To: Dr. John Hutson
From: Jonathan Gusler
Date: 2/20/2019
Subject: Introduction and Analysis of R-Type Instructions

This lab is very similar to Lab 04 which involved implementing a new program using the same instruction. This lab also implements a new program but uses new instruction, namely the R-Type instructions AND, OR, and SUB, along different memory values. This lab also analyzes what their execution looks like in the processor.

This lab uses the exact same hardware simulation as from Lab 03 [1] and Lab 04 [2]. It requires knowledge of MIPS, Quartus, and machine code. This lab is based off Exercise #3 in Chapter 14 Section 12 of *Rapid Prototyping of Digital Systems, SOPC Edition* by Hamblen and Furman [3].

The first step was changing the memory and program files. The pseudocode, shown in Figure 1, shows these changes. This pseudocode comes from the Lab Assignment provided by Dr. Hutson [4].

```
-- Assumes the following memory contents:
--        memory address 00 = ASCII value of char 'A'
--        memory address 01 = ASCII value of char 'B'
--        memory address 02 = ASCII value of char 'C'
START: load reg1 from address 00
       load reg2 from address 01
       load reg3 from address 02
       reg4 = reg1 AND reg2
       reg5 = reg3 – reg4
       reg6 = reg4 OR reg5
       if reg3 = reg6 branch to START
```

Figure 1: Memory changes and pseudocode for the program.

The changed memory and program files are shown in Appendix A. After updating the references to these files in the VHDL, the program was compiled, and a timing simulation was run to view the results and make sure the program ran correctly.

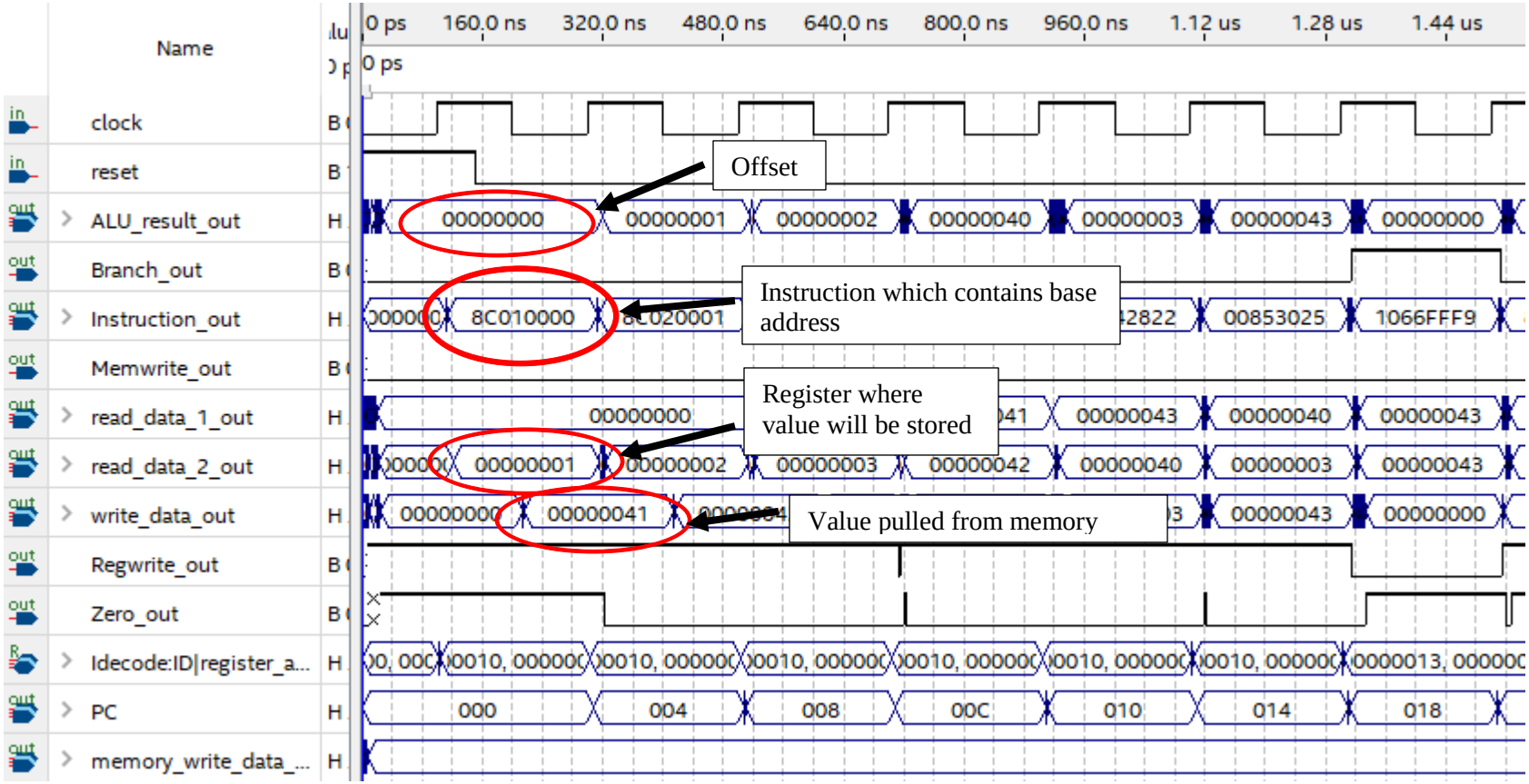
An analysis of each instruction is shown starting after Table 1, but only the first occurrence of each type of instruction. Table 1describes each of the relevant signals in the following analyses.

Signal Name	Description
clock	The clock for the system
reset	Resets system back to instruction on falling edge
ALU_result_out	Output of ALU
Branch_out	Branch control signal
Instruction_out	Instruction fetched from IM
Memwrite_out	Enable write to memory on high
read_data_1_out	Rs output from registers
read_data_2_out	Rd output from registers
write_data_out	Data to write to registers on high
Regwrite_out	Enable write to registers on high
Zero_out	Zero signal from ALU used in branch decisions
PC	Program counter

Table 1: Relevant signal names and descriptions for the following analyses.

The first instruction to be executed is a load word instruction in which the data in memory location 00 is loaded into register $1

8C010000; -- lw $1,0($0) ;memory(00)='A'
100011 | 00000 | 00001 | 0000000000000000
Opcode | s = $0 | t = $1 | offset = 0

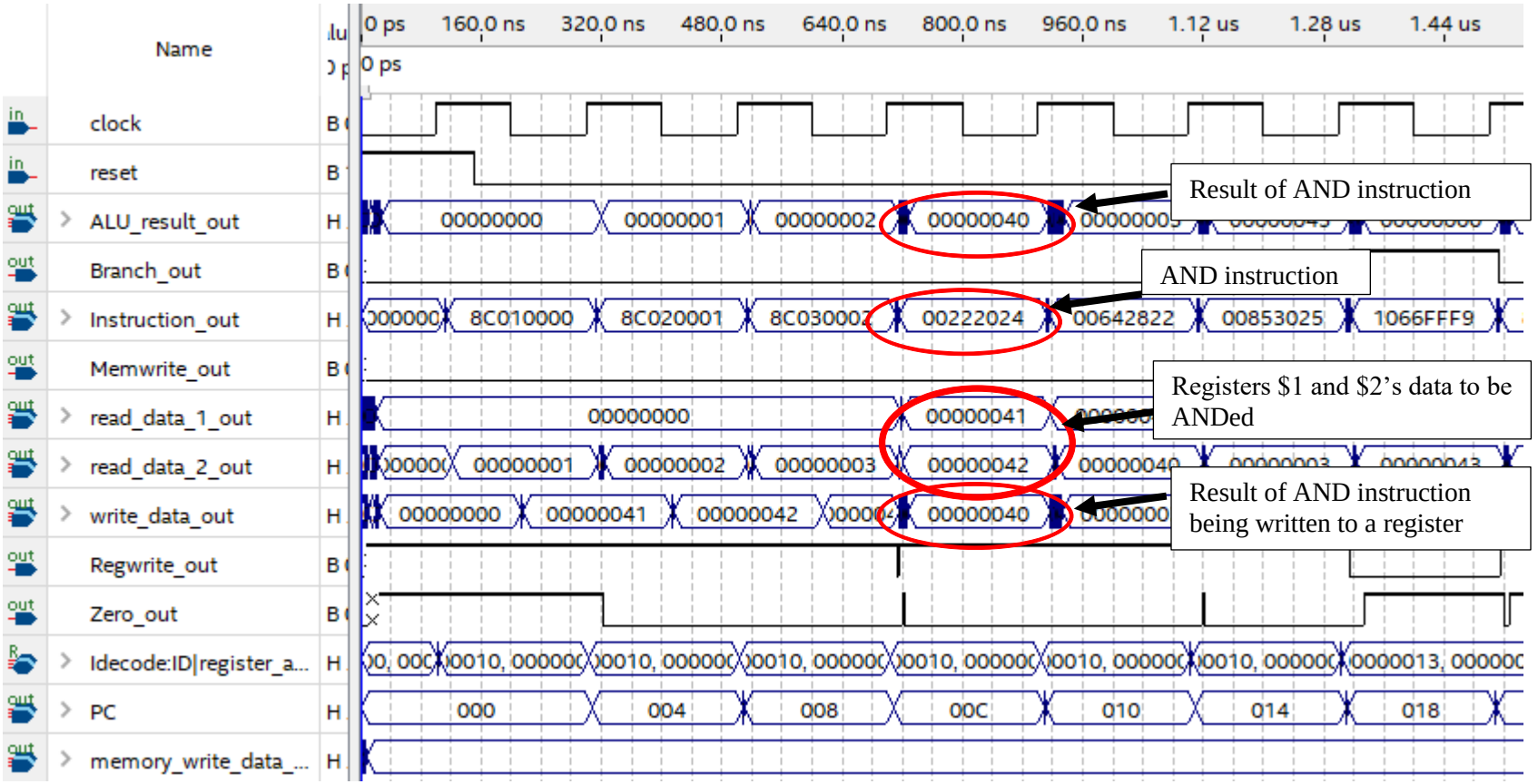


Control Signals Raised
RegWrite, MemRead, MemtoReg, ALUSrc

Control and Data Flow
After the clock is raised for this cycle, the program counter is loaded into the PC register and the instruction at that address is pulled out of program memory and becomes available on the Instruction_out line. Bits 31-26 of Instruction_out are directed to the Control Unit. Bits 25-21 go to the ALU to define the memory read address along with bits 15-0 to define the offset. Bits 20 -16 are sent to register write to define where data from memory will be written to.

The second unique instruction to be executed is AND in which the values from registers 1 and 2 are ANDed together and the result stored in register 4.

00222025; -- and $4,$1,$2
000000 | 00001 | 00010 | 00100 | 00000100101
Opcode | $s = 1 | $t = 2 | $d = 4 | ALU operation = AND

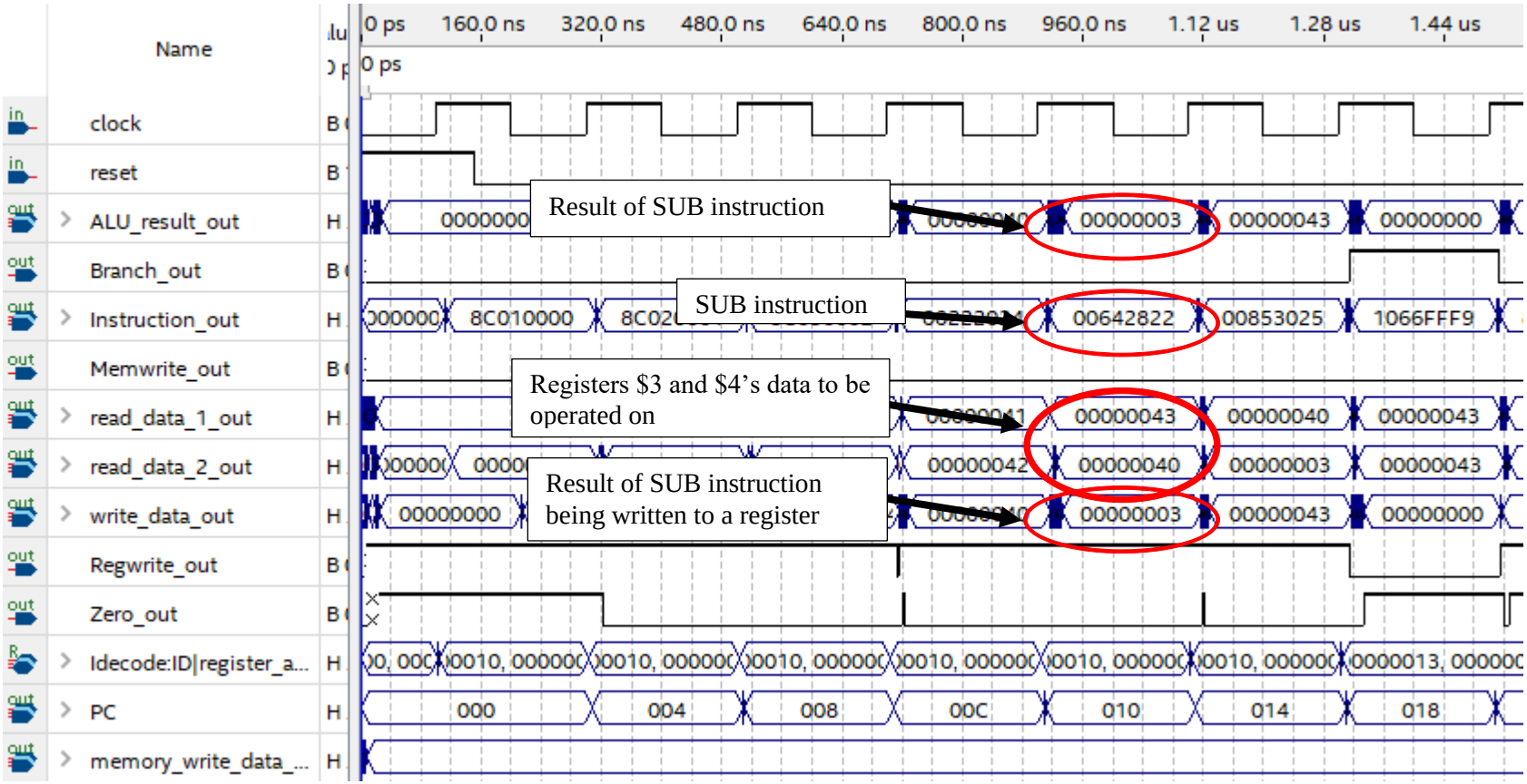


Control Signals Raised
RegWrite, ALUOp

Control and Data Flow
After the clock is raised for this cycle, the incremented program counter is loaded into the PC register and the instruction at that address is pulled out of program memory and becomes available on the Instruction_out line. Bits 31-26 of Instruction_out are directed to the Control Unit. Bits 25-16 are sent to read register 1 and 2 to define which registers are being read. Bits 15-11 go to the write register address. Bits 10-0 go to ALU Control and then to the ALU to define the ALU operation as AND. The result is then sent to register 4.

The third unique instruction to be executed is SUB in which the values from register 4 is subtracted from register 3 and the result is stored in register 5.

00642822; -- sub $5,$3,$4
000000 | 00011 | 00100 | 00101 | 00000100010
Opcode | $s = 3 | $t = 4 | $d = 5 | ALU operation = SUB

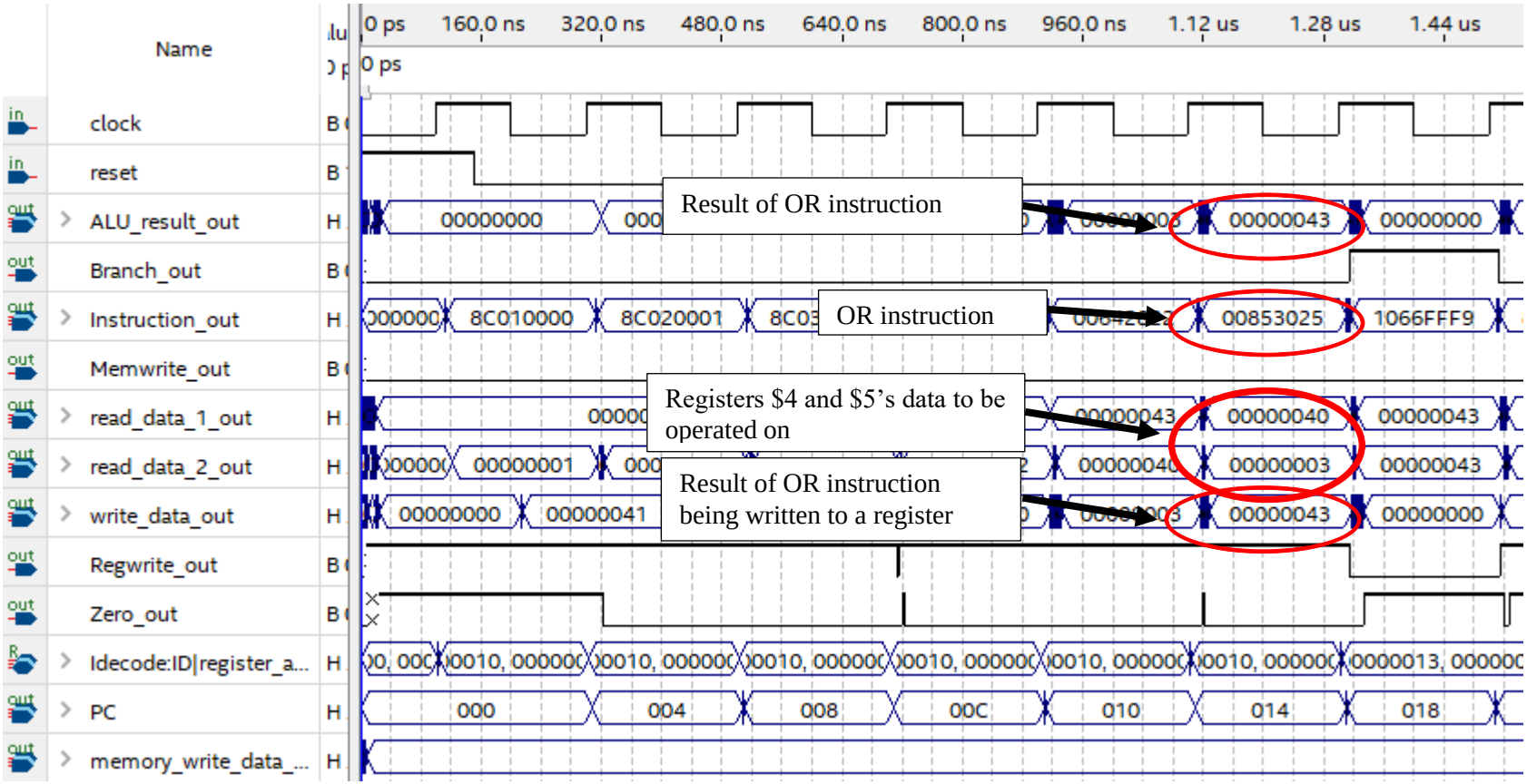


Control Signals Raised
RegWrite, ALUOp

Control and Data Flow
After the clock is raised for this cycle, the incremented program counter is loaded into the PC register and the instruction at that address is pulled out of program memory and becomes available on the Instruction_out line. Bits 31-26 of Instruction_out are directed to the Control Unit. Bits 25-16 are sent to read register 1 and 2 to define which registers are being read. Bits 15-11 go to the write register address. Bits 10-0 go to ALU Control and then to the ALU to define the ALU operation as OR. The result is then sent to register 5.

The fourth unique instruction to be executed is OR in which the values from register 4 and register 5 are ORed together and the result is stored in register 6.

00853025; -- or $6,$4,$5
000000 | 00100 | 00101 | 00110 | 00000100101
Opcode | $s = 4 | $t = 5 | $d = 6 | ALU operation = OR

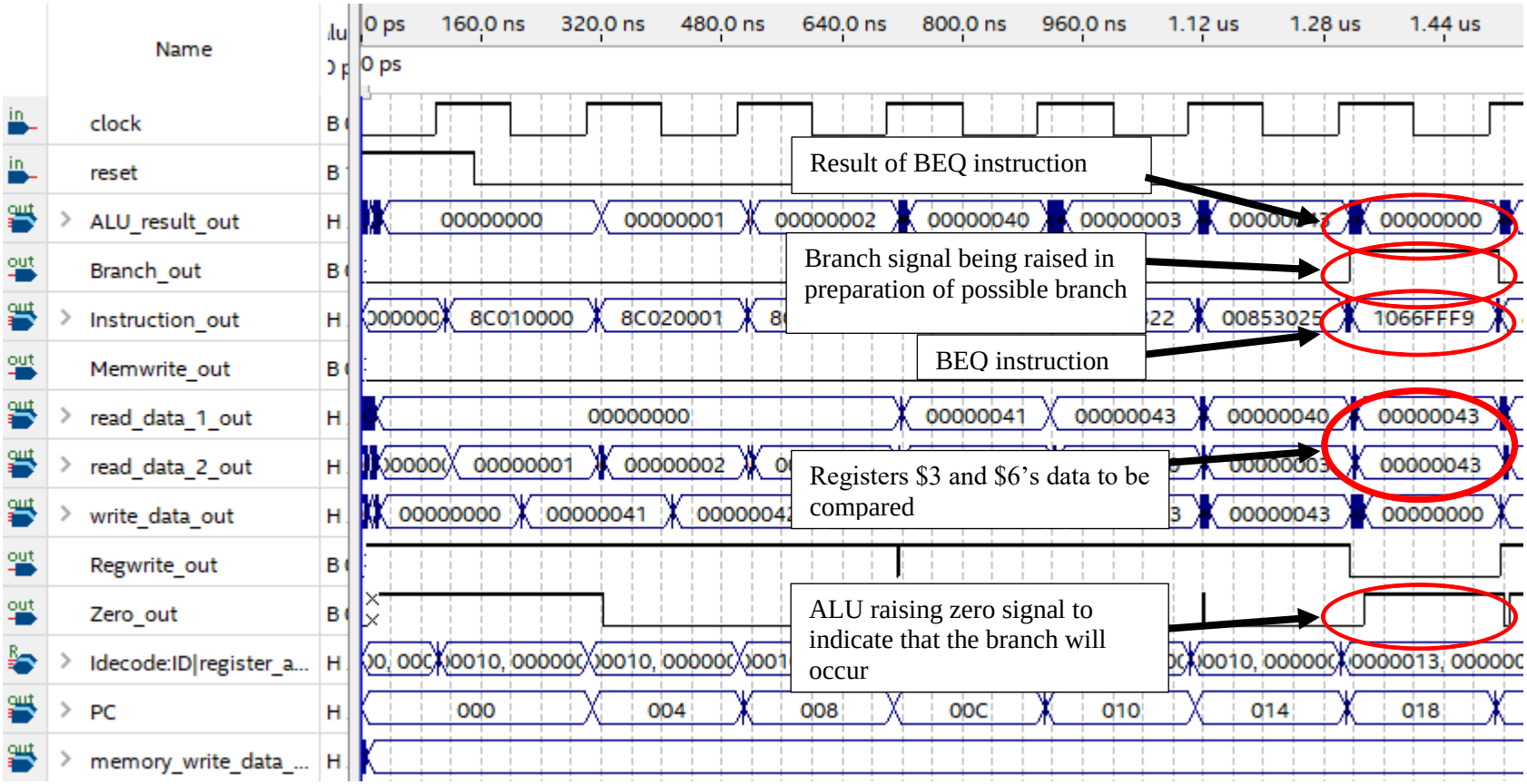


Control Signals Raised
RegWrite, ALUOp

Control and Data Flow
After the clock is raised for this cycle, the incremented program counter is loaded into the PC register and the instruction at that address is pulled out of program memory and becomes available on the Instruction_out line. Bits 31-26 of Instruction_out are directed to the Control Unit. Bits 25-16 are sent to read register 1 and 2 to define which registers are being read. Bits 15-11 go to the write register address. Bits 10-0 go to ALU Control and then to the ALU to define the ALU operation as OR. The result is then sent to register 6.

The fifth and last unique instruction to be executed is BEQ in which the values from register 3 and register 6 are tested for equality and, if equal, result in a return to the beginning of the program.

1066FFF9; -- beq $3,$6,-28
000100 | 00011 | 00110 | 1111111111111001
Opcode | $s = 3 | $t = 6 | PC jump amount, -7 instructions



Control Signals Raised
Zero, ALUOp Branch

Control and Data Flow
After the clock is raised for this cycle, the incremented program counter is loaded into the PC register and the instruction at that address is pulled out of program memory and becomes available on the Instruction_out line. Bits 31-26 of Instruction_out are directed to the Control Unit. Bits 25-16 are sent to read register 1 and 2 to define which registers are being read. Bits 15-0 go to the sign extender and then to the shifter to guarantee word alignment. The PC then jumps back 7 instructions to the beginning of the program due to the branch possibility and the Zero signal from the ALU being set to confirm the branch operation.

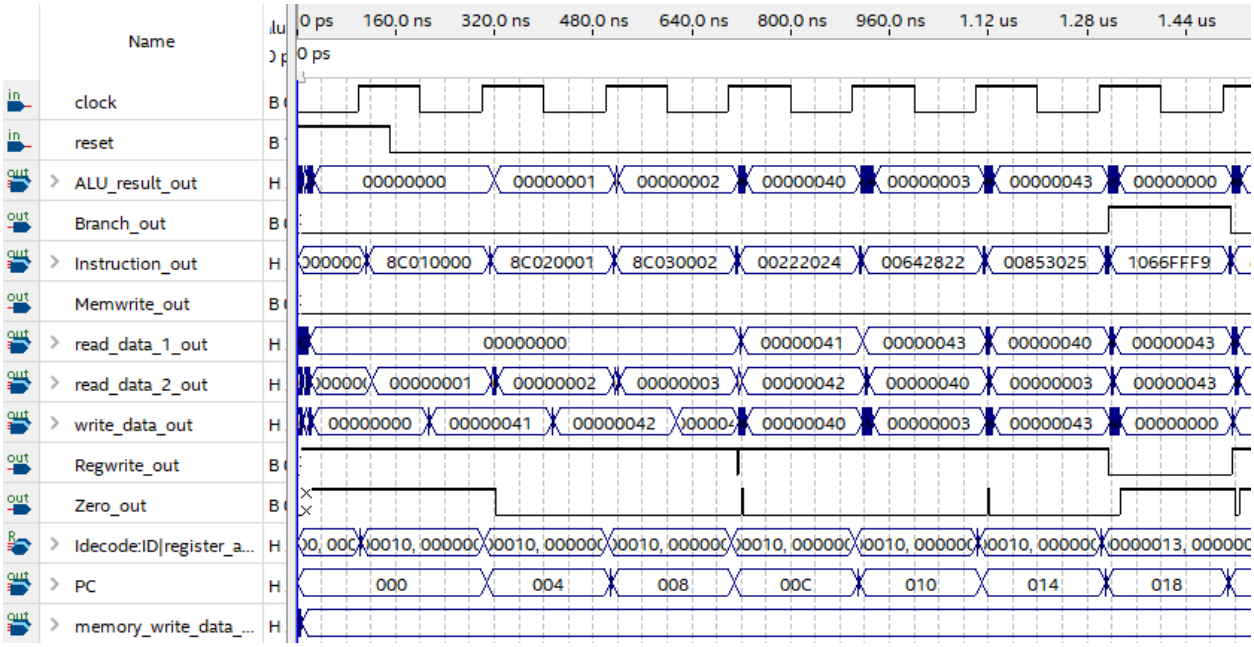


Figure 2: Waveform of simulation.

Figure 2 shows some of the signals and data passing through the processor during the program. The instructions and the results in order are as follows:

1) Memory location 00 is loaded into register 1
2) Memory location 01 is loaded into register 2
3) Memory location 02 is loaded into register 3
4) The contents of registers 1 and 2 are ANDed together with the result stored in register 4
5) The contents of register 4 is subtracted from the contents of register 3 and the result stored in register 5
6) The contents of registers 4 and 5 are ORed together and the result stored in register 6.
7) The contents of register 3 and 6 are compared for equality. The ALU finds that they are equal, raises the Zero signal, and the program loops back to the beginning.

Figure 2 also shows successful execution of the program.

The lab introduced the use of R-Type instructions by having the student translate them into assembly language and into machine language and then test the program using them in a simulation. The program ran successfully according to the pseudocode demonstrating basic understating and ability to implement R-Type instructions.

References

[1] J. Gusler, "Lab 03," 2019.

[2] J. Gusler, "Lab 04," 2019.

[3] H. F. Hamblen, in *Rapid Prototyping of Digital Systems, SOPC Edition*, p. Section 14.

[4] D. J. Hutson. [Online]. Available: https://lipscomb.instructure.com/courses/17375/files/folder/Lab%20Assignments?preview=642730. [Accessed 14 February 2019].

Appendix A

```
--  MIPS Data Memory Initialization File
depth=256;
width=32;
Content
Begin
-- default value for memory
 [00..FF] : 00000000;
-- initial values for test program
 00 : 00000041;     -- ASCII 'A'
 01 : 00000042;     -- ASCII 'B'
 02 : 00000043;     -- ASCII 'C'
End;
```

```
-- MIPS Instruction Memory Initialization File
Depth = 256;
Width = 32;
Address_radix = HEX;
Data_radix = HEX;
Content
Begin
-- Use NOPS for default instruction memory values
    [00..FF]: 00000000; -- nop (sll r0,r0,0)
-- Place MIPS Instructions here
-- Note: memory addresses are in words and not bytes
-- i.e. next location is +1 and not +4
    00: 8C010000;   -- lw $1,0 ;memory(00)='A'
    01: 8C020001;   -- lw $2,1 ;memory(01)='B'
    02: 8C030002;   -- lw $3,2 ;memory(02)='C'
    03: 00222024;   -- and $4,$1,$2
    04: 00642822;   -- sub $5,$3,$4
    05: 00853025;   -- or $6,$4,$5
    06: 1066FFF9;   -- beq $3,$6,-28
End;
```