1ST EDITION 2018

PROGRAMMERS REFERENCE MANUAL

JOCELYN ESPITIA
&
THOMAS VUGIA

"If you do not have the time to do it right, when will you find the time to do it over?"

John Wooden

Authors:

Jocelyn Espitia - 014101709

Thomas Vugia - 013580942

December 3rd, 2018

CECS 341 – Section 7

Mon/Wed 8am

Professor R. W. Allison

Table of Contents

Chapter 1

Purpose

Abstract

Microprocessor without Interlocked Pipeline Stages (MIPS) is a RISC instruction set architecture. This architecture began as part of a research program, led by John Hennessy, with the purpose of creating a microprocessor using RISC principles. MIPS can be considered to be a “classic” RISC architecture.

MIPS consists of registers that can be written or read from with certain instructions. MIPS processors can be found in routers, computer processors, and other small electronics (like the PS2).

This Programmers Reference Manual focuses with a custom Single Instruction Multiple Data (SIMD) instruction set architecture. Instructions with this architecture are designed by using some known MIPS instructions. SIMD is great for working with vector manipulations.

Application

This manual is made with intentions to support Software Developers, Computer Engineers, and Computer Hardware Engineers, with their computer architecture designs. For Developers although hardware may not be in their interest or specialty, it is important to have some level of understanding about how hardware systems are working with your software. If you’re familiar with code, using JAVA as an example, you may find that sometimes when trying to return and print a result you are given the memory address instead of the result. An error you are having is that you are calling the address to be returned.

Fourteen instructions were implemented with SIMD. Each instruction is shown in this manual with how it improves the algorithms necessary for the intended instruction.

1. Vector Add Saturated

a. Specification: Each element of one vector is added to the corresponding element of another vector. The result of each element is placed into the corresponding element of the result vector.

2. Vector Multiply and Add

a. Specification: To multiply and add we are using four vectors, three are being used for arithmetic and one used to hold the resulting values. The elements from vector a are multiplied with the corresponding elements in vector b. The resulting value is then added with the elements in vector c and the final values are stored in vector d.

3. Vector Multiply Even Integer

a. Specification: Two 16-bit vectors, vector a and vector b are used to get an 8-bit resulting vector. Each *even* element (0, 2, 4, 6) in vector a is multiplied to its corresponding element in vector b. The resulting values are stored in vector c.

4. Vector Multiply Odd Integer

a. Specification: Two 16-bit vectors, vector a and vector b are used to get an 8-bit resulting vector. Each *odd* element (1, 3, 5, 7) in vector a is multiplied to its corresponding element in vector b. The resulting values are stored in vector c.

5. Vector Multiply Sum Saturated

a. Specification: Like the vector multiply add instruction, this instruction uses four vectors. The first two vectors, vector a and vector b, have their elements multiplied by each other, these results are a temporary product. Starting from index 0 from the temp. products, two neighboring elements are added with the elements from an 8-bit vector c. These values are stored in an 8-bit vector d.

6. Vector Splat

a. Specification: This instruction is used to copy one element of a vector into every element in another vector.

7. Vector Merge Low

a. Specification: This instruction uses the “lower half” of two vectors. The first vector has its lower half elements replicated into every other element of the resulting vector starting from index 6 to index 0.

8. Vector Merge High

a. Specification: This instruction uses the “higher half” of two vectors. The first vector has its higher half elements replicated into every other element of the resulting vector starting from index 7 to index 1.

9. Vector Pack

a. Specification: This instruction uses the wider elements of two vectors, storing all the wider elements of one vector into the high elements of the resulting vector and storing all the wider elements of the other vector into the low elements of the resulting vector.

10. Vector Permute

a. Specification: This instruction uses three different vectors a, b, and c, of which vector c holds the “element specifier” that specifies from vectors a or b and their respective element. The element specifier has two components: the most-significant half specifies from either vector a or b and the least-significant-half specifies which element within the vector. Based on the corresponding element in vector c, it stores the respective element from one of the two vectors into the resulting vector d.

11. Vector Compare Equal-To

a. Specification: This instruction compares two vectors’ corresponding elements and stores the result of determining whether they are equal or not into the corresponding element of the resulting vector.

12. Vector Compare Less-Than (unsigned)

a. Specification: This instruction compares two vectors’ corresponding elements and stores the result of determining whether the first is greater than the second or not into the corresponding element of the resulting vector.

13. Vector Compare Not Equal-To

a. Specification: This instruction will determine if two corresponding elements from two vectors a and b are not equal or not. If they are not equal, the result (FF) is stored into the corresponding element of vector d. Else, the result (00) is stored.

14. Vector Compare Greater-Than (unsigned)

a. Specification: This instruction will determine if the corresponding element from vector a is greater than the corresponding element in vector b. If true, the result (FF) is stored into the corresponding element of vector d. Else, the result (00) is stored.

Chapter 2

Instruction Set Architecture

Machine Register Set

The machine register set available to you – the user, is:

Name	Number	Use	Preserved across a call?
$zero	0	The constant value 0	N/A
$at	1	Assembler Temporary	No
$v0 - $v1	2 – 3	Values for function results and expression evaluation	No
$a0 - $a3	4 – 7	Arguments	No
$t0 - $t7	8 – 15	Temporaries	No
$s0 - $s7	16 - 23	Saved Temps	Yes
$t8 - $t9	24 – 25	Temporaries	No
$k0 - $k1	26 -27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

Data Types

In MIPS all instructions read and write 32 bits. Below is a table with the different data types and their sizes.

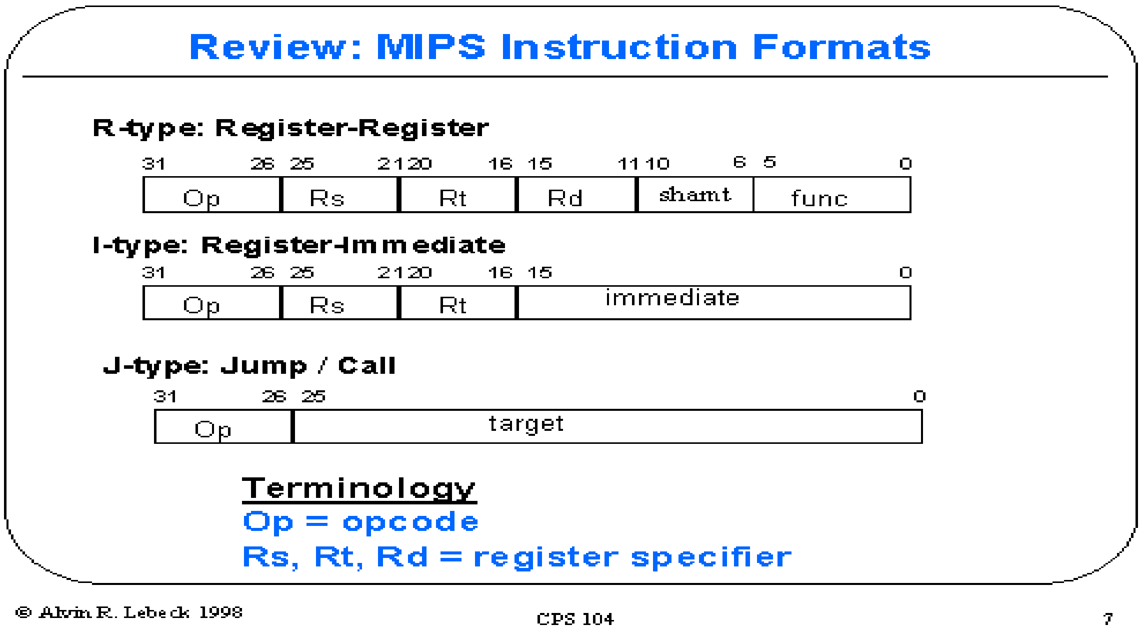
Data Type	**Size**
Halfword	16 bits
Word	32 bits
Byte	8 bits
Int	32 bits
Char	4 bits

Addressing Modes

A simple explanation given by *geeksforgeeks.org* about what Addressing Modes are is “the term addressing modes refers to the way in which the operand of an instruction is specified. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is executed.

There are many 5 basic modes you can use: Immediate, Direct, Register Indirect, Base, and PC Relative. Keep in mind that you can also manipulate these modes into different modes as well.

This diagram shows the type of instruction types and how they vary from registers and opcodes.



Review: MIPS Instruction Formats

R-type: Register-Register

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Op	Rs	Rt	Rd	shamt	func

I-type: Register-Immediate

31 – 26	25 – 21	20 – 16	15 – 0
Op	Rs	Rt	immediate

J-type: Jump / Call

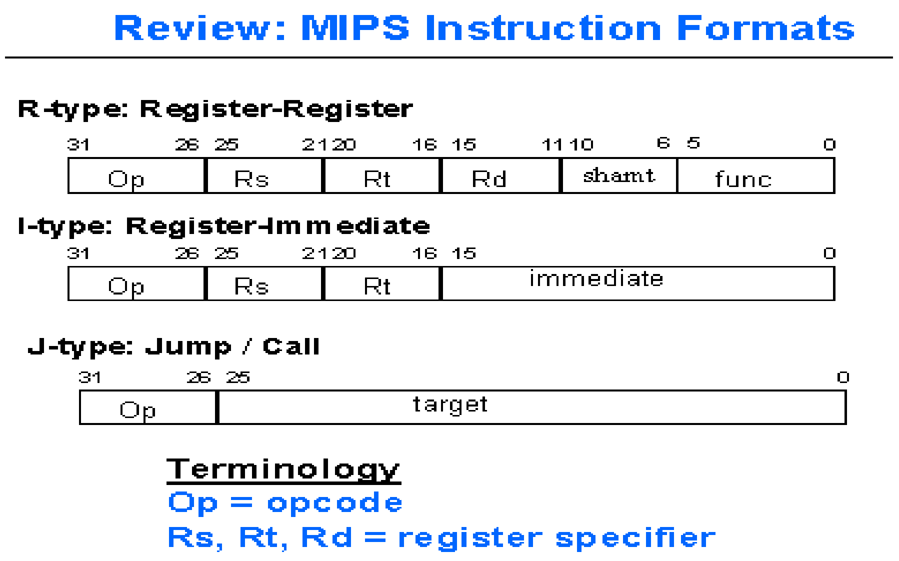
31 – 26	25 – 0
Op	target

Terminology
Op = opcode
Rs, Rt, Rd = register specifier

© Alvin R. Lebeck 1998 CPS 104 7

Fig. 1. MIPS Instruction Formats, Duke University, from Alvin R. Lebeck, *MIPS ISA, Assembly Language*; Powerpoint; Web; 30 Nov. 2018, https://www2.cs.duke.edu/courses/fall98/cps104/lectures/week3-l2/sld007.htm

Instruction Set & Binary Instruction Formats



The types of instructions you will find in this manual are listed below. Along with specific examples.

1) Triple operands
   a) Are typically R-Type instructions
2) Double operands
   a) Are typically I-Type instructions
3) Single operands
   a) Are typically J-Type instructions
4) Conditional branches
   a) Branches are I-Type instructions.
   b) Most of these instructions in our manual use conditional branches such as, "BNE". We can use these to iterate through our elements in the vectors.
5) Unconditional jump and subroutine call
   a) Jumps have their own format called J-Type.
6) Immediate operand instructions
   a) Instructions such as addi, lui, slti, ori. These immediate instructions use I-Type format.
   b) Vector Splat is an example of an immediate operand in this manual

7) **SIMD instructions**
   a) Our instructions used a variety of the instruction formats.

8) **Flags processor instructions**
   a) Individual bits control the action or represent the status of the processor
      i) Control flags (TF, IF, DF)
         1. Determine how the processor responds to certain situations
      ii) Status flags (CF, PF, AF, ZF, SF, OF)
         2. Set to represent the result of certain operations
         3. Used to control conditional jump instructions

Chapter 3

MIPS Implementation / Verification with Annotation

Vector Add Saturated (Unsigned)

```
# File Name: vector_add_sat_uns.asm
 # Version: 1.0
 # Date: Nov 5, 2018
 # Programmer: Jocelyn Espitia & Thomas Vugia
 #
 # Description: vec_addsu d, a, b; add the elements of a and b and store the
 #                    results in d to add 8 bytes to another 8 bytes in one instruction
 #
 # Register Usage:
 #        vector A will be shown with $s0 & $s1
 #        vector B will be shown with $s2 & $s3
 #        vector D will be shown with $s6 & $s7
 #***************************************************************************



 #***************************************************************************
 #                              CODE SEGMENT
 #***************************************************************************
    .text
    .globl main
                                      #this will load the vectors we need to add
 main:
       li $s0, 0x233C475D       #load reg $s0 with elements 0 - 3 from vector A
       li $s1, 0x087F196F       #load reg $s1 with elements 4 - 7 from vector A
          li $s2, 0x981963C5    #load reg $s2 with elements 0 - 3 from vector B
          li $s3, 0x5E80B36E    #load reg $s3 with elements 4 - 7 from vector B


                                      #vector A is placed into temp regs
          add $t0, $s0, $zero   #the elements 0 - 3 of vector A are in $t0
          add $t1, $s0, $zero   #the elements 0 - 3 of vector A are in $t1
          add $t2, $s0, $zero   #the elements 0 - 3 of vector A are in $t2
          add $t3, $s0, $zero   #the elements 0 - 3 of vector A are in $t3


                                      #vector B is placed into temp regs
          add $t4, $s2, $zero   #the elements 0 - 3 of vector B are in $t4
          add $t5, $s2, $zero   #the elements 0 - 3 of vector B are in $t5
          add $t6, $s2, $zero   #the elements 0 - 3 of vector B are in $t6
          add $t7, $s2, $zero   #the elements 0 - 3 of vector B are in $t7


                                      #working with vector A
          sll $t0, $t0, 0
          srl $t0, $t0, 24      #by shifting right, it pushes the value of
                                      #element 0 to element 3
          sll $t1, $t1, 8       #elements 1 - 3 are now held in 0 - 2
          srl $t1, $t1, 24      #by shifting right, it pushes the value of
                                      #element 0 to element 3
          sll $t2, $t2, 16
          srl $t2, $t2, 24      #elements 2 - 3 are now held in 0 - 1
          sll $t3, $t3, 24
          srl $t3, $t3, 24      #element 3 is shifted into element 0
          sll $t4, $t4, 0       #element 0 is shifted into element 3
```

```
                        #working with vector B
        srl $t4, $t4, 24
        sll $t5, $t5, 8         #element 0 is shifted into element 3
        srl $t5, $t5, 24        #elements 1 - 3 are now held in 0 - 2
        sll $t6, $t6, 16        #element 0 is shifted into element 3
        srl $t6, $t6, 24        #elements 2 - 3 are now held in 0 - 1
        sll $t7, $t7, 24        #element 0 is shifted into element 3
        srl $t7, $t7, 24        #element 3 is shifted into element 0
                                #element 0 is shifted into element 3

                                #now we begin to add
        add $t0, $t0, $t4       #adding elements 0 from both vectors
        add $t1, $t1, $t5       #adding elements 1 from both vectors
        add $t2, $t2, $t6       #adding elements 2 from both vectors
        add $t3, $t3, $t7       #adding elements 3 from both vectors
        addi $t8, $zero, 255    #$t8 now holds the result of element 3

        blt $t0, $t8, vector0   #if true, $at holds a 1
        add $t0, $t8, $zero

vector0:
        blt $t1, $t8, vector1
        add $t1, $t8, $zero

vector1:
        blt $t2, $t8, vector2
        add $t2, $t8, $zero
vector2:
```

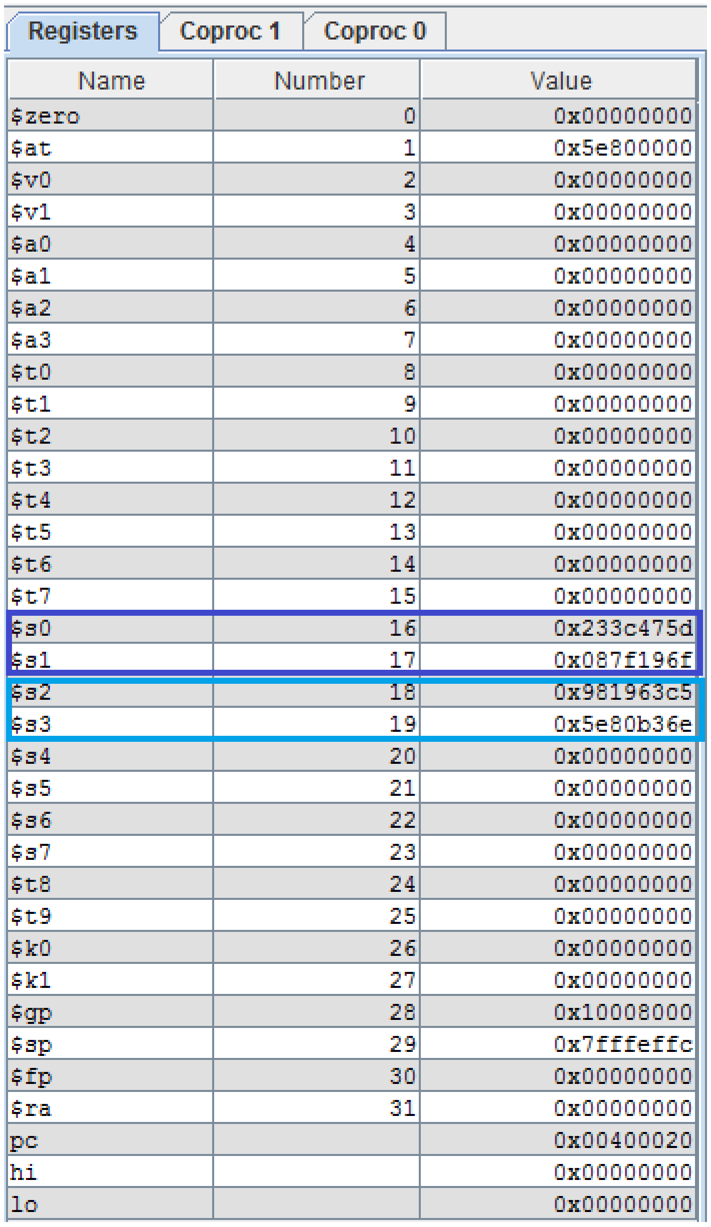
```
          blt $t3, $t8, vector3
          add $t3, $t8, $zero    #$t3 now has result of element 3
vector3:
                                 #when we began to add, the results
                                 #were stored into element 3 of their
                                 #respective registers
          sll $t0, $t0, 24       #element 3 is shifted into element 0
          sll $t1, $t1, 16       #element 3 is shifted into element 1
          sll $t2, $t2, 8        #element 3 is shifted into element 2
          sll $t3, $t3, 0        #element 3 stays in element 3


                                 #now the results are put into the final
                                 #vector.
          add $s6, $t0, $s6      #adding the value/element of $t0 into $s6
          add $s6, $t1, $s6      #adding the value/element of $t1 into $s6
          add $s6, $t2, $s6      #adding the value/element of $t2 into $s6
          add $s6, $t3, $s6      #adding the value/element of $t3 into $s6

                                 #we now have half the results we need
                                 #elements 0 - 3 in the final vector



                                 #this is vector A that holds elements 4 - 7
                                 #vector A is placed into temp regs
          add $t0, $s1, $zero    #adds reg $t0 with elements 4 - 7 from vector A
          add $t1, $s1, $zero    #adds reg $t1 with elements 4 - 7 from vector A
          add $t2, $s1, $zero    #adds reg $t2 with elements 4 - 7 from vector A
          add $t3, $s1, $zero    #adds reg $t3 with elements 4 - 7 from vector A
```

```
                        #this is vector B that holds elements 4 - 7
                        #vector B is placed into temp regs
add $t4, $s3, $zero     #adds reg $t4 with elements 4 - 7 from vector B
add $t5, $s3, $zero     #adds reg $t5 with elements 4 - 7 from vector B
add $t6, $s3, $zero     #adds reg $t6 with elements 4 - 7 from vector B
add $t7, $s3, $zero     #adds reg $t7 with elements 4 - 7 from vector B

sll $t0, $t0, 0
srl $t0, $t0, 24        #by shifting right, it pushes the value of
                        #element 0 to element 3
sll $t1, $t1, 8         #elements 5 - 7 are shifted into 4 - 6 of $t1
srl $t1, $t1, 24        #element 4 is shifted into element 7 of $t1
sll $t2, $t2, 16        #elements 6 - 7 are shifted into 4 - 5 of $t2
srl $t2, $t2, 24        #element 4 is shifted into element 7 of $t2
sll $t3, $t3, 24        #element 7 is shifted into element 4 of $t3
srl $t3, $t3, 24        #element 4 is shifted into element 7 of $t3

                        #working with vector B now
sll $t4, $t4, 0
srl $t4, $t4, 24        #element 4 is shifted into element 7 of $t4
sll $t5, $t5, 8         #elements 5 - 7 are shifted into elements 4 - 6 of $t5
srl $t5, $t5, 24        #element 4 is shifted into element 7 of $t5
sll $t6, $t6, 16        #element 7 is shifted into element 4 of $t6
srl $t6, $t6, 24        #element 6 is shifted into element 7 of $t6
sll $t7, $t7, 24        #element 7 is shifted into element 4 of $t7
srl $t7, $t7, 24        #element 4 is shifted into element 7 of $t7


add $t0, $t0, $t4       #adding elements 4 from both vectors
```

```
        add $t1, $t1, $t5       #adding elements 5 from both vectors
        add $t2, $t2, $t6       #adding elements 6 from both vectors
        add $t3, $t3, $t7       #adding elements 7 from both vectors
        addi $t8, $zero, 255
        blt $t0, $t8, vector4
        add $t0, $t8, $zero

vector4:
        blt $t1, $t8, vector5
        add $t1, $t8, $zero

vector5:
        blt $t2, $t8, vector6
        add $t2, $8, $zero

vector6:
        blt $t3, $t8, vector7
        add $t3, $t8, $zero

vector7:
        sll $t0, $t0, 24        #element 7 is shifted into element 4 of $t0
        sll $t1, $t1, 16        #element 7 is shifted into element 3 of $t1
        sll $t2, $t2, 8         #element 7 is shifted into element 2 of $t2
        sll $t3, $t3, 0         #element 7 is shifted into element 7

        add $s7, $t0, $s7       #adding the value/element of $t0 into $s7
        add $s7, $t1, $s7       #adding the value/element of $t1 into $s7
        add $s7, $t2, $s7       #adding the value/element of $t2 into $s7
        add $s7, $t3, $s7       #adding the value/element of $t3 into $s7
```

```
exit:
	ori $v0, $zero, 10
	syscall

.data
```

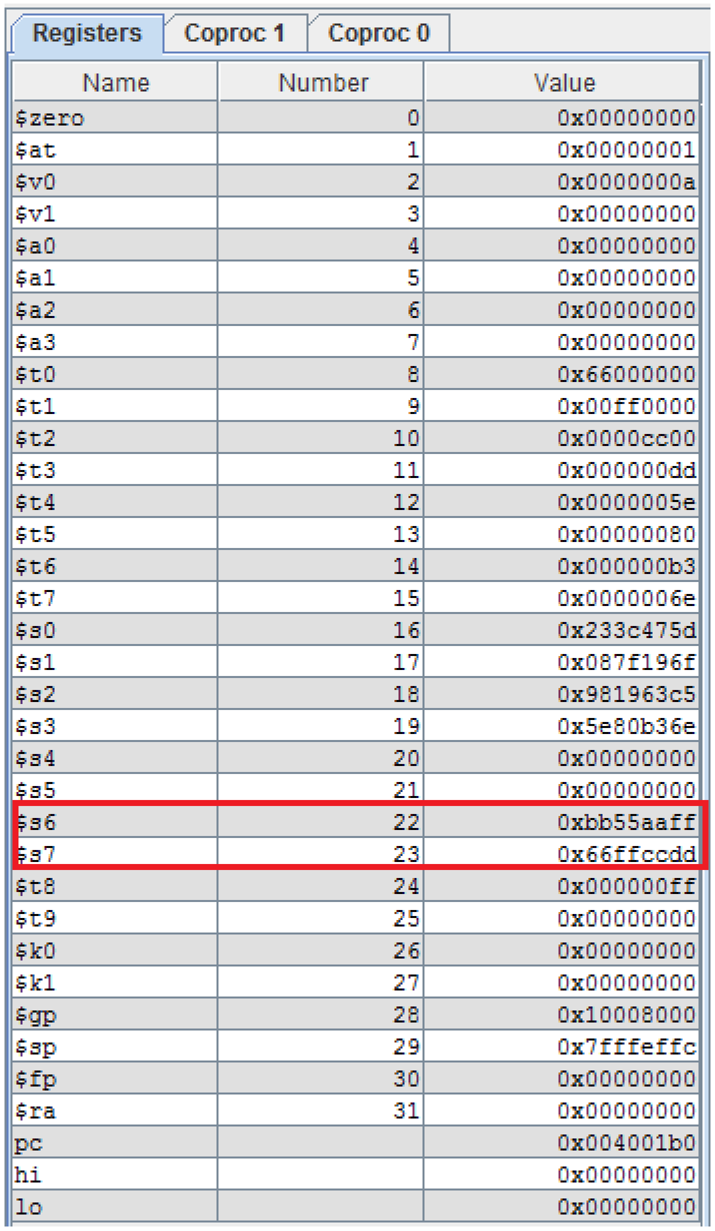
VECTOR ADD SATURATED BEFORE



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x5e800000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x233c475d
$s1	17	0x087f196f
$s2	18	0x981963c5
$s3	19	0x5e80b36e
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400020
hi		0x00000000
lo		0x00000000

VECTOR ADD SATURATED AFTER



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000001
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x66000000
$t1	9	0x00ff0000
$t2	10	0x0000cc00
$t3	11	0x000000dd
$t4	12	0x0000005e
$t5	13	0x00000080
$t6	14	0x000000b3
$t7	15	0x0000006e
$s0	16	0x233c475d
$s1	17	0x087f196f
$s2	18	0x981963c5
$s3	19	0x5e80b36e
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0xbb55aaff
$s7	23	0x66ffccdd
$t8	24	0x000000ff
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x004001b0
hi		0x00000000
lo		0x00000000

Vector Multiply & Add

```
# File Name: vector_mult_and_add.asm
# Version: 1.0
# Date: Nov 24, 2018
# Programmer: Jocelyn Espitia & Thomas Vugia
#
# Description: vec_madd d, a, b, c; multiplies the elements of a and b then, adds
#                       the result to vector c. The results are stored in vector d.
#
# Register Usage:
#       vector A will be shown with $s0 & $s1
#       vector B will be shown with $s2 & $s3
#       vector C will be shown with $s4 & $s5
#       vector D will be shown with $s6 & s7
#********************************************************************************


#********************************************************************************
#                               CODE SEGMENT
#********************************************************************************
        .text
        .globl main

                                #this will load the vectors that will be multipled
main:
        lui $s0, 0x120C         #load reg $s0 with elements 0 & 1 from vector A
        li $s1, 0x00001A0D      #load reg $s1 with elements 2 & 3 from vector A
        or $s0, $s0, $s1        #this instruction ors $s0 & $s1 and stores the
                                #result in $s0
        lui $s1, 0x2305         #load reg $s1 with elements 4 & 5 from vector A
        li $s2, 0x00001912      #load reg $s2 with elements 6 & 7 from vector A
        or $s1, $s1, $s2        #this instruction ors $s1 & $s2 and stores the
                                #result in $s1

        add $s2, $zero, $zero           #$s2 = 0



        lui $s2, 0x3D0C         #load reg $s2 with elements 0 & 1 from vector B
        li $s3, 0x0000104D      #load reg $s3 wuth elements 2 & 3 from vector B
        or $s2, $s2, $s3        #this instruction ors $s2 & $s3 and stores the
                                #result in $s2

        lui $s3, 0x057F         #load reg $s3 with elements 4 & 5 from vector B
        li $s4, 0x0000192B      #load reg $s4 with elements 6 & 7 from vector B
        or $s3, $s3, $s4        #this instruction ors $s3 & $s4 and stores the
                                #result in $s3

        add $s4, $zero, $zero           #$s4 = 0




                                #this will load the vector that will be added
add_vec:
        lui $s4, 0x6009         #load reg $s4 with elements 0 & 1 from vector C
        li $s5, 0x00001B05      #load reg $s5 with elements 2 & 3 from vector C
        or $s4, $s4, $s5        #this instruction ors $s4 & $s5 and stores the
                                #result in $s4
```

```
lui $s5, 0x501e          #load reg $s5 with elements 4 & 5 from vector C
li $s6, 0x00000660       #load reg $s6 with elements 6 & 7 from vector C
or $s5, $s5, $s6         #this instruction ors $s5 & $s6 and stores the
                         #result in $s5

add $s6, $zero, $zero            #$s6 = 0



                         #the elements that's multiplying
srl $t0, $s0, 24
srl $t1, $s2, 24

                         #the element that's added
srl $t2, $s4, 24


mul $t0, $t0, $t1        #begin to multiply
andi $t0, $t0, 0x000000FF


                         #adds to the results
add $t0, $t0, $t2


sltiu $t1, $t0, 255      #if $t0 is < 255, $t1 is set to one, else 0
bne $t1, $zero, br1      #branches when the two regs are not equal
addiu $t0, $zero, 255            #if they are equal, we set to FF
```

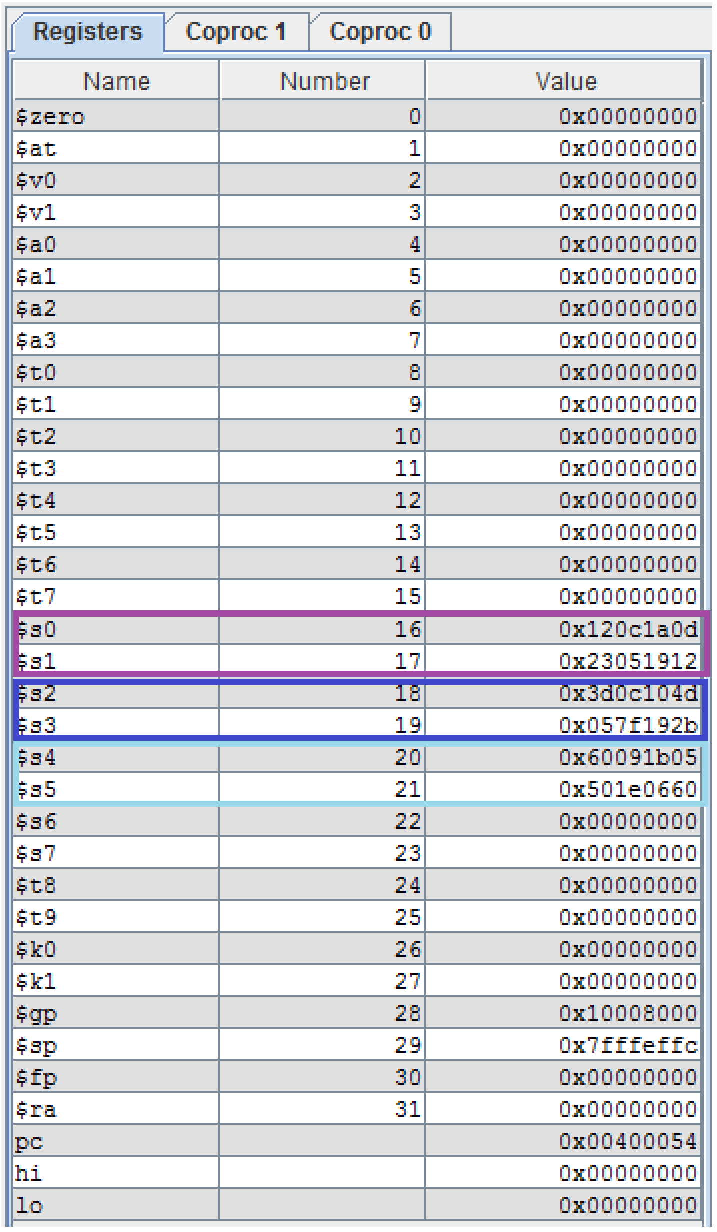
```
                                #1st jump is made here if bne
br1:
        sll $t0, $t0, 24        #shifts to the left & stores result in
                                #$t0

        add $s6, $t0, $s6       #adds to vector D




        srl $t0, $s0, 16        #shifts elements 0 - 1 to elements 2 - 3
        andi $t0, $t0, 0x000000FF
        srl $t1, $s2, 16        #shifts values into elements 3 - 4
        andi $t1, $t1, 0x000000FF

        srl $t2, $s4, 16        #shifts values into elements 3 - 4
        andi $t2, $t2, 0x000000FF

                                #multiply elements together
        mul $t0, $t0, $t1       #multipling elements from $t0 & $t1
        andi $t0, $t0, 0x000000FF

                                #add element to result
        add $t0, $t0, $t2       #adds $t0




        sltiu $t1, $t0, 255     #if $t0 is < 255, $t1 is set to one, else 0
        bne $t1, $zero, br2     #branches when the two regs are not equal
        addiu $t0, $zero, 255 #if they are equal, we set to FF
```

```
br2:
        andi $t0, $t0, 0x000000FF
        sll $t0, $t0, 16            #shifts to the left

        add $s6, $t0, $s6           #adds to vector D



        srl $t0, $s0, 8             #shifts elements into 1 - 3
        andi $t0, $t0, 0x000000FF
        srl $t1, $s2, 8             #shifts elements into 1 - 3
        andi $t1, $t1, 0x000000FF

        srl $t2, $s4, 8             #shifts elements into 1 - 3
        andi $t2, $t2, 0x000000FF


                                    #multiply elements together
        mul $t0, $t0, $t1           #multiplying elements from $t0 & $t1
        andi $t0, $t0, 0x000000FF

        add $t0, $t0, $t2           #adds result to $t0

        sltiu $t1, $t0, 255         #if $t1 is < 255, $t1 is set to one, else 0
        bne $t1, $zero, br3         #branches when the two regs are not equal
        addiu $t0, $zero, 255               #if they are equal, we set to FF

br3:
        andi $t0, $t0, 0x000000FF
```

```
	sll $t0, $t0, 8		#shifts elements into 1 - 3
	add $s6, $t0, $s6	#adds result to vector D
	andi $t0, $s0, 0x000000FF
	andi $t1, $s2, 0x000000FF
	andi $t2, $s4, 0x000000FF



	mul $t0, $t0, $t1	#multiply elements from $t0 & $t1
	andi $t0, $t0, 0x000000FF

	add $t0, $t0, $t2	#adds result to $t0

	sltiu $t1, $t0, 255	#if $t1 is < 255, $t1 is set to one, else 0
	bne $t1, $zero, br4	#branches when the two regs are not equal
	addiu $t0, $zero, 255		#if they are equal, we set to FF

br4:
	andi $t0, $t0, 0x000000FF
	add $s6, $t0, $s6	#adds result to vector D
	srl $t0, $s1, 24	#shifts the result into the vector
	srl $t1, $s3, 24	#shifts the result into the vector
	srl $t2, $s5, 24	#shifts the result into the vector


	mul $t0, $t0, $t1	#multiply elements from $t0 & $t1
	andi $t0, $t0, 0x000000FF
	add $t0, $t0, $t2	#adds result to $t0
	sltiu $t1, $t0, 255	#if $t1 is < 255, $t1 is set to one, else 0
	bne $t1, $zero, br5	#branches when the two regs are not equal
	addiu $t0, $zero, 255		#if they are equal, we set to FF

br5:
	sll $t0, $t0, 24		#shifts to the left & stores result in
					#$t0
	add $s7, $t0, $s7		#adds result to vector D

					#load elements to be multiplied
	srl $t0, $s1, 16		#shifts right elements to $t0

	andi $t0, $t0, 0x000000FF
	srl $t1, $s3, 16		#shifts values into elements 3 - 4
	andi $t1, $t1, 0x000000FF
	srl $t2, $s5, 16		#shifts values into elements 3 - 4
	andi $t2, $t2, 0x000000FF


	mul $t0, $t0, $t1		#multiplying elements from $t0 & $t1
	andi $t0, $t0, 0x000000FF
	add $t0, $t0, $t2		#adds result to $t0



	sltiu $t1, $t0, 255		#if $t1 is < 255, $t1 is set to one, else 0
	bne $t1, $zero, br6		#branches when the two regs are not equal
	addiu $t0, $zero, 255		#if they are equal, we set to FF

br6:
	andi $t0, $t0, 0x000000FF
	sll $t0, $t0, 16		#shifts values into elements 3 - 4
```

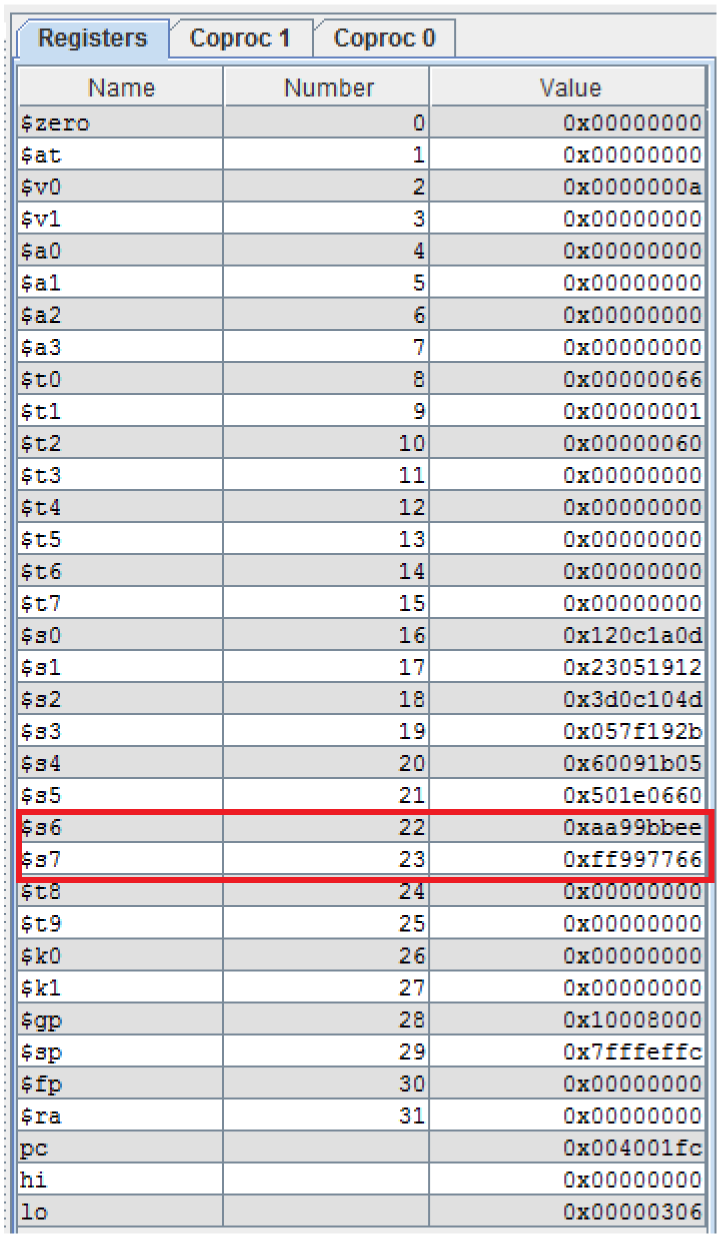
```
        add $s7, $t0, $s7        #adds result to vector D



        srl $t0, $s1, 8          #shifts elements into 1 - 3
        andi $t0, $t0, 0x000000FF
        srl $t1, $s3, 8          #shifts elements into 1 - 3
        andi $t1, $t1, 0x000000FF
      srl $t2, $s5, 8            #shifts elements into 1 - 3
      andi $t2, $t2, 0x000000FF




        mul $t0, $t0, $t1        #multiplying elements from $t0 & $t1
        andi $t0, $t0, 0x000000FF
        add $t0, $t0, $t2        #adds result to $t0

        sltiu $t1, $t0, 255      #if $t1 is < 255, $t1 is set to one, else 0
        bne $t1, $zero, br7      #branches when the two regs are not equal
        addiu $t0, $zero, 255          #if they are equal, we set to FF

br7:
        andi $t0, $t0, 0x000000FF
        sll $t0, $t0, 8
        add $s7, $t0, $s7        #adds result to vector D



        andi $t0, $s1, 0x000000FF
        andi $t1, $s3, 0x000000FF
        andi $t2, $s5, 0x000000FF
        mul $t0, $t0, $t1         #multiplying elements from $t0 & $t1
        andi $t0, $t0, 0x000000FF
        add $t0, $t0, $t2         #adds result to $t0

        sltiu $t1, $t0, 255       #if $t1 is < 255, $t1 is set to one, else 0
        bne $t1, $zero, br8       #branches when the two regs are not equal
        addiu $t0, $zero, 255             #if they are equal, we set to FF

br8:
      andi $t0, $t0, 0x000000FF
      add $s7, $t0, $s7           #adds result to vector D

exit:
        ori $v0, $zero, 10
      syscall

.data
```

VECTOR MULTIPLY & ADD BEFORE


Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x120c1a0d
$s1	17	0x23051912
$s2	18	0x3d0c104d
$s3	19	0x057f192b
$s4	20	0x60091b05
$s5	21	0x501e0660
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400054
hi		0x00000000
lo		0x00000000

VECTOR MULTIPLY & ADD AFTER



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000066
$t1	9	0x00000001
$t2	10	0x00000060
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x120c1a0d
$s1	17	0x23051912
$s2	18	0x3d0c104d
$s3	19	0x057f192b
$s4	20	0x60091b05
$s5	21	0x501e0660
$s6	22	0xaa99bbee
$s7	23	0xff997766
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x004001fc
hi		0x00000000
lo		0x00000306

Vector Multiply Even Integer

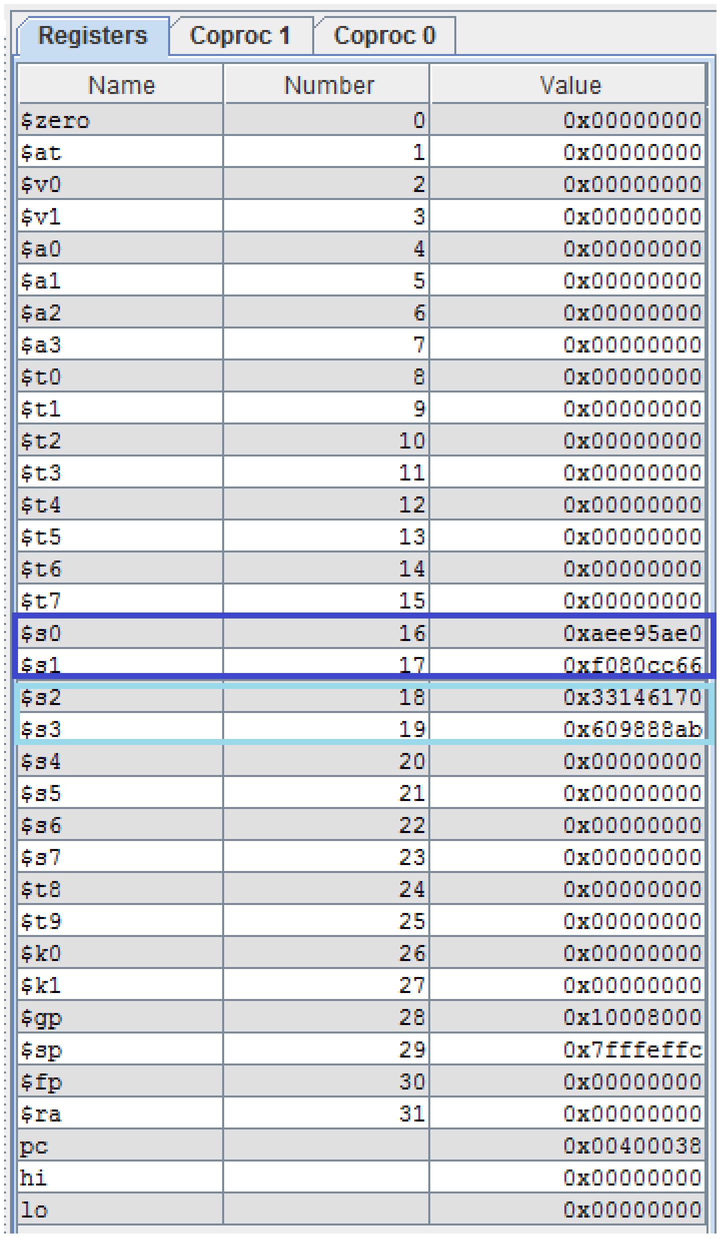
```
# File Name: vector_mult_even_int.asm
 # Version: 1.0
 # Date: Nov 24, 2018
 # Programmer: Jocelyn Espitia & Thomas Vugia
 #
 # Description: vec_mule d, a, b; multiplies the even elements of a and b then adds
 #                  the result to vector d.
 #
 # Register Usage:
 #       vector A will be shown with $s0 & $s1
 #       vector B will be shown with $s2 & $s3
 #       vector D will be shown with $s6 & s7
 #**************************************************************************


 #**************************************************************************
 #                           CODE SEGMENT
 #**************************************************************************
         .text
         .globl main

 main:
         lui $s0, 0xAEE9        #loads $s0 with elements 0 - 1 of vector A
         li  $s1, 0x00005AE0    #loads $s1 with elements 2 - 3 of vector A
         or  $s0, $s0, $s1      #this instruction ors $s0 & $s1 and stores the
                                #result in $s0

         lui $s1, 0xF080        #loads $s1 with elements 4 - 5 of vector A
```

```
li  $s2, 0x0000CC66   #loads $s2 with elements 6 - 7 of vector A
or  $s1, $s1, $s2     #this instruction ors $s1 & $s2 and stores the
                      #result in $s1

add $s2, $zero, $zero         #$s2 = 0

lui $s2, 0x3314       #loads $s2 with elements 0 - 1 of vector B
li  $s3, 0x00006170   #loads $s3 with elements 2 - 3 of vector B
or  $s2, $s2, $s3     #this instruction ors $s2 & $s3 and stores the
                      #result in $s2

lui $s3, 0x6098       #loads $s3 with elements 4 - 5 of vector B
li  $s4, 0x000088AB   #loads $s4 with elements 6 - 7 of vector B
or  $s3, $s3, $s4     #this instruction ors $s3 & $s4 and stores the
                      #result in $s3

add $s4, $zero, $zero         #$s4 = 0



                      #first set of multipled elements
srl $t0, $s0, 24      #shifts org element 0 of vec A to element 3 of $t0
srl $t1, $s2, 24      #shifts org element 0 of vec B to element 3 of $t1
mul $t0, $t0, $t1     #multiplies the elements of $t0 & $t1
sll $t0, $t0, 16      #shifts the result of $t0 to element 0
add $s6, $s6, $t0     #adds result to result vector D
```

```
                        #second set of multiplied elements
  srl $t0, $s0, 8       #shifts org elements 0 - 2 of vec A to element 1 - 3 of $t0

  andi $t0, $t0, 0x000000ff
  srl $t1, $s2, 8       #shifts org elements 0 - 2 of vec B to element 1 - 3 of $t1
  andi $t1, $t1, 0x000000ff
  mul $t0, $t0, $t1     #multiplies the elements of $t0 & $t1
  add $s6, $s6, $t0     #adds result to result vector D



                        #third set of multiplied elements
  srl $t0, $s1, 24      #shifts org element 3 of vec A to element 3 of $t0
  srl $t1, $s3, 24      #shifts org element 3 of vec B to element 3 of $t1
  mul $t0, $t0, $t1     #multiplies the elements of $t0 & $t1
  sll $t0, $t0, 16      #shifts the result of $t0 to element 0
  add $s7, $s7, $t0     #adds result to result vector D



                        #fourth set of multiplied elements
srl $t0, $s1, 8         #shifts org elements 4 - 6 of vec A to element 1 - 3 of $t0
andi $t0, $t0, 0x000000ff
srl $t1, $s3, 8         #shifts org elements 4 - 6 of vec B to element 1 - 3 of $t1
andi $t1, $t1, 0x000000ff
mul $t0, $t0, $t1       #multiplies the elements of $t0 & $t1
add $s7, $s7, $t0#adds result to result vector D
```

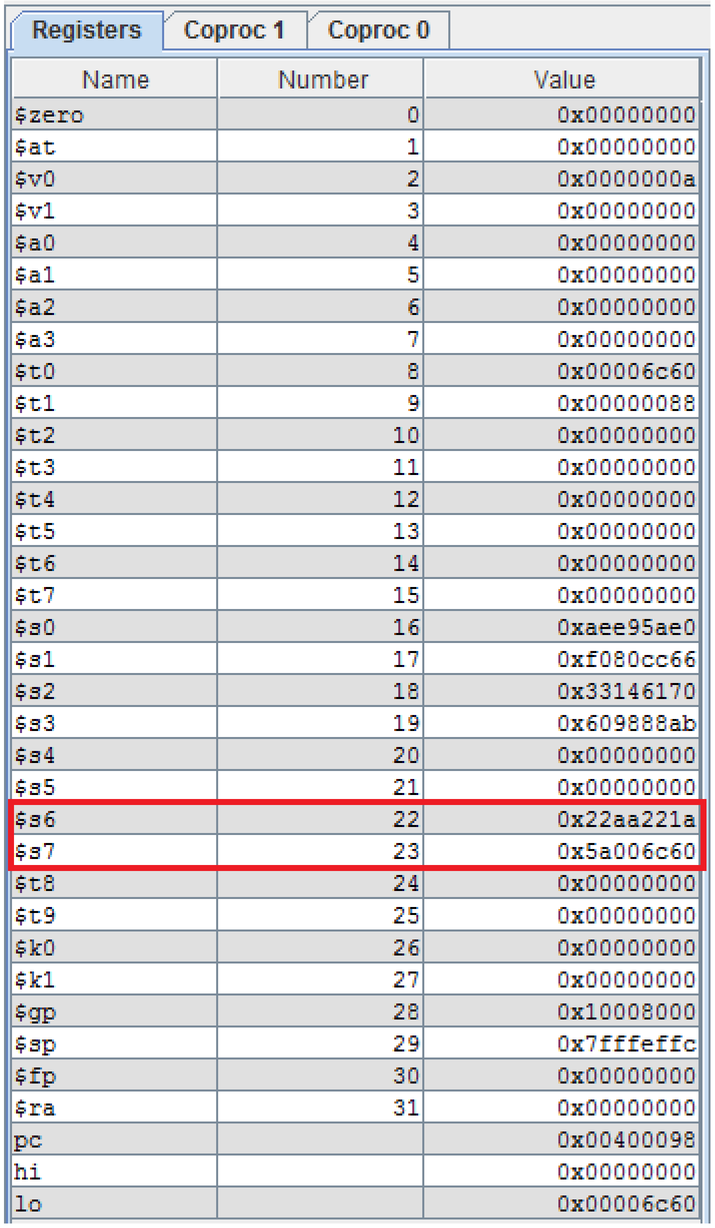
```
exit:
        ori $v0, $zero, 10
        syscall
.data
```

VECTOR MULTIPLY EVEN INTEGER BEFORE



Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0xaee95ae0
$s1	17	0xf080cc66
$s2	18	0x33146170
$s3	19	0x609888ab
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400038
hi		0x00000000
lo		0x00000000

VECTOR MULTIPLY EVEN INTEGER AFTER



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00006c60
$t1	9	0x00000088
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0xaee95ae0
$s1	17	0xf080cc66
$s2	18	0x33146170
$s3	19	0x609888ab
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x22aa221a
$s7	23	0x5a006c60
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400098
hi		0x00000000
lo		0x00006c60

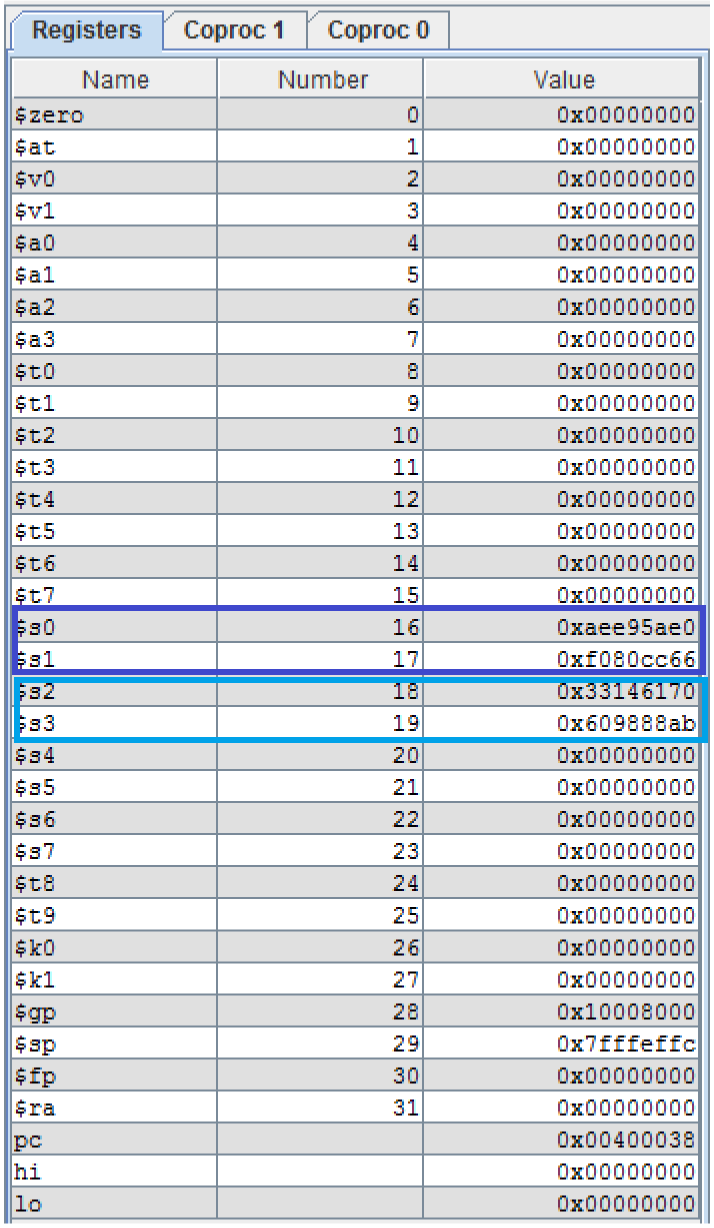
Vector Multiply Odd Integer

```
# File Name: vector_mult_odd_int.asm
# Version: 1.0
# Date: Nov 24, 2018
# Programmer: Jocelyn Espitia & Thomas Vugia
#
# Description: vec_mulo d, a, b; multiplies the odd elements of a and b then adds
#                    the result to vector d.
#
# Register Usage:
#         vector A will be shown with $s0 & $s1
#         vector B will be shown with $s2 & $s3
#         vector D will be shown with $s6 & s7
#********************************************************************************



#********************************************************************************
#                              CODE SEGMENT
#********************************************************************************
          .text
          .globl main

main:
          lui $s0, 0xAEE9         #loads $s0 with elements 0 - 1 of vector A
          li  $s1, 0x00005AE0     #loads $s1 with elements 2 - 3 of vector A
          or  $s0, $s0, $s1       #this instruction ors $s0 & $s1 and stores the
                                  #result in $s0

          lui $s1, 0xF080         #loads $s1 with elements 4 - 5 of vector A
```

```
li  $s2, 0x0000CC66   #loads $s2 with elements 6 - 7 of vector A
or  $s1, $s1, $s2     #this instruction ors $s1 & $s2 and stores the
                      #result in $s0

add $s2, $zero, $zero          #$s2 = 0

lui $s2, 0x3314       #loads $s2 with elements 0 - 1 of vector B
li  $s3, 0x00006170   #loads $s3 with elements 2 - 3 of vector B
or  $s2, $s2, $s3     #this instruction ors $s2 & $s3 and stores the
                      #result in $s2

lui $s3, 0x6098       #loads $s3 with elements 4 - 5 of vector B
li  $s4, 0x000088AB   #loads $s4 with elements 6 - 7 of vector B
or  $s3, $s3, $s4     #this instruction ors $s3 & $s4 and stores the
                      #result in $s3

add $s4, $zero, $zero          #$s4 = 0



                      #first set of multiplied elements
srl $t0, $s0, 16      #shifts org element 0 - 1 of vec A to element 2 - 3 of $t0
andi $t0, $t0, 0x000000ff
srl $t1, $s2, 16      #shifts org element 0 - 1 of vec B to element 2 - 3 of $t0
andi $t1, $t1, 0x000000ff
mul $t0, $t0, $t1     #multiplies the elements of $t0 & $t1

sll $t0, $t0, 16      #shifts results with elements 2 - 3 to 0 - 1
add $s6, $s6, $t0     #adds result to result vector D
```

```
                    #second set of multiplied elements
    andi $t0, $s0, 0x000000ff
    andi $t1, $s2, 0x000000ff
    mul $t0, $t0, $t1   #multiplies the elements of $t0 & $t1
    add $s6, $s6, $t0   #adds result to result vector D



                    #third set of multiplied elements
    srl $t0, $s1, 16    #shifts org element 4 - 5 of vec A to element 2 - 3 of $t0
    andi $t0, $t0, 0x000000ff
    srl $t1, $s3, 16    #shifts org element 4 - 5 of vec B to element 2 - 3 of $t0
    andi $t1, $t1, 0x000000ff
    mul $t0, $t0, $t1   #multiplies the elements of $t0 & $t1
    sll $t0, $t0, 16    #shifts results with elements 2 - 3 to 0 - 1
    add $s7, $s7, $t0   #adds result to result vector D



                    #fourth set of multiplied elements
    andi $t0, $s1, 0x000000ff
    andi $t1, $s3, 0x000000ff
    mul $t0, $t0, $t1   #multiplies the elements of $t0 & $t1
    add $s7, $s7, $t0   #adds result to result vector D
exit:
    ori $v0, $zero, 10
    syscall
.data
```

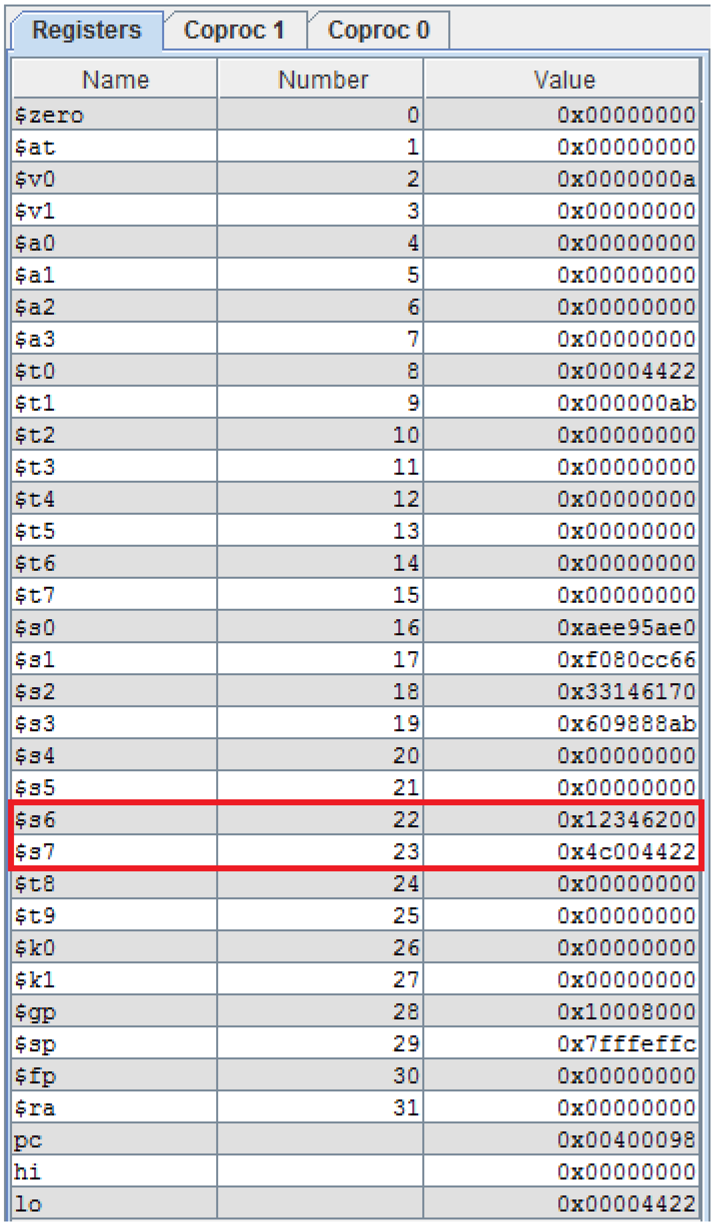
VECTOR MULTIPLY ODD INTEGER



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0xaee95ae0
$s1	17	0xf080cc66
$s2	18	0x33146170
$s3	19	0x609888ab
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400038
hi		0x00000000
lo		0x00000000

VECTOR MULTIPLY ODD INTEGER AFTER


Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00004422
$t1	9	0x000000ab
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0xaee95ae0
$s1	17	0xf080cc66
$s2	18	0x33146170
$s3	19	0x609888ab
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x12346200
$s7	23	0x4c004422
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400098
hi		0x00000000
lo		0x00004422

Vector Multiply Sum Saturated

```
# File Name: vector_mult_sum_sat.asm
# Version: 1.0
# Date: Nov 24, 2018
# Programmer: Jocelyn Espitia & Thomas Vugia
#
# Description: vec_msums d, a, b, c; multiplies the elements of vector a and b and adds
#                   the results (temp) with a 16-bit vector C. The way it is added is with
#                   2 elements of the temp results correspond to the 1st element of
#                   vector C. The final results are stored in vector d.
#
# Register Usage:
#         vector A will be shown with $s0 & $s1
#         vector B will be shown with $s2 & $s3
#         vector C will be shown with $s4 & $s5
#         vector D will be shown with $s6 & s7
#***************************************************************************


#***************************************************************************
#                             CODE SEGMENT
#***************************************************************************
        .text
        .globl main

main:
        lui $s0, 0x230C          #loads $s0 with elements 0 - 1 of vector A
        li  $s1, 0x0000F14D      #loads $s1 with elements 2 - 3 of vector A
        or  $s0, $s0, $s1        #this instruction ors $s0 & $s1 and stores the
```

```
                        #result in $s0

lui $s1, 0x5C7F         #loads $s1 with elements 4 - 5 of vector A
li  $s2, 0x0000191A     #loads $s2 with elements 6 - 7 of vector A
or  $s1, $s1, $s2       #this instruction ors $s1 & $s2 and stores the
                        #result in $s0

add $s2, $zero, $zero           #$s2 = 0

lui $s2, 0xA30C         #loads $s2 with elements 0 - 1 of vector B
li  $s3, 0x00005bfd     #loads $s3 with elements 2 - 3 of vector B
or  $s2, $s2, $s3       #this instruction ors $s2 & $s3 and stores the
                        #result in $s2

lui $s3, 0xC5FF         #loads $s3 with elements 4 - 5 of vector B
li  $s4, 0x0000C9EE     #loads $s4 with elements 6 - 7 of vector B
or  $s3, $s3, $s4       #this instruction ors $s3 & $s4 and stores the
                        #result in $s3

add $s4, $zero, $zero           #$s4 = 0

lui $s4, 0x609E         #loads $s4 with element 0 of vector C
li  $s5, 0x000019F7     #loads $s5 with element 1 of vector C
or  $s4, $s4, $s5       #this instruction ors $s4 & $s5 and stores the
                        #result in $s4

lui $s5, 0x4567         #loads $s5 with element 2 of vector C
li  $s6, 0x00000766     #loads $s6 with element 3 of vector C
or  $s5, $s5, $s6       #this instruction ors $s5 & $s6 and stores the
                        #result in $s5
```

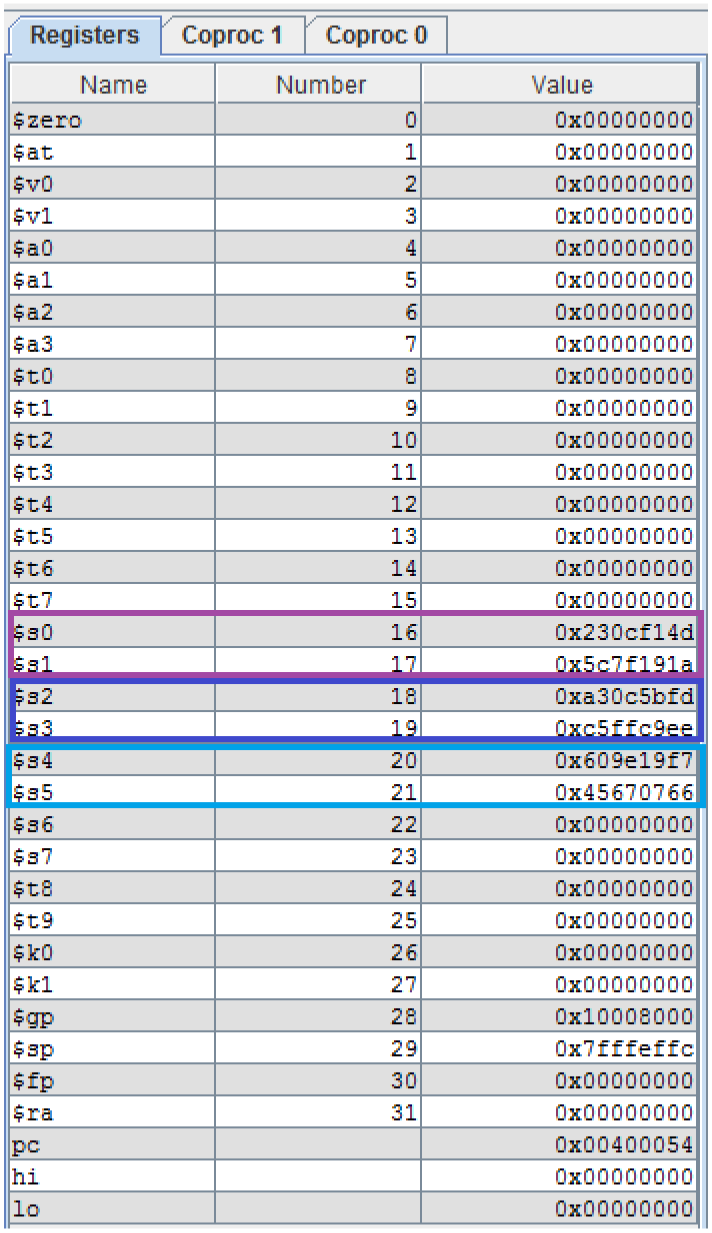
```
add $s6, $zero, $zero          #$s6 = 0
li  $t7, 0x00010000    #loads $t7 with 00010000


                       #load elements to be multiplied
srl $t0, $s0, 24       #shifts original element 0 of vector A to element
                       #3 in $t0
srl $t1, $s2, 24       #shifts original element 0 of vector B to element
                       #3 in $t0
mul $t2, $t0, $t1      #multiplies $t0 and $t1 and stores it in $t2



srl $t0, $s0, 16       #shifts original elements 0 - 1 of vector A to element
                       #2 - 3 in $t0
andi $t0, $t0, 0x000000FF
srl $t1, $s2, 16       #shifts original elements 0 - 1 of vector B to element
                       #2 - 3 in $t0
andi $t1, $t1, 0x000000FF
mul $t3, $t0, $t1      #multiplies $t0 and $t1 and stores it in $t3



srl $t0, $s0, 8        #shifts original elements 0 - 2 of vector A to element
                       #1 - 3 in $t0
andi $t0, $t0, 0x000000FF
srl $t1, $s2, 8        #shifts original elements 0 - 2 of vector B to element
                       #1 - 3 in $t0
andi $t1, $t1, 0x000000FF
mul $t4, $t0, $t1      #multiplies $t0 and $t1 and stores it in $t4
```

```
        andi $t0, $s0, 0x000000FF
        andi $t1, $s2, 0x000000FF
        mul $t5, $t0, $t1       #multiplies $t0 and $t1 and stores it in $t5


        add $t2, $t2, $t3       #add $t2 & $t3, stores in $t2
        add $t3, $t4, $t5       #add $t4 & $t5, stores in $t3


        srl $t0, $s4, 16        #result is shifted into element 2 - 3
        andi $t1, $s4, 0x0000FFFF


        add $t0, $t0, $t2       #add $t0 & $t2, stores in $t0
        slt $t4, $t0, $t7       #if $t0 < $t7, $t4 is set to 1, else 0


        bne $t4, $zero, br1     #branches when the two regs are not equal
        li  $t0, 65535

                                #now we begin to add the multiplicated
                                #results with elements from vec C

br1:
        add $t1, $t1, $t3       #add $t1 & $t3, stores in $t1
        slt $t4, $t1, $t7       #if $t0 < $t7, $t4 is set to 1, else 0
        bne $t4, $zero, br2     #branches when the two regs are not equal
        li  $t1, 65535
```

```
br2:
        sll $t0, $t0, 16            #shifts elements from 2 - 3 to 0 - 1
        or $s6, $t0, $t1            #this instruction ors $t0 & $t1 and stores the
                                    #result in $s6


        srl $t0, $s1, 24            #shifts original element 4 of vector A to
                                    #element 3
        srl $t1, $s3, 24            #shifts original element 4 of vector B to
                                    #element 3
        mul $t2, $t0, $t1           #multiply $t0 & $t1, store in $t2



        srl $t0, $s1, 16            #shift elements 4 - 5 of vector A to 2 - 3
        andi $t0, $t0, 0x000000FF
        srl $t1, $s3, 16            #shift elements 4 - 5 of vector B to 2 - 3
        andi $t1, $t1, 0x000000FF
        mul $t3, $t0, $t1           #multiply $t0 & $t1, store in $t3



        srl $t0, $s1, 8             #shift elements 4 - 6 of vector A to 1 - 3
        andi $t0, $t0, 0x000000FF
        srl $t1, $s3, 8             #shift elements 4 - 6 of vector B to 1 - 3
        andi $t1, $t1, 0x000000FF
        mul $t4, $t0, $t1           #multiply $t0 & $t1, store in $t4


        andi $t0, $s1, 0x000000FF
        andi $t1, $s3, 0x000000FF
        mul $t5, $t0, $t1        #multiply $t0 & $t1, store in $t4



        add $t2, $t2, $t3        #add $t2 & $t3, store result in $t2
        add $t3, $t4, $t5        #add $t4 & $t5, store result in $t3



        srl $t0, $s5, 16         #shifts element 3 of vec C into elements 2 - 3
        andi $t1, $s5, 0x0000FFFF



        add $t0, $t0, $t2        #add $t0 & $t2, store result in $t0
        slt $t4, $t0, $t7        #if $t0 < $t7, $t4 is set to 1, else 0



        bne $t4, $zero, br3      #branches when the two regs are not equal
        li  $t0,  65535

br3:
        add $t1, $t1, $t3        #add $t1 & $t3, store result in $t1
        slt $t4, $t1, $t7        #if $t1 < $t7, $t4 is set to 1, else 0

        bne $t4, $zero, br4  #branches when the two regs are not equal
        li $t1, 65535

br4:
        sll $t0, $t0, 16         #shift results to the right
        or $s7, $t0, $t1         #adds result into final vector D
```

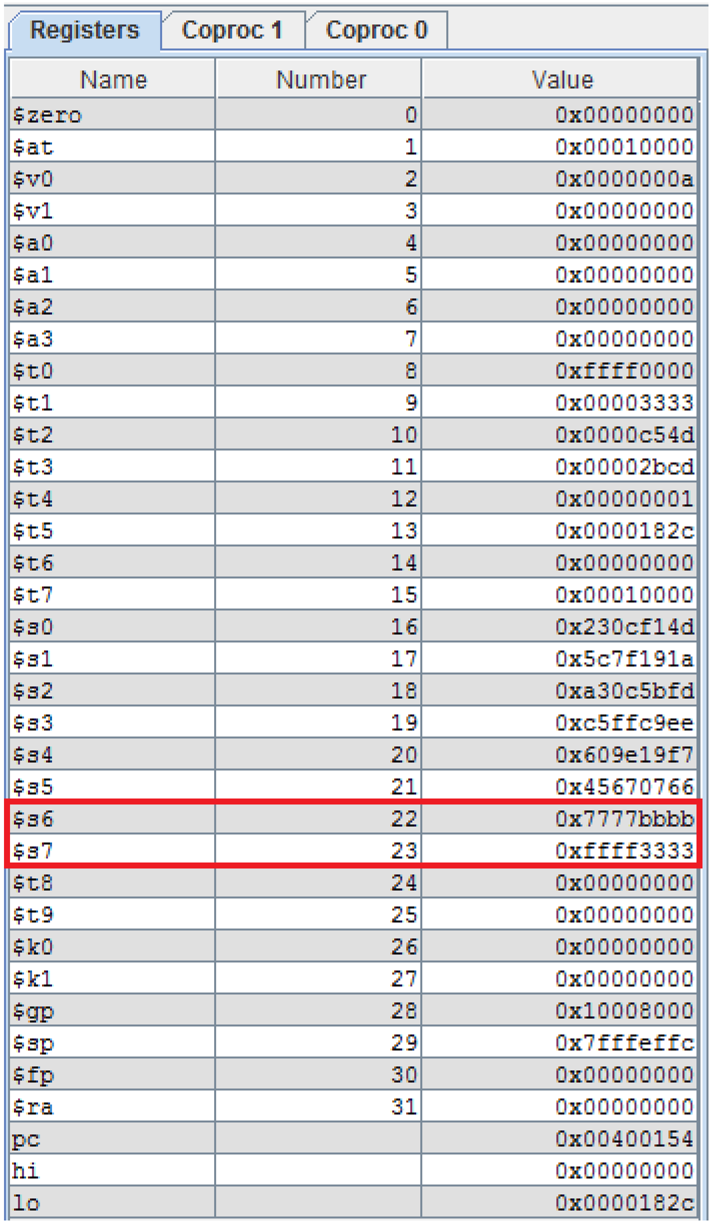
```
exit:
        ori $v0, $zero, 10
        syscall
 .data
```

VECTOR MULTIPLY SUM SATURATED BEFORE



Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x230cf14d
$s1	17	0x5c7f191a
$s2	18	0xa30c5bfd
$s3	19	0xc5ffc9ee
$s4	20	0x609e19f7
$s5	21	0x45670766
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400054
hi		0x00000000
lo		0x00000000

VECTOR MULTIPLY SUM SATURATED AFTER



Registers | Coproc 1 | Coproc 0

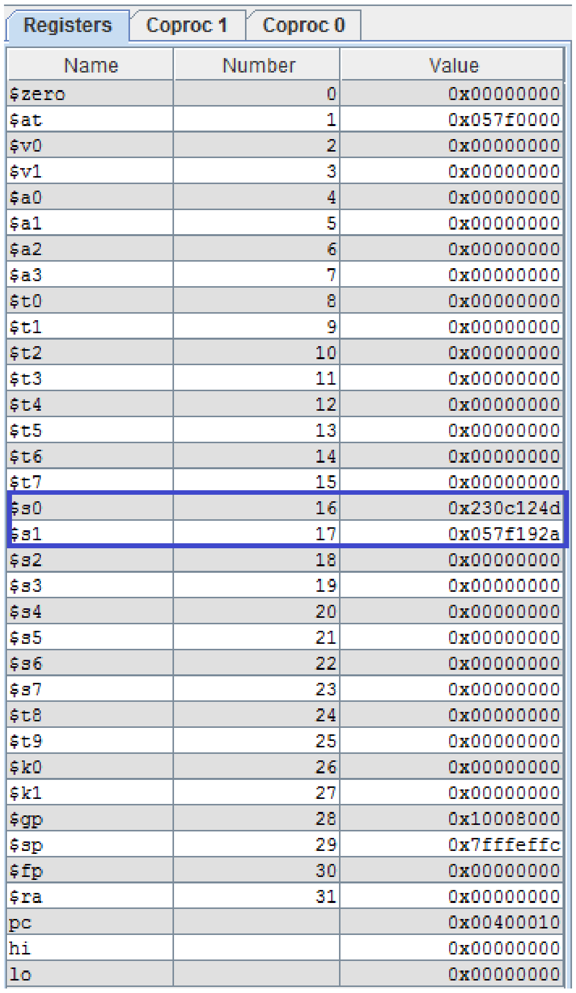
Name	Number	Value
$zero	0	0x00000000
$at	1	0x00010000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0xffff0000
$t1	9	0x00003333
$t2	10	0x0000c54d
$t3	11	0x00002bcd
$t4	12	0x00000001
$t5	13	0x0000182c
$t6	14	0x00000000
$t7	15	0x00010000
$s0	16	0x230cf14d
$s1	17	0x5c7f191a
$s2	18	0xa30c5bfd
$s3	19	0xc5ffc9ee
$s4	20	0x609e19f7
$s5	21	0x45670766
$s6	22	0x7777bbbb
$s7	23	0xffff3333
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400154
hi		0x00000000
lo		0x0000182c

Vector Splat

```
# File Name: vector_splat.asm
# Version: 1.0
# Date: Nov 1, 2018
# Programmer: Jocelyn Espitia & Thomas Vugia
#
# Description: vec_splat d, a, b; copies an element from one vector, into all the
#                       elements of the result vector
#
# Register Usage:
#        vector A will be shown with $s0 & $s1
#        vector D will be shown with $s2 & $s3
#*****************************************************************************


#*****************************************************************************
#                               CODE SEGMENT
#*****************************************************************************
         .text
         .globl main

main:
         li $s0, 0x230C124D    #initializing vector A with elements 0 - 3
         li $s1, 0x057F192A    #initializing vector A with elements 4 - 7
         li $s2, 0x00000000    #vector B will be the element that we want to copy over to D
         li $s3, 0x00000005    #using this register to store vector D with element 5 of vec A


     add $t0, $s0, $zero       #places the elements 0 - 3 of vector A into temp regs
add $t1, $s0, $zero     #places the elements 0 - 3 of vector A into temp regs
add $t2, $s0, $zero     #places the elements 0 - 3 of vector A into temp regs
add $t3, $s0, $zero     #places the elements 0 - 3 of vector A into temp regs
add $t4, $s1, $zero     #places the elements 4 - 7 of vector A into temp regs
add $t5, $s1, $zero     #places the elements 4 - 7 of vector A into temp regs
add $t6, $s1, $zero     #places the elements 4 - 7 of vector A into temp regs
add $t7, $s1, $zero     #places the elements 4 - 7 of vector A into temp regs

  sll $t0, $t0, 0
  srl $t0, $t0, 24
  sll $t1, $t1, 8       #shifts 1 - 3 of vector A to 1 - 3 in $t0
  srl $t1, $t1, 24      #shifts element 1 of vector A to element 3 of $t1
  sll $t2, $t2, 16      #shifts 2 - 3 of vector A to 0 - 1 in $t2
  srl $t2, $t2, 24      #shifts element 2 of vector A to element 3 of $t2
  sll $t3, $t3, 24


  srl $t3, $t3, 24      #element 0 shifted to element 3
  sll $t4, $t4, 0
  srl $t4, $t4, 24      #element 0 shifted to element 3
  sll $t5, $t5, 8       #element 5 - 7 of vec A is shifted to 0 - 2 of $t5
  srl $t5, $t5, 24      #element 0 shifted to element 3
  sll $t6, $t6, 16
  srl $t6, $t6, 24
  sll $t7, $t7, 24      #shifts element 7 of vec A to element 0 of $t7
  srl $t7, $t7, 24      #element 0 shifted to element 3

  addi $t8, $zero, 0

  bne $t8, $s3, loop1   #branches when the two regs are not equal, adds a 1 if so
```

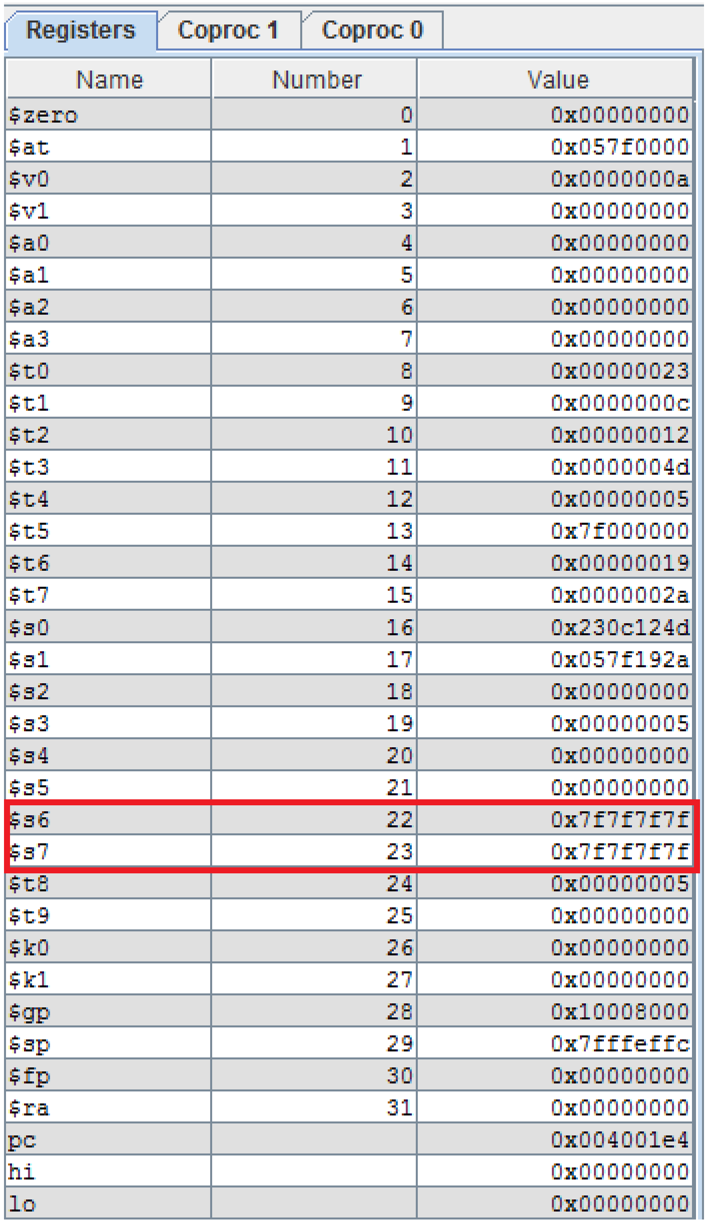
```
        add $s6, $t0, $s6
        sll $t0, $t0, 8
        add $s6, $t0, $s6
        sll $t0, $t0, 8
        add $s6, $t0, $s6
        sll $t0, $t0, 8
        add $s6, $t0, $s6
        add $s7, $s6, $zero
        j exit
 loop1:
        addi $t8, $t8, 1        #adds a one to $t8
        bne $t8, $s3, loop2
        add $s6, $t1, $s6
        sll $t1, $t1, 8
        add $s6, $t1, $s6
        sll $t1, $t1, 8
        add $s6, $t1, $s6
        sll $t1, $t1, 8
        add $s6, $t1, $s6
        add $s7, $s6, $zero
        j exit
loop2:
        addi $t8, $t8, 1        #adds a one to $t8
        bne $t8, $s3, loop3
        add $s6, $t2, $s6
        sll $t2, $t2, 8
        add $s6, $t2, $s6
        sll $t2, $t2, 8
        add $s6, $t2, $s6
        sll $t2, $t2, 8
        add $s6, $t2, $s6
        add $s7, $s6, $zero
        j exit
loop3:
        addi $t8, $t8, 1        #adds a one to $t8
        bne $t8, $s3, loop4
        add $s6, $t3, $s6
        sll $t3, $t3, 8
        add $s6, $t3, $s6
        sll $t3, $t3, 8
        add $s6, $t3, $s6
        sll $t3, $t3, 8

        add $s6, $t3, $s6
        add $s7, $s6, $zero
        j exit
loop4:
        addi $t8, $t8, 1        #adds a one to $t8
        bne $t8, $s3, loop5
        add $s6, $t4, $s6
        sll $t4, $t4, 8
        add $s6, $t4, $s6
        sll $t4, $t4, 8
        add $s6, $t4, $s6
        sll $t4, $t4, 8
        add $s6, $t4, $s6
        add $s7, $s6, $zero
        j exit
loop5:
        addi $t8, $t8, 1        #adds a one to $t8
        bne $t8, $s3, loop6
        add $s6, $t5, $s6       #adds element 5 of vec A to vec D
        sll $t5, $t5, 8
        add $s6, $t5, $s6       #adds element 5 of vec A to vec D
        sll $t5, $t5, 8
        add $s6, $t5, $s6       #adds element 5 of vec A to vec D
        sll $t5, $t5, 8
        add $s6, $t5, $s6       #adds element 5 of vec A to vec D
        add $s7, $s6, $zero
        j exit
loop6:
        addi $t8, $t8, 1
        bne $t8, $s3, loop7
        add $s6, $t6, $s6
        sll $t6, $t6, 8
        add $s6, $t6, $s6
        sll $t6, $t6, 8
        add $s6, $t6, $s6
        sll $t6, $t6, 8
        add $s6, $t6, $s6
        add $s7, $s6, $zero
        j exit
loop7:
        addi $t8, $t8, 1
        bne $t8, $s3, loopexit
        add $s6, $t7, $s6
        sll $t7, $t7, 8
        add $s6, $t7, $s6
        sll $t7, $t7, 8
        add $s6, $t7, $s6
        sll $t7, $t7, 8
        add $s6, $t7, $s6
        add $s7, $s6, $zero
        j exit
loopexit:
        j exit


exit:
        ori $v0, $zero, 10
        syscall
.data
```

VECTOR SPLAT BEFORE



Name	Number	Value
$zero	0	0x00000000
$at	1	0x057f0000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x230c124d
$s1	17	0x057f192a
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400010
hi		0x00000000
lo		0x00000000

VECTOR SPLAT AFTER


Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x057f0000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000023
$t1	9	0x0000000c
$t2	10	0x00000012
$t3	11	0x0000004d
$t4	12	0x00000005
$t5	13	0x7f000000
$t6	14	0x00000019
$t7	15	0x0000002a
$s0	16	0x230c124d
$s1	17	0x057f192a
$s2	18	0x00000000
$s3	19	0x00000005
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x7f7f7f7f
$s7	23	0x7f7f7f7f
$t8	24	0x00000005
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x004001e4
hi		0x00000000
lo		0x00000000

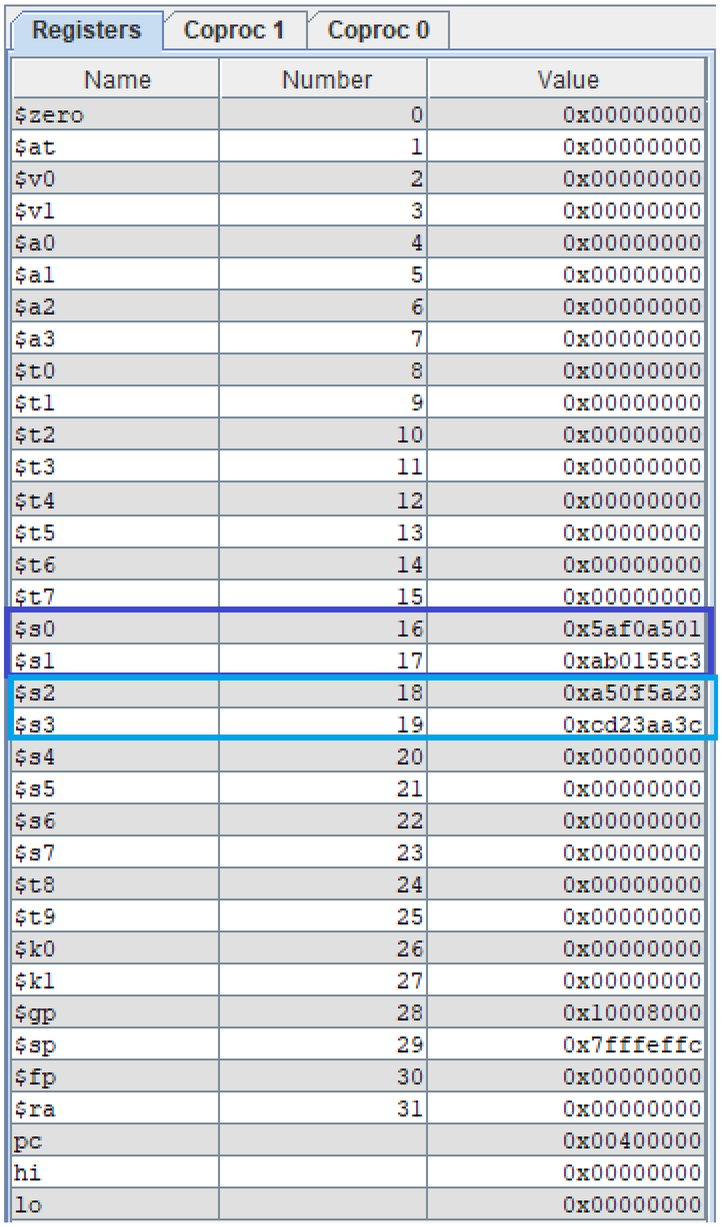
Vector Merge Low

```
# File name:          CECS341_Projecs1_vec_mergel_7.asm
# Version:            1.0
# Date:               November 5th, 2018
# Programmers:        Thomas Vugia, Jocelyn Espitia
#
# Description:        Designing a customized instruction for vectors, specifically
#                     for a vector merge low (vec_mergel d, a, b). This instruction
#                     takes the low elements of vectors a and b and stores them
#                     into the even and odd elements of vector d respectively.
#                     That is, all low elements of vector a are stored into
#                     the even elements of vector d and all low elements of
#                     vector b are stored into the odd elements of vector d.
#
# Register usage:     $s0 - first half of vector a
#                     $s1 - second half of vector a
#                     $s2 - first half of vector b
#                     $s3 - second half of vector b
#                     $s4 - first half of vector d
#                     $s5 - second half of vector d
#                     $t0 - temp register for bit manipulation
#                     $t1 - temp register for storing results into vector d
#
# Notes:              This instruction will only work on vectors with 8 elements
#                     (or 64 bits) each. Vectors smaller or larger than
#                     8 elements will not be read and processed correctly
#                     according to this program.
#
#*******************************************************************************
```

```
        #******************************************************
        #               M A I N   C O D E S E G M E N T
        #******************************************************

        .text
        .global         main

main:
                                        #loading in registers for vectors
                                        #with pre-set value examples
                li      $s0, 0x5AF0A501 #first half of vector a
                li      $s1, 0xAB0155C3 #second half of vector a
                li      $s2, 0xA50F5A23 #first half of vector b
                li      $s3, 0xCD23AA3C #second half of vector b

                li      $t0, 0xFF000000 #temp register for bit manipulation
                                        #starting in 1st element of vectors a & b

vec3_0:         and     $t1, $s1, $t0   #storing 5th element of vector a
                or      $s4, $s4, $t1   #into 1st element of vector c

vec3_1:         and     $t1, $s3, $t0   #storing 5th element of vector b
                srl     $t1, $t1, 8
                or      $s4, $s4, $t1   #into 2nd element of vector c

                srl     $t0, $t0, 8     #moving to 6th element of vectors a and b

vec3_2:         and     $t1, $s1, $t0   #storing 6th element of vector a
```

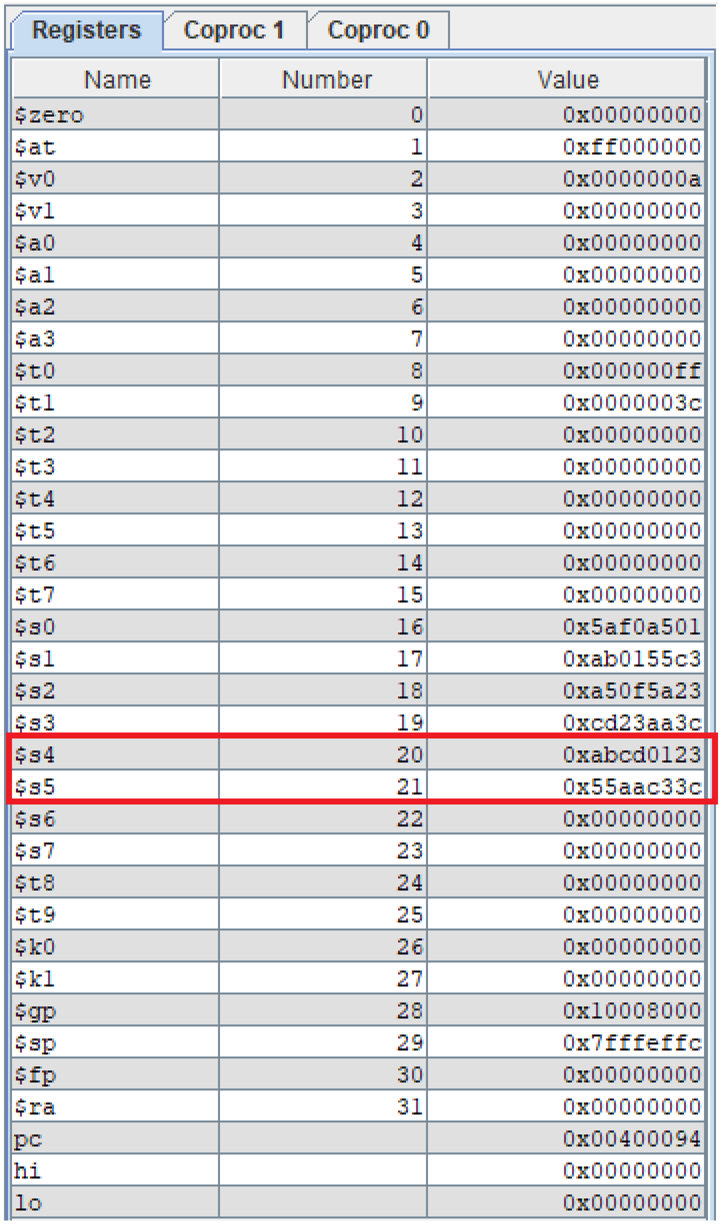
```
                srl     $t1, $t1, 8
                or      $s4, $s4, $t1           #into 3rd element of vector c

vec3_3:         and     $t1, $s3, $t0           #storing 6th element of vector b
                srl     $t1, $t1, 16
                or      $s4, $s4, $t1           #into 4th element of vector c

                srl     $t0, $t0, 8             #moving to 7th element of vectors a and b

vec3_4:         and     $t1, $s1, $t0           #storing 7th element of vector a
                sll     $t1, $t1, 16
                or      $s5, $s5, $t1           #into 5th element of vector c

vec3_5:         and     $t1, $s3, $t0           #storing 7th element of vector b
                sll     $t1, $t1, 8
                or      $s5, $s5, $t1           #into 6th element of vector c

                srl     $t0, $t0, 8             #moving to 8th element of vectors a and b

vec3_6:         and     $t1, $s1, $t0           #storing 8th element of vector a
                sll     $t1, $t1, 8
                or      $s5, $s5, $t1           #into 7th element of vector c

vec3_7:         and     $t1, $s3, $t0           #storing 8th element of vector b
                or      $s5 $s5, $t1            #into 8th element of vector c

exit:           ori     $v0, $zero, 10
                syscall
```

VECTOR MERGE LOW BEFORE



Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x5af0a501
$s1	17	0xab0155c3
$s2	18	0xa50f5a23
$s3	19	0xcd23aa3c
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

VECTOR MERGE LOW AFTER


Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0xff000000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x000000ff
$t1	9	0x0000003c
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x5af0a501
$s1	17	0xab0155c3
$s2	18	0xa50f5a23
$s3	19	0xcd23aa3c
$s4	20	0xabcd0123
$s5	21	0x55aac33c
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400094
hi		0x00000000
lo		0x00000000

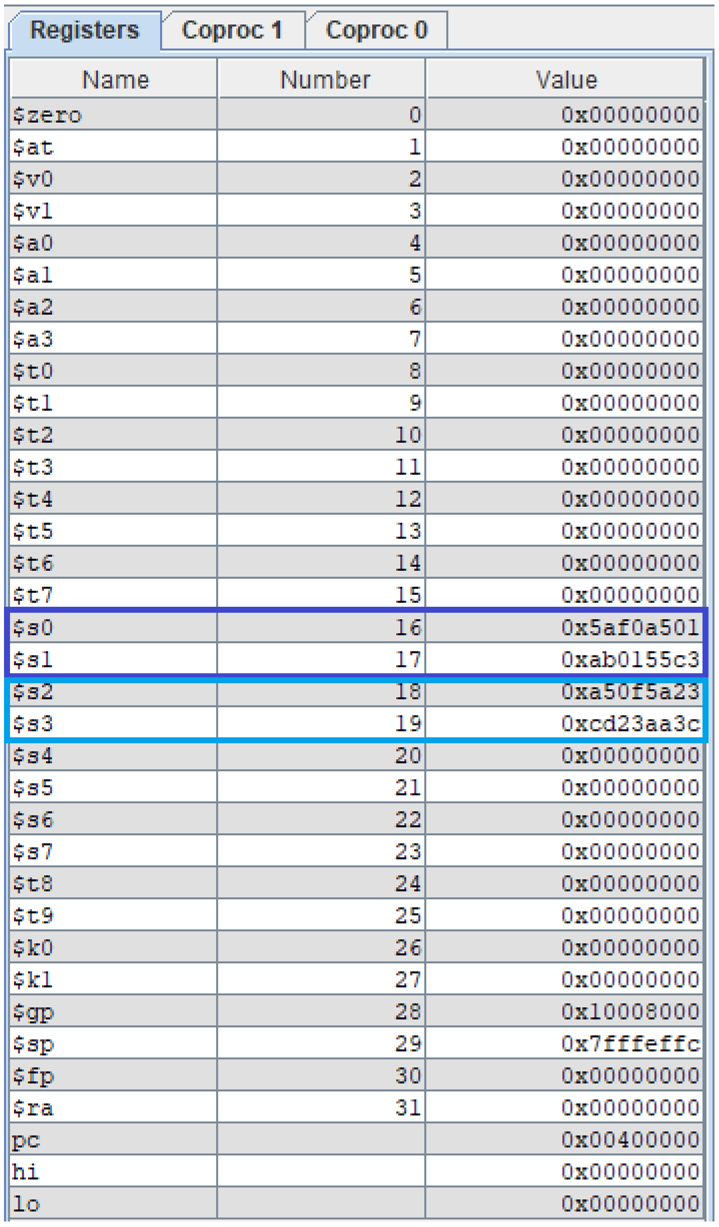
Vector Merge High

```
# File name:            CECS341_Projecs1_vec_mergeh_8.asm
# Version:              1.0
# Date:                 November 5th, 2018
# Programmers:          Thomas Vugia, Jocelyn Espitia
#
# Description:          Designing a customized instruction for vectors, specifically
#                       for a vector merge high (vec_mergeh d, a, b). This instruction
#                       takes the high elements of vectors a and b and stores them
#                       into the even and odd elements of vector d respectively.
#                       That is, all high elements of vector a are stored into
#                       the even elements of vector d and all high elements of
#                       vector b are stored into the odd elements of vector d.
#
#
# Register usage:       $s0 - first half of vector a
#                       $s1 - second half of vector a
#                       $s2 - first half of vector b
#                       $s3 - second half of vector b
#                       $s4 - first half of vector d
#                       $s5 - second half of vector d
#                       $t0 - temp register for element manipulation
#                       $t1 - temp register for storing results into vector d
#
# Notes:                This instruction will only work on vectors with 8 elements
#                       (or 64 elements) each. Vectors smaller or larger than
#                       8 elements will not be read and processed correctly
#                       according to this program.
#
```

```
#*********************************************************************************

          #********************************************************
          #                    M A I N   C O D E S E G M E N T
          #********************************************************

          .text
          .global       main

main:
                                                #loading in registers for vectors
                                                #with pre-set value examples
                    li      $s0, 0x5AF0A501     #first half of vector a
                    li      $s1, 0xAB0155C3     #second half of vector a
                    li      $s2, 0xA50F5A23     #first half of vector b
                    li      $s3, 0xCD23AA3C     #second half of vector b

                    li      $t0, 0xFF000000     #temp register for element manipulation

vec3_1:             and     $t1, $s0, $t0       #storing 5th element of vector a
                    or      $s4, $s4, $t1       #into 1st element of vector d

vec3_2:             and     $t1, $s2, $t0       #storing 5th element of vector b
                    srl     $t1, $t1, 8
                    or      $s4, $s4, $t1       #into 2nd element of vector d

                    srl     $t0, $t0, 8         #moving to 6th element of vectors a and b

vec3_3:             and     $t1, $s0, $t0       #storing 6th element of vector a
```

```
            srl     $t1, $t1, 8
            or      $s4, $s4, $t1           #into 3rd element of vector d

vec3_4:     and     $t1, $s2, $t0           #storing 6th element of vector b
            srl     $t1, $t1, 16
            or      $s4, $s4, $t1           #into 4th element of vector d

            srl     $t0, $t0, 8             #moving to 7th element of vectors a and b

vec3_5:     and     $t1, $s0, $t0           #storing 7th element of vector a
            sll     $t1, $t1, 16
            or      $s5, $s5, $t1           #into 5th element of vector d

vec3_6:     and     $t1, $s2, $t0           #storing 7th element of vector b
            sll     $t1, $t1, 8
            or      $s5, $s5, $t1           #into 6th element of vector d

            srl     $t0, $t0, 8             #moving to 8th element of vectors a and b

vec3_7:     and     $t1, $s0, $t0           #storing 8th element of vector a
            sll     $t1, $t1, 8
            or      $s5, $s5, $t1           #into 7th element of vector d

vec3_8:     and     $t1, $s2, $t0           #storing 8th element of vector b
            or      $s5 $s5, $t1            #into 8th element of vector d


exit:       ori     $v0, $zero, 10
            syscall
```

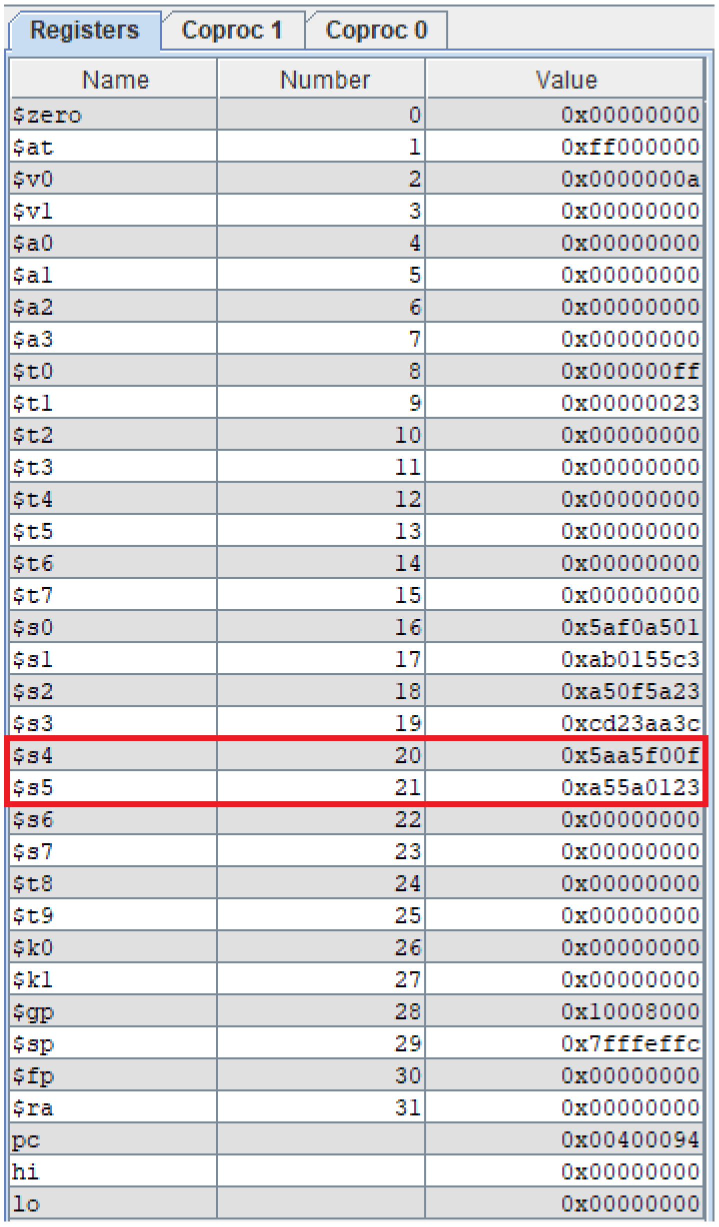
VECTOR MERGE HIGH BEFORE



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x5af0a501
$s1	17	0xab0155c3
$s2	18	0xa50f5a23
$s3	19	0xcd23aa3c
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

VECTOR MERGE HIGH AFTER


Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0xff000000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x000000ff
$t1	9	0x00000023
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x5af0a501
$s1	17	0xab0155c3
$s2	18	0xa50f5a23
$s3	19	0xcd23aa3c
$s4	20	0x5aa5f00f
$s5	21	0xa55a0123
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400094
hi		0x00000000
lo		0x00000000

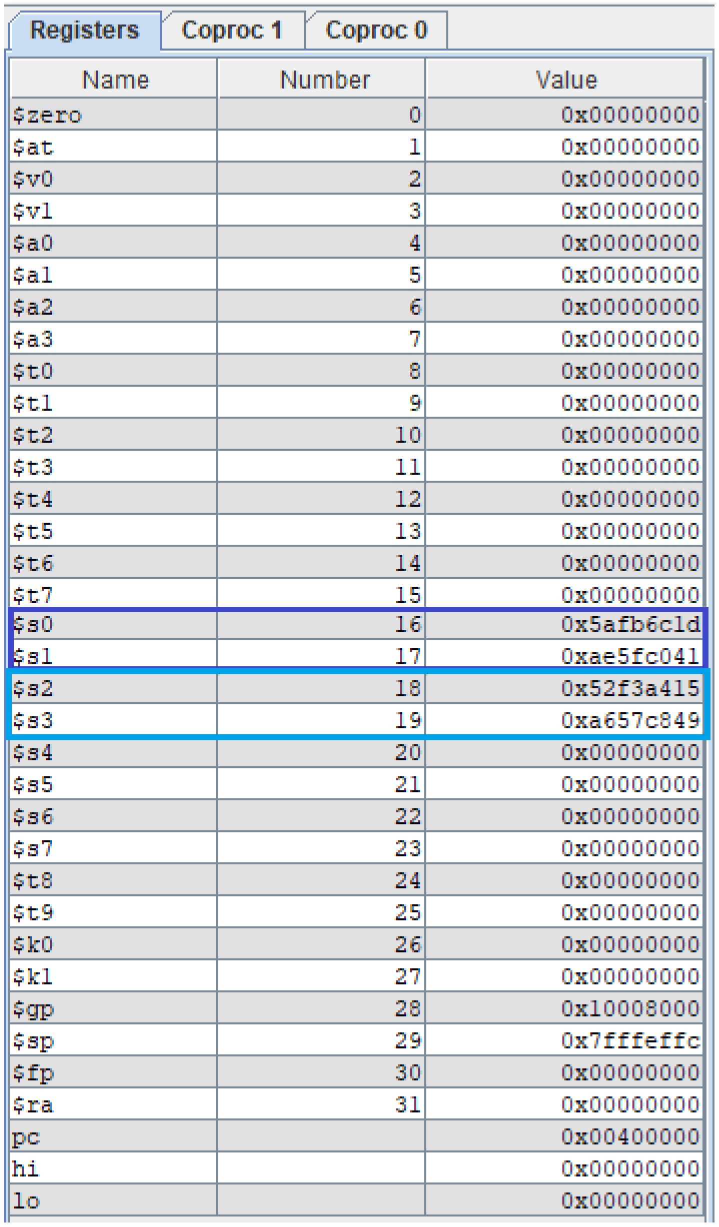
Vector Pack

```
# File name:            CECS341_Projecs1_vec_pack_9.asm
# Version:              1.0
# Date:                 November 5th, 2018
# Programmers:          Thomas Vugia, Jocelyn Espitia
#
# Description:          Designing a customized instruction for vectors, specifically
#                       for a vector pack (vec_pack d, a, b). This instruction takes the
#                       corresponding wider elements of a and b and stores them
#                       into the high and low elements of vector d respectively. That
#                       is, all high elements of vector d are comprised of wide
#                       elements from vector a and all low elements of vector d are
#                       comprised of wider elements from vector b.
#
# Register usage:       $s0 - first half of vector a
#                       $s1 - second half of vector a
#                       $s2 - first half of vector b
#                       $s3 - second half of vector b
#                       $s4 - first half of vector d
#                       $s5 - second half of vector d
#                       $t0 - temp register for bit manipulation
#                       $t1 - temp register for storing wider high elements of vector a
#                       $t2 - temp register for storing wider low elements of vector a
#                       $t3 - temp register for storing wider high elements of vector b
#                       $t4 - temp register for storing wider low elements of vector b
#
# Notes:                This instruction will only work on vectors with 8 elements
#                       (or 64 bits) each. Vectors smaller or larger than
#                       8 elements will not be read and processed correctly
#                                       according to this program.
#
#*********************************************************************************

          #******************************************************
          #                   M A I N   C O D E S E G M E N T
          #******************************************************

          .text
          .global       main

main:
                                              #loading in registers for vectors
                                              #with pre-set value examples
                    li      $s0, 0x5AFB6C1D   #first half of vector a
                    li      $s1, 0xAE5FC041   #second half of vector a
                    li      $s2, 0x52F3A415   #first half of vector b
                    li      $s3, 0xA657C849   #second half of vector b

                    li      $t0, 0x0F000000   #setting $t0 for bit manipulation

pck3_0:             and     $t1, $s0, $t0     #store into temp registers
                    and     $t2, $s1, $t0
                    and     $t3, $s2, $t0
                    and     $t4, $s3, $t0
                    sll     $t1, $t1, 4       #shift to store into high elements of vector d
                    srl     $t2, $t2, 12
                    sll     $t3, $t3, 4       #shift to store into low elements of vector d
                    srl     $t4, $t4, 12
```

```
            or      $s4, $s4, $t1           #storing wider elements into respective
            or      $s4, $s4, $t2           #elements of vector d
            or      $s5, $s5, $t3
            or      $s5, $s5, $t4

            srl     $t0, $t0, 8             #shifting $t0 down for next
                                            #bit manipulation

pck3_1:     and     $t1, $s0, $t0           #store into temp registers
            and     $t2, $s1, $t0
            and     $t3, $s2, $t0
            and     $t4, $s3, $t0
            sll     $t1, $t1, 8             #shift to store into high elements of vector d
            srl     $t2, $t2, 8
            sll     $t3, $t3, 8             #shift to store into low elements of vector d
            srl     $t4, $t4, 8
            or      $s4, $s4, $t1           #storing wider elements into respective
            or      $s4, $s4, $t2           #elements of vector d
            or      $s5, $s5, $t3
            or      $s5, $s5, $t4

            srl     $t0, $t0, 8             #shifting $t0 down for next
                                            #bit manipulation

pck3_2:     and     $t1, $s0, $t0           #store into temp registers
            and     $t2, $s1, $t0
            and     $t3, $s2, $t0
            and     $t4, $s3, $t0
            sll     $t1, $t1, 12            #shift to store into high elements of vector d
            srl     $t2, $t2, 4
```

```
                sll     $t3, $t3, 12            #shift to store into low elements of vector d
                srl     $t4, $t4, 4
                or      $s4, $s4, $t1           #storing wider elements into respective
                or      $s4, $s4, $t2           #elements of vector d
                or      $s5, $s5, $t3
                or      $s5, $s5, $t4

                srl     $t0, $t0, 8             #shifting $t0 down for next
                                                #bit manipulation

pck3_3:         and     $t1, $s0, $t0           #store into temp registers
                and     $t2, $s1, $t0
                and     $t3, $s2, $t0
                and     $t4, $s3, $t0
                sll     $t1, $t1, 8             #shift to store into high and low elements of
                                                #vector d
                sll     $t1, $t1, 8             #no shift necessary for lower wider elements
                sll     $t3, $t3, 8
                sll     $t3, $t3, 8
                or      $s4, $s4, $t1           #storing wider elements into respective
                or      $s4, $s4, $t2           #elements of vector d
                or      $s5, $s5, $t3
                or      $s5, $s5, $t4

                srl     $t0, $t0, 8             #shifting $t0 down for next
                                                #bit manipulation

exit:   ori     $v0, $zero, 10
                syscall
```

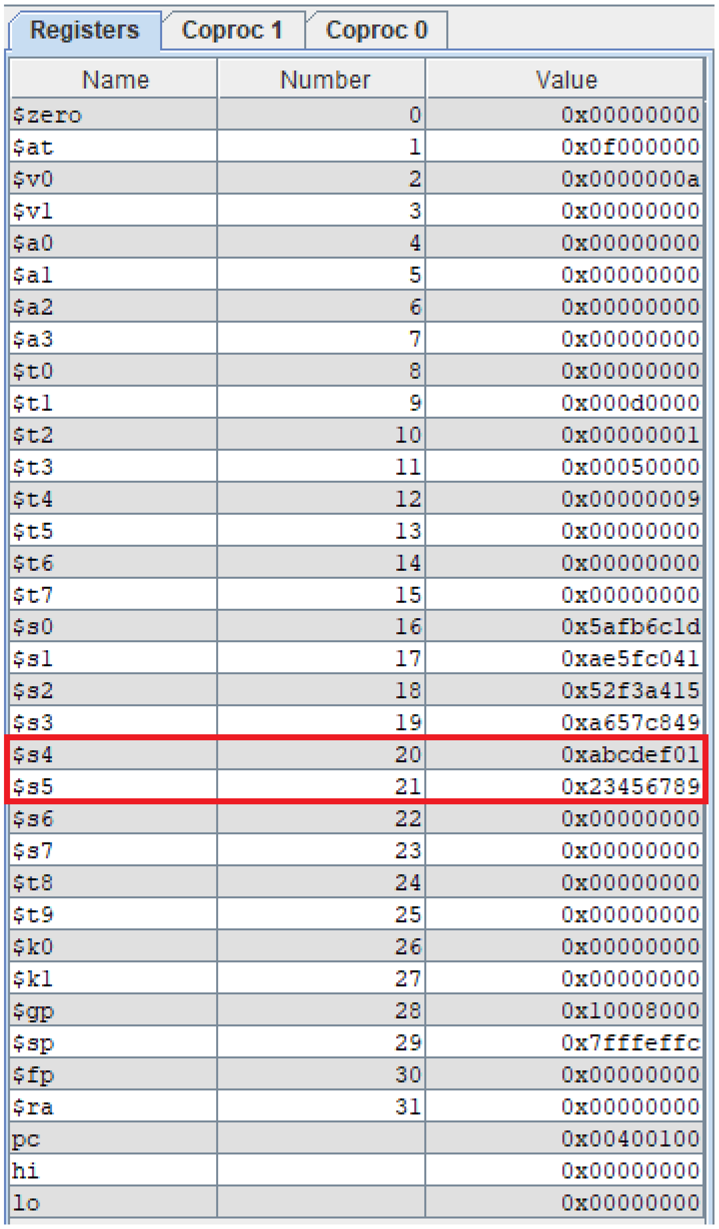
VECTOR PACK BEFORE



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x5afb6c1d
$s1	17	0xae5fc041
$s2	18	0x52f3a415
$s3	19	0xa657c849
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

VECTOR PACK AFTER



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x0f000000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x000d0000
$t2	10	0x00000001
$t3	11	0x00050000
$t4	12	0x00000009
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x5afb6c1d
$s1	17	0xae5fc041
$s2	18	0x52f3a415
$s3	19	0xa657c849
$s4	20	0xabcdef01
$s5	21	0x23456789
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400100
hi		0x00000000
lo		0x00000000

Vector Permute

```
# File name:            CECS341_Project2_vec_perm_#10.asm
# Version:              1.0
# Date:                 November 5th, 2018
# Programmers:          Thomas Vugia, Jocelyn Espitia
#
# Description:          Designing a customized instruction for vectors, specifically
for a
#                       vector permute (vec_perm d, a, b, c). This instruction uses
#                       three different vectors, of which vector c holds the "element
#                       specifier". The element specifier has two parts: the
#                       most-significant-half (or high half-width), which determines
#                       from either vector a or b, and the least-significant-half (or
#                       low half-width), which determines from which element
within
#                       vector a or b. The corresponding element specifier will store
#                       the specified element into the corresponding element of the
#                       result vector d.
#
# Register usage:       $s0 - first half of vector a
#                       $s1 - second half of vector a
#                       $s2 - first half of vector b
#                       $s3 - second half of vector b
#                       $s4 - first half of vector c
#                       $s5 - second half of vector c
#                       $s6 - first half of vector d
#                       $s7 - second half of vector d
#                       $t0 - high half-width element specifier manipulator
#                       $t1 - low half-width element specifier manipulator
#                       $t2 - high half-width element specifier getter
#                               $t3 - low half-width element specifier getter
#                               $t4 - temp store for half of vector c
#                               $t5 - temp store for half of vector d
#                               $t6 - bit manipulator for getting parts of vectors 1 or 2
#                               $t7 - counter for the loop
#
# Notes:                        This instruction will only work on vectors with 8 elements
#                               (or 64 elements) each. Vectors smaller or larger than
#                               8 elements will not be read and processed correctly
#                               according to this program.
#
#*******************************************************************************


        #********************************************************
        #               M A I N   C O D E   S E G M E N T
        #********************************************************

        .text
        .global         main

main:
                                        #loading in registers for vectors
                                        #with pre-set value examples
                li      $s0, 0xA567013D #first half of vector a
                li      $s1, 0xAB45393C #second half of vector a
                li      $s2, 0xEFC54D23 #first half of vector b
                li      $s3, 0x1277AACD #second half of vector b
                li      $s4, 0x04171002 #first half of vector c
                li      $s5, 0x13050105 #second half of vector c
```

```
                    add     $t4, $t4, $s4              #temp store for element specifiers

          #*****************************************************
          #                         E L E M E N T  0  &  4
          #*****************************************************
#First or fifth element to copy from vector a or b
Copy1:
                    li      $t0, 0xF0000000        #set $t0 for bit manipulation, iterating through
                                                   #every high half-width (or most-significant-
                                                   #half) of vector c
                    li      $t1, 0x0F000000        #set $t1 for bit manipulation, iterating through
                                                   #every low half-width (or least-significant-half)
                                                   #of vector c

                    and     $t2, $t0, $t4          #getting element specifiers
                    and     $t3, $t1, $t4

                    srl     $t2, $t2, 16           #shifting into appropriate
                    srl     $t2, $t2, 12           #bit for determining values
                    srl     $t3, $t3, 16
                    srl     $t3, $t3, 8

                    beq     $t2, 0, C1Vec1         #Copy from vector a if high half-width is 0
                    beq     $t2, 1, C1Vec2         #Copy from vector a if high half-width is 1
                    bne     $t2, 0, Copy2          #If not equal to any, move to next element

#Switch-case set for determining from which element of vector a to copy into vector d
C1Vec1:
                    beq     $t3, 0, C1Vec10        #Case 0: if low half-width is 0, copy from
```

```
                                        #element 0
          beq    $t3, 1, C1Vec11        #Case 1: if low half-width is 1, copy from
                                        #element 1
          beq    $t3, 2, C1Vec12        #Case 2: if low half-width is 2, copy from
                                        #element 2
          beq    $t3, 3, C1Vec13        #Case 3: if low half-width is 3, copy from
                                        #element 3
          beq    $t3, 4, C1Vec14        #Case 4: if low half-width is 4, copy from
                                        #element 4
          beq    $t3, 5, C1Vec15        #Case 5: if low half-width is 5, copy from
                                        #element 5
          beq    $t3, 6, C1Vec16        #Case 6: if low half-width is 6, copy from
                                        #element 6
          beq    $t3, 7, C1Vec17        #Case 7: if low half-width is 7, copy from
                                        #element 7
          bne    $t3, 0, Copy2          #If not equal to any, move to next element

#Switch-case set for determining from which element of vector b to copy into vector d
C1Vec2:
          beq    $t3, 0, C1Vec20        #Case 0: if low half-width is 0, copy from
                                        #element 0
          beq    $t3, 1, C1Vec21        #Case 1: if low half-width is 1, copy from
                                        #element 1
          beq    $t3, 2, C1Vec22        #Case 2: if low half-width is 2, copy from
                                        #element 2
          beq    $t3, 3, C1Vec23        #Case 3: if low half-width is 3, copy from
                                        #element 3
          beq    $t3, 4, C1Vec24        #Case 4: if low half-width is 4, copy from
                                        #element 4
          beq    $t3, 5, C1Vec25        #Case 5: if low half-width is 5, copy from
```

```
                                #element 5
        beq     $t3, 6, C1Vec26 #Case 6: if low half-width is 6, copy from
                                #element 6
        beq     $t3, 7, C1Vec27 #Case 7: if low half-width is 7, copy from
                                #element 7
        bne     $t3, 0, Copy2   #If not equal to any, move to next element

#First or fifth element copy from vector a
#Copying first element of vector a into vector d
C1Vec10:
        li      $t6, 0xFF000000
        and     $t6, $t6, $s0
        or      $t5, $t5, $t6
        j       Copy2
#Copying second element; shift required to put element into appropriate element of
#vector d
C1Vec11:
        li      $t6, 0x00FF0000
        and     $t6, $t6, $s0
        sll     $t6, $t6, 8
        or      $t5, $t5, $t6
        j       Copy2
#Copying third element; shift required
C1Vec12:
        li      $t6, 0x0000FF00
        and     $t6, $t6, $s0
        sll     $t6, $t6, 16
        or      $t5, $t5, $t6
        j       Copy2
#Copying fourth element; shift required
```

```
C1Vec13:
            li      $t6, 0x000000FF
            and     $t6, $t6, $s0
            sll     $t6, $t6, 16
            sll     $t6, $t6, 8
            or      $t5, $t5, $t6
            j       Copy2
#Copying fifth element; no shift required
C1Vec14:
            li      $t6, 0xFF000000
            and     $t6, $t6, $s1
            or      $t5, $t5, $t6
            j       Copy2
#Copying sixth element; shift required
C1Vec15:
            li      $t6, 0x00FF0000
            and     $t6, $t6, $s1
            sll     $t6, $t6, 8
            or      $t5, $t5, $t6
            j       Copy2
#Copying seventh element; shift required
C1Vec16:
            li      $t6, 0x0000FF00
            and     $t6, $t6, $s1
            sll     $t6, $t6, 16
            or      $t5, $t5, $t6
            j       Copy2
#Copying eighth element; shift required
C1Vec17:
            li      $t6, 0x000000FF
```

```
                and     $t6, $t6, $s1
                sll     $t6, $t6, 16
                sll     $t6, $t6, 8
                or      $t5, $t5, $t6
                j       Copy2

#First or fifth element copy from vector b
#Copying first element of vector b into vector d
C1Vec20:
                li      $t6, 0xFF000000
                and     $t6, $t6, $s2
                or      $t5, $t5, $t6
                j       Copy2
#Copying second element
C1Vec21:
                li      $t6, 0x00FF0000
                and     $t6, $t6, $s2
                sll     $t6, $t6, 8
                or      $t5, $t5, $t6
                j       Copy2
#Copying third element
C1Vec22:
                li      $t6, 0x0000FF00
                and     $t6, $t6, $s2
                sll     $t6, $t6, 16
                or      $t5, $t5, $t6
                j       Copy2
#Copying fourth element
C1Vec23:
                li      $t6, 0x000000FF
                and     $t6, $t6, $s2
                sll     $t6, $t6, 16
                sll     $t6, $t6, 8
                or      $t5, $t5, $t6
                j       Copy2
#Copying fifth element
C1Vec24:
                li      $t6, 0xFF000000
                and     $t6, $t6, $s3
                or      $t5, $t5, $t6
                j       Copy2
#Copying sixth element
C1Vec25:
                li      $t6, 0x00FF0000
                and     $t6, $t6, $s3
                sll     $t6, $t6, 8
                or      $t5, $t5, $t6
                j       Copy2
#Copying seventh element
C1Vec26:
                li      $t6, 0x0000FF00
                and     $t6, $t6, $s3
                sll     $t6, $t6, 16
                or      $t5, $t5, $t6
                j       Copy2
#Copying eight element
C1Vec27:
                li      $t6, 0x000000FF
                and     $t6, $t6, $s3
                sll     $t6, $t6, 16
                sll     $t6, $t6, 8
                or      $t5, $t5, $t6
                j       Copy2

        #*******************************************************
        #                  E L E M E N T  1  &  5
        #*******************************************************
#Second or sixth element to copy from vector a or b
Copy2:
                li      $t2, 0                  #resetting t2 and t3 to 0 for next bit get
                li      $t3, 0

                srl     $t0, $t0, 8             #shifting bit manipulators to get next element
of
                srl     $t1, $t1, 8             #vector c

                and     $t2, $t0, $t4           #getting element specifiers
                and     $t3, $t1, $t4

                srl     $t2, $t2, 16            #shifting into appropriate bit for determining
values
                srl     $t2, $t2, 4
                srl     $t3, $t3, 16

                beq     $t2, 0, C2Vec1          #Copy from vector a if high half-width is 0
                beq     $t2, 1, C2Vec2          #Copy from vector a if high half-width is 1
                bne     $t2, 0, Copy3           #If not equal to any, move to next element

#Switch-case set for determining from which element of vector a to copy into vector d
C2Vec1:
```

```
                    beq     $t3, 0, C2Vec10         #Case 0: if low half-width is 0, copy from
element 0
                    beq     $t3, 1, C2Vec11         #Case 1: if low half-width is 1, copy from
element 1
                    beq     $t3, 2, C2Vec12                 #Case 2: if low half-width is 2, copy from
element 2
                    beq     $t3, 3, C2Vec13                 #Case 3: if low half-width is 3, copy from
element 3
                    beq     $t3, 4, C2Vec14                 #Case 4: if low half-width is 4, copy from
element 4
                    beq     $t3, 5, C2Vec15                 #Case 5: if low half-width is 5, copy from
element 5
                    beq     $t3, 6, C2Vec16                 #Case 6: if low half-width is 6, copy from
element 6
                    beq     $t3, 7, C2Vec17                 #Case 7: if low half-width is 7, copy from
element 7
                    bne     $t3, 0, Copy3                   #If not equal to any, move to next
element

#Switch-case set for determining from which element of vector b to copy into vector d
C2Vec2:
                    beq     $t3, 0, C2Vec20                 #Case 0: if low half-width is 0, copy from
element 0
                    beq     $t3, 1, C2Vec21                 #Case 1: if low half-width is 1, copy from
element 1
                    beq     $t3, 2, C2Vec22                 #Case 2: if low half-width is 2, copy from
element 2
                    beq     $t3, 3, C2Vec23                 #Case 3: if low half-width is 3, copy from
element 3
                    beq     $t3, 4, C2Vec24                 #Case 4: if low half-width is 4, copy from
element 4
                    beq     $t3, 5, C2Vec25                 #Case 5: if low half-width is 5, copy from
element 5
                    beq     $t3, 6, C2Vec26                 #Case 6: if low half-width is 6, copy from
element 6
```

```
            beq     $t3, 7, C2Vec27             #Case 7: if low half-width is 7, copy from
element 7
            bne     $t3, 0, Copy3           #If not equal to any, move to next element


#Second or sixth element copy from vector a
#Copying first element; shift required
C2Vec10:
            li      $t6, 0xFF000000
            and     $t6, $t6, $s0
            srl     $t6, $t6, 8
            or      $t5, $t5, $t6
            j       Copy3
#Copying second element; no shift required
C2Vec11:
            li      $t6, 0x00FF0000
            and     $t6, $t6, $s0
            or      $t5, $t5, $t6
            j       Copy3
#Copying third element; shift required
C2Vec12:
            li      $t6, 0x0000FF00
            and     $t6, $t6, $s0
            sll     $t6, $t6, 8
            or      $t5, $t5, $t6
            j       Copy3
#Copying fourth element; shift required
C2Vec13:
            li      $t6, 0x000000FF
            and     $t6, $t6, $s0
            sll     $t6, $t6, 16
```

```
                or      $t5, $t5, $t6
                j       Copy3
#Copying fifth element; shift required
C2Vec14:
                li      $t6, 0xFF000000
                and     $t6, $t6, $s1
                srl     $t6, $t6, 8
                or      $t5, $t5, $t6
                j       Copy3
#Copying sixth element; no shift required
C2Vec15:
                li      $t6, 0x00FF0000
                and     $t6, $t6, $s1
                or      $t5, $t5, $t6
                j       Copy3
#Copying seventh element
C2Vec16:
                li      $t6, 0x0000FF00
                and     $t6, $t6, $s1
                sll     $t6, $t6, 8
                or      $t5, $t5, $t6
                j       Copy3
#Copying eighth element
C2Vec17:
                li      $t6, 0x000000FF
                and     $t6, $t6, $s1
                sll     $t6, $t6, 16
                or      $t5, $t5, $t6
                j       Copy3
```

```
#Second or sixth element copy from vector b
#Copying first element; shift required
C2Vec20:
            li      $t6, 0xFF000000
            and     $t6, $t6, $s2
            srl     $t6, $t6, 8
            or      $t5, $t5, $t6
            j       Copy3
#Copying second element; no shift required
C2Vec21:
            li      $t6, 0x00FF0000
            and     $t6, $t6, $s2
            or      $t5, $t5, $t6
            j       Copy3
#Copying third element; shift required
C2Vec22:
            li      $t6, 0x0000FF00
            and     $t6, $t6, $s2
            sll     $t6, $t6, 8
            or      $t5, $t5, $t6
            j       Copy3
#Copying fourth element; shift required
C2Vec23:
            li      $t6, 0x000000FF
            and     $t6, $t6, $s2
            sll     $t6, $t6, 16
            or      $t5, $t5, $t6
            j       Copy3
#Copying fifth element; shift required
C2Vec24:
```

```
                li      $t6, 0xFF000000
                and     $t6, $t6, $s3
                srl     $t6, $t6, 8
                or      $t5, $t5, $t6
                j       Copy3
#Copying sixth element; no shift required
C2Vec25:
                li      $t6, 0x00FF0000
                and     $t6, $t6, $s3
                or      $t5, $t5, $t6
                j       Copy3
#Copying seventh element; shift required
C2Vec26:
                li      $t6, 0x0000FF00
                and     $t6, $t6, $s3
                sll     $t6, $t6, 8
                or      $t5, $t5, $t6
                j       Copy3
#Copying eight element; shift required
C2Vec27:
                li      $t6, 0x000000FF
                and     $t6, $t6, $s3
                sll     $t6, $t6, 16
                or      $t5, $t5, $t6
                j       Copy3

        #*******************************************************
        #               E L E M E N T   2   &   6
        #*******************************************************
#Third or sixth element to copy from vector a or b
```

```
Copy3:
                    li      $t2, 0                  #resetting t2 and t3 to 0 for next bit get
                    li      $t3, 0

                    srl     $t0, $t0, 8             #shifting element manipulators to get next
element of
                    srl     $t1, $t1, 8             #vector c

                    and     $t2, $t0, $t4           #getting element specifiers
                    and     $t3, $t1, $t4

                    srl     $t2, $t2, 12            #shifting into appropriate bit for determining
values
                    srl     $t3, $t3, 8

                    beq     $t2, 0, C3Vec1          #Copy from vector a if high half-width is 0
                    beq     $t2, 1, C3Vec2          #Copy from vector a if high half-width is 1
                    bne     $t2, 0, Copy4           #If not equal to any, move to next element

#Switch-case set for determining from which element of vector a to copy into vector d
C3Vec1:
                    beq     $t3, 0, C3Vec10         #Case 0: if low half-width is 0, copy from
                                                    #element 0
                    beq     $t3, 1, C3Vec11         #Case 1: if low half-width is 1, copy from
                                                    #element 1
                    beq     $t3, 2, C3Vec12         #Case 2: if low half-width is 2, copy from
                                                    #element 2
                    beq     $t3, 3, C3Vec13         #Case 3: if low half-width is 3, copy from
                                                    #element 3
                    beq     $t3, 4, C3Vec14         #Case 4: if low half-width is 4, copy from
```

```
                                    #element 4
          beq    $t3, 5, C3Vec15    #Case 5: if low half-width is 5, copy from
                                    #element 5
          beq    $t3, 6, C3Vec16    #Case 6: if low half-width is 6, copy from
                                    #element 6
          beq    $t3, 7, C3Vec17    #Case 7: if low half-width is 7, copy from
                                    #element 7
          bne    $t3, 0, Copy4      #If not equal to any, move to next element

#Switch-case set for determining from which element of vector b to copy into vector d
C3Vec2:
          beq    $t3, 0, C3Vec20    #Case 0: if low half-width is 0, copy from
                                    #element 0
          beq    $t3, 1, C3Vec21    #Case 1: if low half-width is 1, copy from
                                    #element 1
          beq    $t3, 2, C3Vec22    #Case 2: if low half-width is 2, copy from
                                    #element 2
          beq    $t3, 3, C3Vec23    #Case 3: if low half-width is 3, copy from
                                    #element 3
          beq    $t3, 4, C3Vec24    #Case 4: if low half-width is 4, copy from
                                    #element 4
          beq    $t3, 5, C3Vec25    #Case 5: if low half-width is 5, copy from
                                    #element 5
          beq    $t3, 6, C3Vec26    #Case 6: if low half-width is 6, copy from
                                    #element 6
          beq    $t3, 7, C3Vec27    #Case 7: if low half-width is 7, copy from
                                    #element 7
          bne    $t3, 0, Copy4      #If not equal to any, move to next element

#Third or seventh element copy from vector a
```

```
#Copying first element; shift required
C3Vec10:
            li      $t6, 0xFF000000
            and     $t6, $t6, $s0
            srl     $t6, $t6, 16
            or      $t5, $t5, $t6
            j       Copy4
#Copying second element; shift required
C3Vec11:
            li      $t6, 0x00FF0000
            and     $t6, $t6, $s0
            srl     $t6, $t6, 8
            or      $t5, $t5, $t6
            j       Copy4
#Copying third element; no shift required
C3Vec12:
            li      $t6, 0x0000FF00
            and     $t6, $t6, $s0
            or      $t5, $t5, $t6
            j       Copy4
#Copying fourth element; shift required
C3Vec13:
            li      $t6, 0x000000FF
            and     $t6, $t6, $s0
            sll     $t6, $t6, 8
            or      $t5, $t5, $t6
            j       Copy4
#Copying fifth element; shift required
C3Vec14:
            li      $t6, 0xFF000000
```

```
                and     $t6, $t6, $s1
                srl     $t6, $t6, 16
                or      $t5, $t5, $t6
                j       Copy4
#Copying sixth element; shift required
C3Vec15:
                li      $t6, 0x00FF0000
                and     $t6, $t6, $s1
                srl     $t6, $t6, 8
                or      $t5, $t5, $t6
                j       Copy4
#Copying seventh element; no shift required
C3Vec16:
                li      $t6, 0x0000FF00
                and     $t6, $t6, $s1
                or      $t5, $t5, $t6
                j       Copy4
#Copying eighth element; shift required
C3Vec17:
                li      $t6, 0x000000FF
                and     $t6, $t6, $s1
                sll     $t6, $t6, 8
                or      $t5, $t5, $t6
                j       Copy4

#Third or seventh element copy from vector b
#Copying first element; shift required
C3Vec20:
                li      $t6, 0xFF000000
                and     $t6, $t6, $s2
```

```
            srl     $t6, $t6, 16
            or      $t5, $t5, $t6
            j       Copy4
#Copying second element; shift required
C3Vec21:
            li      $t6, 0x00FF0000
            and     $t6, $t6, $s2
            srl     $t6, $t6, 8
            or      $t5, $t5, $t6
            j       Copy4
#Copying third element; no shift required
C3Vec22:
            li      $t6, 0x0000FF00
            and     $t6, $t6, $s2
            or      $t5, $t5, $t6
            j       Copy4
#Copying fourth element; shift required
C3Vec23:
            li      $t6, 0x000000FF
            and     $t6, $t6, $s2
            sll     $t6, $t6, 8
            or      $t5, $t5, $t6
            j       Copy4
#Copying fifth element; shift required
C3Vec24:
            li      $t6, 0xFF000000
            and     $t6, $t6, $s3
            srl     $t6, $t6, 16
            or      $t5, $t5, $t6
            j       Copy4
```

```
#Copying sixth element; shift required
C3Vec25:
                li      $t6, 0x00FF0000
                and     $t6, $t6, $s3
                srl     $t6, $t6, 8
                or      $t5, $t5, $t6
                j       Copy4
#Copying seventh element; no shift required
C3Vec26:
                li      $t6, 0x0000FF00
                and     $t6, $t6, $s3
                or      $t5, $t5, $t6
                j       Copy4
#Copying eight element; shift required
C3Vec27:
                li      $t6, 0x000000FF
                and     $t6, $t6, $s3
                sll     $t6, $t6, 8
                or      $t5, $t5, $t6
                j       Copy4



        #********************************************************
        #                       E L E M E N T  3  &  7
        #********************************************************
#Fourth or seventh element copy from vector a or b
Copy4:
                li      $t2, 0                  #resetting t2 and t3 to 0 for next bit get
                li      $t3, 0
                srl     $t0, $t0, 8             #shifting element manipulators to get next
                srl     $t1, $t1, 8             # element of vector c

                and     $t2, $t0, $t4           #getting element specifiers; no need to shift $t3
                and     $t3, $t1, $t4           # as it's already in the desired bit for
                srl     $t2, $t2, 4             # determining its element specifier

                beq     $t2, 0, C4Vec1          #Copy from vector a if high half-width is 0
                beq     $t2, 1, C4Vec2          #Copy from vector a if high half-width is 1
                bne     $t2, 0, Hf1or2          #If not equal to any, move to next half of vector

#Switch-case set for determining from which element of vector a to copy into vector d
C4Vec1:
                beq     $t3, 0, C4Vec10         #Case 0: if low half-width is 0, copy from
                                                #element 0
                beq     $t3, 1, C4Vec11         #Case 1: if low half-width is 1, copy from
                                                #element 1
                beq     $t3, 2, C4Vec12         #Case 2: if low half-width is 2, copy from
                                                #element 2
                beq     $t3, 3, C4Vec13         #Case 3: if low half-width is 3, copy from
                                                #element 3
                beq     $t3, 4, C4Vec14         #Case 4: if low half-width is 4, copy from
                                                #element 4
                beq     $t3, 5, C4Vec15         #Case 5: if low half-width is 5, copy from
                                                #element 5
                beq     $t3, 6, C4Vec16         #Case 6: if low half-width is 6, copy from
                                                #element 6
                beq     $t3, 7, C4Vec17         #Case 7: if low half-width is 7, copy from
                                                #element 7
                bne     $t3, 0, Hf1or2          #If not equal to any, move to next half of vector
```

```
#Switch-case set for determining from which element of vector b to copy into vector d
C4Vec2:
            beq     $t3, 0, C4Vec20     #Case 0: if low half-width is 0, copy from
                                        #element 0
            beq     $t3, 1, C4Vec11     #Case 1: if low half-width is 1, copy from
                                        #element 1
            beq     $t3, 2, C4Vec12     #Case 2: if low half-width is 2, copy from
                                        #element 2
            beq     $t3, 3, C4Vec13     #Case 3: if low half-width is 3, copy from
                                        #element 3
            beq     $t3, 4, C4Vec14     #Case 4: if low half-width is 4, copy from
                                        #element 4
            beq     $t3, 5, C4Vec15     #Case 5: if low half-width is 5, copy from
                                        #element 5
            beq     $t3, 6, C4Vec16     #Case 6: if low half-width is 6, copy from
                                        #element 6
            beq     $t3, 7, C4Vec17     #Case 7: if low half-width is 7, copy from
                                        #element 7
            bne     $t3, 0, Hf1or2      #If not equal to any, move to next half of vector

#Fourth or eighth element copy from vector a
#Copying first element; shift required
C4Vec10:
            li      $t6, 0xFF000000
            and     $t6, $t6, $s0
            srl     $t6, $t6, 16
            srl     $t6, $t6, 8
            or      $t5, $t5, $t6
            j       Hf1or2
```

```
#Copying second element; shift required
C4Vec11:
            li      $t6, 0x00FF0000
            and     $t6, $t6, $s0
            srl     $t6, $t6, 16
            or      $t5, $t5, $t6
            j       Hf1or2
#Copying third element; shift required
C4Vec12:
            li      $t6, 0x0000FF00
            and     $t6, $t6, $s0
            srl     $t6, $t6, 8
            or      $t5, $t5, $t6
            j       Hf1or2
#Copying fourth element; no shift required
C4Vec13:
            li      $t6, 0x000000FF
            and     $t6, $t6, $s0
            or      $t5, $t5, $t6
            j       Hf1or2
#Copying fifth element; shift required
C4Vec14:
            li      $t6, 0xFF000000
            and     $t6, $t6, $s1
            srl     $t6, $t6, 16
            srl     $t6, $t6, 8
            or      $t5, $t5, $t6
            j       Hf1or2
#Copying sixth element; shift required
C4Vec15:
```

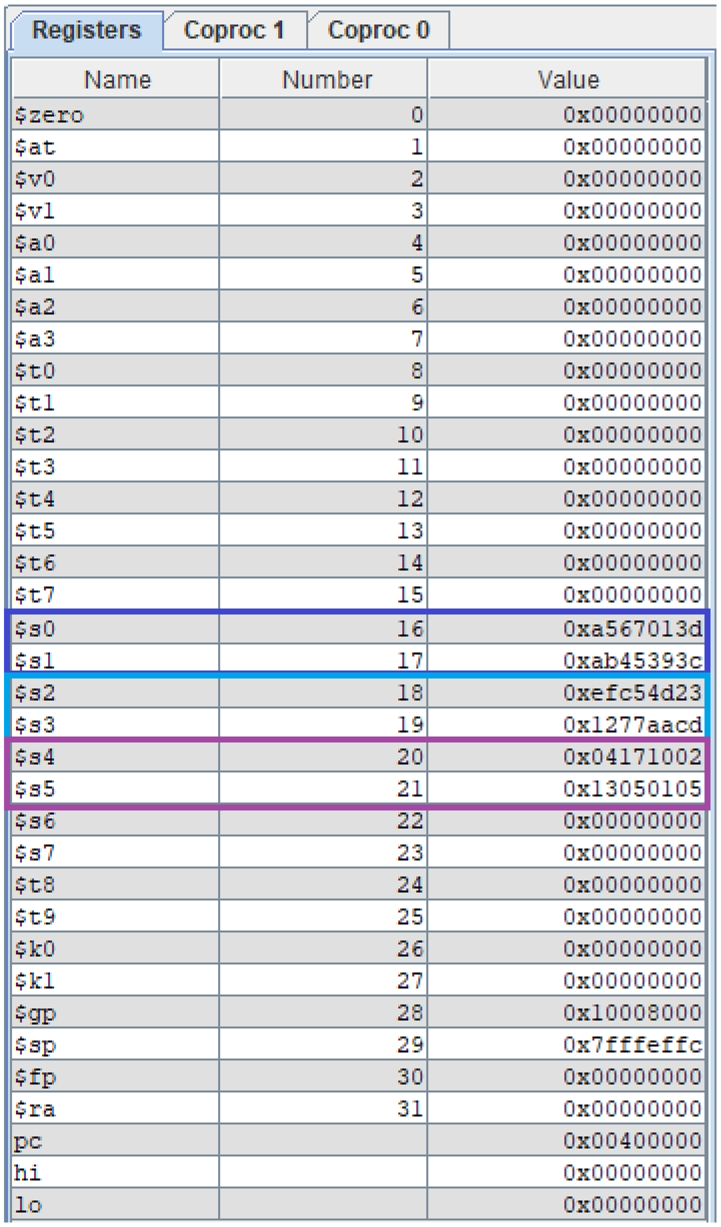
```
                    li      $t6, 0x00FF0000
                    and     $t6, $t6, $s1
                    srl     $t6, $t6, 16
                    or      $t5, $t5, $t6
                    j       Hf1or2
#Copying seventh element; shift required
C4Vec16:
                    li      $t6, 0x0000FF00
                    and     $t6, $t6, $s1
                    srl     $t6, $t6, 8
                    or      $t5, $t5, $t6
                    j       Hf1or2
#Copying eighth element; no shift required
C4Vec17:li          $t6, 0x000000FF
                    and     $t6, $t6, $s1
                    or      $t5, $t5, $t6
                    j       Hf1or2

#Fourth or eigth element copy from vector b
#Copying first element; shift required
C4Vec20:
                    li      $t6, 0xFF000000
                    and     $t6, $t6, $s2
                    srl     $t6, $t6, 16
                    srl     $t6, $t6, 8
                    or      $t5, $t5, $t6
                    j       Hf1or2
#Copying second element; shift required
C4Vec21:
                    li      $t6, 0x00FF0000
```

```
            and     $t6, $t6, $s2
            srl     $t6, $t6, 16
            or      $t5, $t5, $t6
            j       Hf1or2
#Copying third element; shift required
C4Vec22:
            li      $t6, 0x0000FF00
            and     $t6, $t6, $s2
            srl     $t6, $t6, 8
            or      $t5, $t5, $t6
            j       Hf1or2
#Copying fourth element; no shift required
C4Vec23:
            li      $t6, 0x000000FF
            and     $t6, $t6, $s2
            or      $t5, $t5, $t6
            j       Hf1or2
#Copying fifth element; shift required
C4Vec24:
            li      $t6, 0xFF000000
            and     $t6, $t6, $s3
            srl     $t6, $t6, 16
            srl     $t6, $t6, 8
            or      $t5, $t5, $t6
            j       Hf1or2
#Copying sixth element; shift required
C4Vec25:
            li      $t6, 0x00FF0000
            and     $t6, $t6, $s3
            srl     $t6, $t6, 16
```

```
                    or      $t5, $t5, $t6
                    j       Hf1or2
#Copying seventh element; shift required
C4Vec26:
                    li      $t6, 0x0000FF00
                    and     $t6, $t6, $s3
                    srl     $t6, $t6, 8
                    or      $t5, $t5, $t6
                    j       Hf1or2
#Copying eight element; no shift required
C4Vec27:
                    li      $t6, 0x000000FF
                    and     $t6, $t6, $s3
                    or      $t5, $t5, $t6
                    j       Hf1or2

#Determines which half the entire loop has done / is doing
Hf1or2:
                    beq     $t7, 0, Half1           #If counter ($t7) = 0, then it has completed the
first half
                    beq     $t7, 1, Half2           #If counter ($t7) = 1, then it has completed the
second half

#Completion of first half; program stores then resets for next loop
Half1:
                    addi    $t7, $t7, 1             #Increments loop counter by 1
                    add     $s6, $s6, $t5           #Store first half of vector d
                    li      $t4, 0                  #Reset temp stores for halves of vectors 3 and 4
                    li      $t5, 0
                    add     $t4, $t4, $s5           #Gets next half of vector c
```

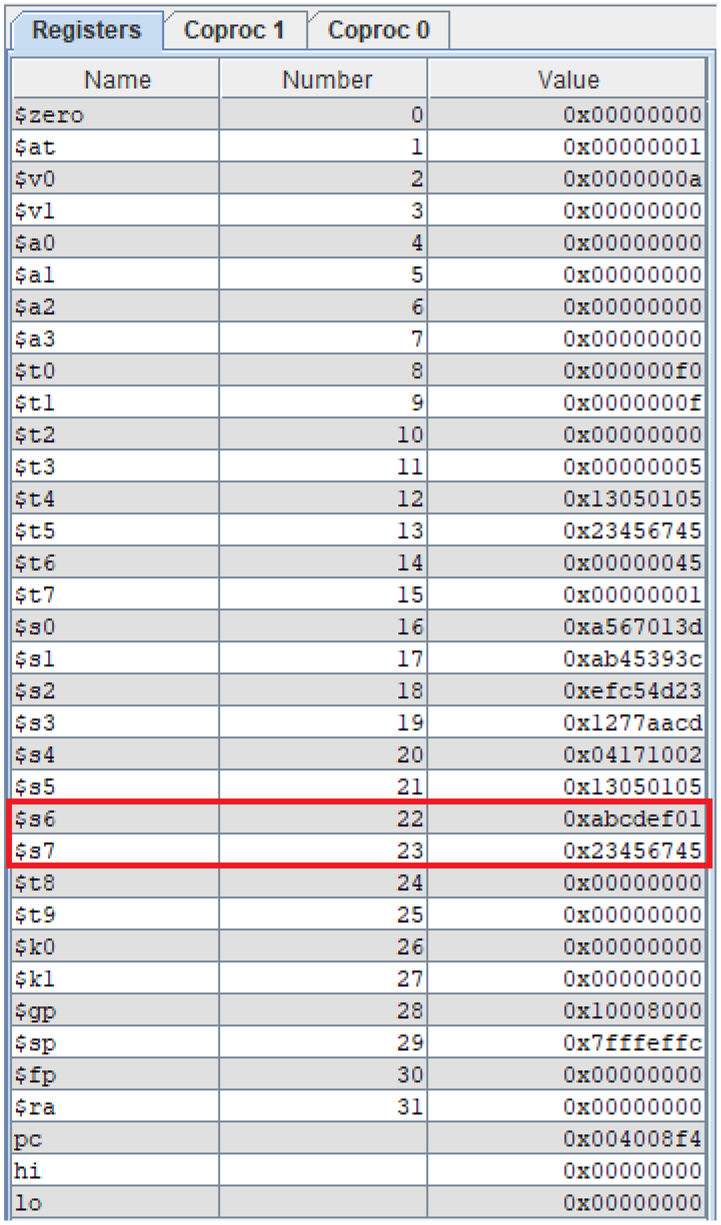
```
                j       Copy1
#Completion of second half; program stores then exits
Half2:
                add     $s7, $s7, $t5           #Store last half of vector d
                j       exit                    #and exits

exit:
                ori $v0, $zero, 10
                syscall
```

VECTOR PERMUTE BEFORE


Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0xa567013d
$s1	17	0xab45393c
$s2	18	0xefc54d23
$s3	19	0x1277aacd
$s4	20	0x04171002
$s5	21	0x13050105
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

VECTOR PERMUTE AFTER


Registers | Coproc 1 | Coproc 0

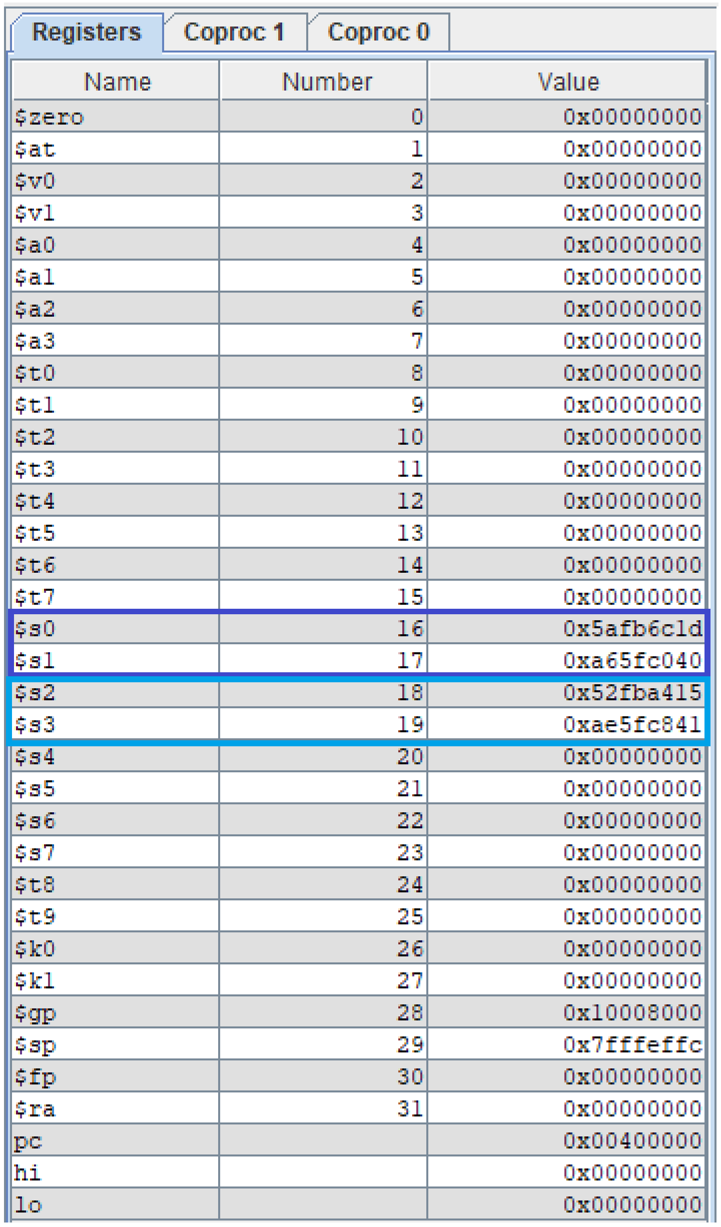
Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000001
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x000000f0
$t1	9	0x0000000f
$t2	10	0x00000000
$t3	11	0x00000005
$t4	12	0x13050105
$t5	13	0x23456745
$t6	14	0x00000045
$t7	15	0x00000001
$s0	16	0xa567013d
$s1	17	0xab45393c
$s2	18	0xefc54d23
$s3	19	0x1277aacd
$s4	20	0x04171002
$s5	21	0x13050105
$s6	22	0xabcdef01
$s7	23	0x23456745
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x004008f4
hi		0x00000000
lo		0x00000000

Vector Compare Equal-To

```
# File name:           CECS341_Project2_vec_cmpeq_11.asm
# Version:             1.0
# Date:                November 5th, 2018
# Programmers:         Thomas Vugia, Jocelyn Espitia
#
# Description:         Designing a customized instruction for vectors, specifically
#                      for a vector compare equal-to (vec_cmpeq d, a, b). This
#                      instruction will determine if two corresponding elements
#                      from two vectors a and b are equal or not. If they are equal,
#                      the result (FF) is stored into the corresponding element of
#                      vector d.
#
# Register usage:      $s0 - first 4 elements of vector a
#                      $s1 - second 4 elements of vector a
#                      $s2 - first 4 elements of vector b
#                      $s3 - second 4 elements of vector b
#                      $s4 - first 4 elements of vector d
#                      $s5 - second 4 elements of vector d
#                      $t0 - temp register for bit manipulation and for storing result
#                              into elements of vector d
#                      $t1 - temp register for element comparison; holds an element
#                              from vector a
#                      $t2 - temp register for element comparison; holds an element
#                              from vector b
#
# Notes:               This instruction will only work on vectors with 8 elements
#                      (or 64 bits) each. Vectors smaller or larger than
#                      8 elements will not be read and processed correctly
#                           according to this program.
#
#*********************************************************************************

        #*******************************************************
        #             M A I N   C O D E S E G M E N T
        #*******************************************************

        .text
        .global       main

main:
                                               #loading in registers for vectors
                                               #with pre-set value examples
              li      $s0, 0x5AFB6C1D          #first 4 elements of vector a
              li      $s1, 0xA65FC040          #second 4 elements of vector a
              li      $s2, 0x52FBA415          #first 4 elements of vector b
              li      $s3, 0xAE5FC841          #second 4 elements of vector b

              li      $t0, 0xFF000000          #Set $t5 for bit manipulation, with one
                                               #element of each vectors a and b. It is
                                               #also used for storing the result into
                                               #vector d

loop:         and     $t1, $t0, $s0            #Get an element of vector a
              and     $t2, $t0, $s2            #Get an element of vector b

              bne     $t1, $t2, skip      #If elements are not the same then loop back,
                                          #else store the result ($t5)
```

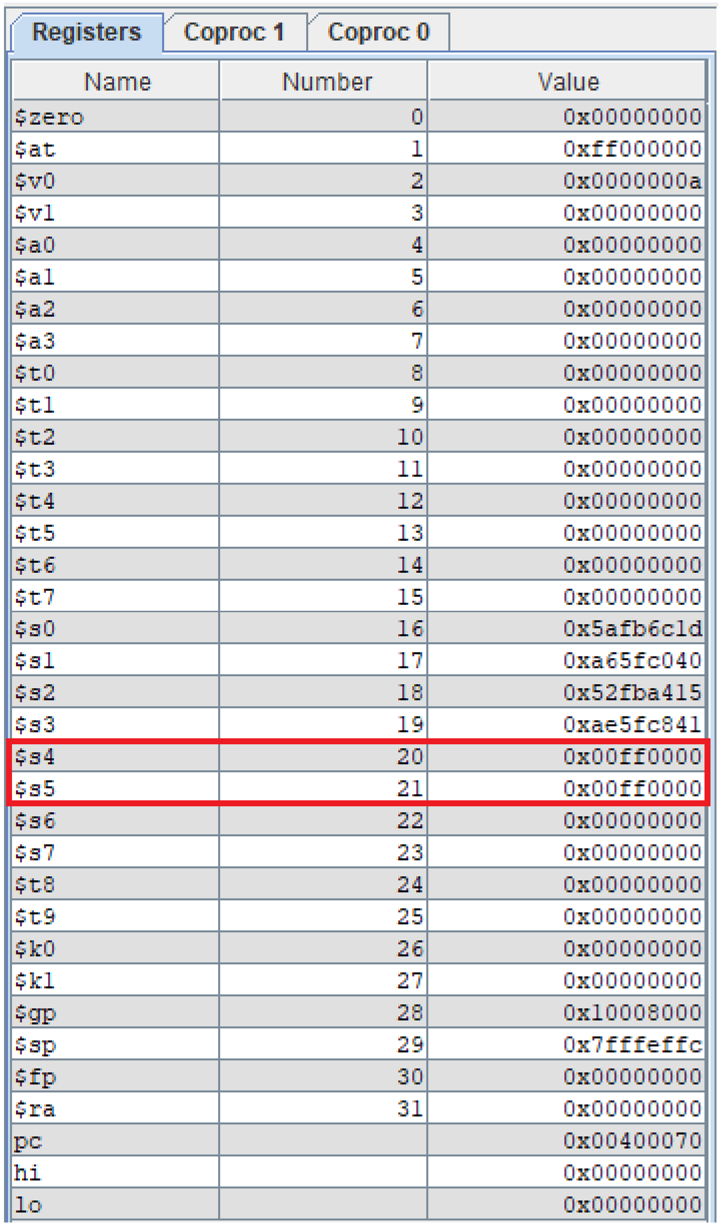
```
            or      $s4, $s4, $t0       #Storing results of findings in 1st 1/2 of vector d

skip:       beq     $t0, $zero, next    #If searching through first half of vectors a & b
                                        #is completed, then move to next halves
                                        #of vectors a & b.
            srl     $t0, $t0, 8         #Shift right to get next two bits
            j       loop                #Looping back

next:       li      $t0, 0xFF000000     #Reset bit manipulator / result finder for next
                                        #halves

loop2:      and     $t1, $t0, $s1       #Get an element of vector a
            and     $t2, $t0, $s3       #Get an element of vector b

            bne     $t1, $t2, skip2     #If elements are not the same, then loop back,
                                        #else store the result ($t5)
            or      $s5, $s5, $t0       #Store results of findings in second half of
                                        #vector d

skip2:      beq     $t0, $zero, exit    #If searching through second half of vectors a
                                        #and b is complete, exit program.
            srl     $t0, $t0, 8         #Shift right to get next two bits
            j       loop2               #Looping back


exit:       ori     $v0, $zero, 10
            syscall
```

VECTOR COMPARE EQUAL TO BEFORE



Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x5afb6c1d
$s1	17	0xa65fc040
$s2	18	0x52fba415
$s3	19	0xae5fc841
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

VECTOR COMPARE EQUAL TO AFTER



Registers | Coproc 1 | Coproc 0

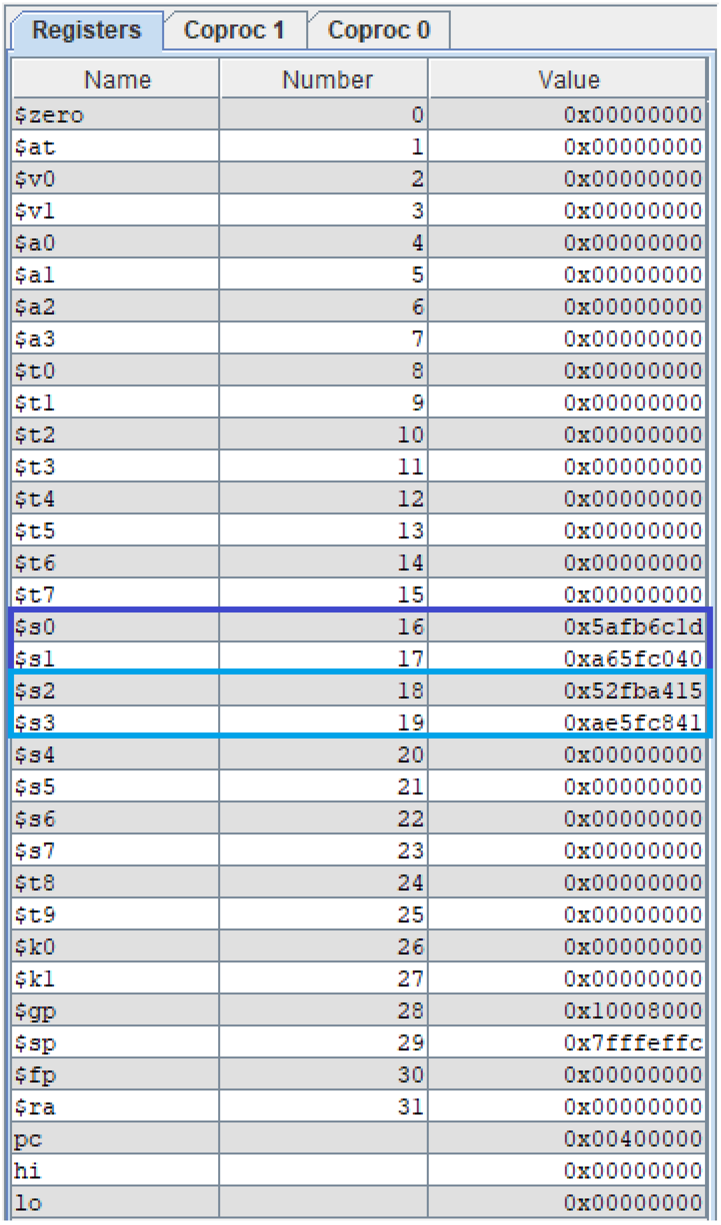
Name	Number	Value
$zero	0	0x00000000
$at	1	0xff000000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x5afb6c1d
$s1	17	0xa65fc040
$s2	18	0x52fba415
$s3	19	0xae5fc841
$s4	20	0x00ff0000
$s5	21	0x00ff0000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400070
hi		0x00000000
lo		0x00000000

Vector Compare Less-Than (Unsigned)

```
# File name:          CECS341_Project2_vec_cmpltu_12.asm
# Version:            1.0
# Date:               November 5th, 2018
# Programmers:        Thomas Vugia, Jocelyn Espitia
#
# Description:        Designing a customized instruction for vectors, specifically
#                     for a vector compare less than (unsigned)
#                     (vec_cmpltu d, a, b). This instruction will determine if the
#                     corresponding element from vector a is less than the
#                     corresponding element in vector b. If true, the result (FF) is
#                     stored into the corresponding element of vector d. Else, the
#                     result (00) is stored.
#
# Register usage:     $s0 - first half of vector a
#                     $s1 - second half of vector a
#                     $s2 - first half of vector b
#                     $s3 - second half of vector b
#                     $s4 - first half of vector d
#                     $s5 - second half of vector d
#                     $t0 - temp register for bit manipulation
#                     $t1 - temp register for element comparison; holds an element
#                             from vector a
#                     $t2 - temp register for element comparison; holds an element
#                             from vector b
#                     $t3 - temp register for determining comparison result; if true,
#                             then $t0 is stored into resulting vector d
#
# Notes:              This instruction will only work on vectors with 8 elements
#                                   (or 64 bits) each. Vectors smaller or larger than
#                                   8 elements will not be processed correctly
#                                   according to this program.
#
#*******************************************************************************

        #*******************************************************
        #               M A I N   C O D E S E G M E N T
        #*******************************************************

        .text
        .global         main

main:
                                        #loading in registers for vectors
                                        #with pre-set value examples
                li      $s0, 0x5AFB6C1D #first half of vector a
                li      $s1, 0xA65FC040 #second half of vector a
                li      $s2, 0x52FBA415 #first half of vector b
                li      $s3, 0xAE5FC841 #second half of vector b

                li      $t0, 0xFF000000 #set $t0 for bit manipulation,
                                        #beginning with first two bits of
                                        #each vector a and b

loop:           and     $t1, $t0, $s0   #get two bits of vector a
                and     $t2, $t0, $s2   #get two bits of vector b
                sltu    $t3, $t1, $t2   #test whether bits of vector a
                                        #is less than vector b
                beq     $t3, $zero, skip        #if results in $t3 is false, then
```

```
                                    #move to next two bits of each vector

           or     $s4, $s4, $t0     #else store results of findings
                                    #in first half of vector c

skip:      beq    $t0, $zero, next  #if searching through first half of vectors a
                                    #and b complete, move to next halves
           srl    $t0, $t0, 8       #shift right to get next two bits
           j      loop              #looping back

next:      li     $t0, 0xFF000000

loop2:     and    $t1, $t0, $s1     #get two bits of vector a
           and    $t2, $t0, $s3     #get two bits of vector b
           sltu   $t3, $t1, $t2     #test whether bits of vector a
                                    #is less than vector b

           beq    $t3, $zero, skip2 #if results $t3 is false, then
                                    #move to next two bits of each vector

           or     $s5, $s5, $t0     #else store results of findings
                                    #in second half of vector c

skip2:     beq    $t0, $zero, exit  #if searching through first half of vectors a
                                    #and b complete, move to next halves
           srl    $t0, $t0, 8       #shift right to get next two bits
           j      loop2             #looping back

exit:      ori    $v0, $zero, 10
           syscall
```

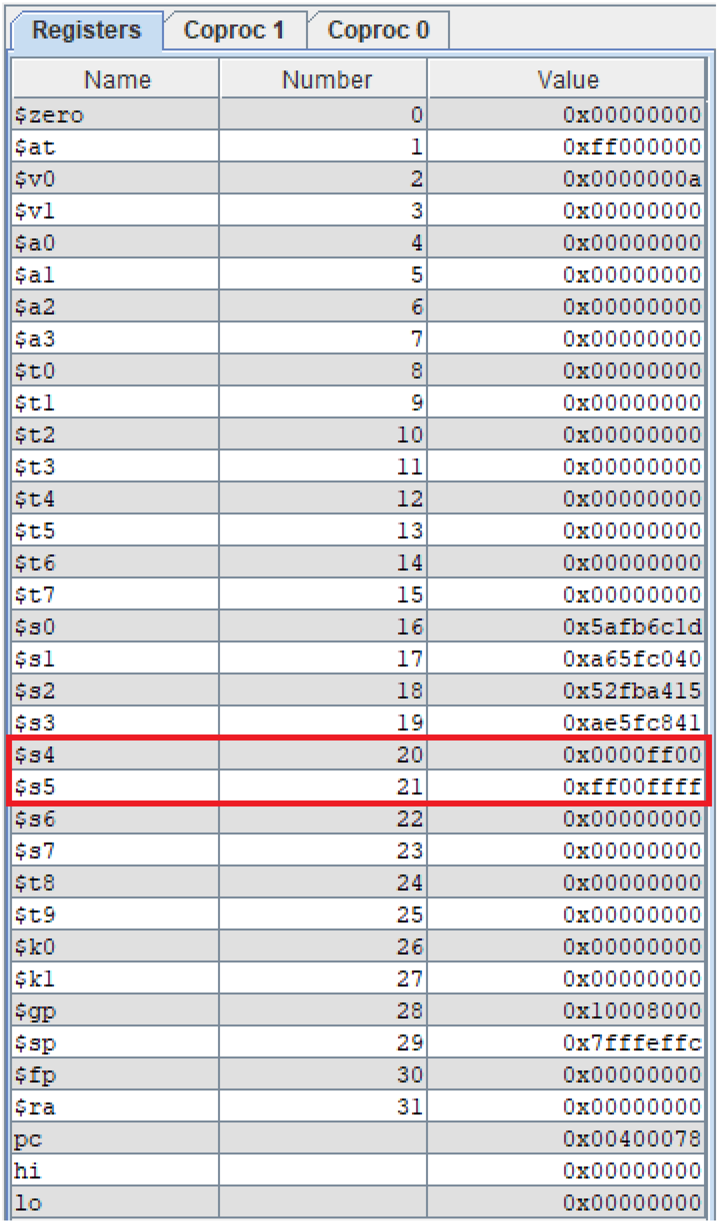
VECTOR COMPARE LESS THAN (UNSIGNED) BEFORE



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x5afb6c1d
$s1	17	0xa65fc040
$s2	18	0x52fba415
$s3	19	0xae5fc841
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

VECTOR COMPARE LESS THAN (UNSIGNED) AFTER



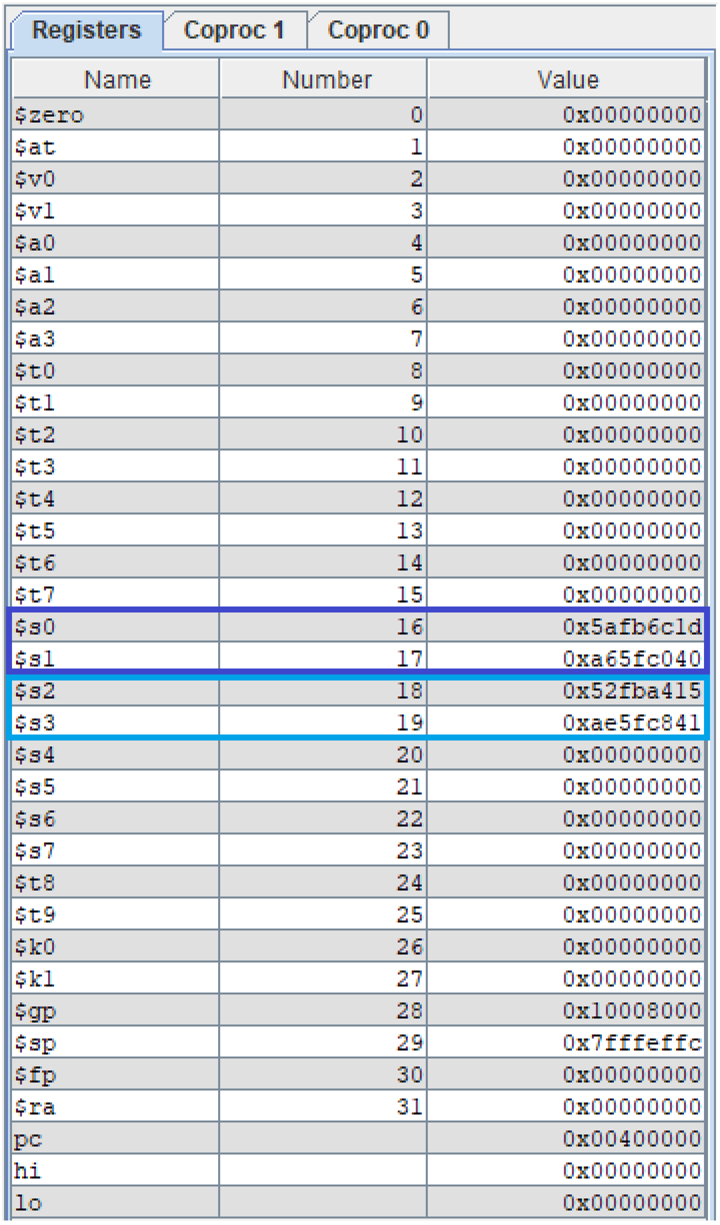
Name	Number	Value
$zero	0	0x00000000
$at	1	0xff000000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x5afb6c1d
$s1	17	0xa65fc040
$s2	18	0x52fba415
$s3	19	0xae5fc841
$s4	20	0x0000ff00
$s5	21	0xff00ffff
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400078
hi		0x00000000
lo		0x00000000

Vector Compare Not Equal-To

```
# File name:          CECS341_Project2_vec_cmpneq_11.1.asm
# Version:            1.0
# Date:               November 5th, 2018
# Programmers:        Thomas Vugia, Jocelyn Espitia
#
# Description:        Designing a customized instruction for vectors, specifically
#                     for a vector compare not-equal-to (vec_cmpneq d, a, b). This
#                     instruction will determine if two corresponding elements
#                     from two vectors a and b are not equal or not. If they are
#                     not equal, the result (FF) is stored into the corresponding
#                     element of vector d. Else, the result (00) is stored.
#
# Register usage:     $s0 - first 4 elements of vector a
#                     $s1 - second 4 elements of vector a
#                     $s2 - first 4 elements of vector b
#                     $s3 - second 4 elements of vector b
#                     $s4 - first 4 elements of vector d
#                     $s5 - second 4 elements of vector d
#                     $t0 - temp register for bit manipulation and for storing result
#                           into elements of vector d
#                     $t1 - temp register for element comparison; holds an element
#                           from vector a
#                     $t2 - temp register for element comparison; holds an element
#                           from vector b
#
# Notes:          This instruction will only work on vectors with 8 elements
#                     (or 64 bits) each. Vectors smaller or larger than
#                     8 elements will not be read and processed correctly
#                                       according to this program.
#
#**********************************************************************************


          #*******************************************************
          #                   M A I N   C O D E S E G M E N T
          #*******************************************************

          .text
          .global        main

main:
                                             #loading in registers for vectors
                                             #with pre-set value examples
                    li      $s0, 0x5AFB6C1D  #first 4 elements of vector a
                    li      $s1, 0xA65FC040  #second 4 elements of vector a
                    li      $s2, 0x52FBA415  #first 4 elements of vector b
                    li      $s3, 0xAE5FC841  #second 4 elements of vector b

                    li      $t0, 0xFF000000  #Set $t5 for bit manipulation, with one element
                                             #of each vectors a and b. It is also used for
                                             #storing the result into vector d

loop:               and     $t1, $t0, $s0    #Get an element of vector a
                    and     $t2, $t0, $s2    #Get an element of vector b

                    beq     $t1, $t2, skip   #If elements are the same then loop back,
                                             #else store the result ($t5)
                    or      $s4, $s4, $t0    #Storing results of findings in first half of
```

```
                                        # vector d

skip:       beq   $t0, $zero, next      #If searching through first half of vectors a & b
                                        #is completed, then move to next halves
                                        #of vectors a & b.
            srl   $t0, $t0, 8           #Shift right to get next two bits
            j     loop                  #Looping back

next:       li    $t0, 0xFF000000       #Reset bit manipulator / result finder for
                                        # next halves

loop2:      and   $t1, $t0, $s1         #Get an element of vector a
            and   $t2, $t0, $s3         #Get an element of vector b

            beq   $t1, $t2, skip2       #If elements are not the same, then loop back,
                                        #else store the result ($t5)
            or    $s5, $s5, $t0         #Store results of findings in second half of
                                        # vector d


skip2:
            beq   $t0, $zero, exit      #If searching through second half of vectors a
                                        #and b is complete, exit program.
            srl   $t0, $t0, 8           #Shift right to get next two bits
            j     loop2                 #Looping back



exit:       ori   $v0, $zero, 10
            syscall
```

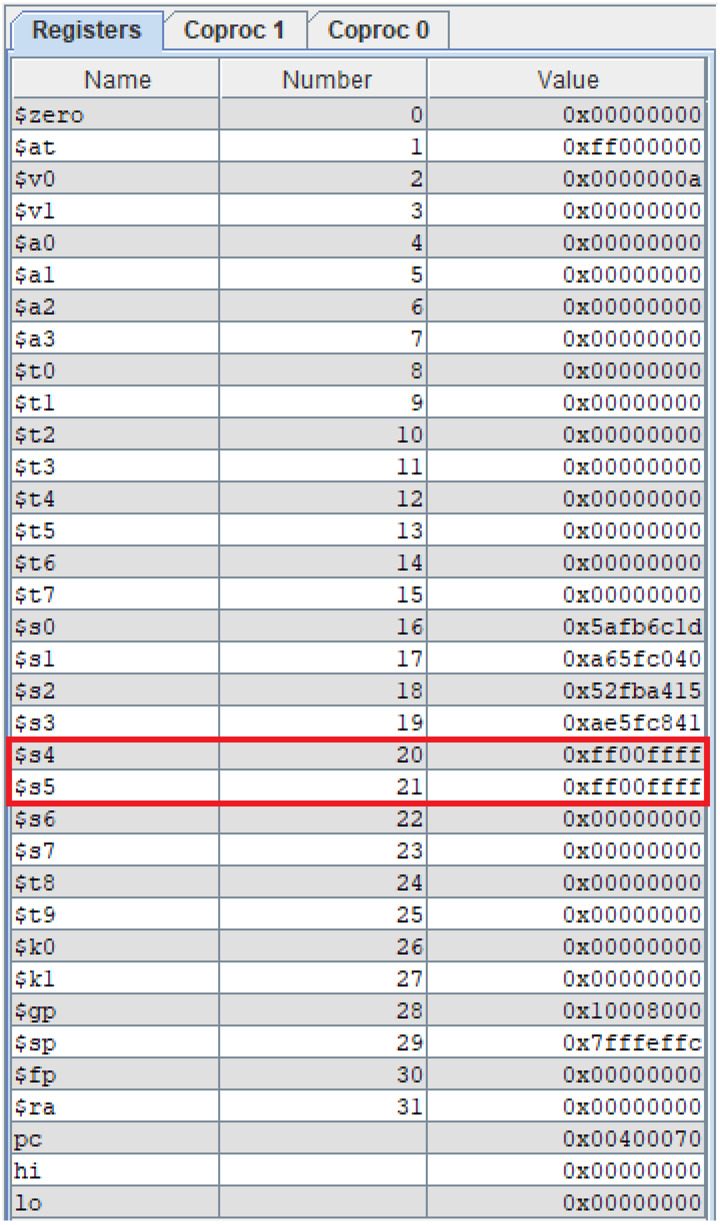
VECTOR COMPARE NOT EQUAL-TO BEFORE



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x5afb6c1d
$s1	17	0xa65fc040
$s2	18	0x52fba415
$s3	19	0xae5fc841
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

VECTOR COMPARE NOT EQUAL-TO AFTER



Registers | Coproc 1 | Coproc 0

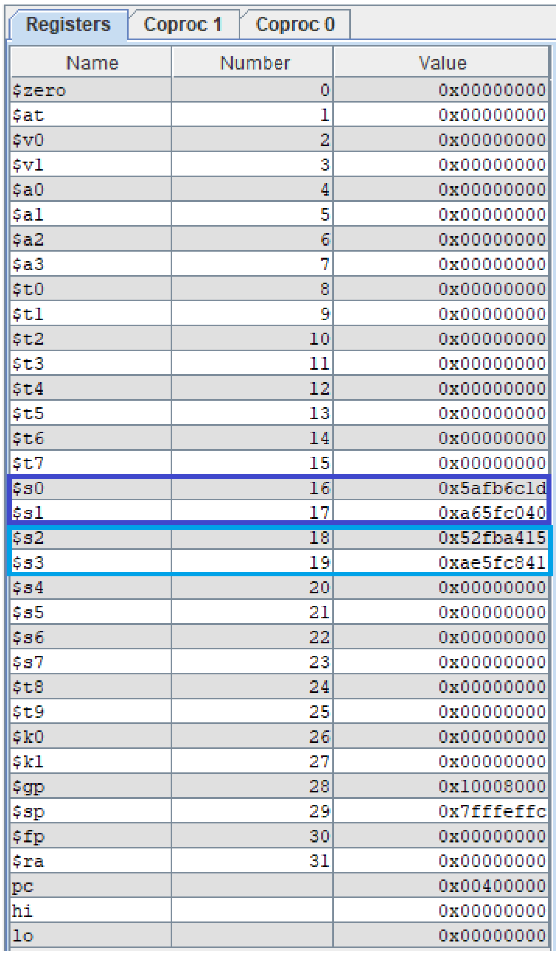
Name	Number	Value
$zero	0	0x00000000
$at	1	0xff000000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x5afb6c1d
$s1	17	0xa65fc040
$s2	18	0x52fba415
$s3	19	0xae5fc841
$s4	20	0xff00ffff
$s5	21	0xff00ffff
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400070
hi		0x00000000
lo		0x00000000

Vector Compare Greater-Than (Unsigned)

```
# File name:        CECS341_Project2_vec_cmpgtu_12.1.asm
# Version:          1.0
# Date:             November 5th, 2018
# Programmers:      Thomas Vugia, Jocelyn Espitia
#
# Description:      Designing a customized instruction for vectors, specifically for a
#                           vector compare greater than (unsigned) (vec_cmpgtu d, a, b).
#                           This instruction will determine if the corresponding
#                           element from vector a is greater than the corresponding
#                           element in vector b. If true, the result (FF) is stored
#                           into the corresponding element of vector d. Else, the
#                           result (00) is stored.
#
# Register usage:   $s0 - first half of vector a
#                   $s1 - second half of vector a
#                   $s2 - first half of vector b
#                   $s3 - second half of vector b
#                   $s4 - first half of vector d
#                   $s5 - second half of vector d
#                   $t0 - temp register for bit manipulation
#                   $t1 - temp register for element comparison; holds an element
#                           from vector a
#                   $t2 - temp register for element comparison; holds an element
#                           from vector a
#                   $t3 - temp register for determining comparison result; if true,
#                           then $t0 is stored into resulting vector d
#
# Notes:            This instruction will only work on vectors with 8 elements
#                                   (or 64 bits) each. Vectors smaller or larger than
#                                   8 elements will not be processed correctly
#                                   according to this program.
#
#*********************************************************************************

         #********************************************************
         #              M A I N   C O D E S E G M E N T
         #********************************************************

         .text
         .global        main

main:
                                           #loading in registers for vectors
                                           #with pre-set value examples
                  li     $s0, 0x5AFB6C1D   #first half of vector a
                  li     $s1, 0xA65FC040   #second half of vector a
                  li     $s2, 0x52FBA415   #first half of vector b
                  li     $s3, 0xAE5FC841   #second half of vector b

                  li     $t0, 0xFF000000   #set $t0 for bit manipulation,
                                           #beginning with first 8 bits
                                           #of each vector a and b

loop:             and    $t1, $t0, $s0     #get 8 bits of vector a
                  and    $t2, $t0, $s2     #get 8 bits of vector b
                  sltu   $t3, $t2, $t1     #test whether bits of vector a is greater than
                                           #vector b
                  beq    $t3, $zero, skip  #if results in $t3 is false, then
```

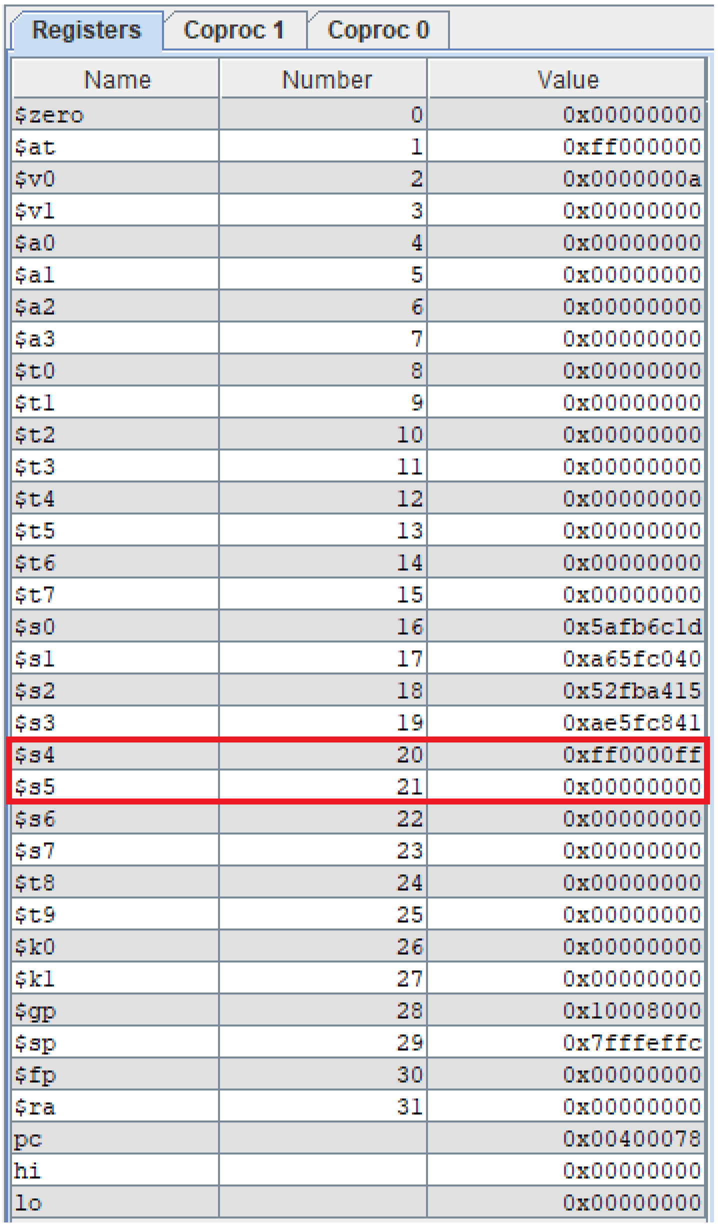
```
                                    #move to next 8 bits of each vector

          or    $s4, $s4, $t0       #else store results of findings
                                    #in first half of vector d

skip:     beq   $t0, $zero, next    #if searching through first half of vectors a
                                    # and b is complete, move to next halves
          srl   $t0, $t0, 8         #shift right to get next 8 bits
          j     loop                #looping back

next:     li    $t0, 0xFF000000     #resets manipulator for next loop

loop2:    and   $t1, $t0, $s1       #get 8 bits of vector a
          and   $t2, $t0, $s3       #get 8 bits of vector b
          sltu  $t3, $t2, $t1       #test whether bits of vector a
                                    #is greater than vector b

          beq   $t3, $zero, skip2   #if results $t3 is false, then
                                    #move to next 8 bits of each vector

          or    $s5, $s5, $t0       #else store results of findings
                                    #in second half of vector d

skip2:    beq   $t0, $zero, exit    #if searching through first half of vectors a
                                    #and b complete, move to next halves
          srl   $t0, $t0, 8         #shift right to get next 8 bits
          j     loop2               #looping back

exit:     ori   $v0, $zero, 10
          syscall
```

VECTOR COMPARE GREATER-THAN (UNSIGNED) BEFORE


Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x5afb6c1d
$s1	17	0xa65fc040
$s2	18	0x52fba415
$s3	19	0xae5fc841
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

VECTOR COMPARE GREATER-THAN (UNSIGNED) AFTER



Name	Number	Value
$zero	0	0x00000000
$at	1	0xff000000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x5afb6c1d
$s1	17	0xa65fc040
$s2	18	0x52fba415
$s3	19	0xae5fc841
$s4	20	0xff0000ff
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400078
hi		0x00000000
lo		0x00000000

Chapter 4

Data Path Block Diagrams

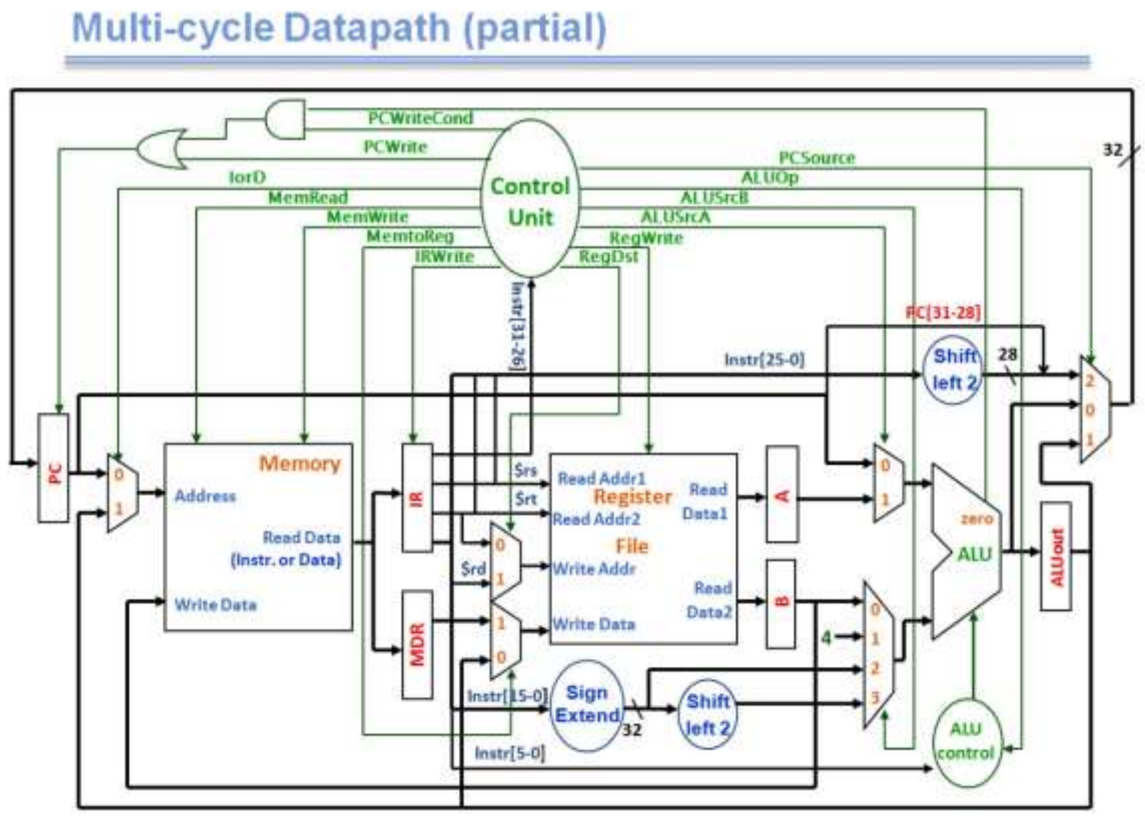


Fig. 2. Multi-cycle Datapath (partial), from R. W. Allison; CD-ROM; 30 Nov. 2018

This is the current multi-cycle datapath that is common in all technology today. The instructions created for this project were not supported on this datapath. Thus, several additions and modifications were made.

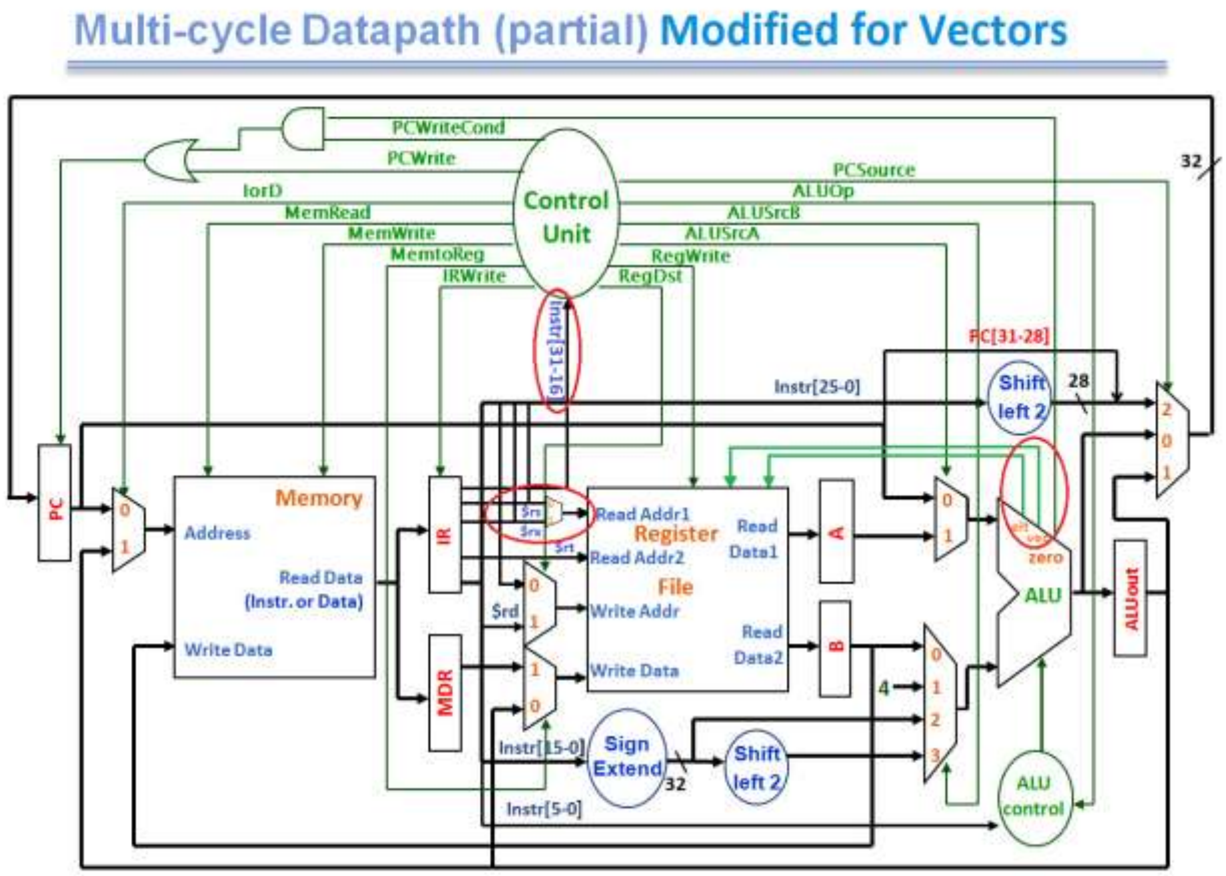


Fig. 3. Multi-cycle Datapath (partial) Modified for Vectors

This Datapath was modified to accommodate our processor with fourteen instructions. What has been modified is circled in red.

Because we have only fourteen instructions to accommodate, we extended the instruction path to be 16 bits long. This also applies to the ALUOp path to allow the ALU to process each instruction properly and correctly. Additionally, with regards to the ALU, we've added two extra paths to accommodate for several of the instructions such as splat and permute. As this set of custom instructions deals mainly with vectors, this deemed to be a necessary addition to the current existing multicycle datapath.

Chapter 5

Additional Discussion and/or Comments

Many of the instructions in this project were found to be difficult to implement, as the design process was rather frustrating, and the simplification process proved to be a challenge. Few others were very easy to build, yet tedious. It is important to note that once an instruction was implemented, it was easy to build another instruction using that instruction base. However, upon completion of one of the difficult and lengthy instructions, there was a sense of accomplishment and pride to be had. Overall, this project was both challenging and rewarding. The authors are proud of the design and content of this manual.

Works Cited

“Addressing Modes.” *GeeksforGeeks*, 10 Feb. 2018, www.geeksforgeeks.org/addressing-modes/.

Fig. 1. MIPS Instruction Formats, Duke University, from Alvin R. Lebeck, *MIPS ISA, Assembly Language*; Powerpoint; Web; 30 Nov. 2018, https://www2.cs.duke.edu/courses/fall98/cps104/lectures/week3-l2/sld007.htm

Fig. 2. Multi-cycle Datapath (partial), from R. W. Allison; CD-ROM; 30 Nov. 2018

Fig. 2. Multi-cycle Datapath (partial) Modified for Vectors, from R. W. Allison; CD-ROM; 30 Nov. 2018

Lebeck, Alvin. “CPS 104 Lecture.” *LZW Data Compression*, www2.cs.duke.edu/courses/fall98/cps104/lectures/week3-l2/sld007.htm.