Dario Marasco

Zachary Bryceland

August 26th, 2015

ECEC-355

MIPS Processor Report

Design Principles:

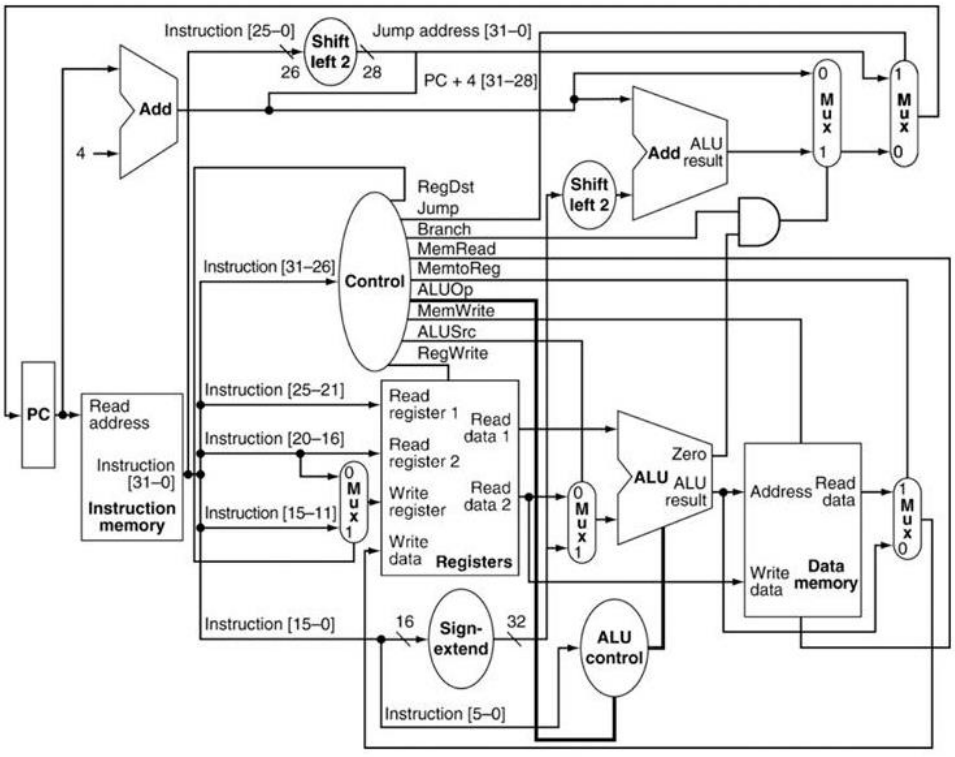

Figure 1, Single-cycle MIPS Processor

Our processor design is based upon that shown in Fig. 1. Each visible component is implemented as a VHDL component. Because there are so few components, generics are only utilized for the mux. Each ALU and left-shifter has its own unique code. The data memory, program counter, and registers are the only components whose processes are driven by the clock. All other components' processes are driven by signals output from other components.

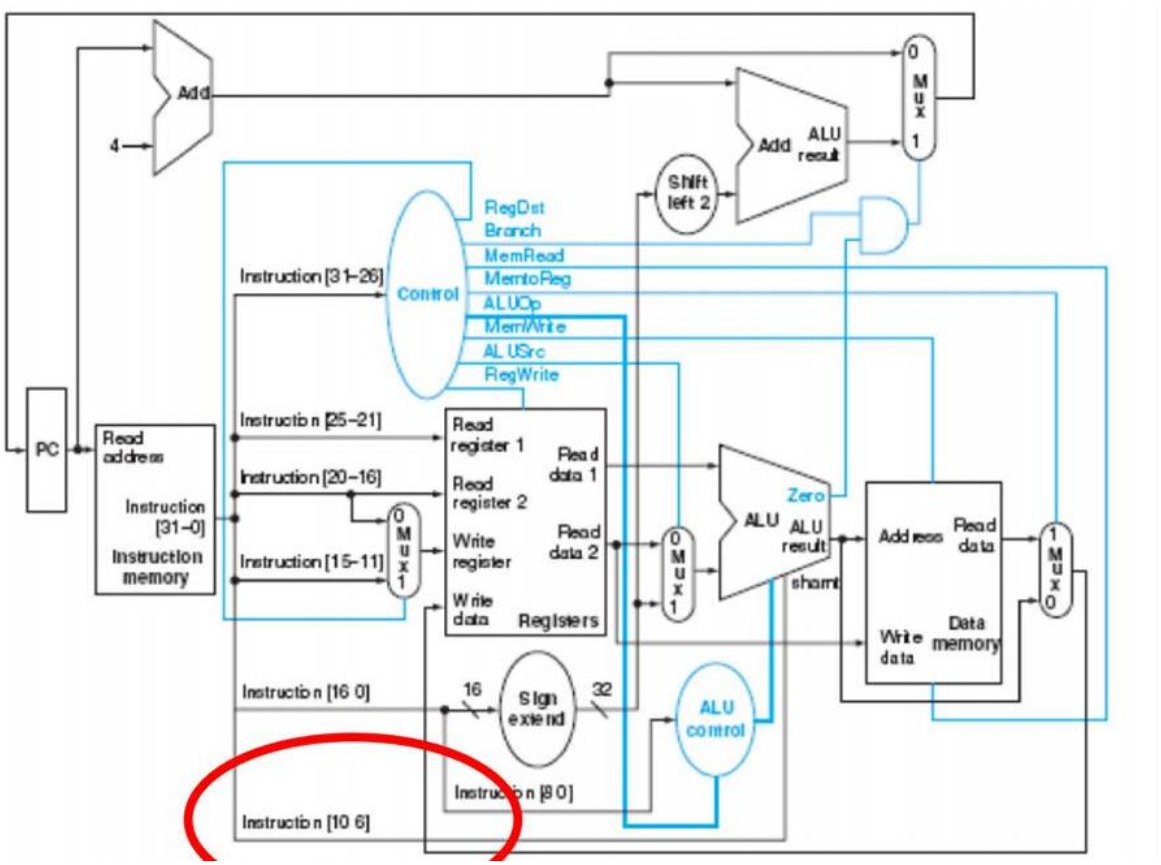

Figure 2, Single-cycle MIPS processor with sll implemented.

Modifications to the design were needed to implement *bne* and *sll*. To implement *bne,* an additional control signal (branch_s) is sent to the branching and-gate. If this signal is high, *beq* is used. If this signal is low, *bne* is used. To implement *sll*, instructions [10-6] are sent to the main ALU. An additional operation was added to the ALU to shift by *shamt*. This implementation can be seen in Fig. 2.

Challenges Faced:

Connecting the components was the most difficult part of the processor design. Once the components were connected, it became more difficult to debug. For complex issues, unit testing was used on each .vhd file to determine which component(s) was responsible.

Before implementing *sll*, incrementing i within the while-loop presented an issue. Because i must be multiplied by four to be used for addressing, *sll* seems like the natural solution. Without an implementation, we first thought to add i to the base address four times. This was inelegant and consumed too much of the instruction memory. Instead, we elected to use a human optimization, incrementing i by four on each iteration of the loop.

Part C MIPS Code:

```
add $s0, $zero, $zero ; A = 0x0
addi $s1, $zero, 48 ; B = 0x30
addi $t0, $zero, 10 ; x = 10;
addi $t1, $zero, 20; y = 20;
addi $t2, $zero, $zero ; i = 0;
addi $t3, $zero, 1 ; store 1 as a comparator and operand
addi $t4, $zero, 3; store 3 as operand

LOOP: ; while (y >= x)
slt $t5, $t1, $t0
beq $t3, $t5, END

add $t6, $s0, $t2 ; A[i]
lw $t6, 0($t6)

add $t6, $t6, $t0 ; A[i] + x
add $t6, $t6, $t1 ; (A[i] + x) + y
add $t7, $s1, $t2 ; B[i]
sw $t6, 0($t7) ; B[i] = A[i] + x + y
sub $t0, $t0, $t3 ; x -= 1
sub $t1, $t1, $t4 ; y -= 3
addi $t2, $t2, 4 ; i += 1;
j LOOP ; re-evaluate
END:
```

Figure 3, MIPS code for the given C program. Plain text available in Appendix A.

Waveform Screenshots:

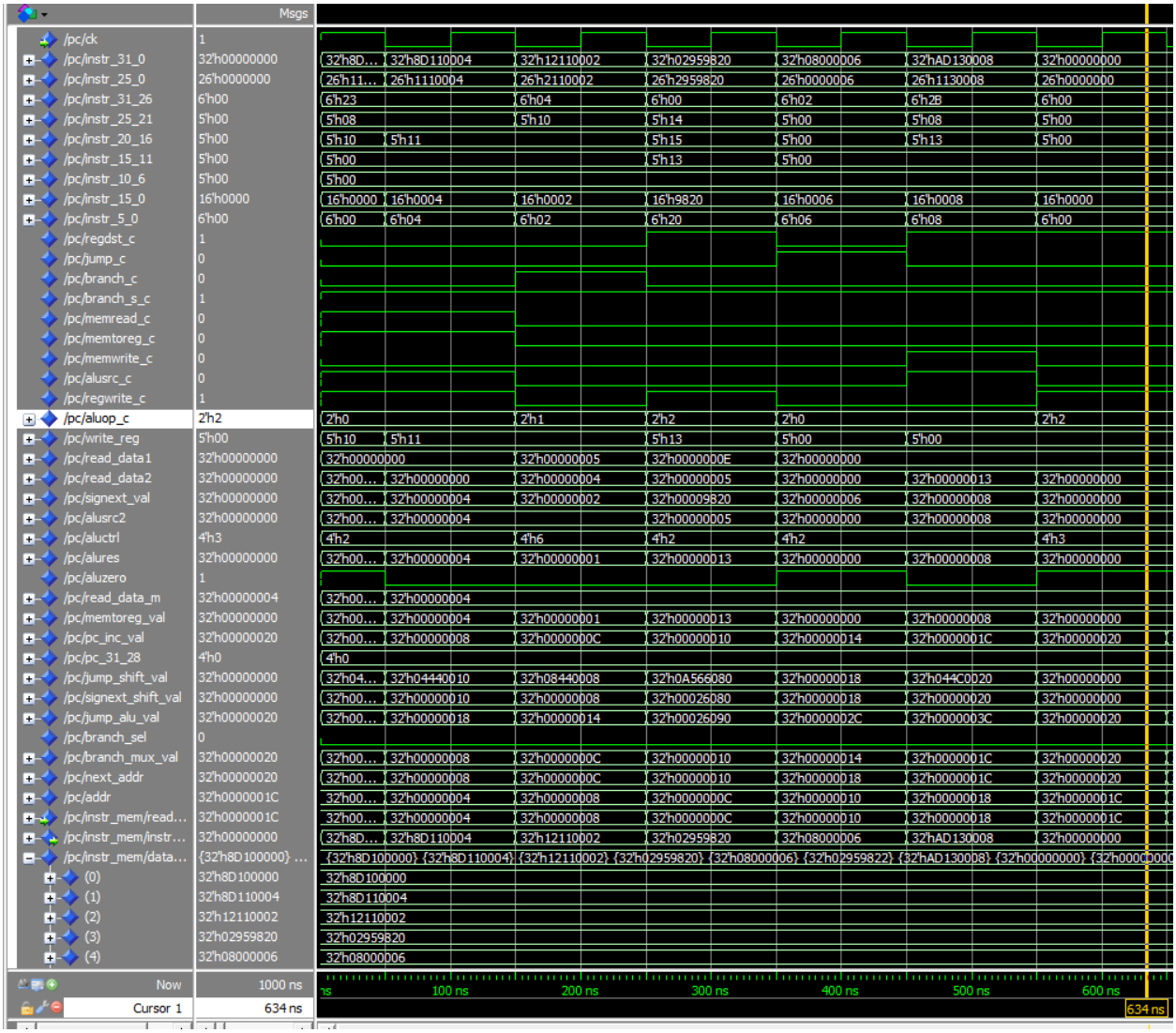

Figure 4, Screen capture of waveforms for part A.

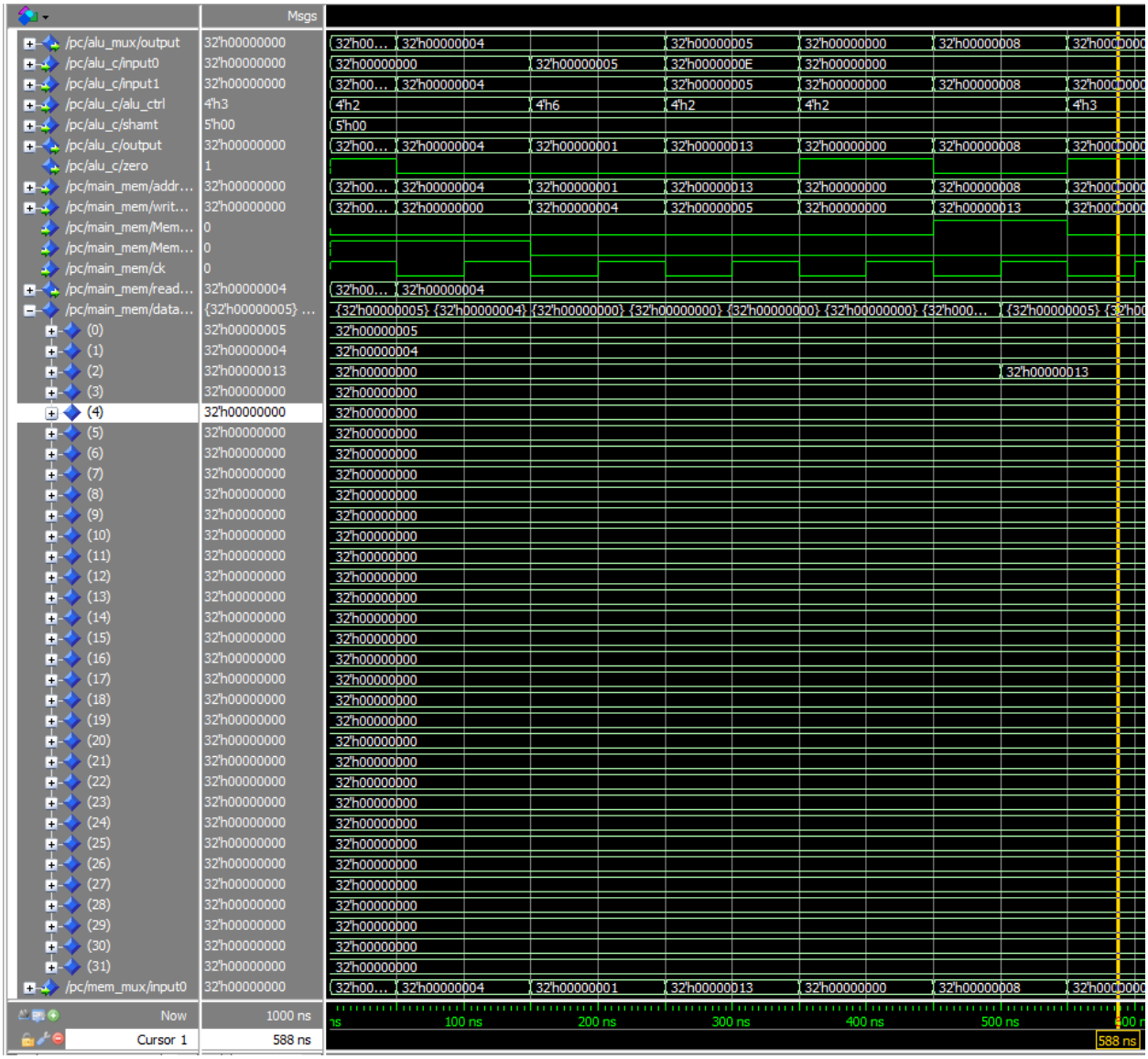

Figure 5, Screen captures of waveforms for memory in part A.

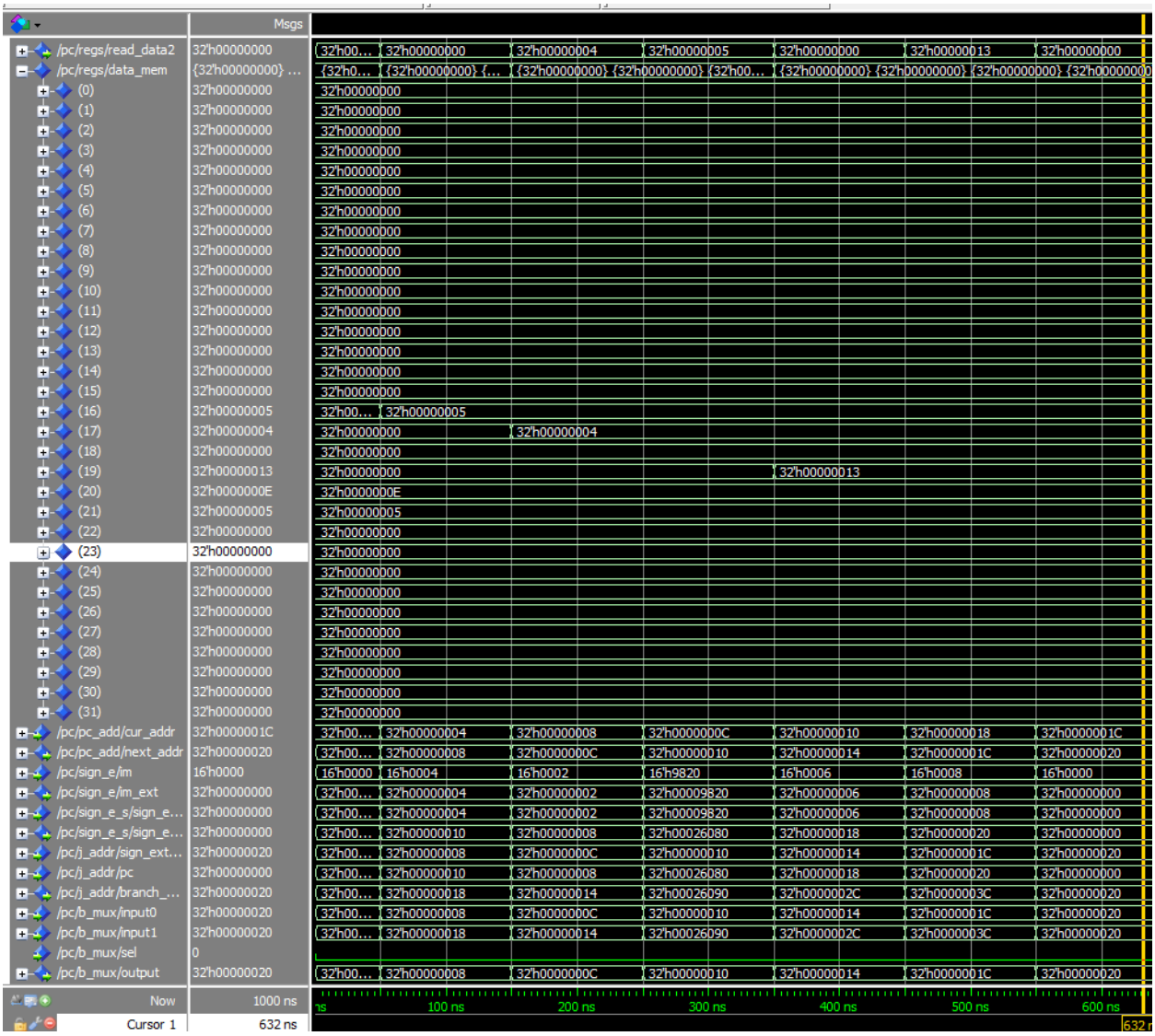

Figure 6, Screen captures for waveforms of registers in part A.

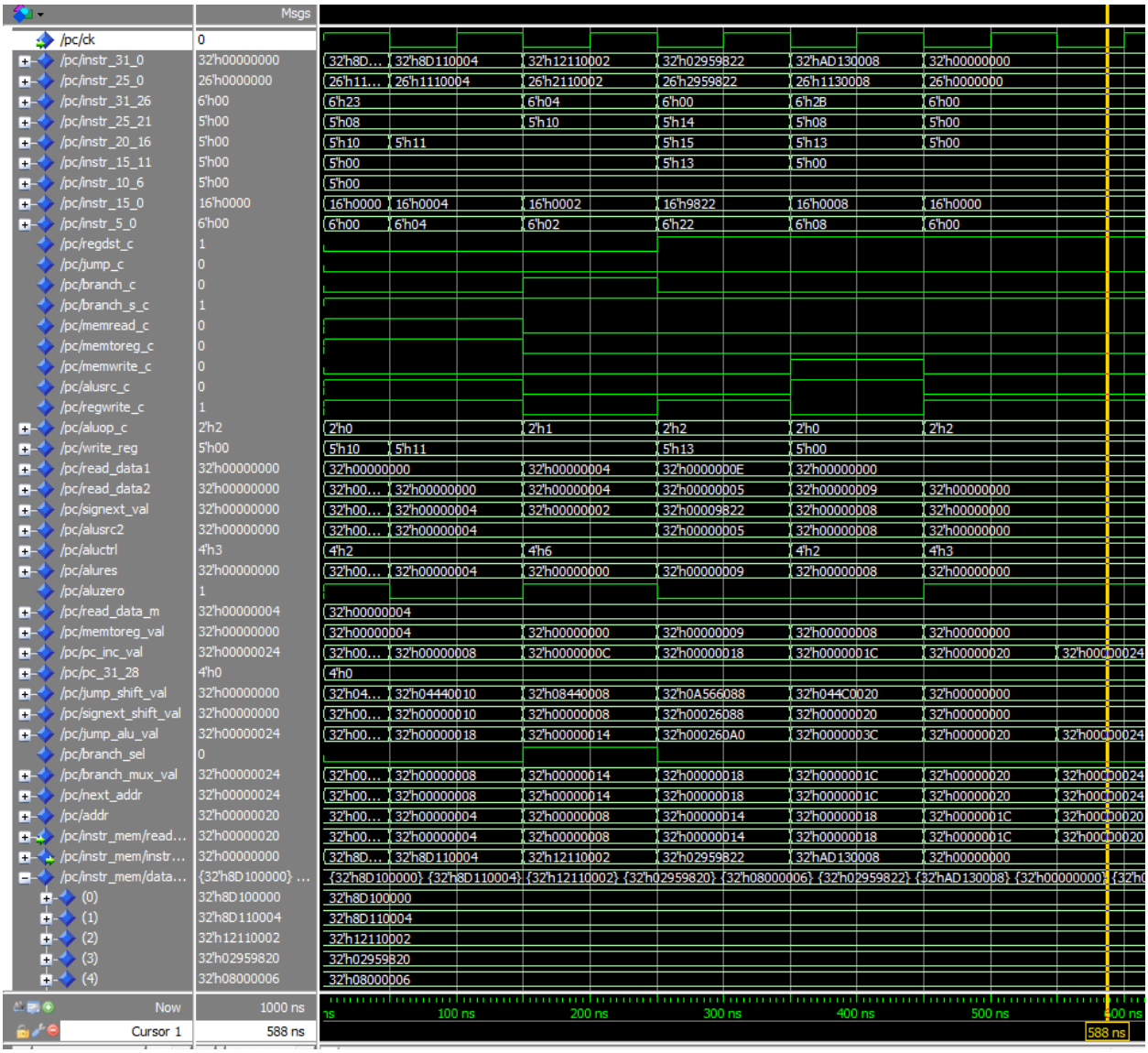

Figure 7, Screen captures for waveforms in part B.

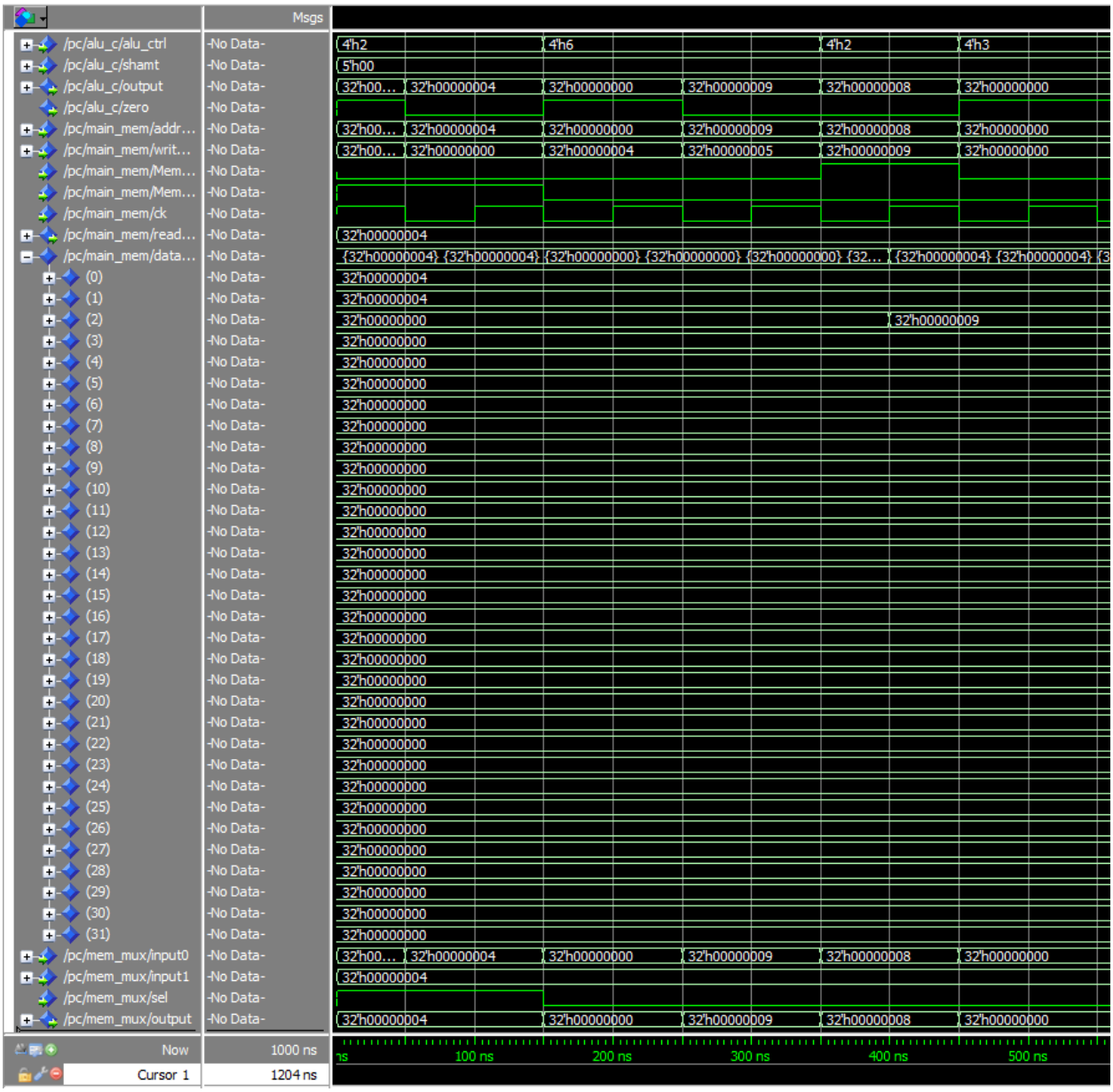

Figure 8, Screen capture for waveforms of memory in part B.

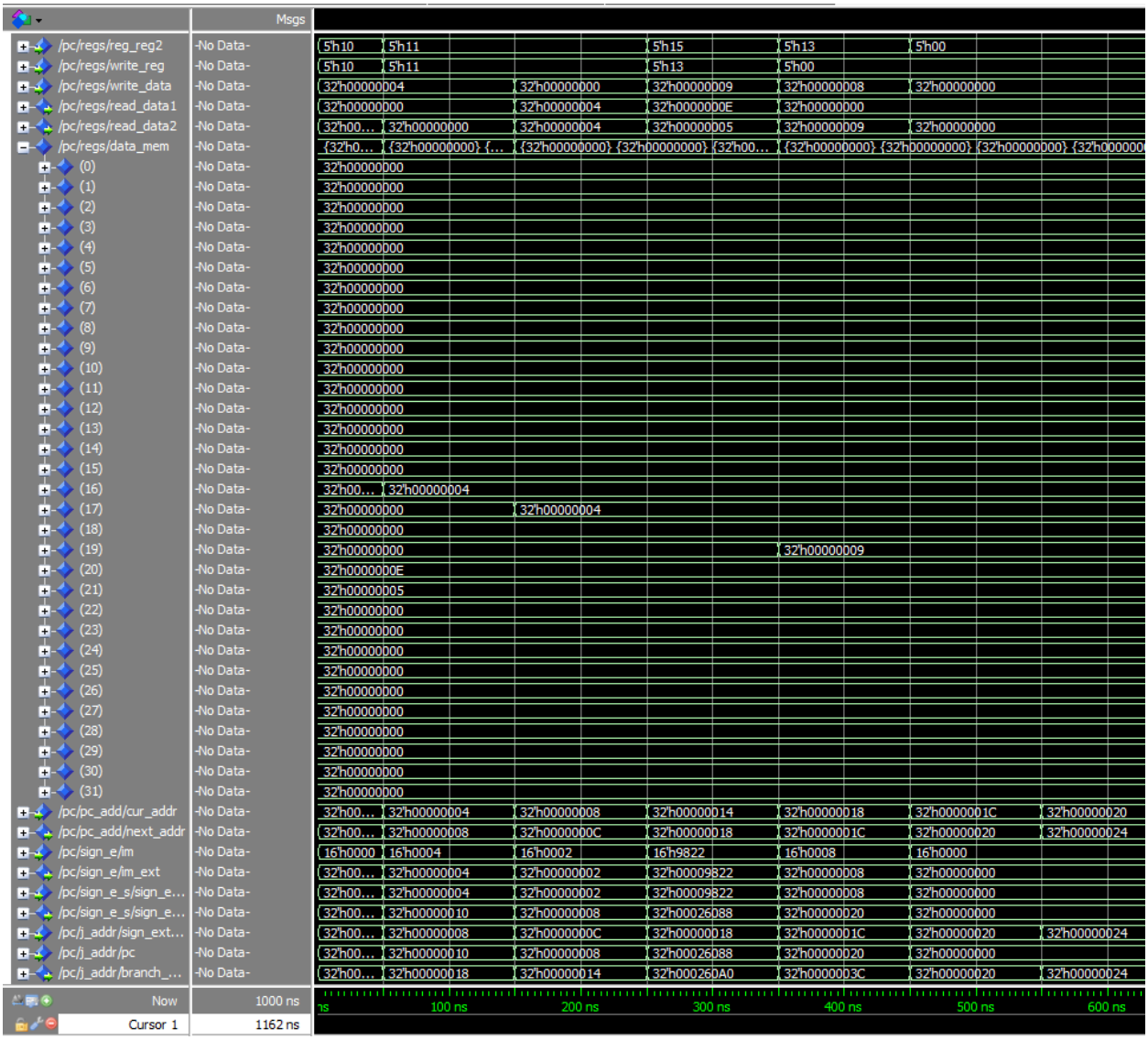

Figure 9, Screen capture of waveforms of registers in part B.

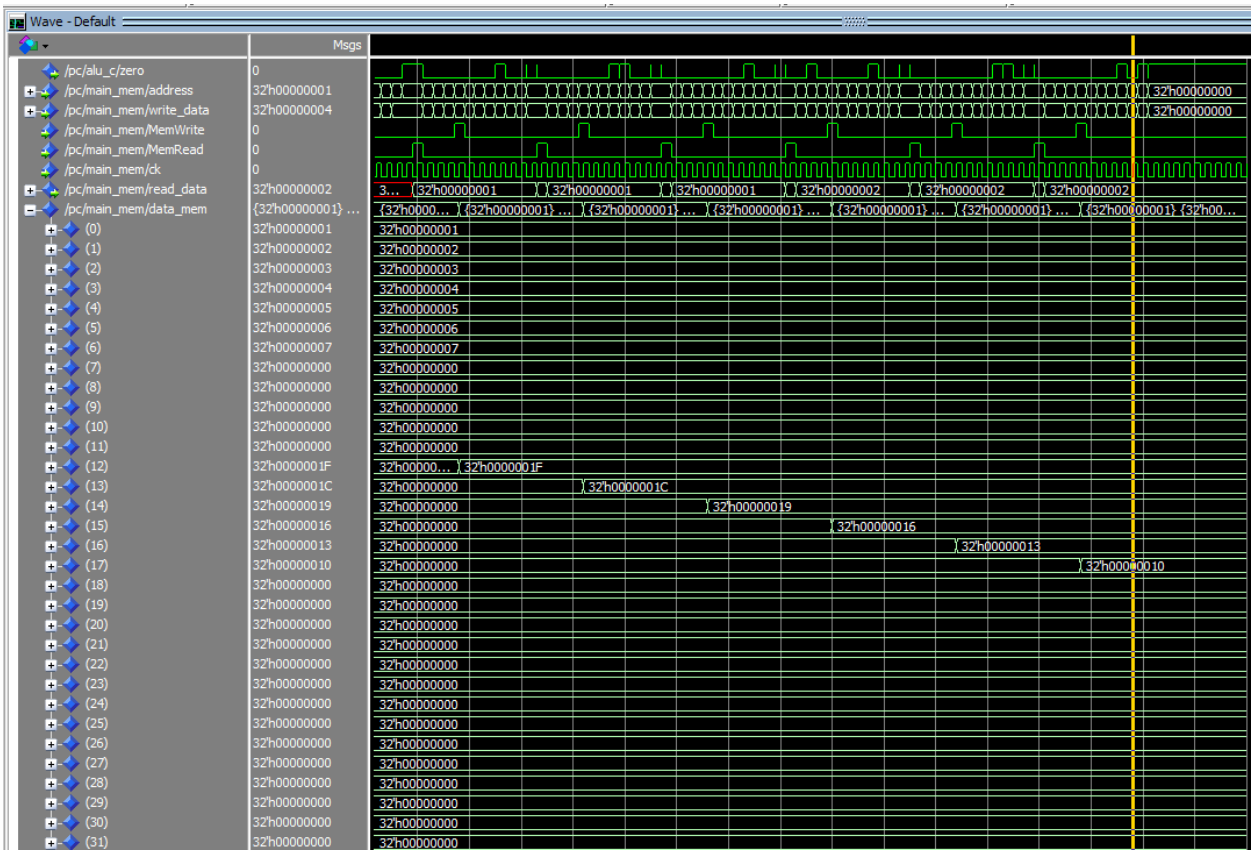

Figure 10, Screen capture for waveforms of memory in part C.

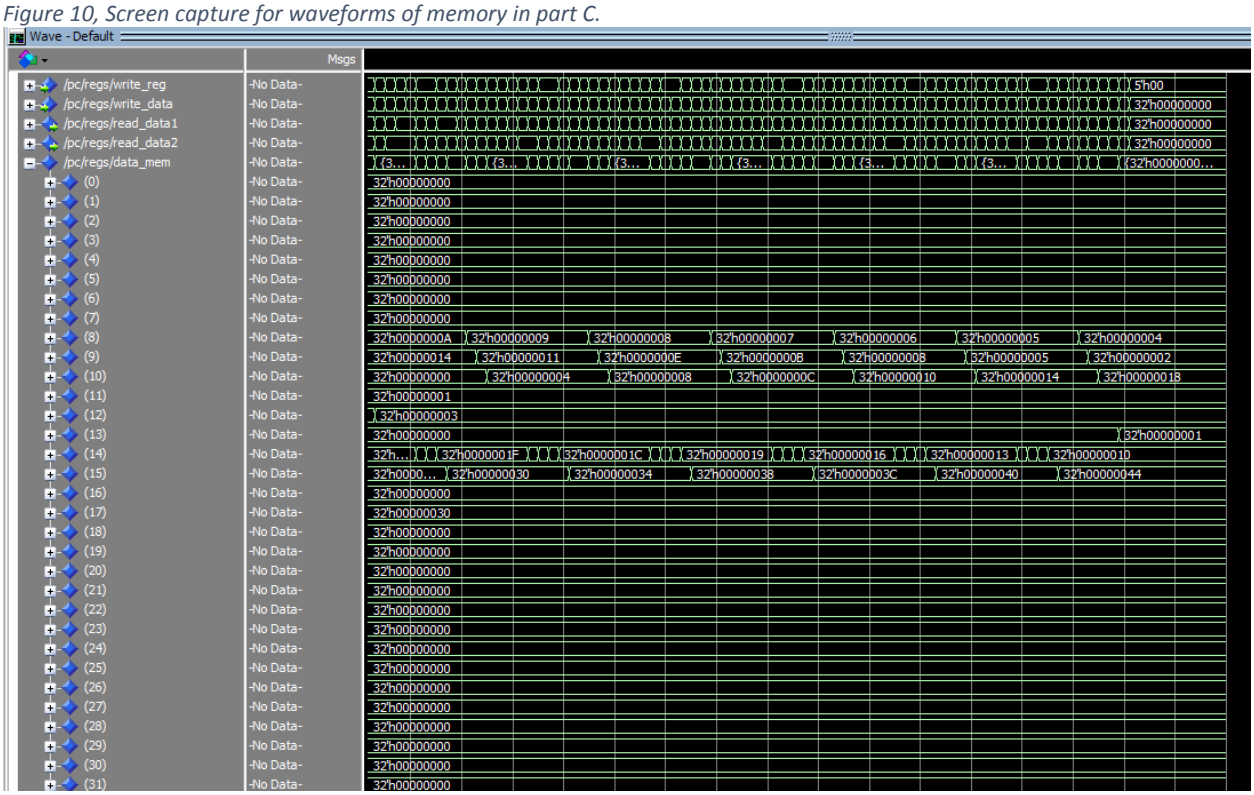

Figure 11, Screen capture for waveforms of registers in part C.

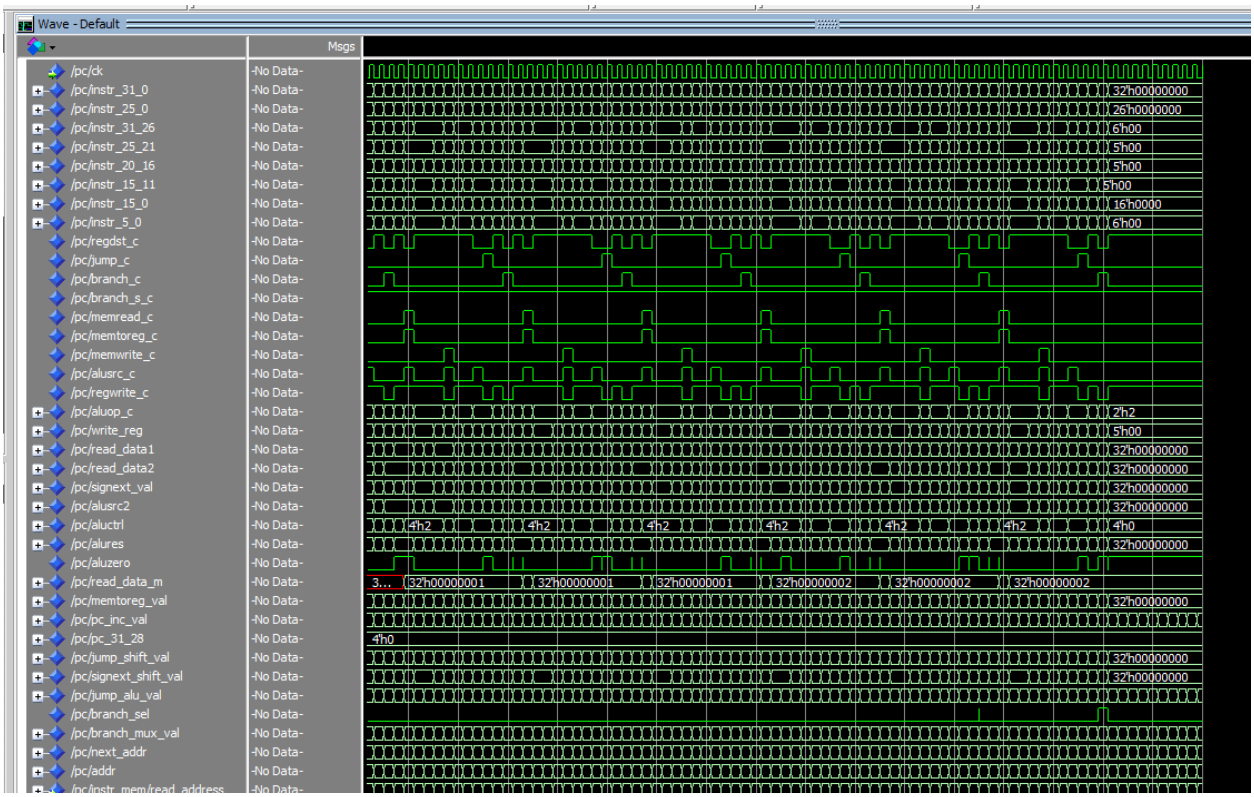

Figure 12, Screen capture of signal waveforms in part C.

Question Answers:

Part A:

Final Values:

$t0 = 0

$s0 = 5

$s1 = 4

$s3 = 19

$s4 = 14

$s5 = 5

0x00000000: 5

0x00000004: 4

0x00000008: 19

Part B:

Final Values:

$t0 = 0

$s0 = 4

$s1 = 4

$s3 = 9

$s4 = 14

$s5 = 5

0x00000000: 4

0x00000004: 4

0x00000008: 9

Part C:

Final Register Values:

x = 4 (in register $t0)

y = 2 (in register $t1)

i = 24 (in register $t2)

Final Data Memory Values:

0x00000000: 1

0x00000004: 2

0x00000008: 3

0x0000000C: 4

0x00000010: 5

0x00000014: 6

0x00000018: 7

0x00000030: 31

0x00000034: 28

0x00000038: 25

0x0000003C: 22

0x00000040: 19

0x00000044: 16

81 cycles were executed to complete part C.

Appendix A:

```
add $s0, $zero, $zero ; A = 0x0
addi $s1, $zero, 48 ; B = 0x30
addi $t0, $zero, 10 ; x = 10;
addi $t1, $zero, 20; y = 20;
addi $t2, $zero, $zero ; i = 0;
addi $t3, $zero, 1 ; store 1 as a comparator and operand
addi $t4, $zero, 3; store 3 as operand

LOOP: ; while (y >= x)
slt $t5, $t1, $t0
beq $t3, $t5, END

add $t6, $s0, $t2 ; A[i]
lw $t6, 0($t6)

add $t6, $t6, $t0 ; A[i] + x
add $t6, $t6, $t1 ; (A[i] + x) + y
add $t7, $s1, $t2 ; B[i]
sw $t6, 0($t7) ; B[i] = A[i] + x + y
sub $t0, $t0, $t3 ; x -= 1
sub $t1, $t1, $t4 ; y -= 3
addi $t2, $t2, 4 ; i += 1;
j LOOP ; re-evaluate
END:
```