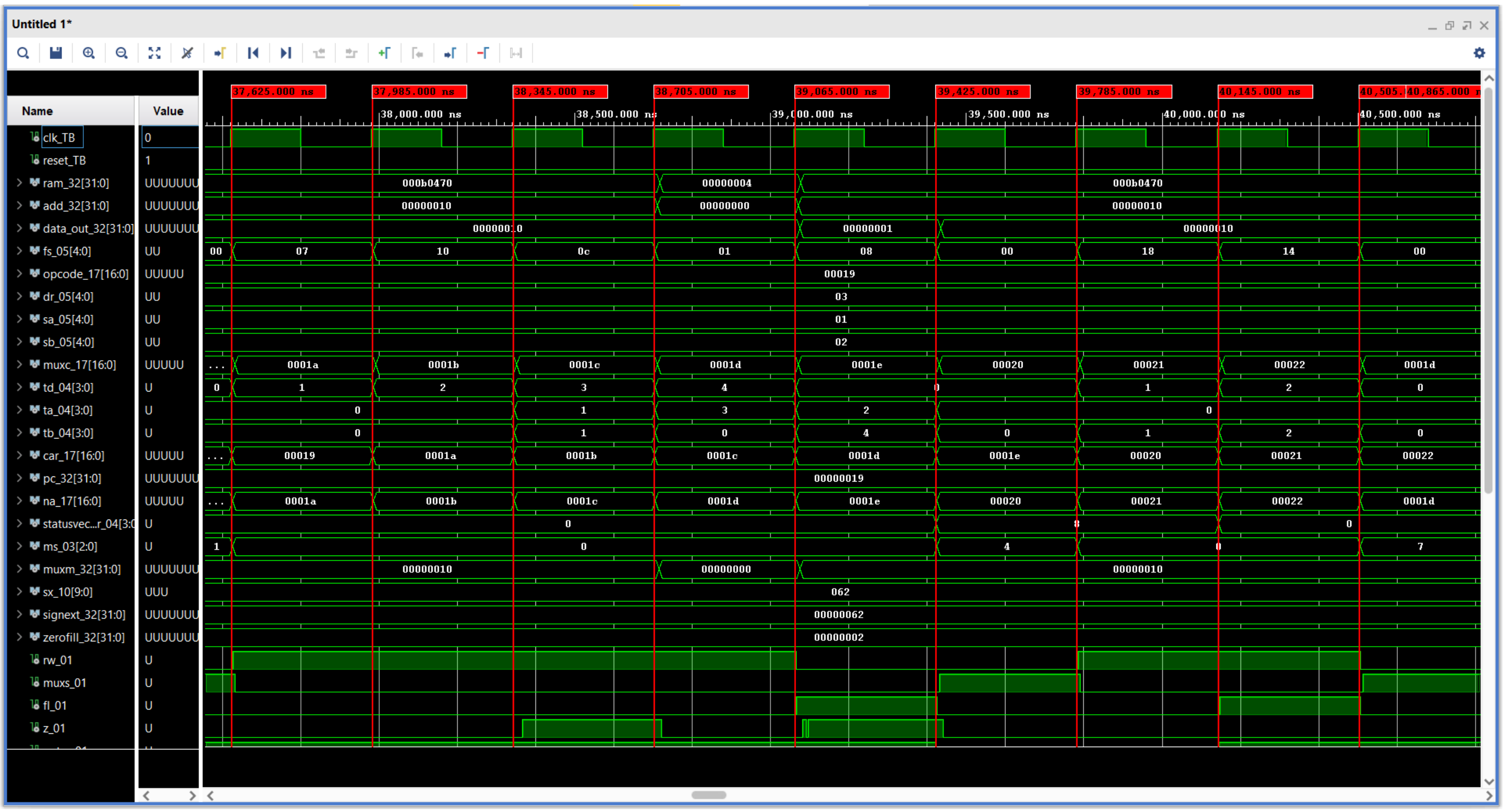


4. Branch to sl4_R2, B<<1 x4, Branch to main, A

CPU_Processor_21363904

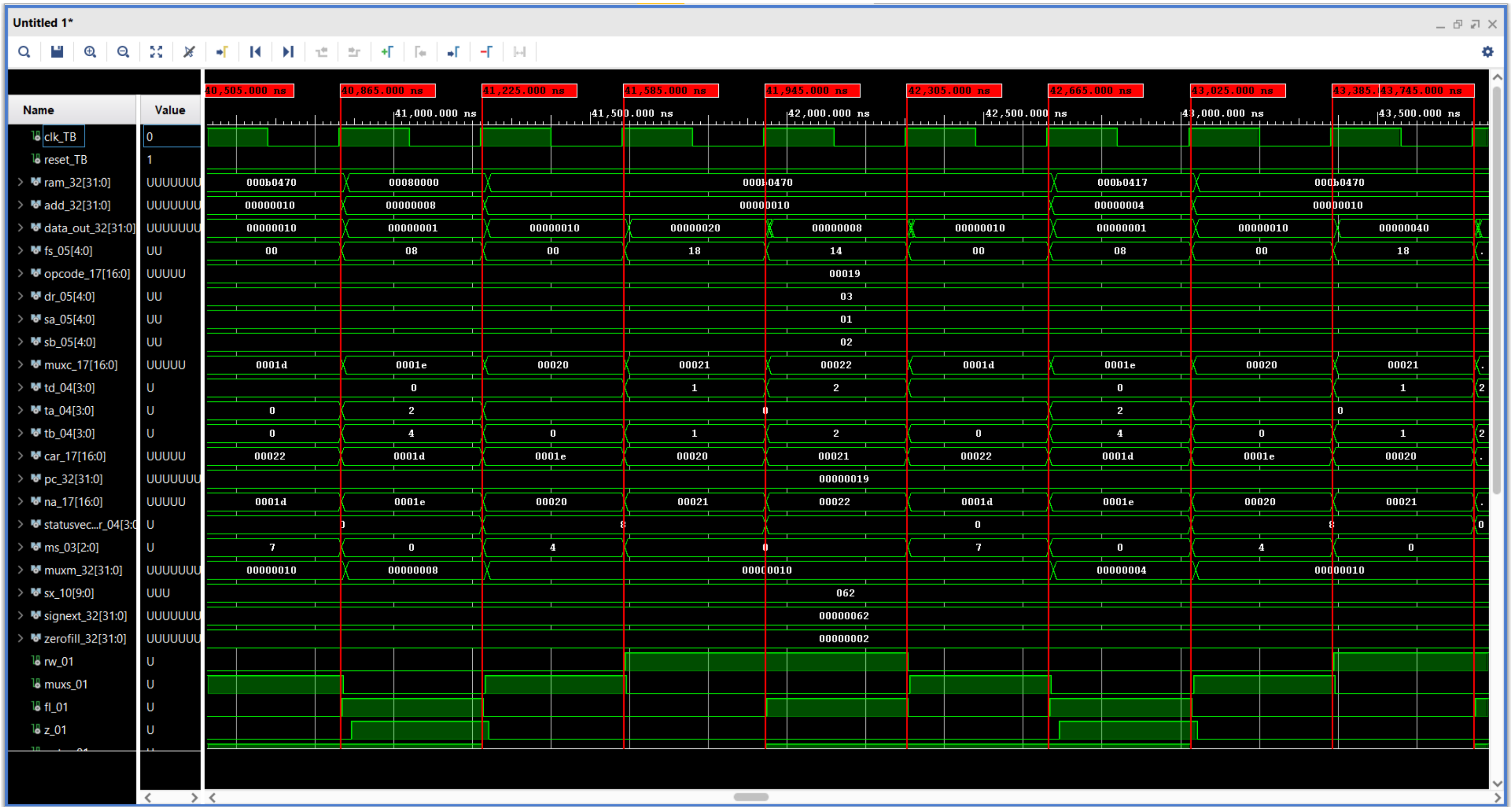


5. B, A+1, A XOR B, A OR B, A-1, A AND B, A * B start

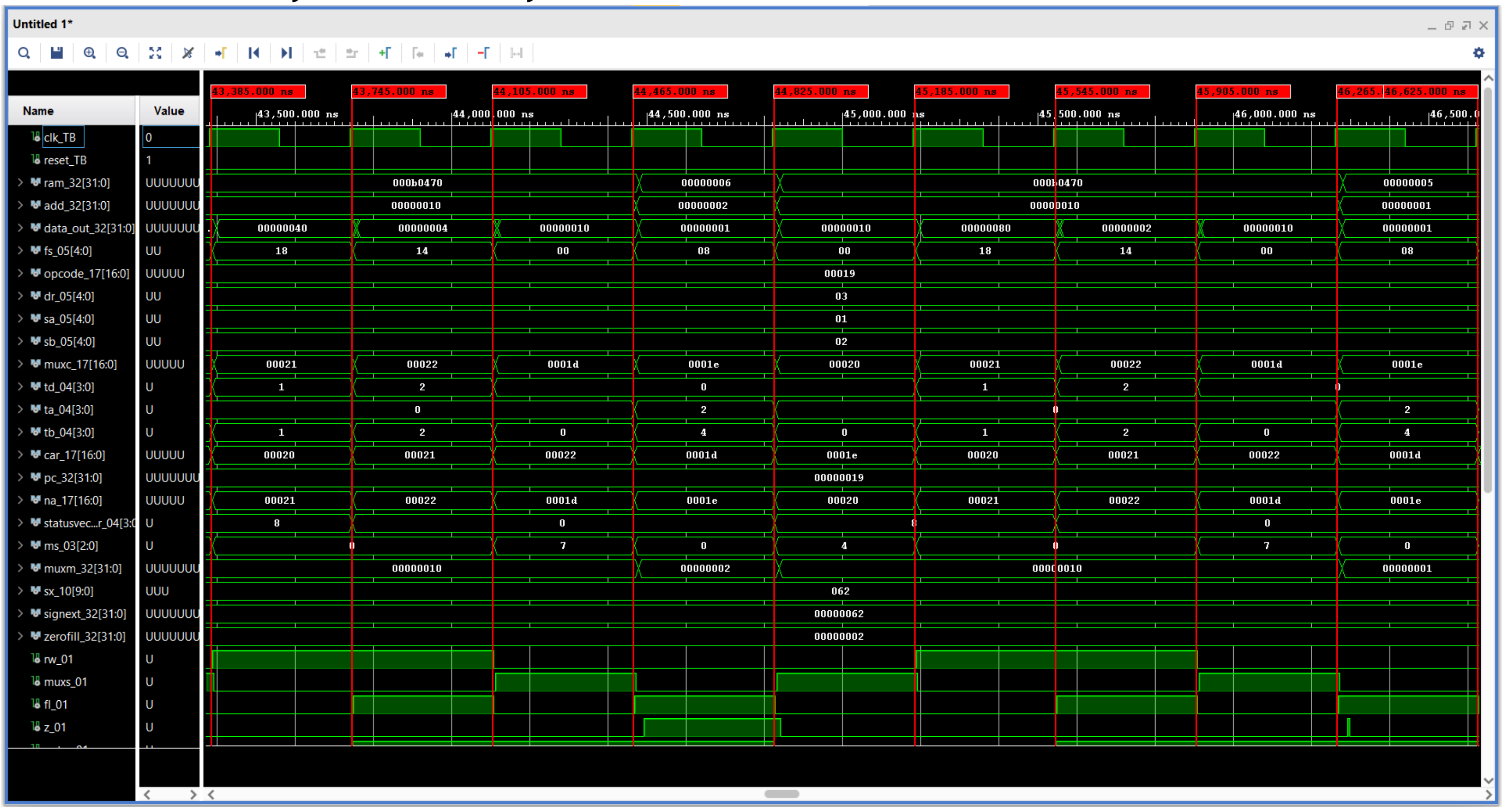


6. TR1 = R[SA], TR2 = R[SB], TR3 = TR1 XOR TR1 (0), TR4 = TR3+1 (1), TR3 AND TR4, Z=1 branch, TR1<<1, TR2>>1

CPU_Processor_21363904

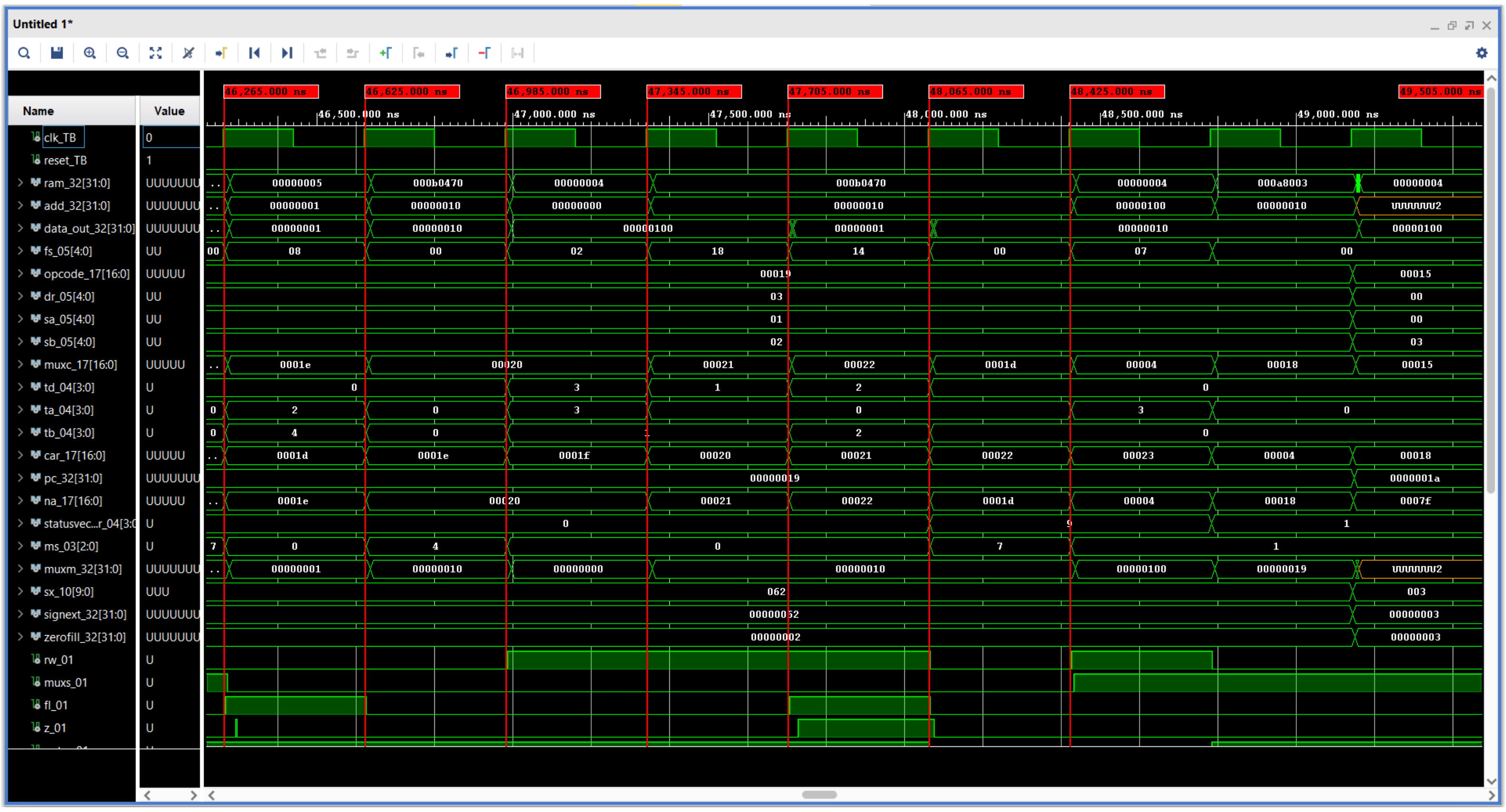


7. NZ=1 branch, TR3 AND TR4, Z=1 branch, TR1<<1, TR2>>1, NZ=1 branch, TR3 AND TR4, Z=1 branch

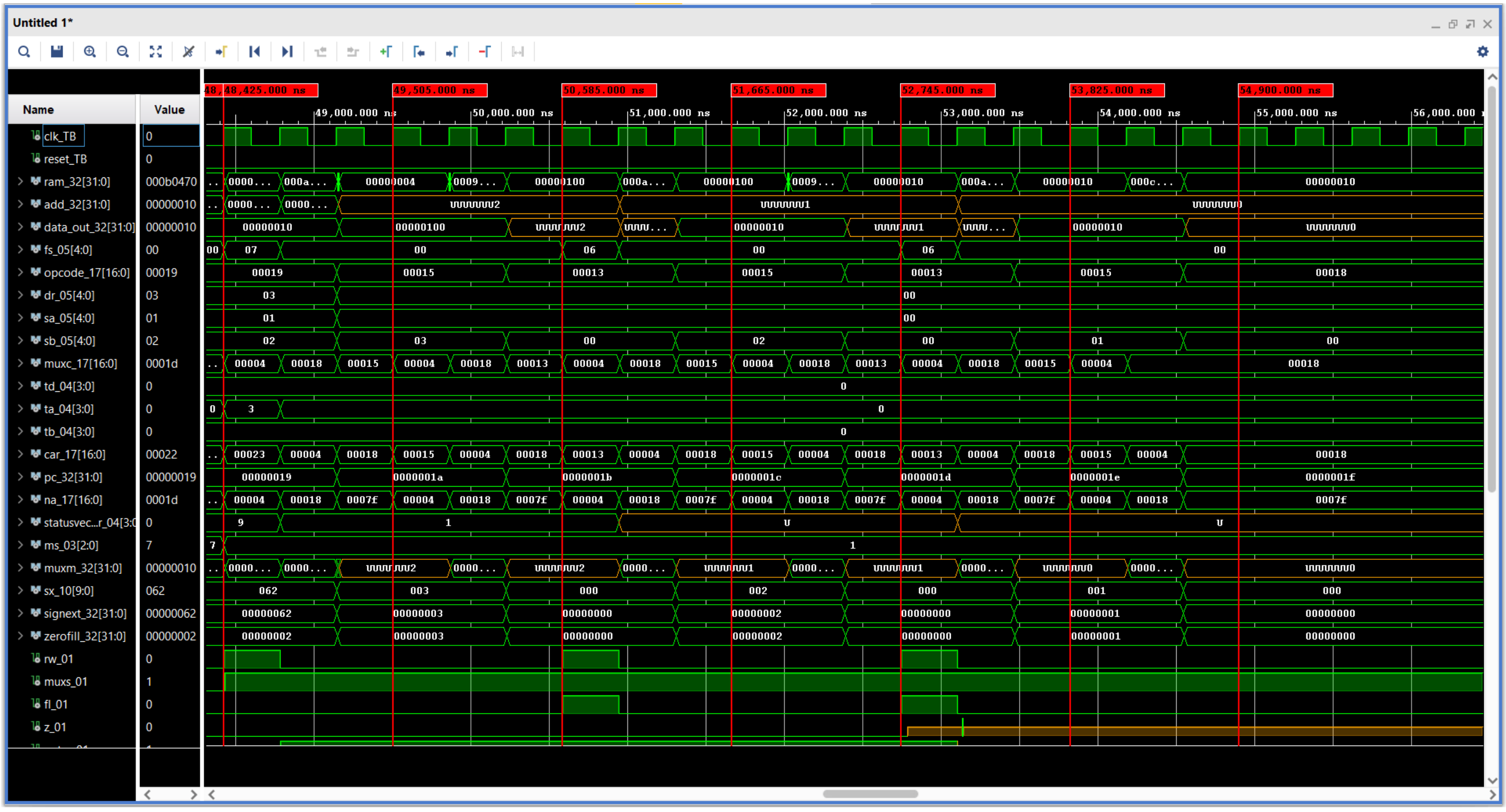


8. TR1<<1, TR2>>1, NZ=1 branch, TR3 AND TR4, Z=1 branch, TR1<<1, TR2>>1, NZ=1 branch

CPU_Processor_21363904



9. Z=0, TR3 = TR3 + TR1, TR1<<1, TR2>>1, NZ=0, R[DR] = TR3



10. R3 (0x100) stored at 0x72, R0-1, R2 (0x10) stored at 0x71, R0-1, R1 (0x10) stored at 0x70, finish.