**HW#03: Program Counter, Branch Instructions**

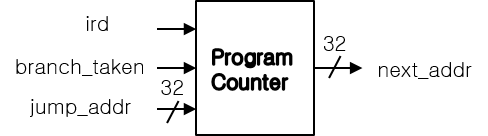
**Issued:** March 28 (Tue.), 2023 **Due:** April 3 (Mon.), 2023

**What to turn in**: **Copy the text from your MODIFIED codes and paste it into a document**. If a question asks you to plot or display something on the screen, also include the plot and screen output your code generates. Submit either a \*.doc or \*.pdf file.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Problem 1 (15p): Program Counter**

Design a program counter in Verilog. Please see the description in the lecture note for details.



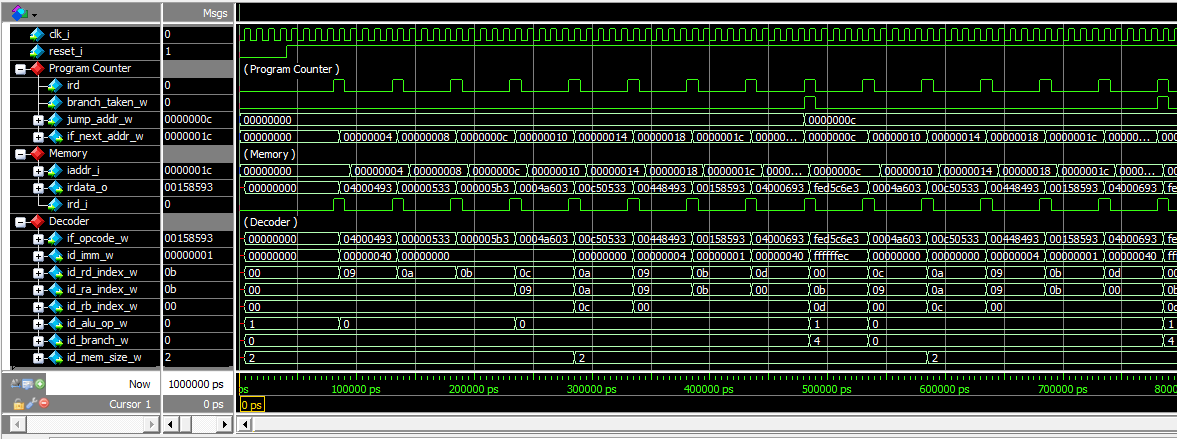
What you have to do:

1. Program Counter (10p)

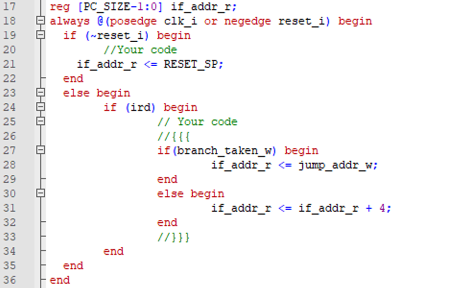
* Complete a program counter and its test bench based on the baseline codes.
* Do simulation and capture its result, i.e., waveform and transcript window.

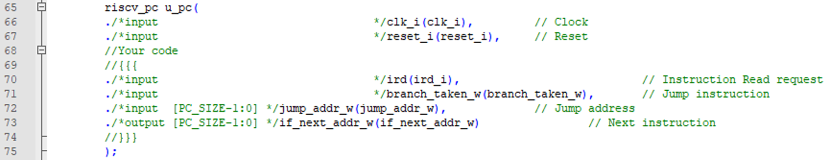
1. The extended program counter (5p): Compared to riscv\_pc\_tb1, riscv\_pc\_tb2 connects iaddr\_i to the output of the Program Counter.

* Complete the test bench (riscv\_pc\_tb2.v).
* Do simulation and capture its result, i.e., waveform and transcript window.

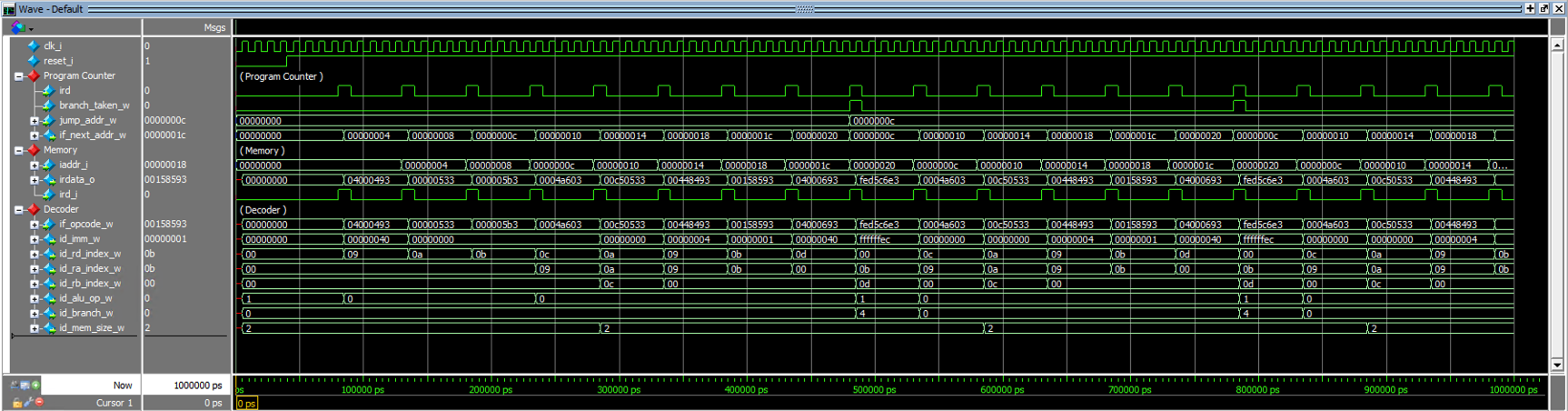


(a) Program Counter



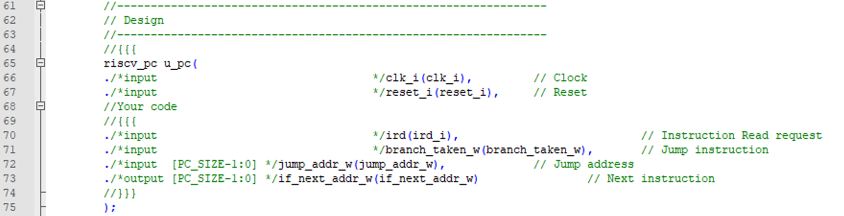






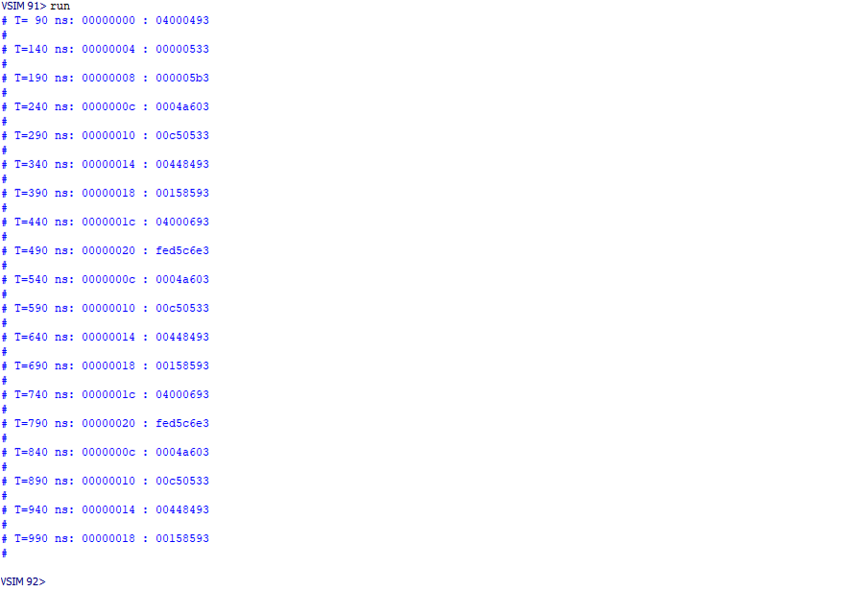


(b) The extended program counter



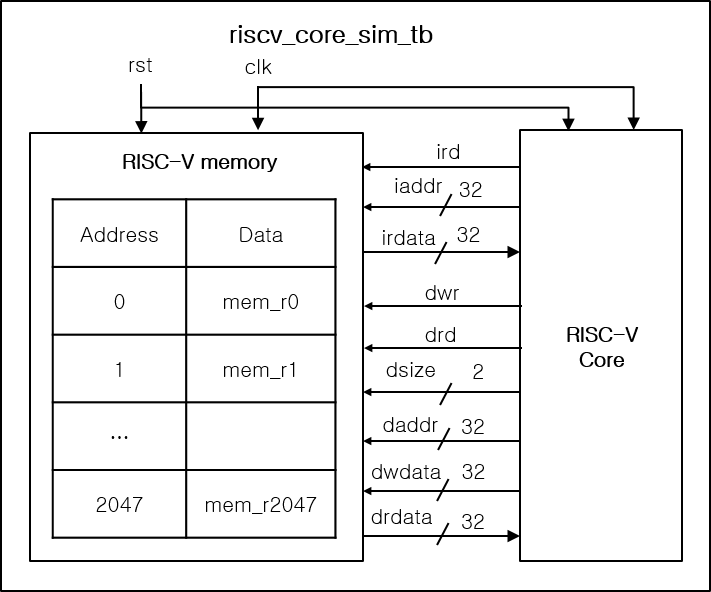






**Problem 2 (20): Program**

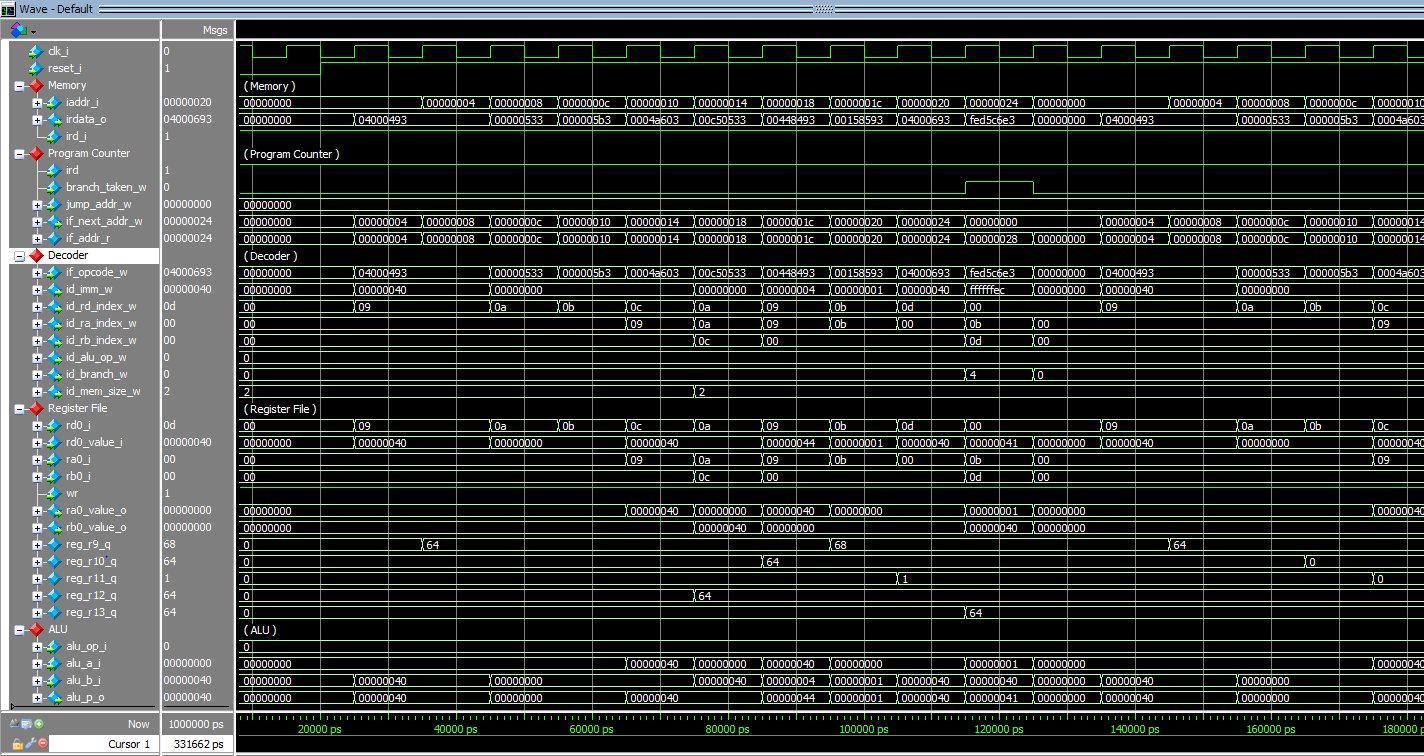
Design a simplified RISC-V core in Verilog. Please see the description in the lecture note.



What you have to do:

1. Baseline core(10p)

* Use Program Counter in Problem 1, complete all connections for ALU, Decoder, and Register File modules.
* Do a simulation and capture the waveform.



|  |  |  |
| --- | --- | --- |
| **C code** | **Assembly Code** | |
| int A[64];  int sum = 0;  for (int i=0; i<64; i++)  sum += A[i]; | addi x9, x0, 64  add x10, x0, x0  add x11, x0, x0  Loop:    lw x12, 0(x9)    add x10, x10, x12    addi x9, x9, 4    addi x11, x11, 1    addi x13, x0, 64    blt x11, x13, Loop | # x9=&A[0]  # sum=0  # i=0  # x12=A[i]  # sum+= A[i]  # &A[i++]  # i++  # x13=64 |



Fig. 2-1: Simulation result.

1. Default target address and branch enable signal for Brach-Less-Than BLT (2p)

In the baseline mode for (BLT), the target address for Branch-Less-Than (BLT) is set to 0, while the branch “enable” signal is **always set to 1**. Explain why r11 in Register File only has two values, 0 and 1.

(c) Branch instruction Brach-Less-Than (BLT) (8p)

1. Target Address for BLT (4p)

* Make code to calculate a target address for BLT.
* Do a simulation and capture its result, i.e., waveform.

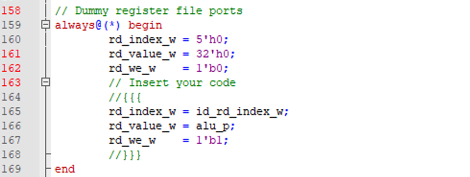
b. Target enable for BLT (4p)

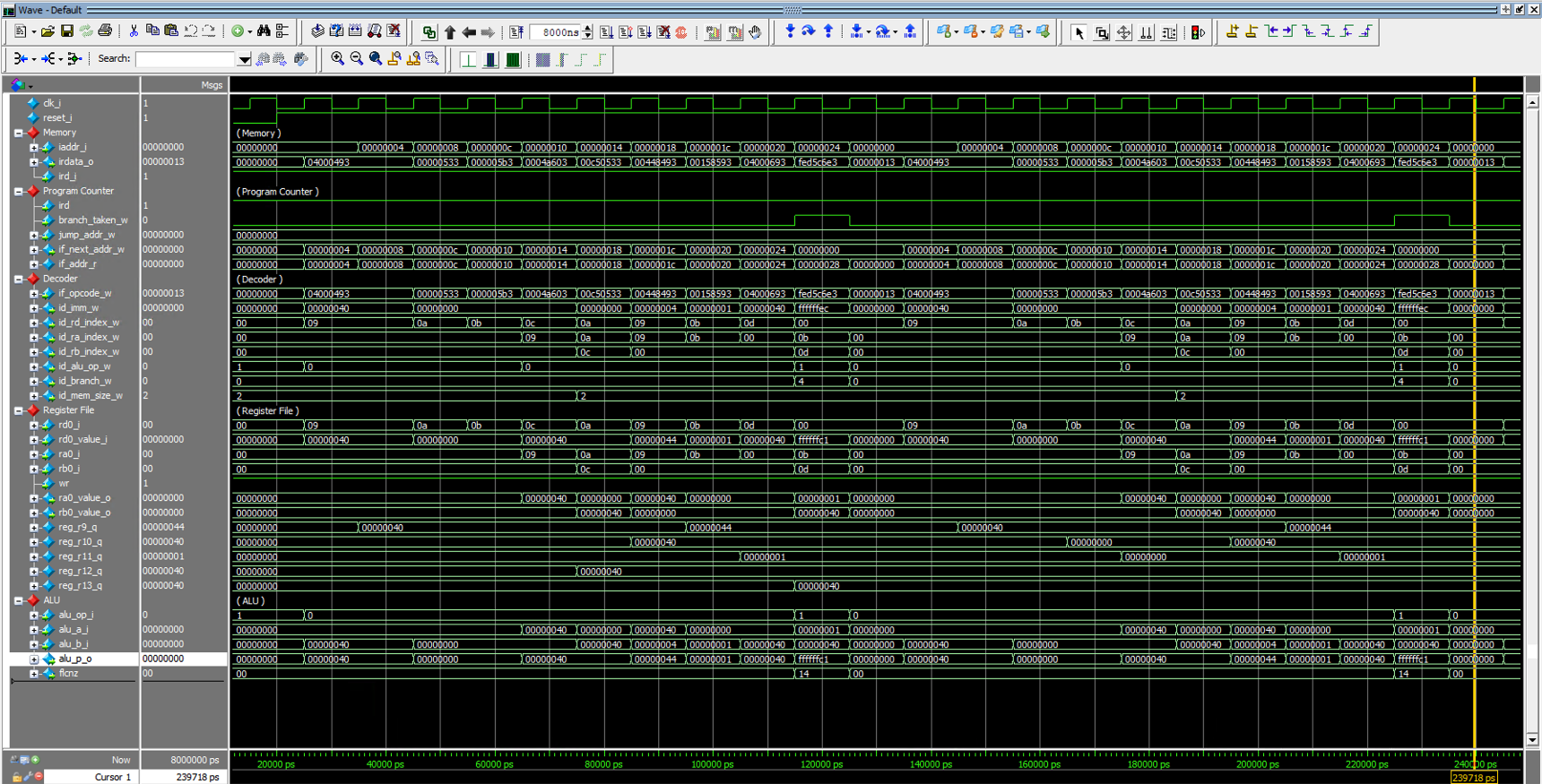
* Make code to generate a branch enable signal for BLT
* Do a simulation and capture the waveform.

----------------------------------------------------------------------------------------------------------------------------------------

(a) Baseline Code





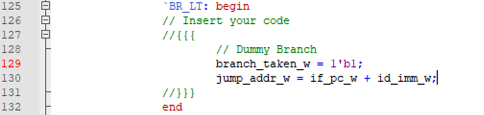


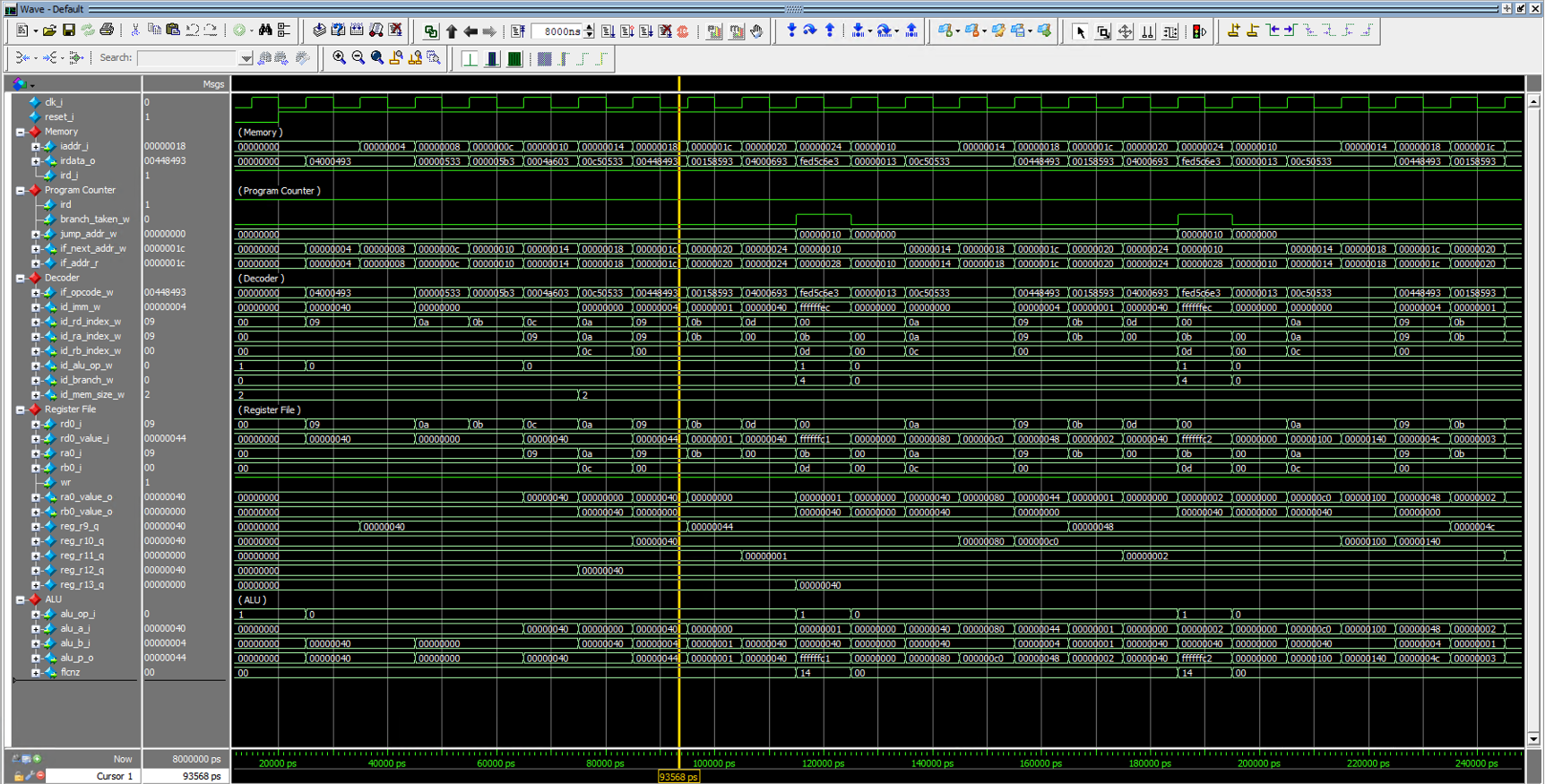
(b) Default target address and branch enable signal for Brach-Less-Than BLT

Answer : Because the target address for Branch-Less-Than is set to 0, and branch enable signal is always set to 1, when the branch instruction arrives, program counter always head to the target address regardless of result of the ALU. Moreover, if the program counter gets to the target address 0, core will again initialize the variables like code “addi x9, x0, 64; add x10, x0, x0; add x11, x0, x0”. These will set the r11 0 again. Since r11 increases by “addi x11, x11, 1” in the loop, as it continues, the sequence of “initialize r11 to 0”, “increase r11 1” will be repeated. So, the value of r11 in Register File only has 0 and 1 repeatedly.

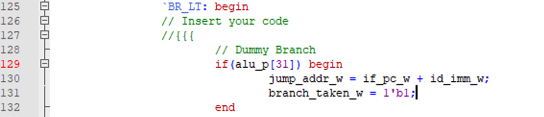
(c) Branch instruction Brach-Less-Than (BLT)

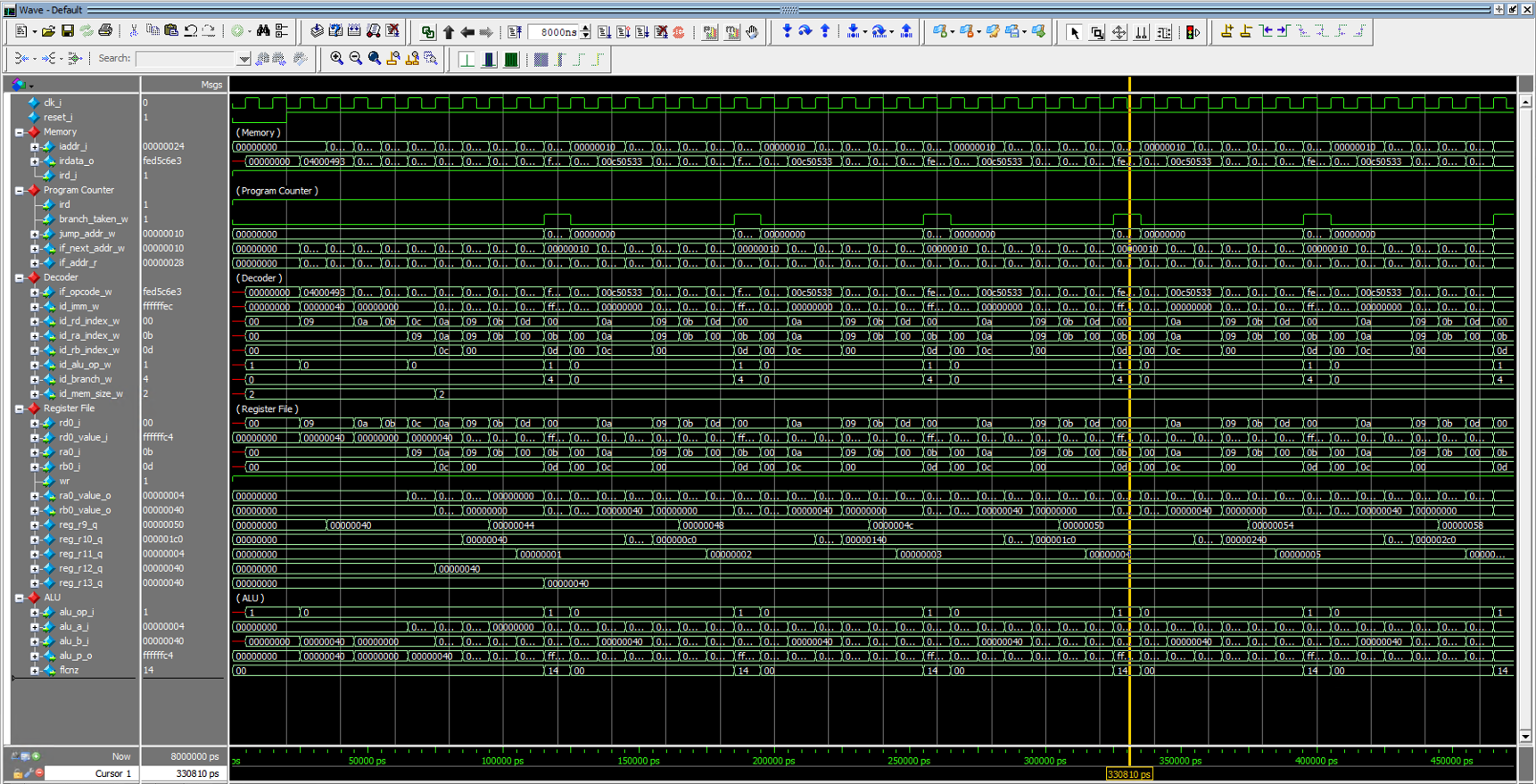
a. Target Address for BLT





b. Target enable for BLT



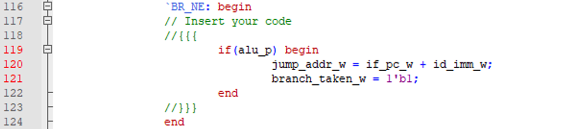


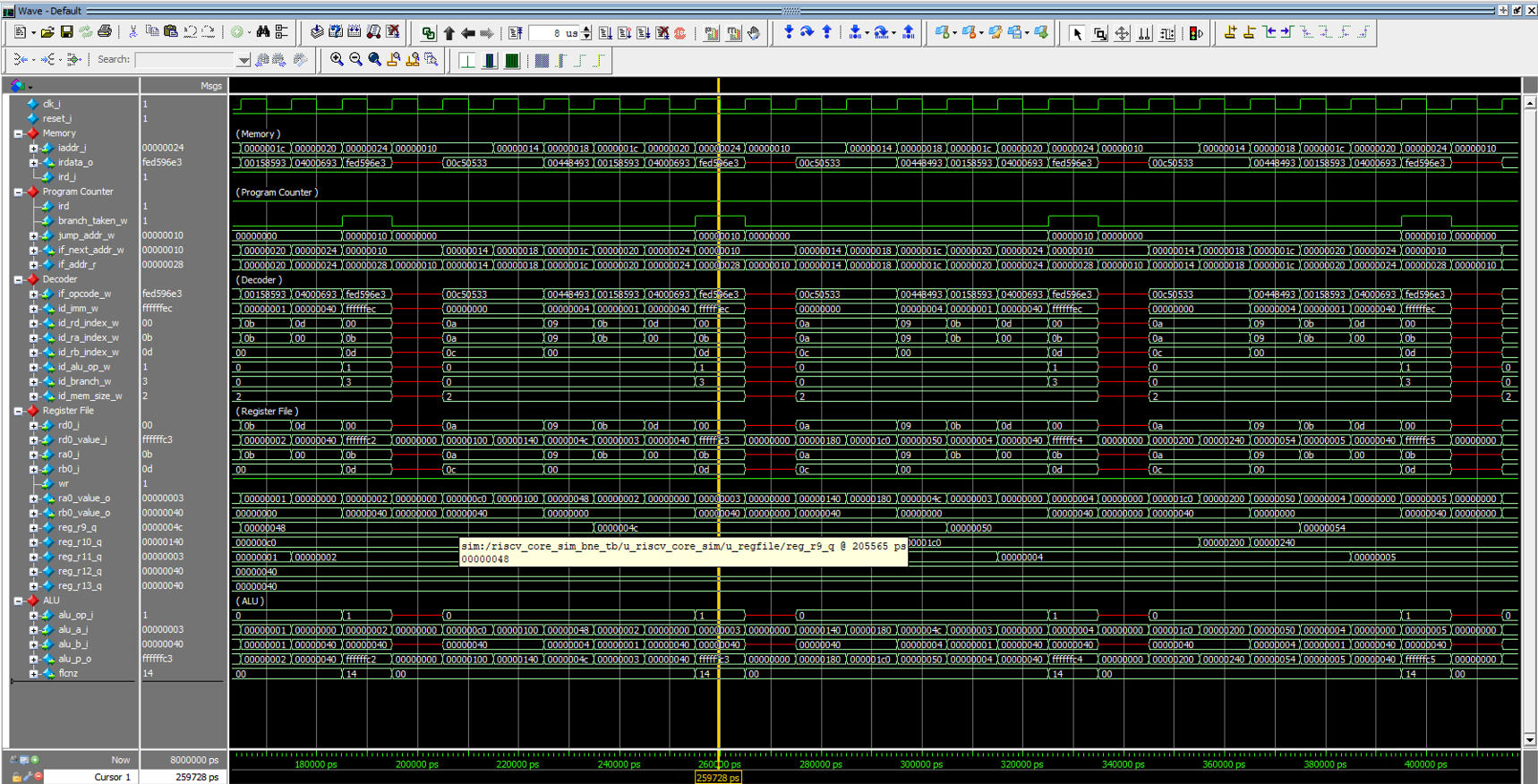
**Problem 3 (2p) Branch-Not-Equal(BNE) (Optional): Bonus**

The following table shows C codes and their assembly codes for a program that calculates an array’s sum of all elements. Two code versions use LESS-THAN and NOT-EQUAL operations for a conditional branch in a loop. In this problem, you have to do the following:

1. Generate a memory file for the modified assembly code using a NOT-EQUAL operation, called mem\_bne.hex.
2. Modify a test bench to use a new memory file (riscv\_core\_sim\_bne\_tb.v).
3. Make code to compute a target address and a branch enable signal for Branch-Not-Equal (BEQ).
4. Do a simulation and capture its waveform.

|  |  |  |
| --- | --- | --- |
| **C code** | **Assembly Code** | |
| int A[64];  int sum = 0;  for (int i=0; i< 64; i++)  sum += A[i]; | addi x9, x0, 64  add x10, x0, x0  add x11, x0, x0  Loop:  lw x12, 0(x9)  add x10, x10, x12  addi x9, x9, 4  addi x11, x11, 1  addi x13, x0, 64  blt x11, x13, Loop | # x9=&A[0]  # sum=0  # i=0  # x12=A[i]  # sum+= A[i]  # &A[i++]  # i++  # x13=64 |
| int A[64];  int sum = 0;  for (int i=0; i≠64; i++)     sum += A[i]; | addi x9, x0, 64  add x10, x0, x0  add x11, x0, x0  Loop:    lw x12, 0(x9)    add x10, x10, x12    addi x9, x9, 4    addi x11, x11, 1    addi x13, x0, 64    bne x11, x13, Loop | # x9=&A[0]  # sum=0  # i=0  # x12=A[i]  # sum+= A[i]  # &A[i++]  # i++  # x13=64 |

****

****