**HW#04: Load, Store Instructions, RICS-V Core**

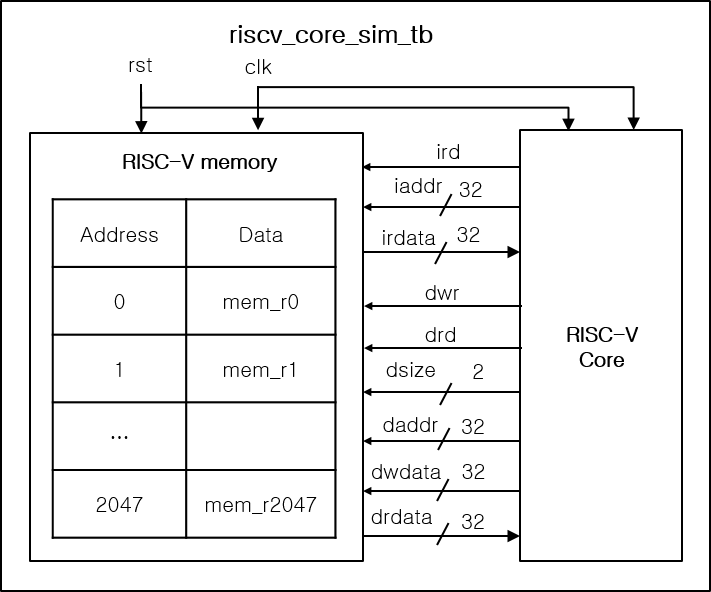
**Issued:** April 04 (Tuesday), 2023 **Due:** April 10 (Monday), 2023

**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 (10p): Load, Store**

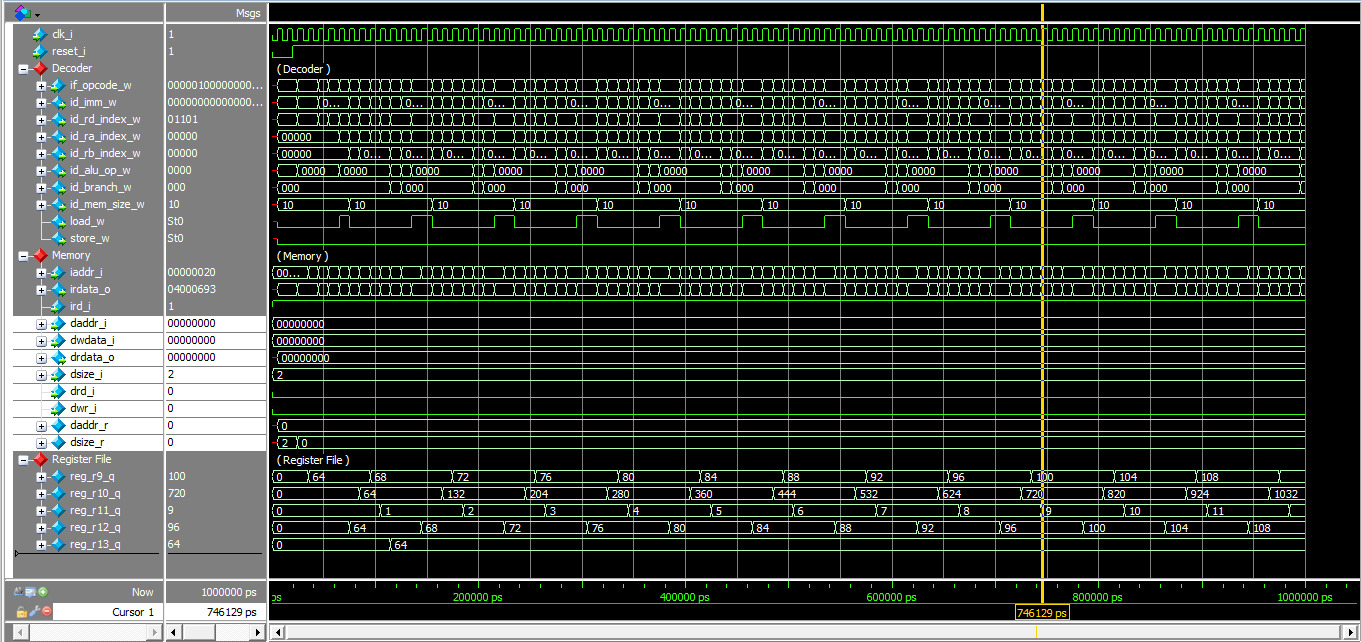
Implement Load, Store instructions of a RISC-V core in Verilog. Please see the description in the lecture note for details.



What you have to do:

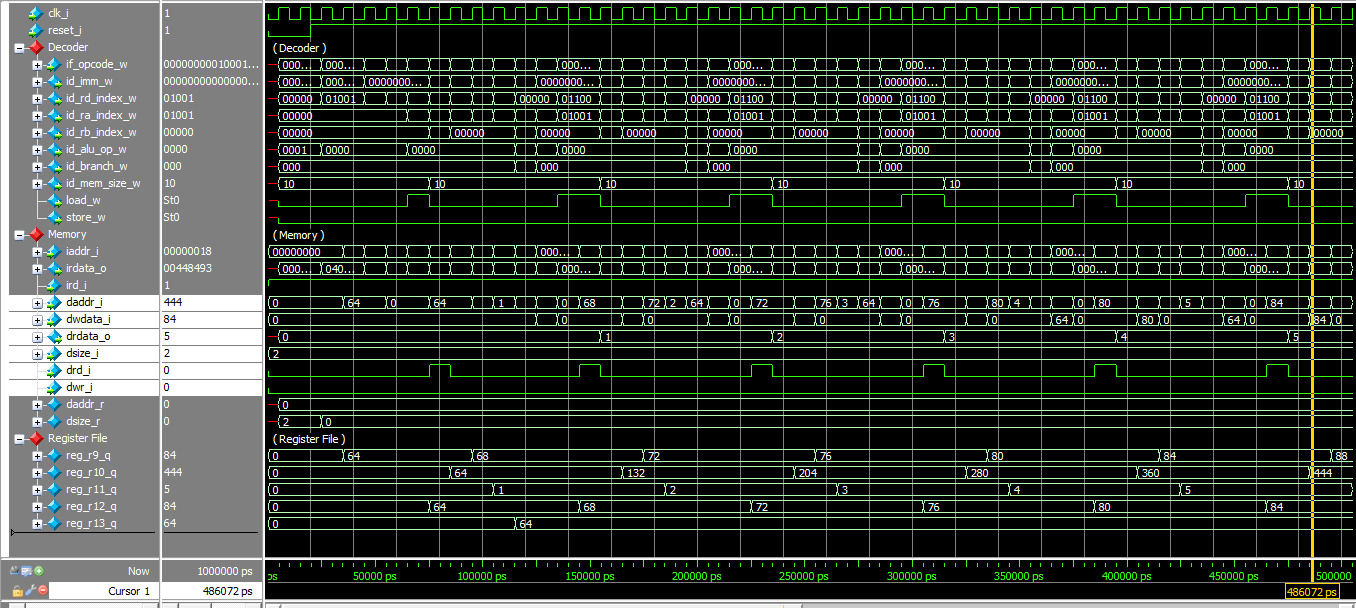
1. Baseline Code (5p)

* Complete Program Counter in Assignment 4.
* Uncomment dummy codes to complete Connections for ALU, Decoder and Register File modules.
* Complete the code for Branch-Less-Than as Assignment 4.
* Do a simulation and capture the waveform.

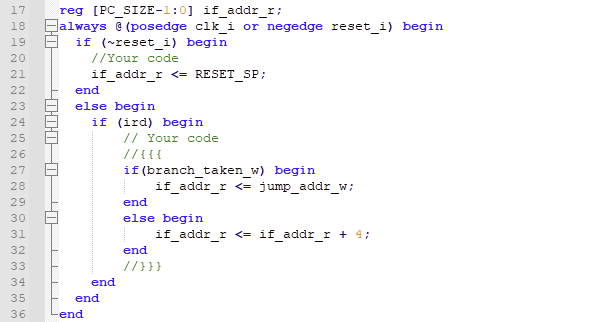


1. Load, Store instructions (5p)

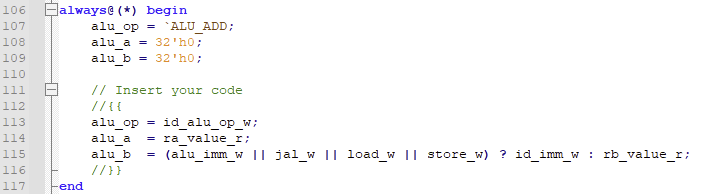
* Replace the dummy code of load, store instructions by your code.
* Do a simulation and capture the waveform.



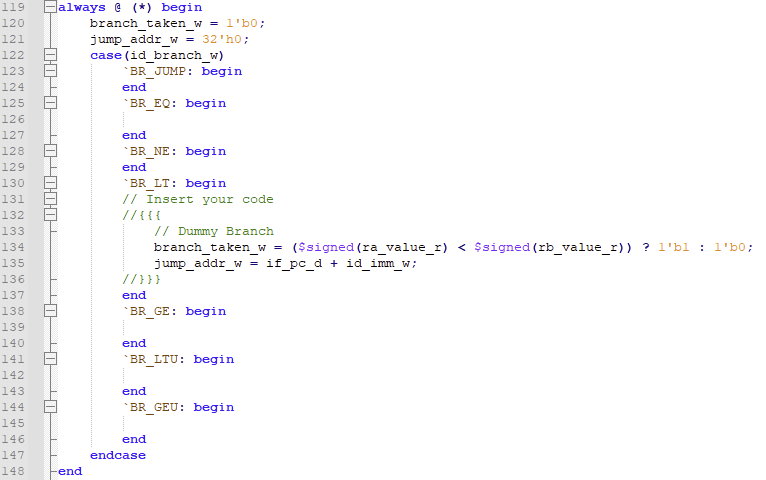
Solution :



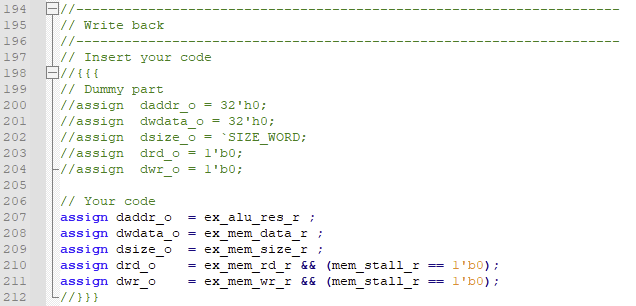
Complete Program Counter



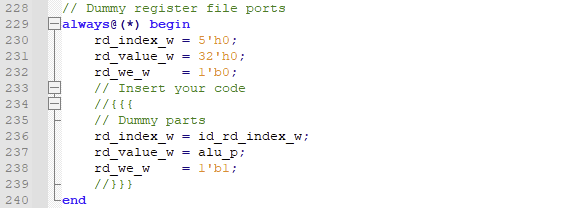
Complete ALU



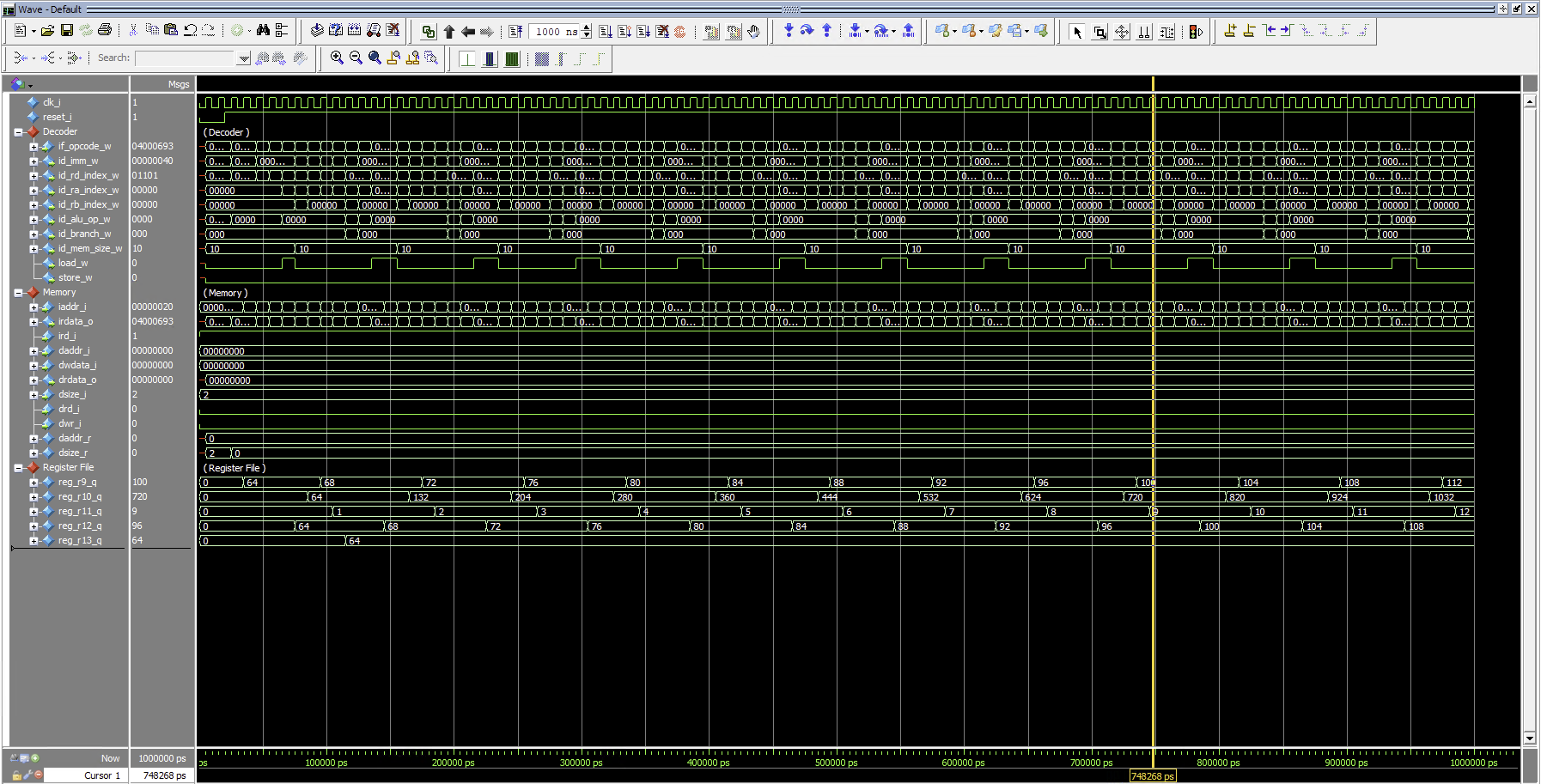
Complete BLT instruction



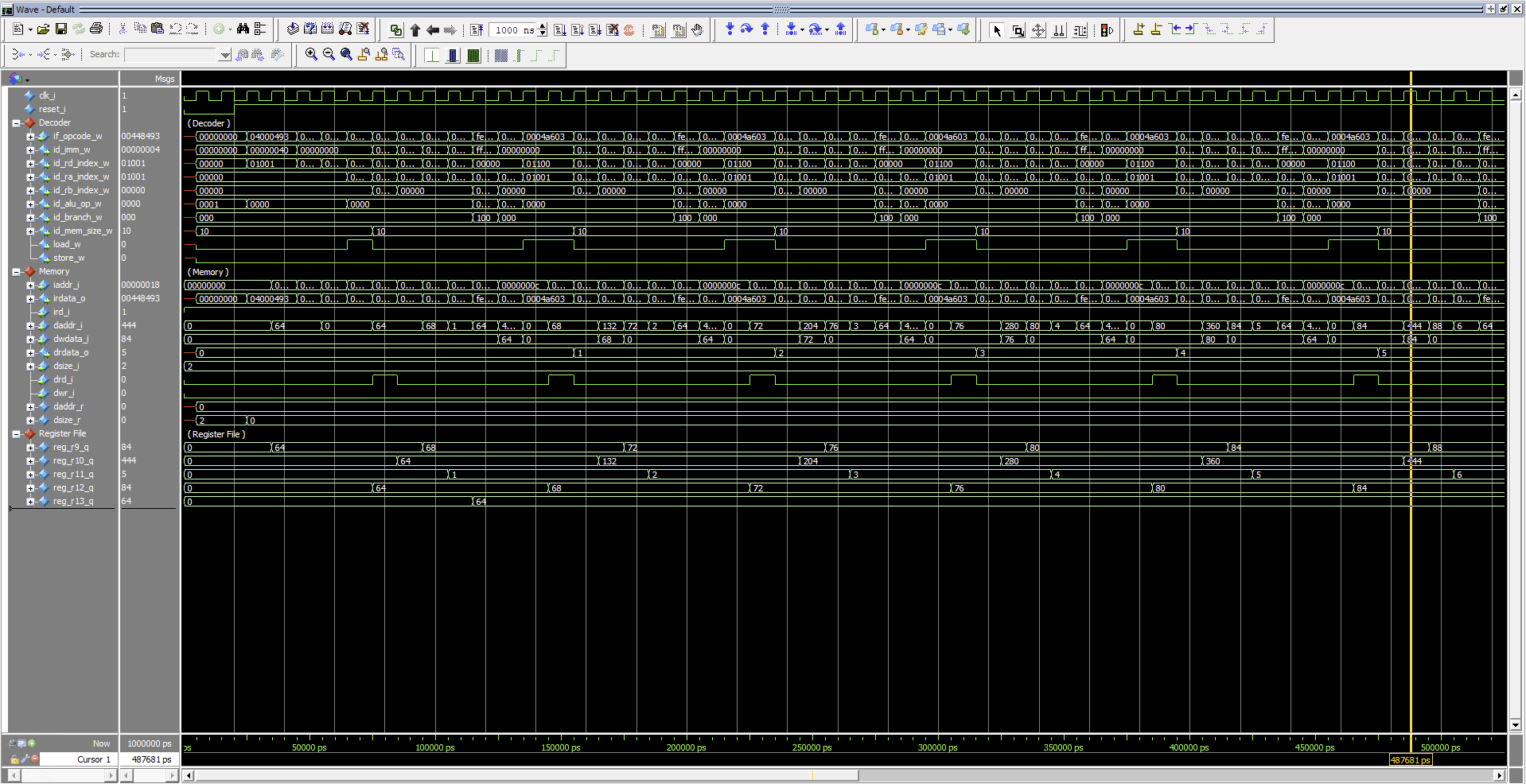
Complete Memory Write Back



Complete Register File



Baseline Code WaveForm



WaveForm of Memory Write Back implemented Verilog Code

**Problem 2 (15p): RISC-V**

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

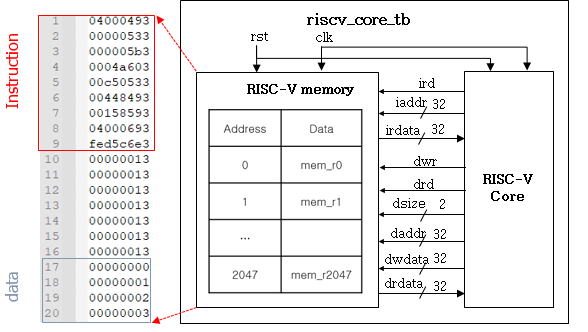
**C code:**

int A[64];

int sum = 0;

for (int i=0; i<64; i++)

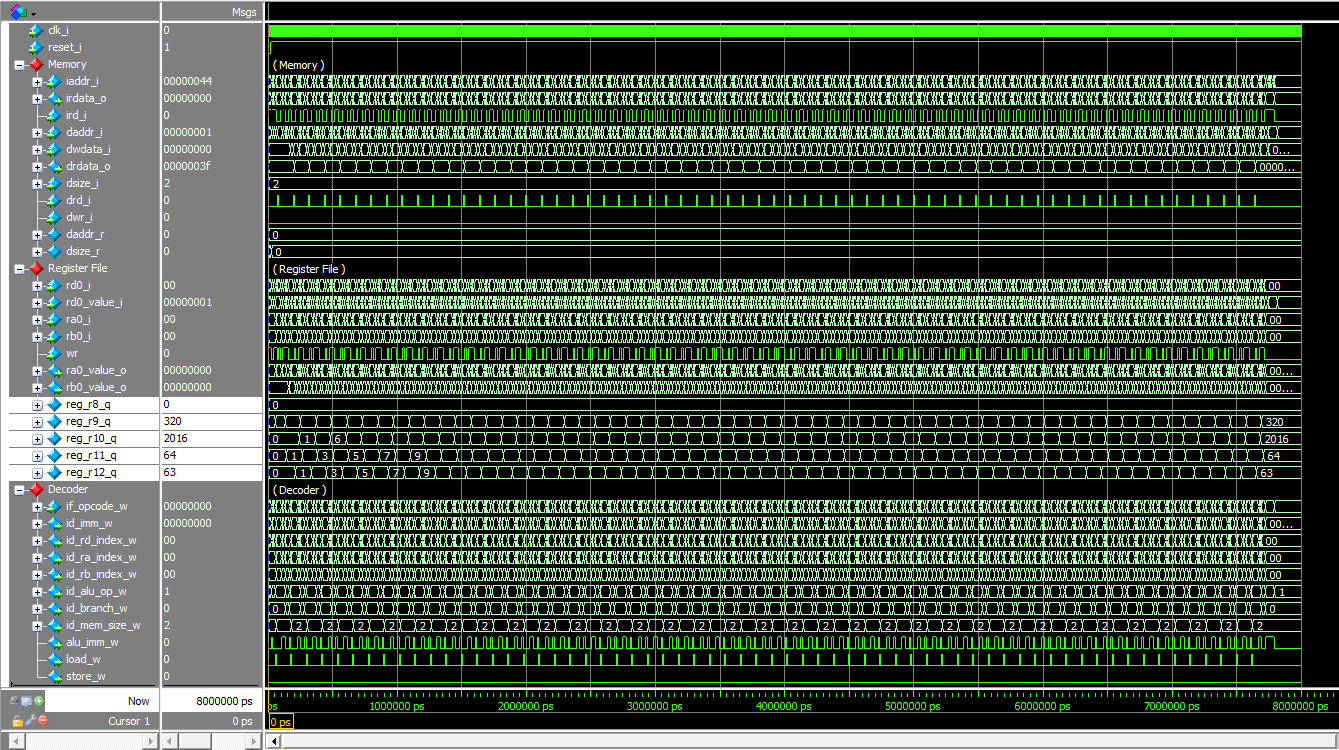
sum += A[i];



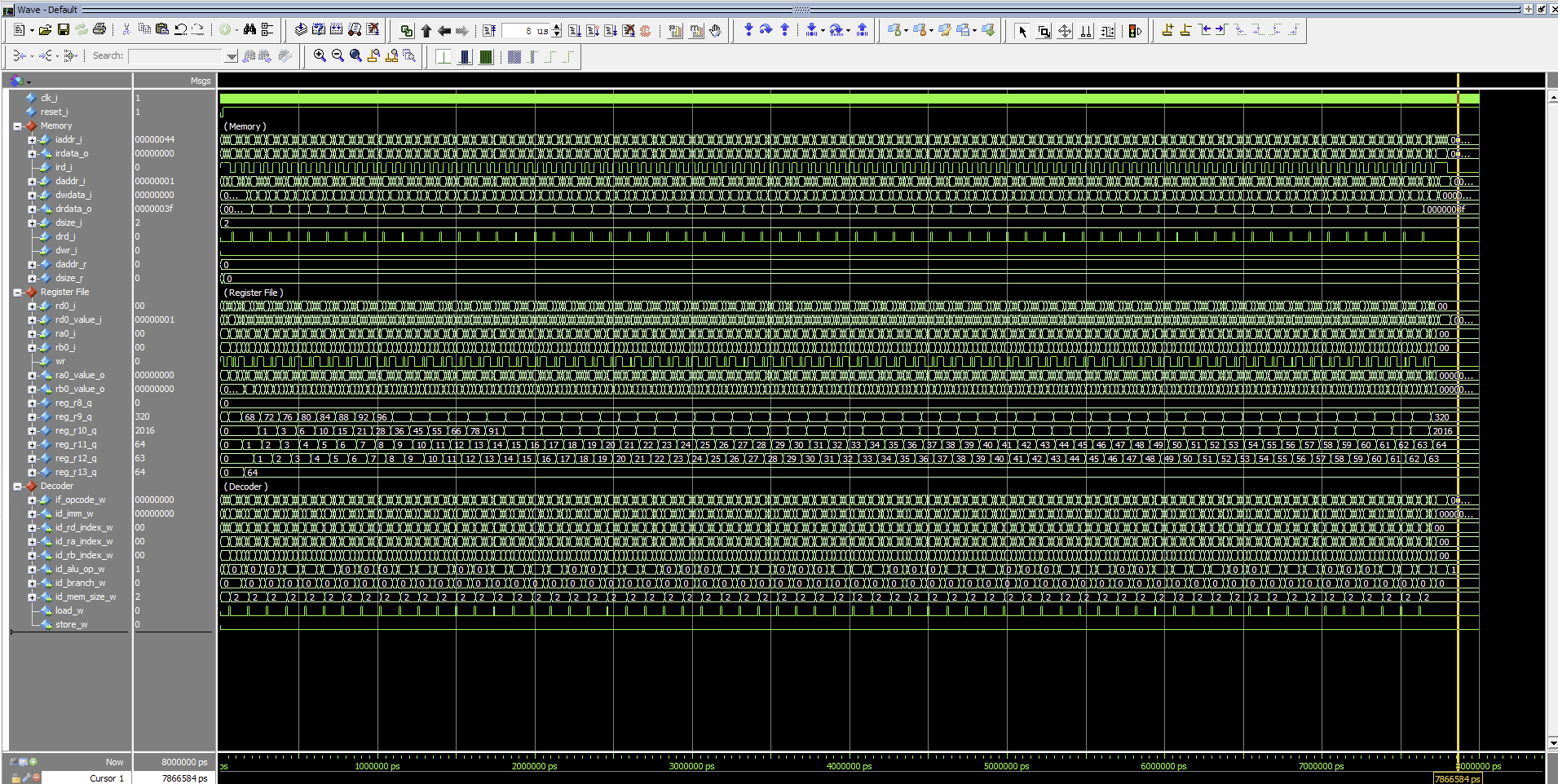
What you have to do:

1. **Baseline core (5p)**

* Do a simulation and capture the waveform. (3p)
* Explain why the final values of r9 and r10 are 320 and 2016, respectively. (2p)



Solution (a) :



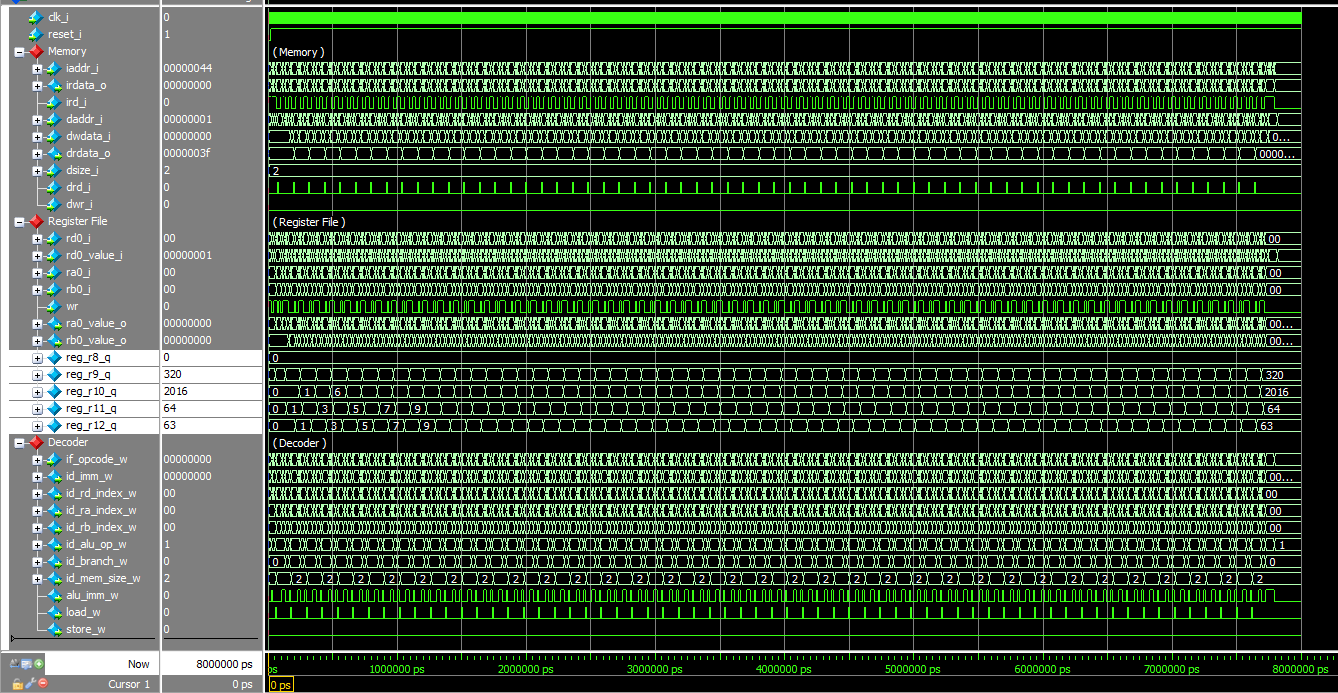
Baseline WaveForm

Explanation of r9 and r10 have 320 and 2016, respectively

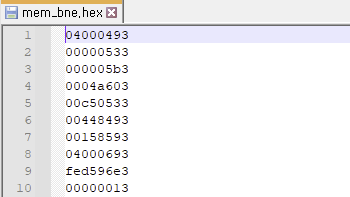
* + r9 : r9 holds the address of array element, which increase from &A[0] to &A[63]. Since the &A[0] starts from the address 64, the address of &A[63] should be 80.
  + r10 : r10 holds the summation of total elements. Since the program is written to sum all the integers in array, which is from 0 to 63, the result should be 2016.

1. **Branch-Not-Equal (BNE) (5p)**
2. Generate a memory file for the modified assembly code using a NOT-EQUAL operation, called mem\_bne.hex.
3. Modify a test bench to use a new memory file (riscv\_core\_bne\_tb.v).
4. Make code to calculate a branch enable signal for Branch-Not-Equal (BNE).
5. Do simulation and capture its result, i.e., 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 |



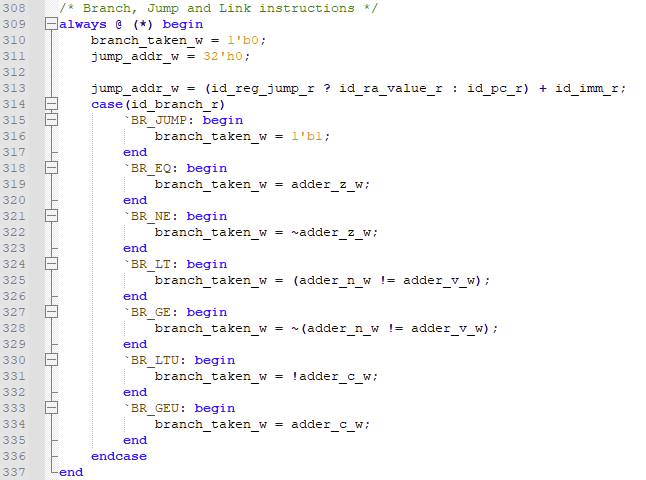
Solution (b) :



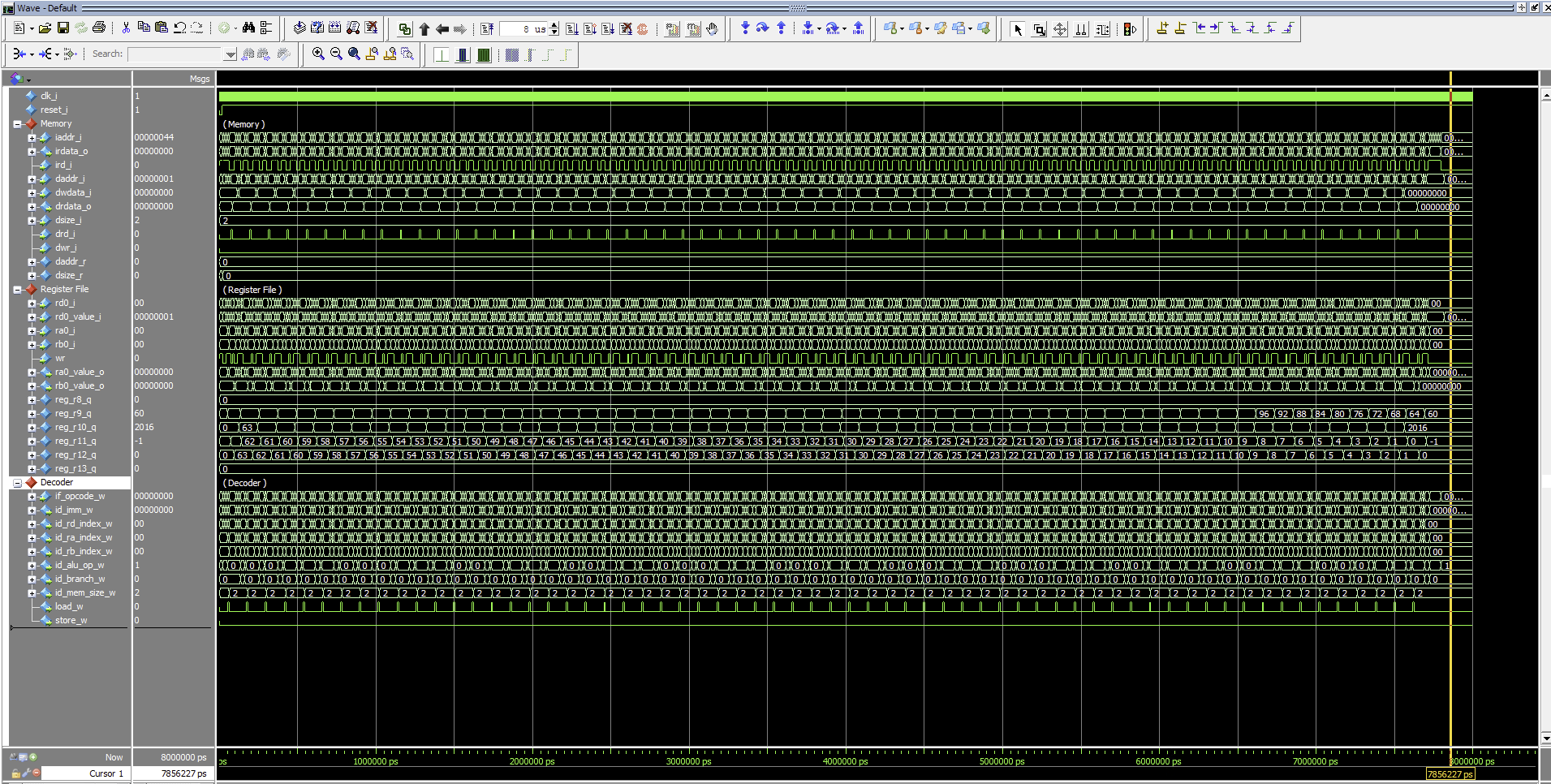
Memory file of modified assembly code(BNE)



Modified Test Bench



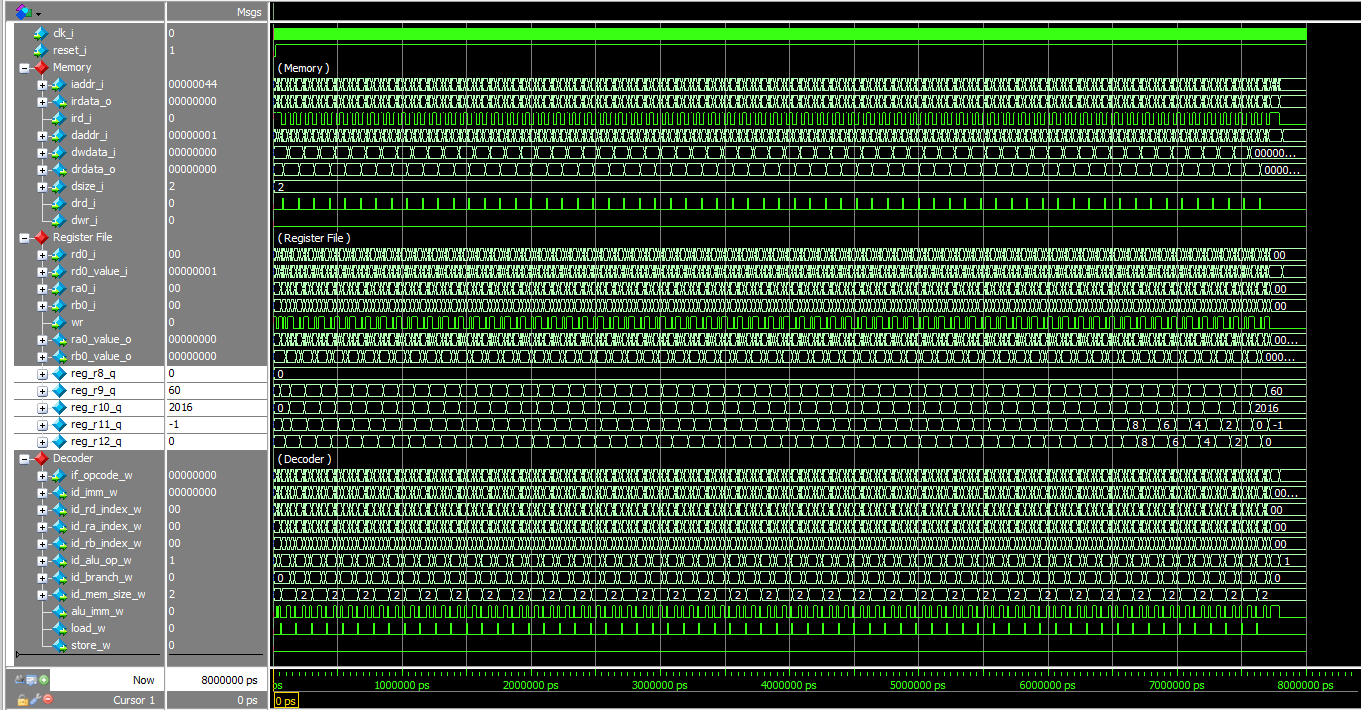
Modified RISC-V core source code



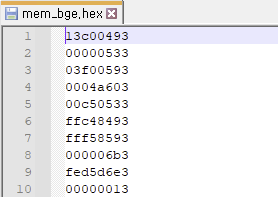
WaveForm

1. **Branch-Greater-or-Equal (BGE) (5p)**
2. Generate a memory file for the modified assembly code using a Greater-or-Equal operation, called mem\_bge.hex.
3. Modify a test bench to use a new memory file (riscv\_core\_bge\_tb.v).
4. Make code to calculate a branch enable signal for Branch-Greater-or-Equal (BGE).
5. Do simulation and capture its result, i.e., 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=63; i>=0; i--)  sum += A[i]; | addi x9, x0, 316  add x10, x0, x0  addi x11, x0, 63  Loop:  lw x12, 0(x9)  add x10, x10, x12  addi x9, x9, -4  addi x11, x11, -1  add x13, x0, x0  bge x11, x13, Loop | # x9=&A[0]  # sum=0  # i=63  # x12=A[i]  # sum+= A[i]  # &A[i++]  # i++  # x13=0 |



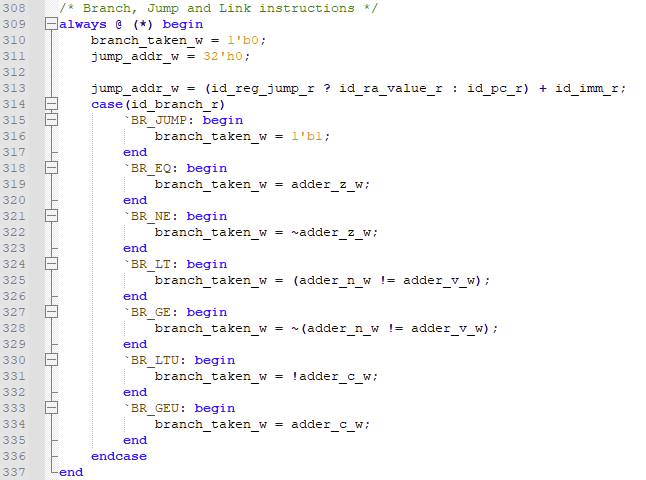
Solution (c) :



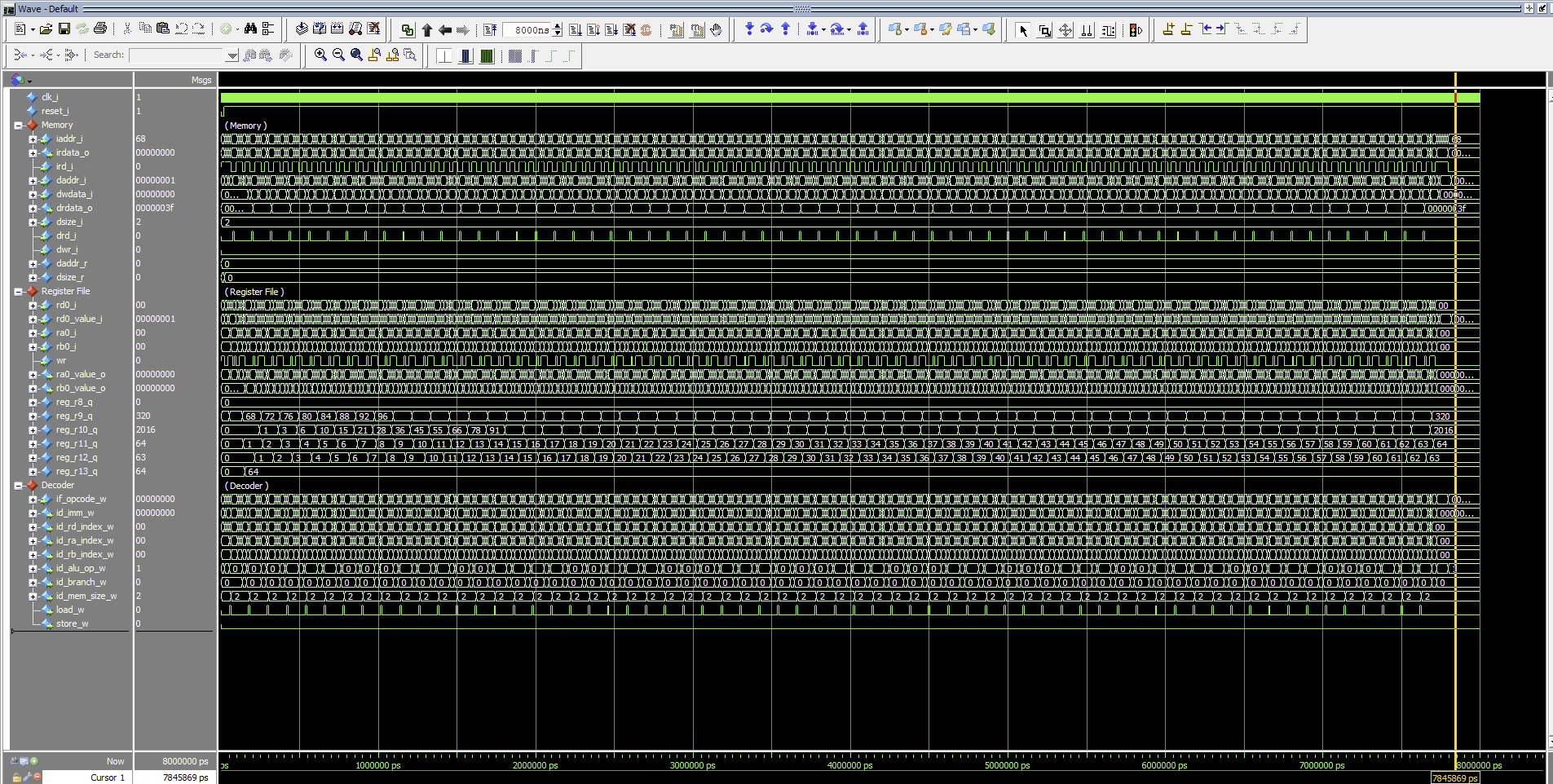
Modified Memory file(BGE)



Modified Test Bench Code



Modified RISC-V core source code



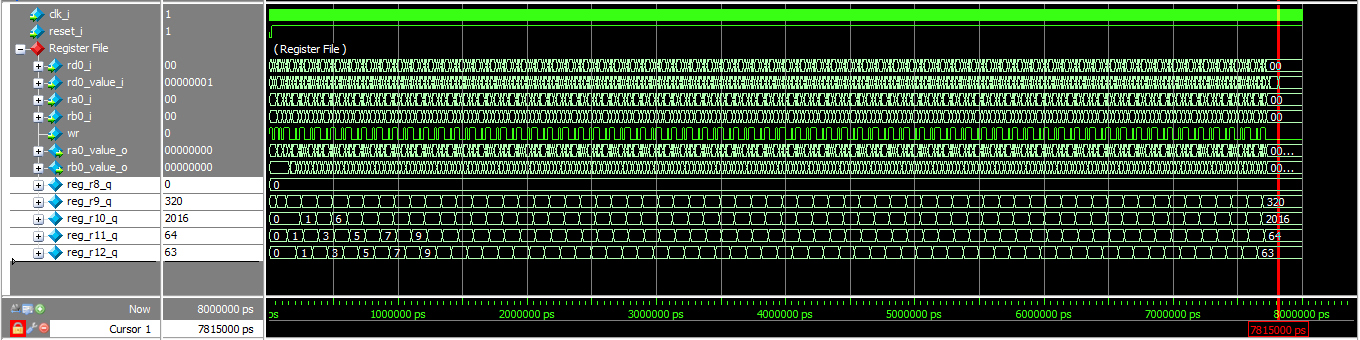
WaveForm

1. **Instruction Reordering (5p)**

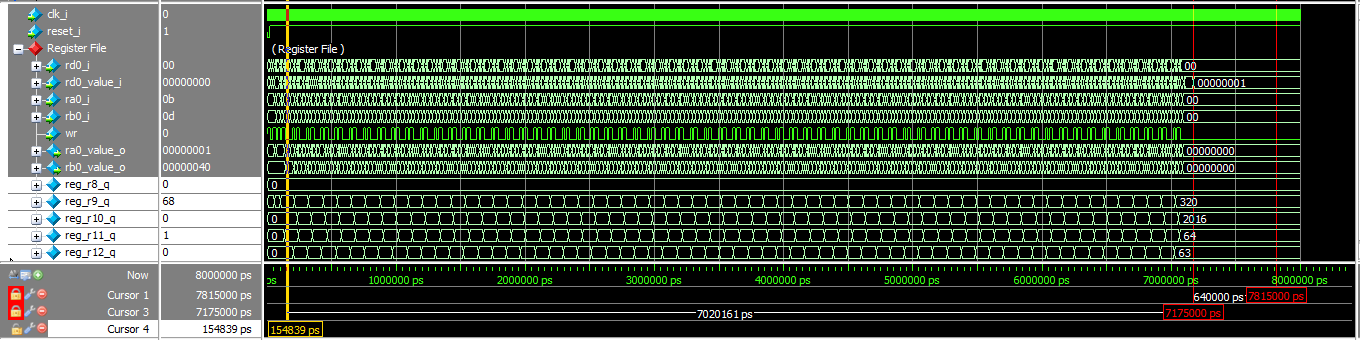
There are three versions of assembly codes as shown in Table.

|  |  |  |  |
| --- | --- | --- | --- |
| **C code** | **Assembly Code** | | |
| **Baseline** | **Optimized 1 (Opt1)** | **Optimized 2 (Opt2)** |
| 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 | addi x9, x0, 64  add x10, x0, x0  add x11, x0, x0  Loop:  lw x12, 0(x9)  addi x9, x9, 4  add x10, x10, x12  addi x11, x11, 1  addi x13, x0, 64  blt x11, x13, Loop | addi x9, x0, 64  add x10, x0, x0  add x11, x0, x0  Loop:  lw x12, 0(x9)  addi x9, x9, 4  addi x11, x11, 1  add x10, x10, x12  addi x13, x0, 64  blt x11, x13, Loop |

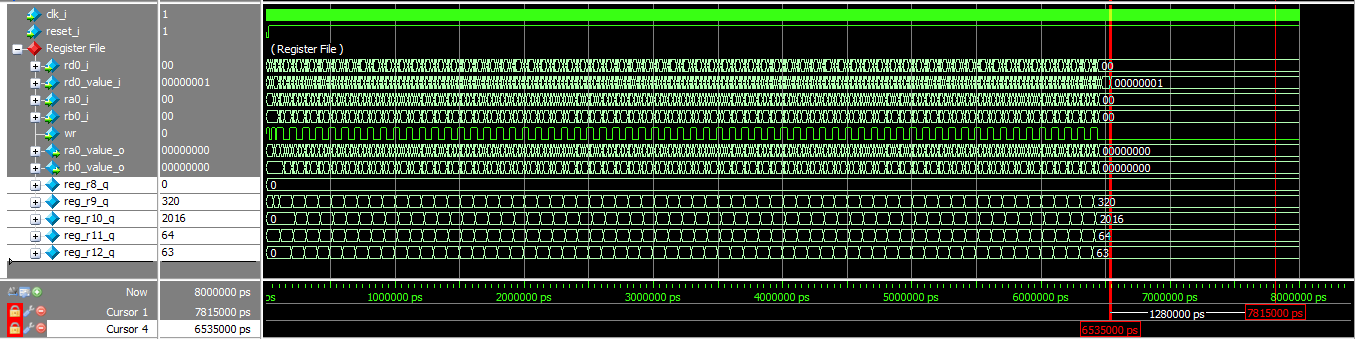
1. Create memory files (mem\_opt1.hex and mem\_opt2.hex).
2. Modify a test bench with new memory files (riscv\_core\_tb.v)
3. Do simulation and captures the waveforms.
4. Compare the running times of three versions: baseline, opt1 and opt2. Explain the experimental results.



Waveform of the baseline version (mem.hex)

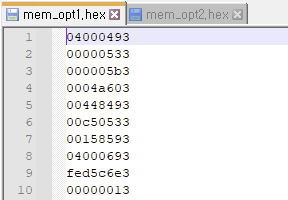


Waveform of the optimized version 1 (mem\_opt1.hex)

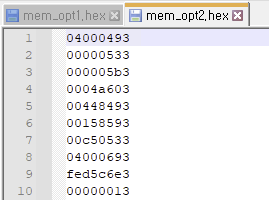


Waveform of the optimized version 2 (mem\_opt2.hex)

Solution (d) :



Modified Memory File ( Opt 1 )



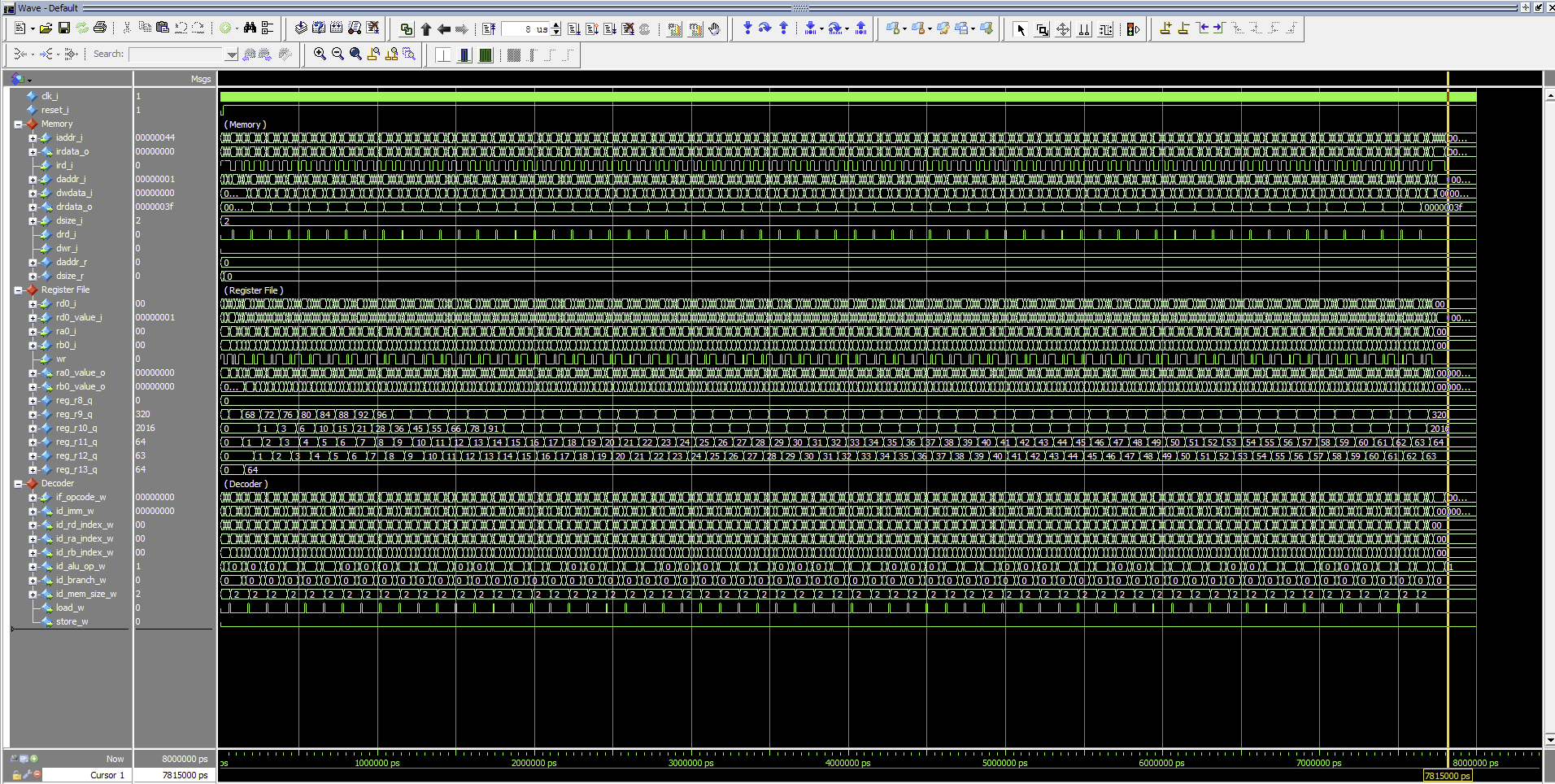
Modified Memory File ( Opt 2 )



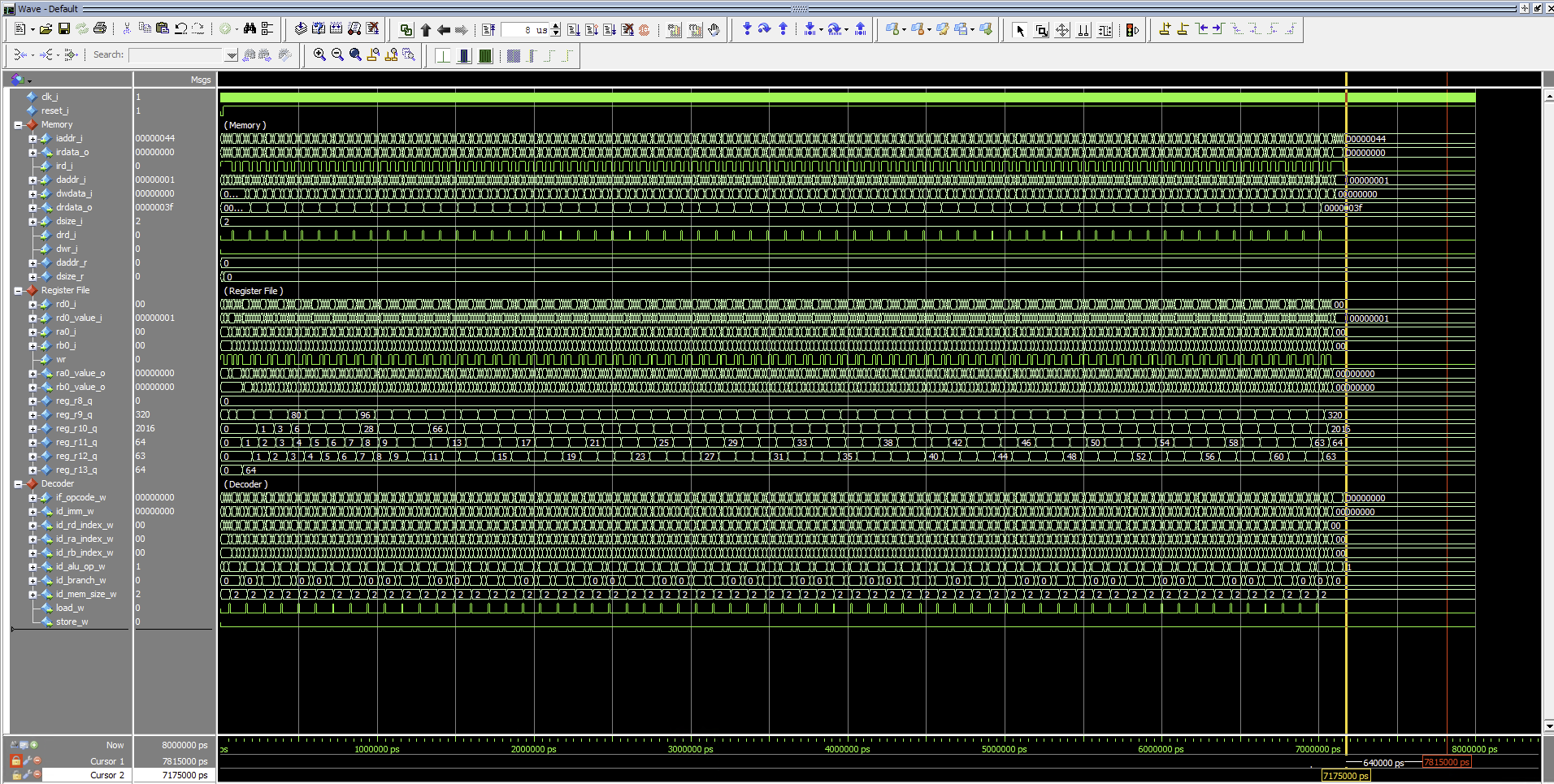
Modified TestBench ( Opt 1 )



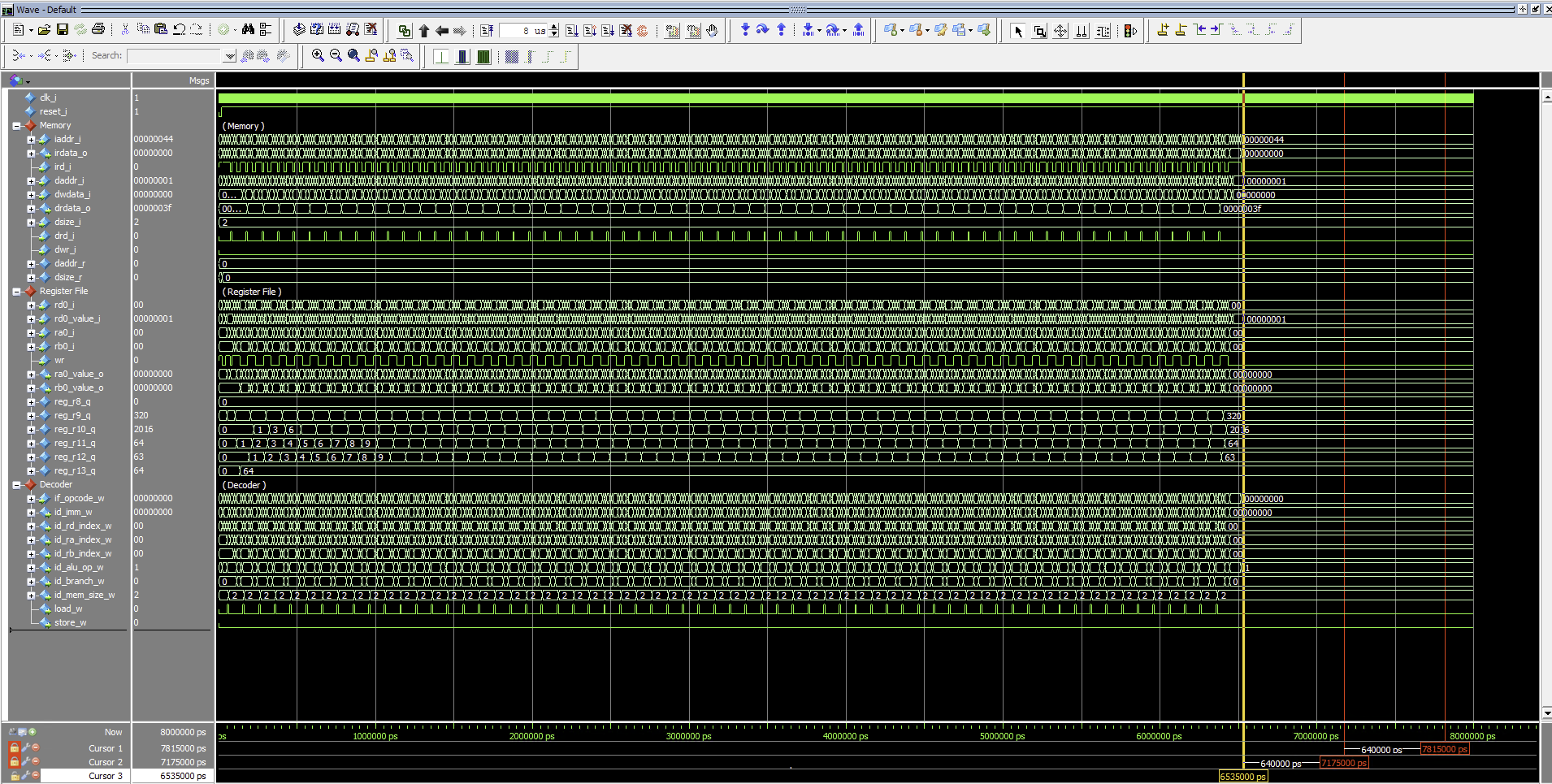
Modified TestBench ( Opt 2 )



Baseline WaveForm



Opt1 WaveForm



Opt2 WaveForm

Result of experiments : The running time of each memory file decreased in order of baseline(7815000ps), opt1(7175000ps), opt2(6535000). Each’s running time is differ to roughly 640000 ps.

Discussion about the result : By re-ordering the instructions, we can decrease the data hazard by data forwarding. Since some of instructions use the result of previous instructions, so we do not have to load the data from memory again. We can reduce this sort of redundancy by re-ordering the instructions like above. So, the opt 1 and opt 2 eventually decrease the data hazard and running time by data forwarding which can be performed with instruction re-ordering.