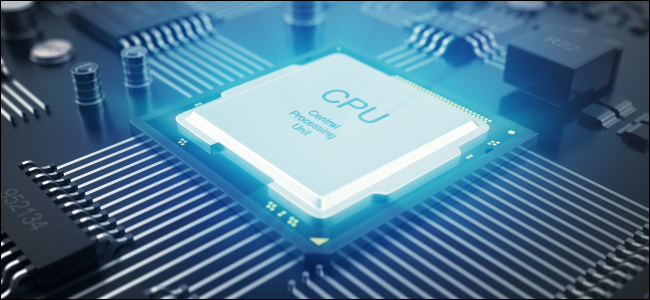
SIMD Enhanced MIPS Instructions

****

Brian Nguyen - 015188925

Randy Thiem - 015432506

CECS 341

MW 8:00 AM - 10:15 AM

R.W. Allison

Table Of Contents

1. Purpose…………………………………………………………………………………….3
2. Abstract………………………………………………………………………………..4
3. Application…………………………………………………………………………….4
4. Instruction Set Architecture……………………………………………………………….5
5. Machine Register Set………………………………………………………………….6
6. Data Types…………………………………………………………………………….8
7. Addressing Modes…………………………………………………………………...10
8. Instruction Set……………………………………………………………………..…12
9. Triple Operand Instructions……………………………………………………...12
10. Double Operand Instructions…………………………………………………….21
11. Single Operand Instructions……………………………………………………...23
12. Conditional Branches…………………………………………………………….24
13. Unconditional Jump and Subroutine Call/Return Instructions…………………..26
14. Immediate Operand Instructions…………………………………………………27
15. Single Instruction Multiple Data (SIMD) Instructions…………………………..28
    1. Vector Add Saturated…………………………………………….28
    2. Vector Multiply Add……………………………………………..30
    3. Vector Multiply Even…………………………………………….32
    4. Vector Multiply Sum Saturated…………………………………..36
    5. Vector Splat………………………………………………………38
    6. Vector Merge Low……………………………………………….40
    7. Vector Merge High……………………………………………….42
    8. Vector Pack………………………………………………………44
    9. Vector Permute…………………………………………………...46
    10. Vector Compare Equal-To………………………………………..48
    11. Vector Compare Less-Than………………………………………50
    12. Vector Swap……………………………………………………...52
    13. Vector Shift Left………………………………………………….54
    14. Vector Shift Right……………………………………………......56
    15. Vector Copy……………………………………………………...58
    16. Vector Contains…………………………………………………..60
    17. Vector AND……………………………………………………...61
    18. Vector OR………………………………………………………..63
    19. Vector Subtract Unsigned………………………………………..65
16. Summary of the Instruction Formats of “Baseline SIMD Enhancements” and “Application Specific” enhancements……………………………………………….67

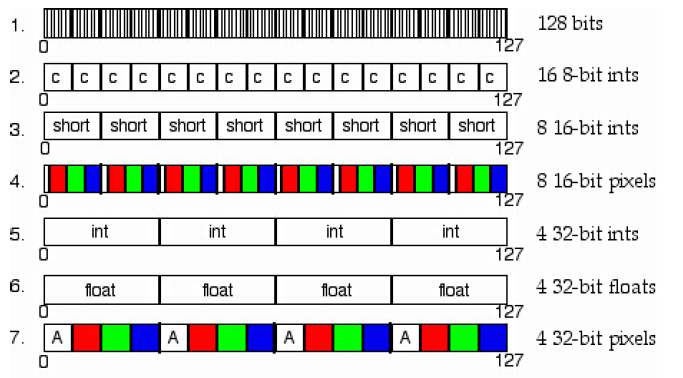
1.

1. MIPS Implementation / Verification with Annotation…………………………………...69
2. Vector Add Saturated………………………………………………………………...70
3. Vector Multiply Add…………………………………………………………………73
4. Vector Multiply Even………………………………………………………………...76
5. Vector Multiply Odd…………………………………………………………………78
6. Vector Multiply Sum Saturated………………………………………………………80
7. Vector Splat…………………………………………………………………………..83
8. Vector Merge Low…………………………………………………………………...85
9. Vector Merge High…………………………………………………………………..88
10. Vector Pack…………………………………………………………………………..91
11. Vector Permute……………………………………………………………………….95
12. Vector Compare Equal-To……………………………………………………………98
13. Vector Compare Less Than…………………………………………………………101
14. Vector Swap………………………………………………………………………...104
15. Vector Shift Left……………………………………………………………………106
16. Vector Shift Right…………………………………………………………………..108
17. Vector Copy…………………………………………………………………………110
18. Vector Contains……………………………………………………………………..112
19. Vector AND…………………………………………………………………………114
20. Vector OR…………………………………………………………………………...116
21. Vector Subtract Unsigned…………………………………………………………...118
    1. Verifications………………………………………………………..121-139
22. Datapath Block Diagram(s)…………………………………………………………….140
23. Additional Discussion and/or Comments………………………………………………143
24. Bibliography…………………………………………………………………………....144
25. Purpose

A. Abstract

The purpose of this project is to create our own Instruction Set Architecture (ISA) utilizing the Million Instructions Per Second (MIPS) ISA. Throughout this process, we will document each step which then will be combined into a Programmers Reference Manual. This manual is designed with the intent to help people who want an in-depth understanding of computer architecture with the intent to further apply the knowledge learned to their work on software applications. We will be discussing the specific details surrounding registers and their applications, along with operands, data types, addressing modes, and binary instructions.

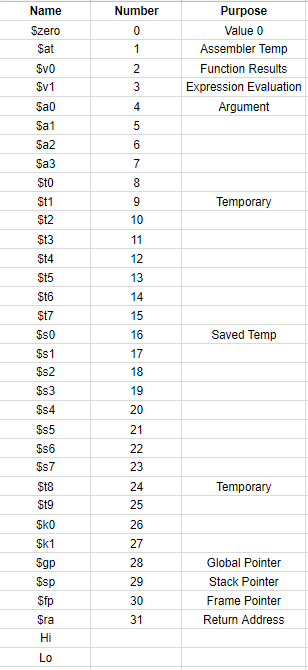
B. Application

MIPS programming offers many different features applicable to programmers. What is dynamic about this is that programmers are able to build their own conventional projects by using the available registers.One case of this is Single Instruction Multiple Data (SIMD). An example of using SIMD is generating graphics. AltiVec Technology uses 32 x 128-bit “vector” registers to display graphics. The following table shows how this is made:

Each vector stores 4 different RGB elements of 16 bit values, 4-bits per component.

II. Instruction Set Architecture

1. Machine Register Set



A machine register set consists of all the registers in the Central Processing Unit (CPU) that are accessible to the programmer. The most common types of registers that are used may include data registers, address registers, flag registers, instruction pointers, and stack pointers. These registers are utilized by instructions sets in order for them to be executed. Each of these registers completes a specific task for the instruction.

**Registers**

* $zero - used to hold the constant value zero in case there is another register that needs to incremented.
* $t - caller-saved registers whose purpose is to hold the value of other registers temporarily until they need to be used again.
* $s - callee-saved registers that hold long-lived values that should be preserved across calls
* $a - used to pass the first four arguments
* $v - used to return values from functions

B. Data Types

**8-bit integer**

**2’s complement Byte-wide Integer**

|  |  |
| --- | --- |
| **s** |  |

7 6 0

**Unsigned Byte-wide Integer**

|  |
| --- |
|  |

7 0

**16-bit integer**

**2’s complement Word-wide Integer**

|  |  |
| --- | --- |
| **s** |  |

7 6 0

**Unsigned Word-wide Integer**

|  |
| --- |
|  |

15 0

**32-bit integer**

**2’s complement Dword-wide Integer**

|  |  |
| --- | --- |
| **s** |  |

31 30 0

**Unsigned Dword-wide Integer**

|  |
| --- |
|  |

31 0

**Single-precision Floating Point**

|  |  |  |
| --- | --- | --- |
| **s** | **exponent** | **mantissa** |

31 30 23 22 0

**64-bit integer**

**Double-precision Floating Point**

|  |  |  |
| --- | --- | --- |
| **s** | **exponent** | **mantissa** |

63 62 52 51 32

|  |
| --- |
| **mantissa-lo** |

31 0

C. Addressing Modes

1. Immediate mode

The operand itself is part of the instruction. The program recognizes operand as an immediate mode when it decodes the addressing mode.



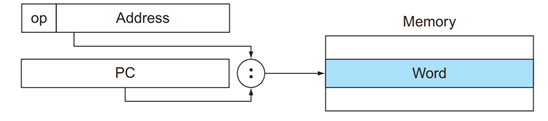
1. Register mode

A mode whose purpose is to access a set of general registers for storing operands.



1. Register Indirect mode

By utilizing base pointers stored in registers indirect mode or register deferred accesses operands.



**D. Instruction Set**

**Triple Operand Instructions**

**Add**

**Description**

Takes the values stored in two registers and adds them together. The result is stored in a specified register.

**Instruction Format**

**R - type**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode | source | source | destination | Shift amount | Function |
| 000000 | rs | rt | rd | 00000 | 100000 |
| 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |

**MIPS Format**

add rd, rs, rt

**Operation**

R[rd] = R[rs] + R[rt]

**Example**

add $t2, $t1, $t0 # $t1 = 10, $t2 = 15

After the instruction executes, $t2 will have the value of 25

**Add Unsigned**

**Description**

Puts the sum of two values into a specific register without overflow.

**Instruction Format**

**R - type**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode | source | source | destination | Shift amount | Function |
| 000000 | rs | rt | rd | 00000 | 100001 |
| 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |

**MIPS Format**

addu rd, rs, rt

**Operation**

R[rd] = R[rs] + R[rt]

**Example**

addu $rd, $rs, $rt #$rd = 2, $rt = 2

After this instruction executes the result of 4 will be stored in register rd.

**Subtract**

**Description**

Takes two values and subtracts the second value from the first value and stores it into a register.

**Instruction Format**

**R - type**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode | source | source | destination | Shift amount | Function |
| 000000 | rs | rt | rd | 00000 | 100010 |
| 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |

**MIPS Format**

sub rd, rs, rt

**Operation**

R[rd] = R[rs] - R[rt]

**Example**

sub $rd, $rs, $rt #$rd = 5, $rt = 3

After this instruction executes 3 will be subtracted from 5 and the value of 2 will be stored into $rd.

**Subtract Unsigned**

**Description**

Takes two values and subtracts the second value from the first value and stores it into a register without overflow.

**Instruction Format**

**R - type**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode | source | source | destination | Shift amount | Function |
| 000000 | rs | rt | rd | 00000 | 100010 |
| 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |

**MIPS Format**

subu rd, rs, rt

**Operation**

R[rd] = R[rs] - R[rt]

**Example**

sub $rd, $rs, $rt #$rd = 5, $rt = 3

After this instruction executes 3 will be subtracted from 5 and the value of 2 will be stored into $rd.

**Set Less Than**

**Description**

Compares two values if the first value is less than the second value then set a register equal to one if not then set the register equal to zero.

**Instruction Format**

**R - type**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode | source | source | destination | Shift amount | Function |
| 000000 | rs | rt | rd | 00000 | 100000 |
| 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |

**MIPS Format**

slt rd, rs, rt

**Operation**

R[rd] = (R[rs] < R[rt]) ? 1 : 0

**Example**

slt rd, rs, rt #$rs = 2, $rt = 3

When this statement executes since rs is less than rt the statement is true and the value of one will be stored in rd.

**Shift Left Logical**

**Description**

Shifts all the bits in a register left and fills the emptied bits with zeros.

**Instruction Format**

**R - type**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode | source | source | destination | Shift amount | Function |
| 000000 | rs | rt | rd | 00000 | 000000 |
| 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |

**MIPS Format**

sll rd, rt, shamt

**Operation**

R[rd] = R[rt] << shamt

**Example**

sll $rd, $rt, 4 #$rt = 9

$rt = 0000 0000 0000 0000 0000 0000 0000 1001

After instruction executes all the bits of $rt will be shifted left by 4 and saved in $rd

$rt = 0000 0000 0000 0000 0000 0000 1001 0000

**Shift Right Logical**

**Description**

Shifts all the bits in a register right and fills the emptied bits with zeros.

**Instruction Format**

**R - type**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode | source | source | destination | Shift amount | Function |
| 000000 | rs | rt | rd | 00000 | 000010 |
| 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |

**MIPS Format**

srl rd, rt, shamt

**Operation**

R[rd] = R[rt] >> shamt

**Example**

srl $rd, $rt, 4 #$rt = 144

$rt = 0000 0000 0000 0000 0000 0000 1001 0000

After instruction executes all the bits of $rt will be shifted right by 4 and saved in $rd

$rt = 0000 0000 0000 0000 0000 0000 0000 1001

**And**

**Description**

The and instruction compares two hexadecimal values by converting them to decimal values and it compares each bit at a time and if both bits are the same it returns that value and if not it will return a zero.

**Instruction Format**

**R - type**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode | source | source | destination | Shift amount | Function |
| 000000 | rs | rt | rd | 00000 | 100100 |
| 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |

**MIPS Format**

and rd, rs, rt

**Operation**

R[rd] = R[rs] & R[rt]

**Example**

and $rd, $rs, $rt #$rs = 2 hex, $rt = 2 hex

After this statement executes the 2 hexadecimal values will be converted into binary values and each bit will be compared and if they are both the same it will return that value and if they are different it will return a zero.

**Or**

**Description**

The or instruction compares the binary values of two registers bit by bit and if at least one of the bits is one then one is returned but if both bits are zero then zero is returned.

**Instruction Format**

**R - type**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode | source | source | destination | Shift amount | Function |
| 000000 | rs | rt | rd | 00000 | 100101 |
| 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |

**MIPS Format**

or rd, rs, rt

**Operation**

R[rd] = R[rs] | R[rt]

**Example**

or $rd, $rs, $rt #$rd = 2, $rt = 3

After this instruction executes the hexadecimal values will be converted into binary values and their bits will be compared bit by bit and if at least one of the bits is one then one will be returned into $rd if both are zero then zero will be returned into $rd.

**Double Operand Instructions**

**Divide**

**Description**

Takes to numbers and divides the first value by the second value and returns the remainder and quotient

**Instruction Format**

**R - type**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode | source | source | destination | Shift amount | Function |
| 000000 | rs | rt | rd | 00000 | 011010 |
| 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |

**MIPS Format**

div rs, rt

**Operation**

Lo = R[rs] / R[rt]; Hi = R[rs] % R[rt]

**Example**

div $rs, $rt #$rs = 99, $rt = 2

When this instruction executes rs will be divided by rt and the result will be 49 R 1.

**Multiply**

**Description**

Takes two values and multiplies them together and outputs the product.

**Instruction Format**

**R - type**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode | source | source | destination | Shift amount | Function |
| 000000 | rs | rt | rd | 00000 | 011000 |
| 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |

**MIPS Format**

mult rs, rt

**Operation**

{Hi, Lo} = R[rs] \* R[rt]

**Example**

mult $rs, $rt #$rs = 2, $rt = 3

After this instruction executes the value that is returned is 6.

**Single Operand Instructions**

**Move From Hi**

**Description**

Loads the upper 32 bits from a product register during multiplication and the remainder is moved into a quotient register during division.

**Instruction Format**

**R - type**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode | source | source | destination | Shift amount | Function |
| 000000 | rs | rt | rd | 00000 | 010000 |
| 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |

**MIPS Format**

mfhi rd

**Operation**

R[rd] = Hi

**Example**

mult $s0, $s1 #Multiply the numbers stored in these registers

#This results in a 64 bit number that is stored in two 32 bits parts Hi and Lo

mfhi $rd #loads the upper 32 bits from the product register

**Conditional Branches**

**Branch On Equal**

**Description**

Branch if equal means go to the label statement if the value in the first register equals the value in the second register

**Instruction Format**

**I - type**

|  |  |  |  |
| --- | --- | --- | --- |
| 0100 | rs | rt | Immediate |

31 26 25 21 20 16 15 0

**MIPS Format**

beq rs, rt, label

**Operation**

if(R[rs] == R[rt])

PC = PC + 4 + BranchAddr

**Example**

sltu $t0, $s1, $t2 #$t0 = 0 if $s1 >= length or $s1 < 0

beq $t0, $zero, IndexOutOfBounds #if bad, go to Error

In this example we use branch on equal to do an index out of bounds check.

**Branch On Not Equal**

**Description**

Branch if not equal means go to the label statement if the value of register one does not equal the value in register two

**Instruction Format**

**I - type**

|  |  |  |  |
| --- | --- | --- | --- |
| 0101 | rs | rt | Immediate |

31 26 25 21 20 16 15 0

**MIPS Format**

bne rs, rt

**Operation**

if(R[rs] != R[rt])

PC = PC + 4 + BranchAddr

**Example**

f, g, h, i, j correspond to the registers $s0 through $s4

if(i == j) f = g + h;

else f = g - h

bne $s3, $s4, Else #go to else if i != j

In this example we can use branch on not equal to implement an else statement if the two registers s3 and s4 are not equal to each other.

**Unconditional Jump and Subroutine Call/Return Instructions**

**Jump**

**Description**

Jump is used to go to a desired instruction.

**Instruction Format**

**J - type**

|  |  |
| --- | --- |
| 0010 | 26 bit address |

31 26 25 0

**MIPS Format**

j label

**Operation**

PC = JumpAddr

**Example**

f, g, h, i , j correspond to registers $s0 through $s4

if (i ==j) f = g + h; else f = g - h;

ben $s3, $s4, Else #go to Else if i != j

add $s0, $s1, $s2 #f = g + h (skipped if != j)

j Exit #go to Exit

Else:

sub $s0, $s1, $s2 #f = g - h (skipped if i = j)

Exit:

In this example we can use the jump instruction to break out of an if statement.

**Immediate Operand Instructions**

**Add Immediate**

**Description**

Add immediate is used to add constants to registers.

**Instruction Format**

**I - type**

|  |  |  |  |
| --- | --- | --- | --- |
| 1100 | rs | rt | constant or address |

31 26 25 21 20 16 15 0

**MIPS Format**

addi rt, rs, imm

**Operation**

R[rt] = R[rs] + SignExtImm

**Example**

addi $s1, $s2, 20 #$s1 = $s2 + 20

After this instruction executes s1 will have the value of s2 plus 20.

**SIMD Instructions**

**Vector Add Saturated (unsigned)**

**Description**

Each element of *a* is added to the corresponding element of *b*. The unsigned-integer is placed into the corresponding element of d.

**Instruction Format**

**V - Type**

opcode function destination source source2 source3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0111 | 0000 | vd | vs | vt | 00000 |

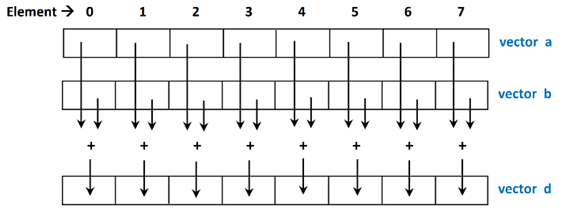
31 28 27 24 23 18 17 12 11 6 5 0

**MIPS Format**

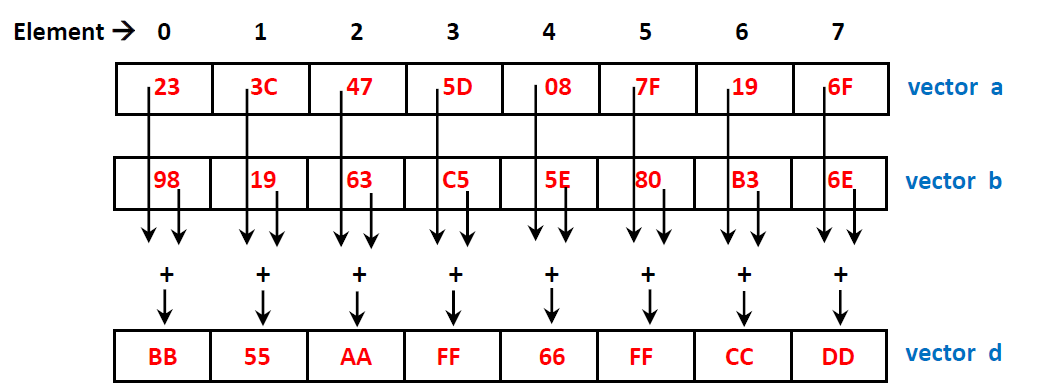
vec\_addsu d, a, b

**Operation**

V[d] = V[a] + V[b]



**Example**

****

**Vector Multiply and Add**

**Description**

Each element in vector *a* is multiplied by the element in vector *b*. The intermediate result is then added to the element in vector *c*. The final result is placed into the corresponding element in vector *d.*

**Instruction Format**

**V - Type**

opcode function destination source source2 source3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0111 | 0001 | vd | vs | vt | vu |

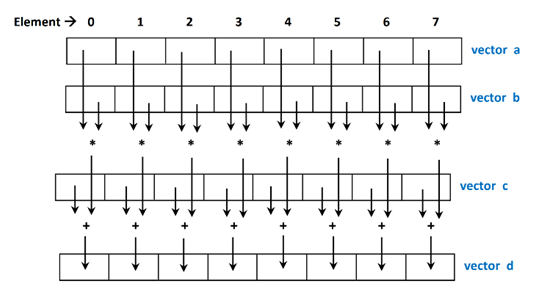
31 28 27 24 23 18 17 12 11 6 5 0

**MIPS Format**

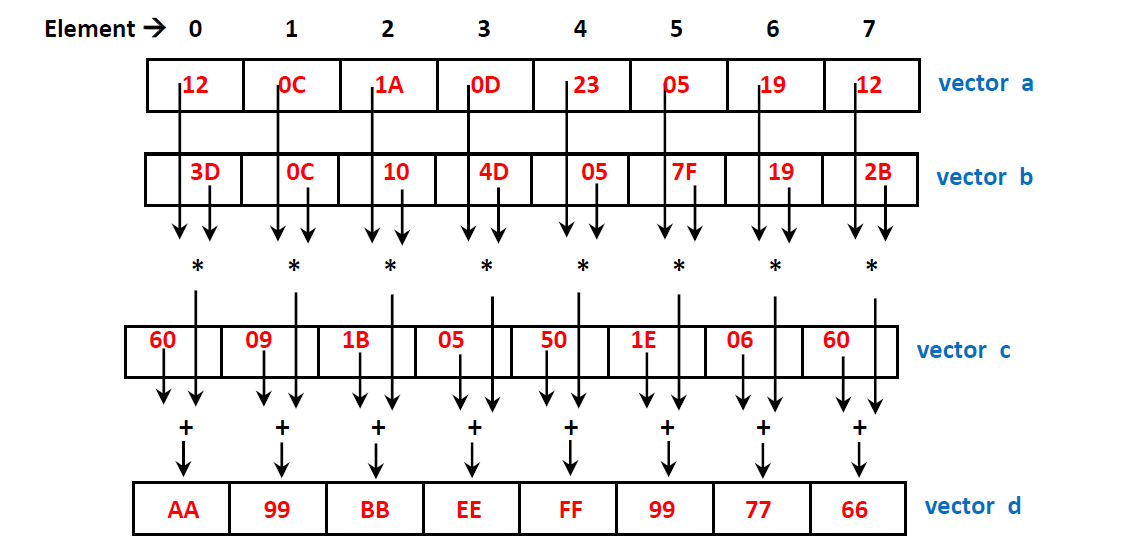
vec\_madd d, a, b, c

**Operation**

V[d] = V[a] + V[b] \* V[c]



**Example**

****

**Vector Multiply Even Integer**

**Description**

Each element of vector *d* is the full-length (16 bit) product of the corresponding high half-width elements of vector *a* and vector *b*.

**Instruction Format**

**V - Type**

opcode function destination source source2 source3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0111 | 0010 | vd | vs | vt | 00000 |

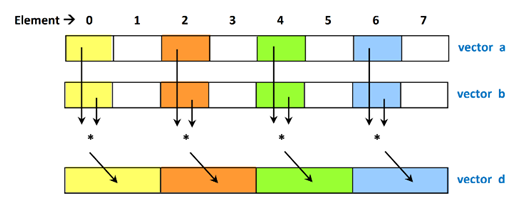
31 28 27 24 23 18 17 12 11 6 5 0

**MIPS Format**

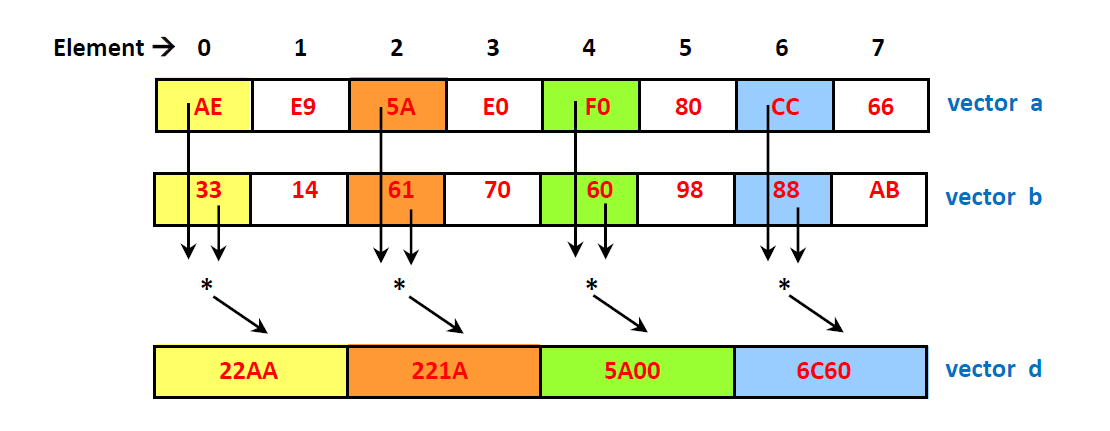
vec\_mule d, a, b

**Operation**

V[d] = V[a] \* V[b]



**Example**

****

**Vector Multiply Odd Integer**

**Description**

Each element of vector *d* is the full-length (16 bit) product of the corresponding low half-width elements of vector *a* and vector *b*.

**Instruction Format**

**V - Type**

opcode function destination source source2 source3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0111 | 0011 | vd | vs | vt | 00000 |

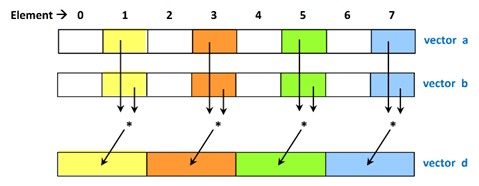
31 28 27 24 23 18 17 12 11 6 5 0

**MIPS Format**

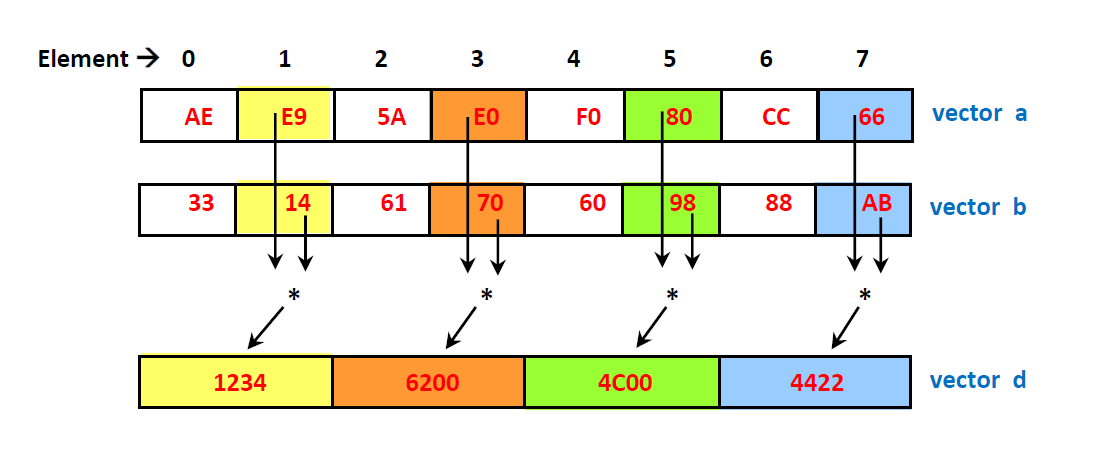
vec\_mulo d, a, b

**Operation**

V[d] = V[a] \* V[b]



**Example**

****

**Vector Multiply Sum Saturated**

**Description**

Each element of vector *d* is the 16-bit sum of the corresponding elements of vector *c* and the 16-bit “temp” products of the 8-bit elements of vector *a* and vector *b* which overlap the positions in *c*. The sum is performed with 16-bit saturating addition.

**Instruction Format**

**V - Type**

opcode function destination source source2 source3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0111 | 0100 | vd | vs | vt | vu |

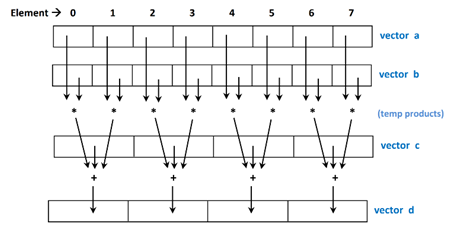
31 28 27 24 23 18 17 12 11 6 5 0

**MIPS Format**

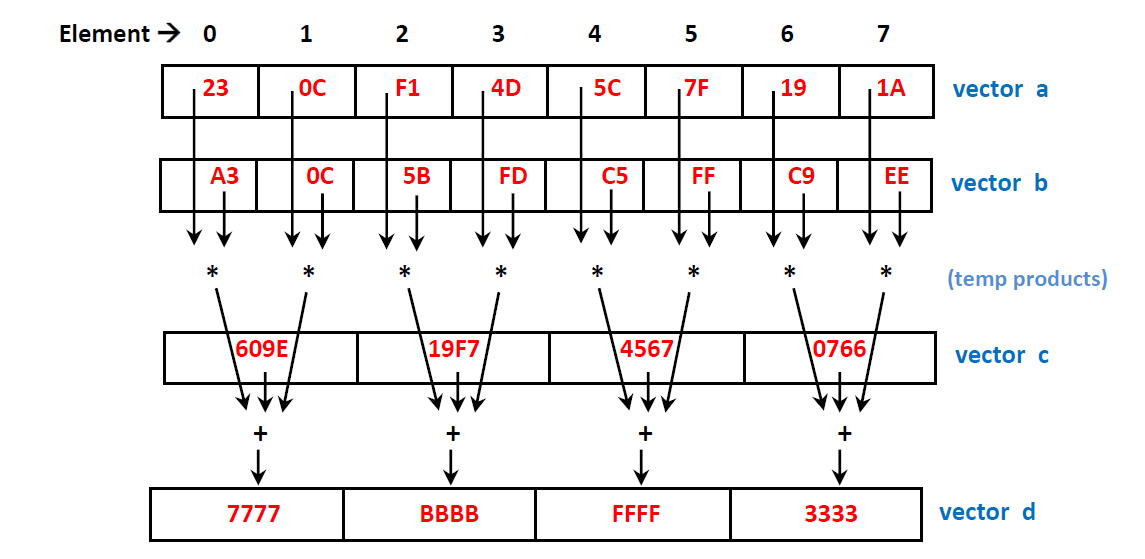
vec\_msums d, a, b, c

**Operation**

V[d] = V[a] + V[b] \* V[c]



**Example**

****

**Vector Splat**

**Description**

Used to copy an element from one vector into all elements of another vector.

**Instruction Format**

**VI - Type**

opcode function destination source 11-bit immediate value

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0111 | 10000 | vd | vs | immediate |

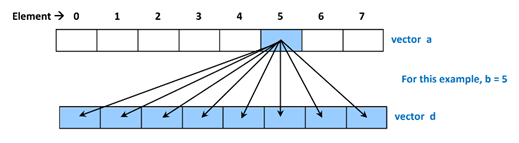
31 28 27 23 22 17 16 11 10 0

**MIPS Format**

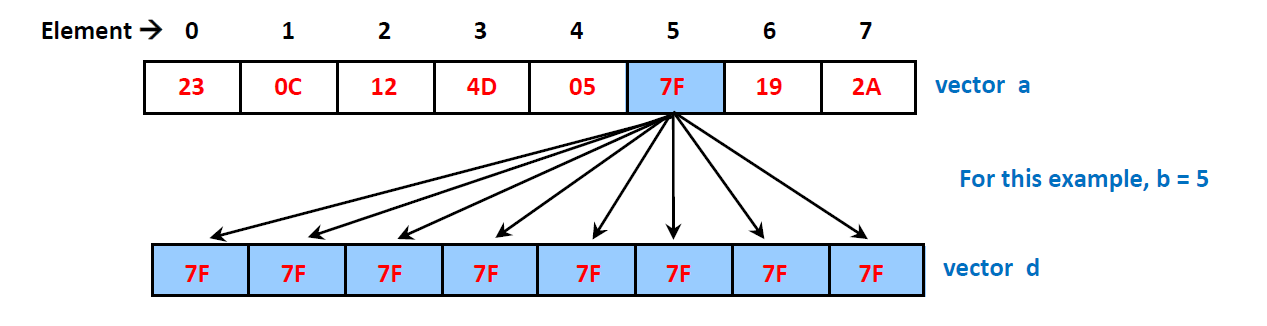
Vec\_splat d, a, b

**Operation**

V[d] = V[a](imm11)



**Example**

****

**Vector Merge Low**

**Description**

The even elements of the result vector *d* are obtained left-to-right from the low elements of vector *a*. The odd elements of the result are obtained left-to-right from the low elements of vector *b*.

**Instruction Format**

**V - Type**

opcode function destination source source2 source3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0111 | 0101 | vd | vs | vt | 000000 |

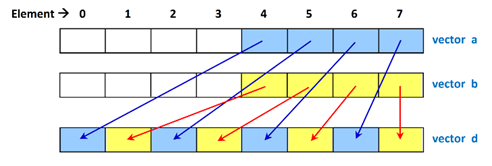
31 28 27 24 23 18 17 12 11 6 5 0

**MIPS Format**

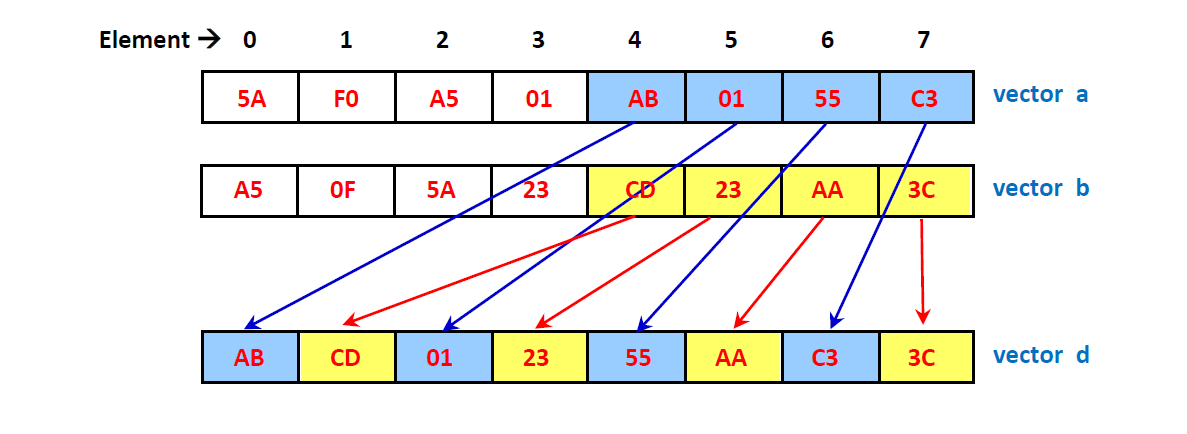
vec\_mergel d, a, b

**Operation**

V[d] = V[a], V[d] = V[b]



**Example**

****

**Vector Merge High**

**Description**

The even elements of the result vector *d* are obtained left-to-right from the high elements of vector *a*. The odd elements of the result are obtained left-to-right from the high elements of vector *b*.

**Instruction Format**

**V - Type**

opcode function destination source source2 source3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0111 | 0110 | vd | vs | vt | 000000 |

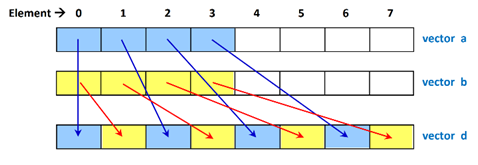
31 28 27 24 23 18 17 12 11 6 5 0

**MIPS Format**

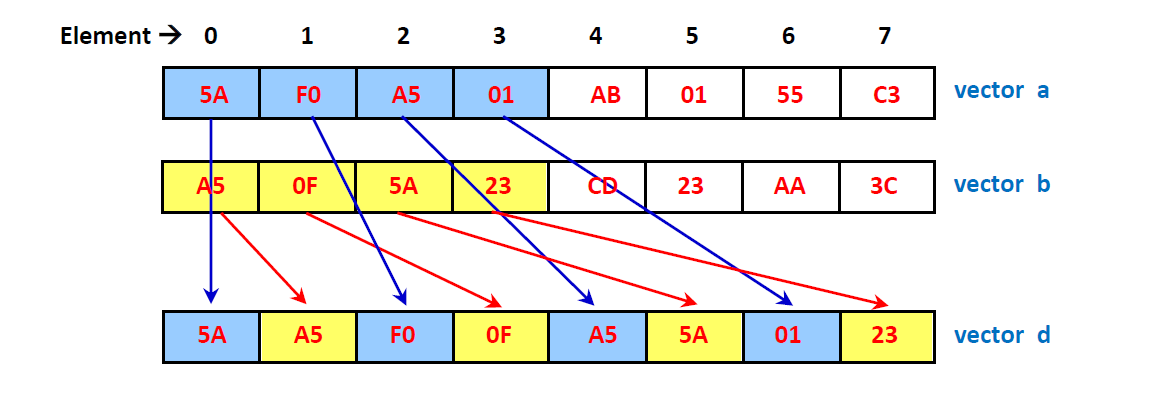
vec\_mergeh d, a, b

**Operation**

V[d] = V[a], V[d] = V[b]



**Example**

****

**Vector Pack**

**Description**

Each high element of the result vector *d* is the truncation of the corresponding wider element of vector *a*. Each low element of the result is the truncation of the corresponding wider element of vector *b*.

**Instruction Format**

**V - Type**

opcode function destination source source2 source3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0111 | 0111 | vd | vs | vt | 000000 |

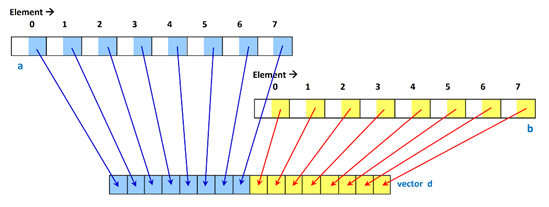
31 28 27 24 23 18 17 12 11 6 5 0

**MIPS Format**

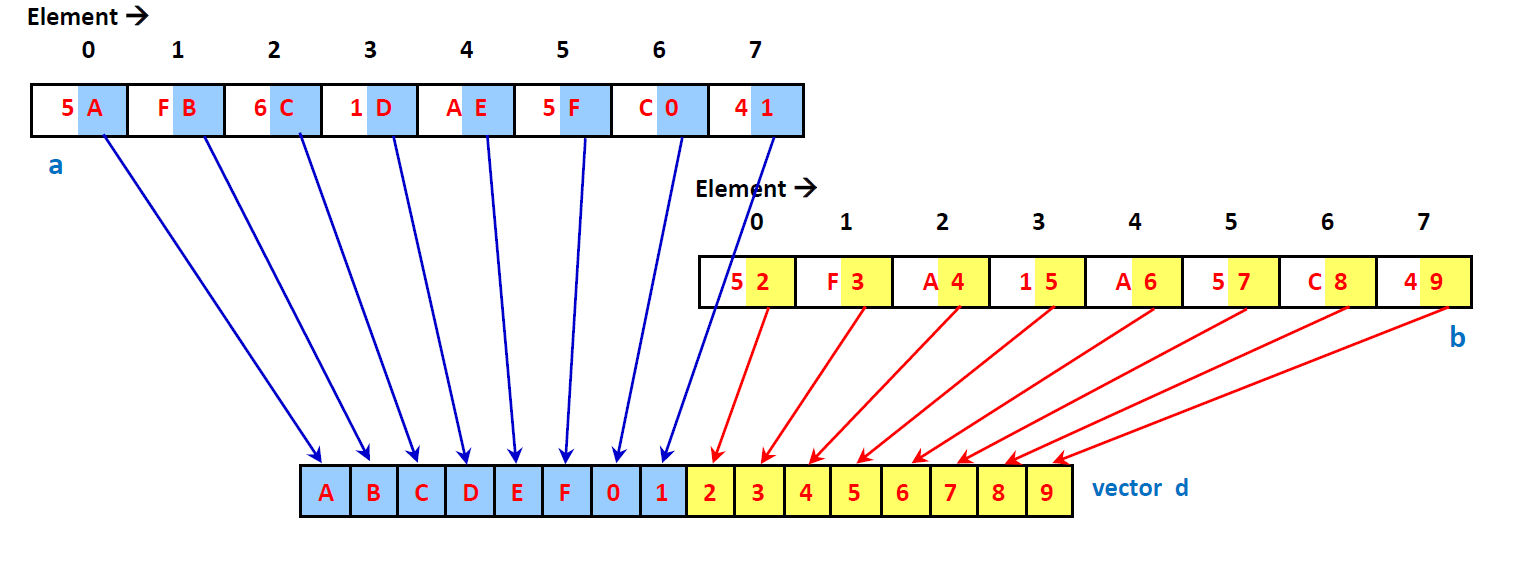
vec\_pack d, a, b

**Operation**

V[d] = V[a], V[d] = V[b]



**Example**

****

**Vector Permute**

**Description**

The instruction fills the result vector *d* with elements from either vector *a* or vector *b*, depending upon the “element specifier” in vector c. The vector elements can be specified in any order.

**Instruction Format**

**V - Type**

opcode function destination source source2 source3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0111 | 1000 | vd | vs | vt | vu |

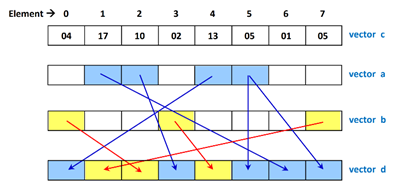
31 28 27 24 23 18 17 12 11 6 5 0

**MIPS Format**

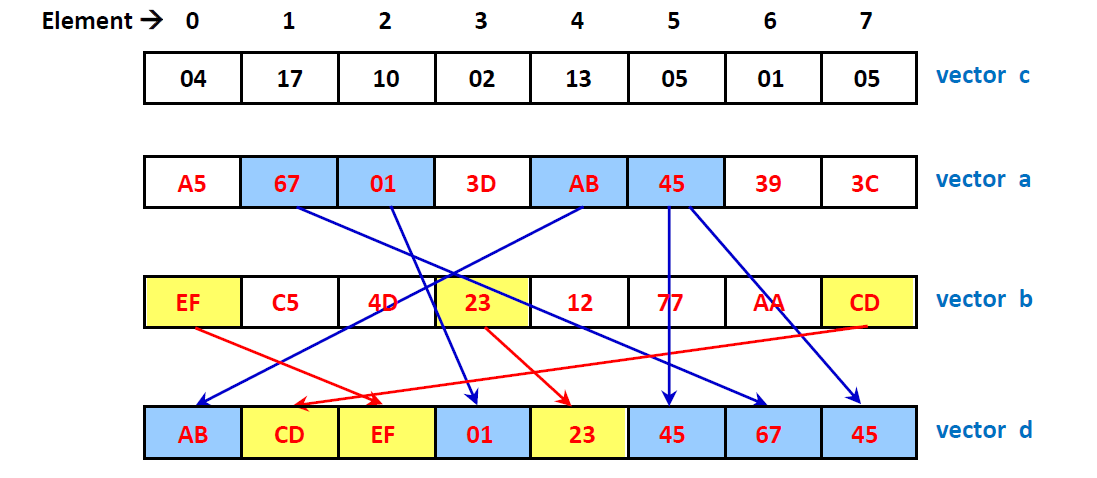
vec\_perm d, a, b, c

**Operation**

V[d] = V[c]decode(V[a]), V[d] = V[c]decode(V[b])



**Example**

****

**Vector Compare Equal-To**

**Description**

Each element of the result vector *d* is TRUE (all bits = 1) if the corresponding element of vector *a* is equal to the corresponding element of vector *b*. Otherwise the element of result is FALSE (all bits = 0).

**Instruction Format**

**V - Type**

opcode function destination source source2 source3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0111 | 1001 | vd | vs | vt | 000000 |

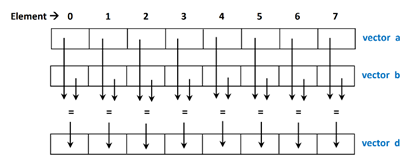
31 28 27 24 23 18 17 12 11 6 5 0

**MIPS Format**

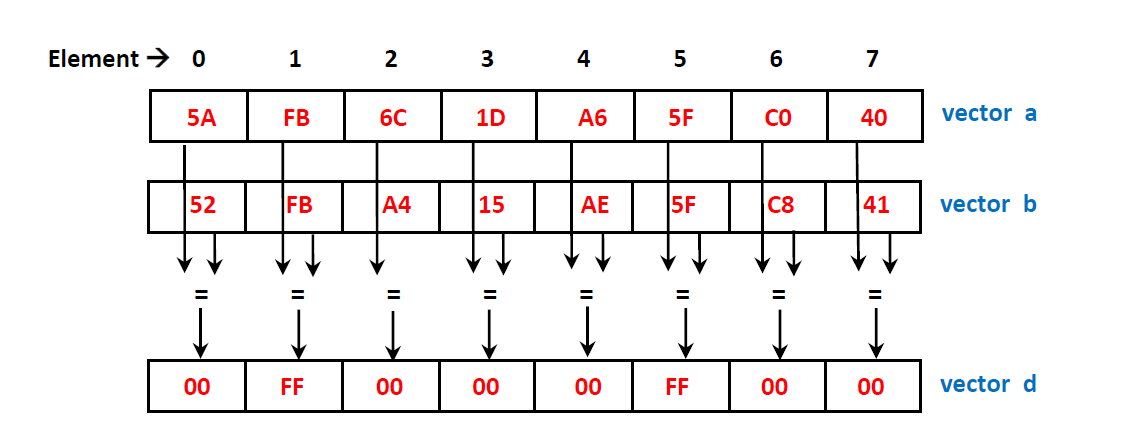
vec\_cmpeq d, a, b

**Operation**

V[d] = (V[a] == V[b])



**Example**

****

**Vector Compare Less-Than (unsigned)**

**Description**

Each element of the result vector *d* is TRUE (all bits = 1) if the corresponding element of vector *a* is less-than the corresponding element of vector *b*. Otherwise the element of result is FALSE (all bits = 0).

**Instruction Format**

**V - Type**

opcode function destination source source2 source3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0111 | 1010 | vd | vs | vt | 000000 |

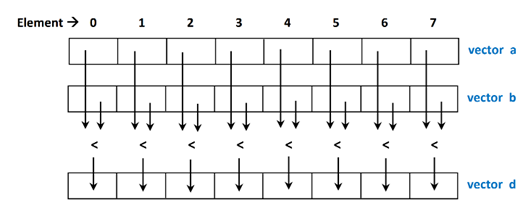
31 28 27 24 23 18 17 12 11 6 5 0

**MIPS Format**

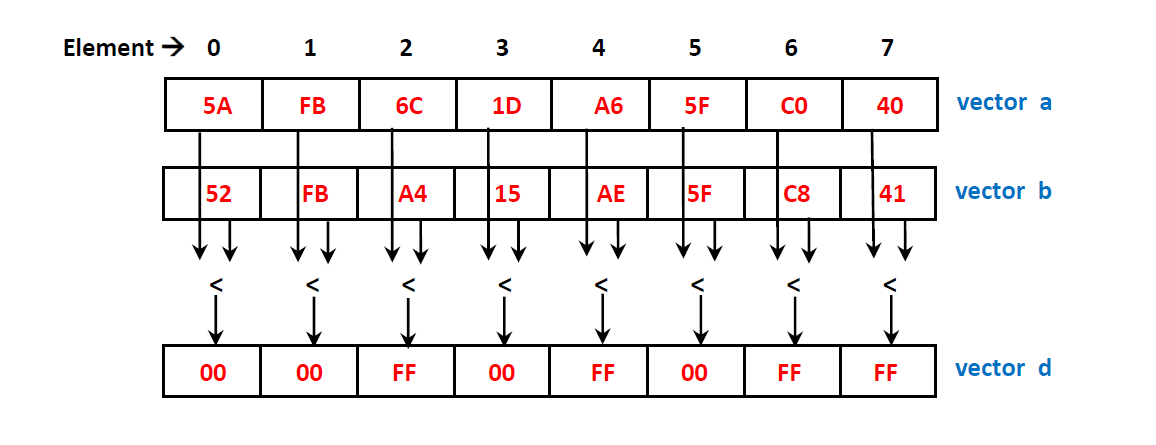
Vec\_cmpltu d, a, b

**Operation**

V[d] = (V[a] < V[b])



**Example**

****

**Vector Swap (16-bit)**

**Description**

Takes elements from the lower half of the vector and replaces them with the higher half. The higher half elements are replaced with the lower half.

**Instruction Format**

**V-type**

opcode function destination source 11-bit immediate

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0111 | 1010 | vd | vs | 0000000000 |

31 28 27 23 22 17 16 11 10 0

**MIPS Format**

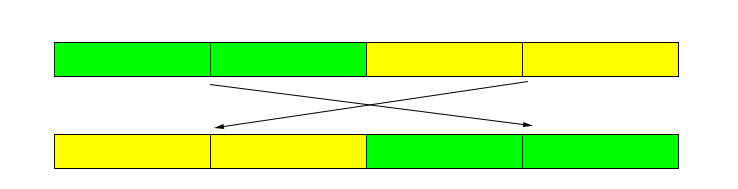
Vec\_swap $vd, $vs

**Operation**

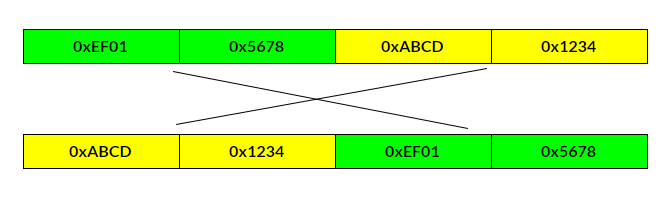
$t = $vs

$vs = $vd

$vd = $vs



**Example**



**Vector Shift Left (16-bit)**

**Description**

Each element in vector *a* is shifted to the left once; the result is then placed into vector *d*. If there is an overflow, then the value will convert back to 16-bits (0xFFFF)

**Instruction Format**

**V-type**

opcode function destination source 11-bit immediate

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0111 | 1010 | vd | vs | 0000000000 |

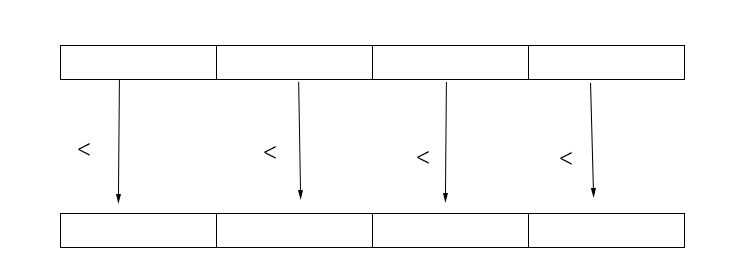
31 28 27 23 22 17 16 11 10 0

**MIPS Format**

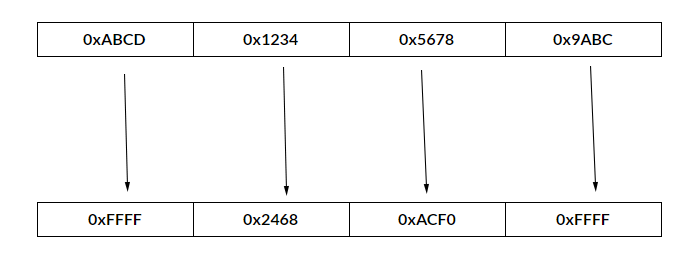
Vec\_sll $vd, $vs

**Operation**

V[d] = (V[a] < < 1)



**Example**

****

**Vector Shift Right (16-bit)**

**Description**

Each element in vector *a* is shifted to the right once; the result is then placed into vector *d*.

**Instruction Format**

**V-type**

opcode function destination source 11-bit immediate

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0111 | 1010 | vd | vs | 0000000000 |

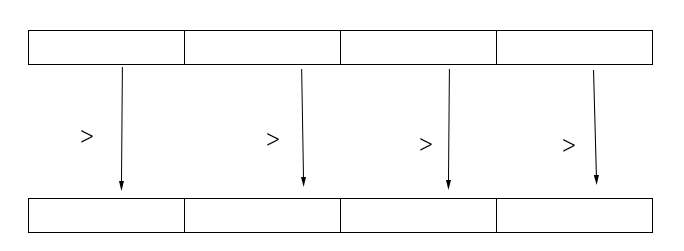
31 28 27 23 22 17 16 11 10 0

**MIPS Format**

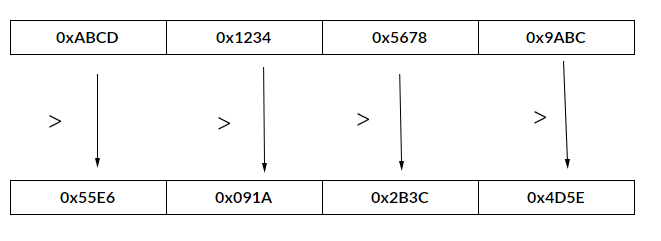
Vec\_srl $vd, $vs

**Operation**

V[d] = (V[a] >> 1)



**Example**



**Vector Copy**

**Description**

Takes elements from vector *a* and copies them into vector *d*

**Instruction Format**

**V-type**

opcode function destination source 11-bit immediate

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0111 | 1010 | vd | vs | 0000000000 |

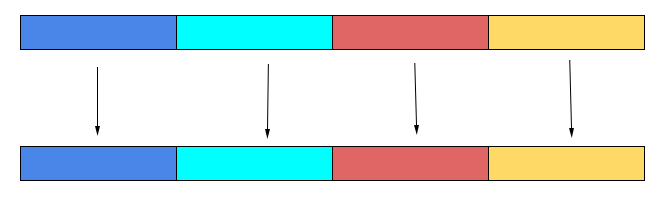
31 28 27 23 22 17 16 11 10 0

**MIPS Format**

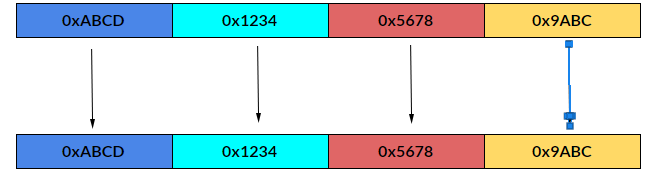
Vec\_cpy $vd, $vs

**Operation**

V[d] = V[a]

****

**Example**



**Vector Contains**

**Description**

Using an element specifier, if an element in vector *a* is the same as element specifier *b*, then insert 0xFF for all occurrences in vector *d*.

**Instruction Format**

**VI-type**

opcode function destination source 11-bit immediate

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0111 | 1010 | vd | vs | 0000000000 |

31 28 27 23 22 17 16 11 10 0

**MIPS Format**

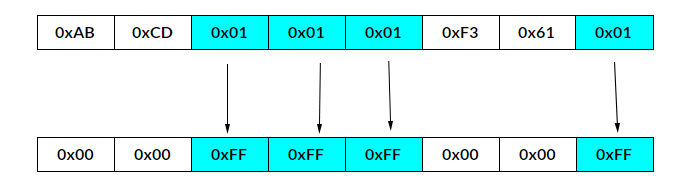
Vec\_cnt $vd, $vs, b

**Operation**

V[d] = V[a], b

**Example**

B = 0x01



**Vector AND**

**Description**

The elements in vector *a* are compared with the elements in vector *b* using the AND operator. The result is placed in vector *d*

**Instruction Format**

**V-type**

opcode function destination source source2 source3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0111 | 1010 | vd | vs | vt | 000000 |

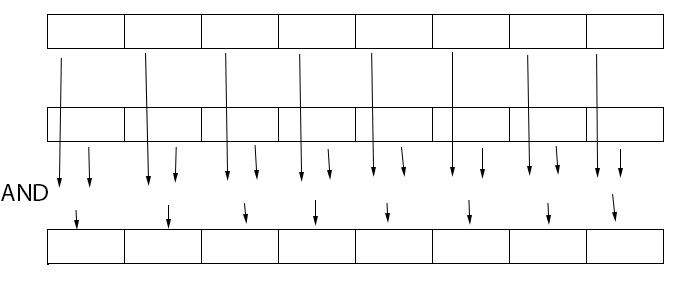
31 28 27 24 23 18 17 12 11 6 5 0

**MIPS Format**

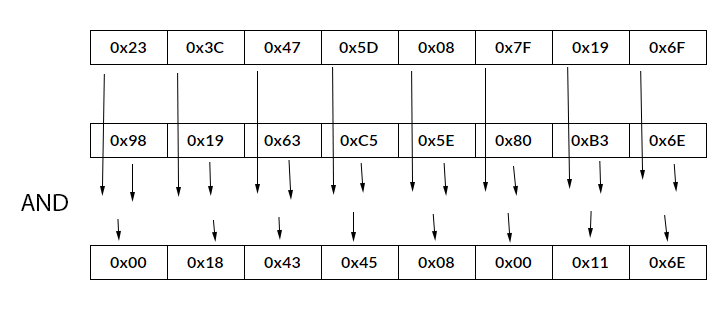
Vec\_and $vd, $vs, $vt

**Operation**

V[d] = V[a] AND V[b]

****

**Example**

****

**Vector OR**

**Description**

The elements in vector *a* are compared with the elements in vector *b* using the OR operator. The result is placed in vector *d*

**Instruction Format**

**V-type**

opcode function destination source source2 source3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0111 | 1010 | vd | vs | vt | 000000 |

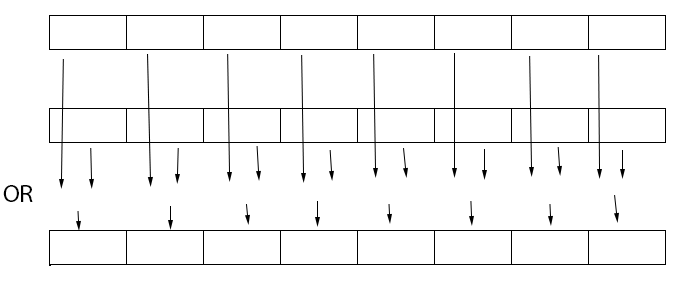
31 28 27 24 23 18 17 12 11 6 5 0

**MIPS Format**

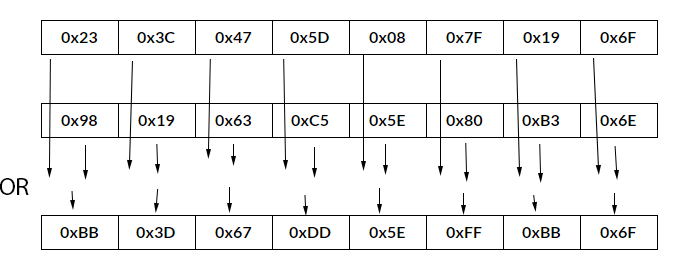
Vec\_or $vd, $vs, $vt

**Operation**

V[d] = V[a] OR V[b]

****

**Example**

****

**Vector Subtract Unsigned**

**Description**

Each element of *a* is subtracted to the corresponding element of *b*. The unsigned-integer is placed into the corresponding element of d.

**Instruction Format**

**V - Type**

opcode function destination source source2 source3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0111 | 0000 | vd | vs | vt | 00000 |

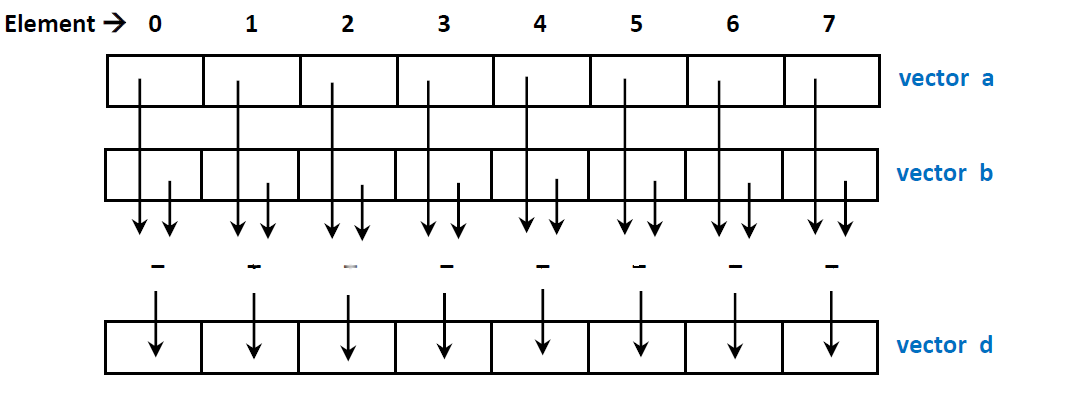
31 28 27 24 23 18 17 12 11 6 5 0

**MIPS Format**

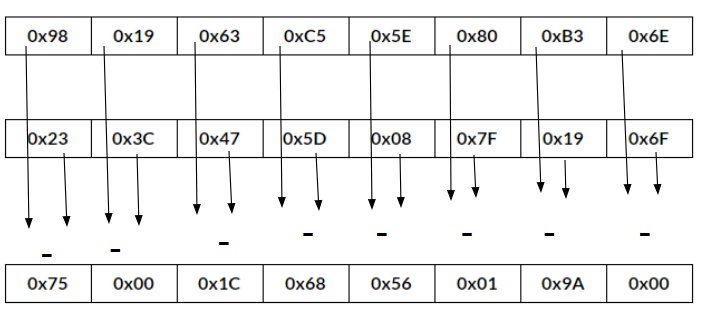
vec\_subu d, a, b

**Operation**

V[d] = V[a] - V[b]

****

**Example**

****

**E. Summary of the Instruction Formats of “Baseline SIMD Enhancements” and “Application Specific” enhancements**

**Basic Formats**

**R-Type**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| opcode | source | source | destination | Shift amount | Function |
| 000000 | rs | rt | rd | shamt |  |
| 31 26 | 25 21 | 20 16 | 15 11 | 10 6 | 5 0 |

**I-Type**

opcode source destination 16-bit immediate value

|  |  |  |  |
| --- | --- | --- | --- |
|  | rs | rt | Immediate value |

31 26 25 21 20 16 15 0

**J-Type**

opcode

|  |  |
| --- | --- |
|  | 26 bit address |

31 26 25 0

**V-type**

opcode function destination source source2 source3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0111 | 1010 | vd | vs | vt | 000000 |

31 28 27 24 23 18 17 12 11 6 5 0

**VI-type**

opcode function destination source 11-bit immediate

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0111 | 1010 | vd | vs | 0000000000 |

31 28 27 23 22 17 16 11 10 0

III. MIPS Implementations

1. **Vector Add Saturated**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: 1\_vector\_add\_sat.asm

# Verson: 2.0

# Date: November 23, 2018

# Programmer: Brian Nguyen

# Description: Each element of vector a is added to the

# corresponding

# element of vector b.

# The unsigned-integer is placed into the corresponding

# element of vector d

#

# vec\_addsu d, a, b

#

#

#

#

# Notes: vec\_addsu is the AltiVec analog of the awdd unsigned # bytes

# available in the PowerPC scalar instruction set

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

main:

la $a0, a # initialize vector a

la $a1, b # initialize vector b

la $a2, d # initialize vector d

lw $t3, size # initialize the loop breaker

li $t4, 0 # initialize count variable

li $t5, 0xFF # load the limit

j loop

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Branch used to store 0xFF in an index

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

store:

sw $t5, 0($a2) # Stores 0xFF into vector d

addi $a0,$a0, 4 # iterate to next element in vector a

addi $a1,$a1, 4 # iterate to next element in vector b

addi $a2, $a2, 4 # iterate to the next element in vector d

addi $t4, $t4, 1 # increment the count variable

b loop

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Go through each element in both

# vectors and add them

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

loop:

lw $t0, 0($a0) # load element from vector a

lw $t1, 0($a1) # load element from vector b

addu $t1, $t1, $t0 # sum of elements unsigned

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

slt $t6, $t5, $t1 # if the sum of the registers is greater than

bne $t6, $zero, store # 0xFF, then go to the "store" branch

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

sw $t1, 0($a2) # store sum in vector d

addi $a0,$a0, 4 # iterate to next element in vector a

addi $a1,$a1, 4 # iterate to next element in vector b

addi $a2, $a2, 4 # iterate to the next element in vector d

addi $t4, $t4, 1 # increment the count variable

bne $t4, $t3, loop # once the count variable reaches the end of

# the sizes of the vectors (8), the loop will

# break

mul $t3, $t3, 4

sub $a2, $a2, $t3

lw $t0, 0($a2)

lw $t1, 4($a2)

sll $t0, $t0, 8

add $t0, $t0, $t1

sll $t0, $t0, 8

addi $a2, $a2, 4

lw $t1, 4($a2)

add $t0, $t0, $t1

sll $t0, $t0, 8

lw $t1, 8($a2)

add $t0, $t0,$t1

lw $t1, 12($a2)

lw $t2, 16($a2)

sll $t1, $t1, 8

add $t1, $t1, $t2

sll $t1, $t1, 8

lw $t2, 20($a2)

add $t1, $t1, $t2

sll $t1, $t1, 8

lw $t2, 24($a2)

add $t1, $t1, $t2

exit: ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data # [0] [1] [2] [3] [4] [5] [6] [7]

a: .word 0x23, 0x3C, 0x47, 0x5D, 0x08, 0x7F, 0x19, 0x6F

b: .word 0X98, 0X19, 0X63, 0XC5, 0X5E, 0X80, 0XB3, 0X6E

d: .space 8

size: .word 8

**2. Vector Multiply Add**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: 2\_vector\_multiply\_\_add.asm

# Verson: 1.0

# Date: November 22, 2018

# Programmer: Brian Nguyen

# Description: Multiply the vector elements in a by the vector

# elements in b and then add the immediate result to

# the vector elements in c, storing each result

# in vector d, in one instruction and in one rounding #

#

# vec\_madd d, a, b, c

#

#

#

#

#

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text # main (must be global)

.globl main

main:

la $a0, vec\_a # load vector a

la $a1, vec\_b # load vector b

la $a2, vec\_c # load vector c

la $a3, vec\_d # load vector d (empty vector)

li $s0, 0x100 # load the divisor

lw $t3, size # initialize the size of the vectors

li $t4, 0 # initialize counter variable

j loop

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# branch used for performing modulus on an overflow

# so that it remains 8-bit

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

str\_limit:

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

div $t2, $s0 # Pseudoinstruction for

mfhi $t2 # modulus: $d = $s % $t

sw $t2, 0($a3) #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

addi $a0, $a0, 4 # increment positions for all vectors

addi $a1, $a1, 4

addi $a2, $a2, 4

addi $a3, $a3, 4

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

addi $t4, $t4, 1 # increment counter variable

beq $t3, $t4, exit # when counter variable reaches its limit,

# then exit the loop

b loop

loop:

lw $t0, 0($a0) # load element from vector a

lw $t1, 0($a1) # load element from vector b

mul $t1, $t1, $t0 # multiply the two elements together

lw $t2, 0($a2) # load element from vector c

add $t2, $t2, $t1 # add the result of the products and

# the element loaded from vector d

slt $t5, $s0, $t2 # if the result is greater than 0x100, go

# to branch str\_limit

bne $t5, $zero, str\_limit

sw $t2, 0($a3) # store result in vector d

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

addi $a0, $a0, 4 # increment positions in all vectors

addi $a1, $a1, 4

addi $a2, $a2, 4

addi $a3, $a3, 4

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

addi $t4, $t4, 1 # increment counter variable

bne $t3, $t4, loop # when counter variable reaches its

# limit,then exit the loop

exit:

mul $t3, $t3, 4

sub $a3, $a3, $t3

lw $t0, 0($a3)

lw $t1, 4($a3)

sll $t0, $t0, 8

add $t0, $t0, $t1

sll $t0, $t0, 8

addi $a2, $a2, 4

lw $t1, 8($a3)

add $t0, $t0, $t1

sll $t0, $t0, 8

lw $t1, 12($a3)

add $t0, $t0, $t1

lw $t1, 16($a3)

lw $t2, 20($a3)

sll $t1, $t1, 8

add $t1, $t1, $t2

sll $t1, $t1, 8

lw $t2, 24($a3)

add $t1, $t1, $t2

sll $t1, $t1, 8

lw $t2, 28($a3)

add $t1, $t1, $t2

ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data # [0] [1] [2] [3] [4] [5] [6] [7]

vec\_a: .word 0x12, 0x0C, 0x1A, 0x0D, 0x23, 0x05, 0x19, 0x12

vec\_b:.word 0X3D, 0X0C, 0X10, 0X4D, 0X05, 0X7F, 0X19, 0X2B

vec\_c: .word 0x60, 0x09, 0x1B, 0x05, 0x50, 0x1E, 0x06, 0x60

vec\_d: .space 8

size: .word 8

**3. Vector Multiply Even**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: 3\_vector\_mul\_even.asm

# Verson: 1.0

# Date: November 4, 2018

# Programmer: Brian Nguyen

# Description: Using a sequence of MIPS instructions, have two

# vectors where each even element of one vector is

# multiplied to the corresponding even element of

# another vector. The result is placed into an empty

# vector

#

# vec\_mule d, a, b

#

#

#

#

#

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text # main (must be global)

.globl main

main:

la $a0, a # initialize vector a

la $a1, b # initialize vector b

la $a2, d # initialize d (an empty vector)

lw $t3, size # the size of the vector (4)

li $t4, 0 # count variable

j loop

loop:

lw $t0, 0($a0) # retrieve element from vector a

lw $t1, 0($a1) # retrieve element from vector b

mul $t2, $t0, $t1 # multiply elements

sw $t2, 0($a2) # store result in vector d

# increment by two to retrieve even elements

addi $a0,$a0, 8

addi $a1,$a1, 8

addi $a2,$a2,4 # increment vector d by one

addi $t4, $t4, 1 # increment counter variable

bne $t4,$t3,loop

mul $t3, $t3, 4

sub $a2, $a2, $t3

lw $t0, 0($a2)

lw $t1, 4($a2)

sll $t0, $t0, 8

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t2, 8($a2)

lw $t3, 12($a2)

sll $t2, $t2, 8

sll $t2, $t2, 8

add $t2, $t2, $t3

exit: ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

proc1: j proc1 # placeholder stub

.data # [0] [1] [2] [3] [4] [5] [6] [7]

a: .word 0xAE, 0xE9, 0x5A, 0x50, 0xF0, 0x80, 0xCC, 0x66

b: .word 0X33, 0X14, 0X61, 0X70, 0X60, 0X98, 0X88, 0XAB

d: .space 4

size: .word 4

**4. Vector Multiply Odd**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: 4\_vector\_mul\_odd.asm

# Verson: 1.0

# Date: November 4, 2018

# Programmer: Brian Nguyen

# Description: Using a sequence of MIPS instructions, have two of

# one vectors where each odd element

# vector is multiplied to the corresponding odd element # of another vector. The result

# is placed into an empty vector

#

# vec\_mulo d, a, b

#

#

#

# Notes:

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text # main (must be global)

.globl main

main:

la $a0, vec\_a # initialize vector a

la $a1, vec\_b # initialize vector b

la $a2, vec\_d # initialize d (an empty vector)

lw $t3, size # the size of the vector (4)

li $t4, 0 # count variable

# the pointers are automatically incremented by

# 1 before the loop so they start at index 1

# of each vector

addi $a0,$a0, 4

addi $a1,$a1, 4

j loop

loop:

lw $t0, 0($a0) # retrieve element from vector a

lw $t1, 0($a1) # retrieve element from vector b

mul $t2, $t0, $t1 # multiply elements

sw $t2, 0($a2) # store result in vector d

# increment by two to retrieve odd elements

addi $a0,$a0, 8

addi $a1,$a1, 8

addi $a2,$a2,4 # increment vector d by one

addi $t4, $t4, 1 # increment counter variable

bne $t4,$t3,loop # if the pointer reaches the end of both

# arrays, then exit loop

mul $t3, $t3, 4

sub $a2, $a2, $t3

lw $t0, 0($a2)

lw $t1, 4($a2)

sll $t0, $t0, 8

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t2, 8($a2)

lw $t3, 12($a2)

sll $t0, $t0, 8

sll $t0, $t0, 8

add $t2, $t2, $t3

exit:

ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

.data # [0] [1] [2] [3] [4] [5] [6] [7]

vec\_a: .word 0xAE, 0xE9, 0x5A, 0xE0, 0xF0, 0x80, 0xCC, 0x66

vec\_b: .word 0X33, 0X14, 0X61, 0X70, 0X60, 0X98, 0X88, 0XAB

vec\_d: .space 4

size: .word 4

**5. Vector Multiply Sum Saturated**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: 5\_vector\_multiply\_sum\_sat.asm

# Verson: 1.4

# Date: November 21, 2018

# Programmer: Brian Nguyen

# Description: Each element of vector d is the 16-bit sum of the

# corresponding elements of vector c and the 16-bit

# "temp" products of the 8-bit

# elements of vector a and vector b which overlap the

# positions of that in vector c. The sum is performed

# with 16-bit saturating addition (no-wrap)

#

# vec\_msums d, a, b

#

#

#

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text # main (must be global)

.globl main

main:

la $a0, a # load vector a

la $a1, b # load vector b

la $a2, c # load vector c

la $a3, d # load vector d

lw $t3, size # initialize the size of the vectors

li $s0, 0xFFFF # load word limit

li $t4, 0 # initialize counter variable

j loop

str\_limit:

sw $t6, 0($a3) # store 0xFFFF into vector d

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

addi $a0, $a0, 4 # the positions of all vectors get incremented

addi $a1, $a1, 4 #

addi $a2, $a2, 4 #

addi $a3, $a3, 4 #

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

addi $t4, $t4, 1 # loop counter gets incremented

loop:

lw $t0, 0($a0) # load element from vector a

lw $t1, 0($a1) # load element from vector b

mul $t1, $t1, $t0 # multiply two elements from vector a and b

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

addi $a0, $a0, 4 # the positions of both vectors

addi $a1, $a1, 4 # get incremented

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

lw $t5, 0($a0) # load element from vector a

lw $t6, 0($a1) # load element from vector b

mul $t6, $t6, $t5 # multiply two elements from vector a and b

add $t6, $t6, $t1 # add the products from both resultants

lw $t7, 0($a2) # load element from vector c

add $t7, $t7, $t6 # add the product from

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

slt $s1, $s0, $t7 # checks if the resultant

bne $s1, $zero, str\_limit # is greater than 0xFFFF

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

sw $t7, 0($a3) # store the resultant into vector d

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

addi $a0, $a0, 4 # the positions of all vectors

Addi $a1, $a1, 4 # get incremented

addi $a2, $a2, 4

addi $a3, $a3, 4

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

addi $t4, $t4, 1 # loop counter gets incremented

bne $t4, $t3, loop # if loop counter reaches its limit, then exit.

combine:

mul $t3, $t3, 4

sub $a2, $a2, $t3

lw $t0, 16($a2)

lw $t1, 20($a2)

sll $t0, $t0, 8

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t2, 24($a2)

lw $t3, 28($a2)

sll $t2, $t2, 8

sll $t2, $t2, 8

add $t2, $t2, $t3

exit: ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

proc1: j proc1 # placeholder stub

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data # [0] [1] [2] [3] [4] [5] [6] [7]

a: .word 0x23, 0x0C, 0xF1, 0x4D, 0x5C, 0x7F, 0x19, 0x1A

b: .word 0XA3, 0X0C, 0X5B, 0XFD, 0XC5, 0XFF, 0XC9, 0XEE

c: .word 0x609E, 0x19F7, 0x4567, 0x0766

d: .space 4

size: .word 4

**6. Vector Splat**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: 6\_vector\_splat.asm

# Verson: 1.0

# Date: November 4, 2018

# Programmer: Brian Nguyen

# Description: The "splat" instruction is used to copy any element

# from one vector into all the elements

# of another vector. Each element of the result of

# vector d is a component b of vector a

#

# vec\_splat d, a, b

#

#

#

#

#

#

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text # main (must be global)

.globl main

main:

la $a0, vec\_a # initialize vector a

la $a1, vec\_d # initialize vector d (empty vector)

lw $t0, b # initialize the index

add $t0, $t0, $t0 # double the index

add $t0, $t0, $t0 # double the index again (4x)

add $t1, $t0, $a0

lw $t2, 0($t1) # load index b into register $t4

lw $t3, size # the size of the vector (8)

li $t4, 0 # count variable

j loop

loop:

sw $t2, 0($a1) # $t2 gets stored into the empty vector

addi $a1, $a1, 4 # increment pointer

addi $t4, $t4, 1 # increment count variable

bne $t4, $t3, loop # once the pointer reaches the end of the

# vector, exit

mul $t3, $t3, 4

sub $a1, $a1, $t3

lw $t0, 0($a1)

lw $t1, 4($a1)

sll $t0, $t0, 4

sll $t0, $t0, 4

add $t0, $t0, $t1

lw $t2, 8($a1)

lw $t3, 12($a1)

sll $t2, $t2, 4

sll $t2, $t2, 4

add $t2, $t2, $t3

sll $t0, $t0, 4

sll $t0, $t0, 4

sll $t0, $t0, 4

sll $t0, $t0, 4

add $t0, $t0, $t2

exit:

ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data # [0] [1] [2] [3] [4] [5] [6] [7]

vec\_a: .word 0x23, 0x0C, 0x12, 0x4D, 0x05, 0x7F, 0x19, 0x2A

d: .space 8

b: .word 5

size: .word 8

**7. Vector Merge Low**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: 7\_vector\_merge\_low.asm

# Verson: 1.0

# Date: November 18, 2018

# Programmer: Brian Nguyen

# Description: The even elements of the result of vector d are

# obtained left-to-right from the low elements

# of vector a. The odd elements

# of the result are obtained left-to-right

# from the low elements of vector b

#

# vec\_mergel d, a, b

#

#

#

#

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text # main (must be global)

.globl main

main:

la $a0, vec\_a # initialize vector a

la $a1, vec\_b # initialize vector b

la $a2, vec\_d # initialize vector d

li $t3, 4 # initialize end count

li $t4, 0 # initialize count variable

li $t5, 0 # initialize second count variable

j half\_array # jump to label half\_array

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# The purpose of this label is to reach the middle of the vector

# which will then use the low elements to store into the new array

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

half\_array:

addi $a0, $a0, 4

addi $a1, $a1, 4

addi $t4, $t4, 1

bne $t3, $t4, half\_array

loop:

lw $t0, 0($a0) # retrieve element from vector a

sw $t0, 0($a2) # store element into vector d

addi $a2, $a2, 4 # increment to next element in vector a

lw $t0, 0($a1) # retrieve element from vector b

sw $t0, 0($a2) # store element into vector d

addi $a2, $a2, 4 # increment to next element in vector b

addi $a0,$a0, 4 # iterate to next element in vector a

addi $a1,$a1, 4 # iterate to next element in vector b

addi $t5, $t5, 1 # increment count variable by one

bne $t5, $t4, loop

mul $t3,$t3, 4

mul $t3, $t3, 2

sub $a2, $a2, $t3

lw $t0, 0($a2)

lw $t1, 4($a2)

sll $t0, $t0, 8

add $t0, $t0, $t1

sll $t0, $t0, 8

sll $t0, $t0, 8

lw $t2, 8($a2)

lw $t3, 12($a2)

sll $t2, $t2, 8

add $t2, $t2, $t3

add $t0, $t0, $t2 # first half of vector

lw $t2, 16($a2)

lw $t3, 20($a2)

sll $t2, $t2, 8

add $t2, $t2, $t3

sll $t2, $t2, 8

lw $t3, 24($a2)

add $t2, $t2, $t3

sll $t2, $t2, 8

lw $t3, 28($a2)

add $t2, $t2, $t3

exit:

ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data # [0] [1] [2] [3] [4] [5] [6] [7]

vec\_a: .word 0x5A, 0xF0, 0xA5, 0x01, 0xAB, 0x01, 0x55, 0xC3

vec\_b: .word 0XA5, 0X0F, 0X5A, 0X23, 0XCD, 0X23, 0XAA, 0X3C

vec\_d: .space 8

**8. Vector Merge High**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: 8\_vector\_merge\_high.asm

# Verson: 1.0

# Date: November 18, 2018

# Programmer: Brian Nguyen

# Description: The even elements of the result of vector d are # # obtained left-to-right from the high elements

# of vector a. The odd elements of the

# result are obtained left-to-right

# from the high elements of vector b

#

# vec\_mergeh d, a, b

#

#

#

#

#

#

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text # main (must be global)

.globl main

main:

la $a0, vec\_a # initialize vector a

la $a1, vec\_b # initialize vector b

la $a2, vec\_d # initialize vector d

li $t3, 4 # initialize end count

li $t4, 0 # initialize count variable

j loop

loop:

lw $t0, 0($a0) # retrieve element from vector a

sw $t0, 0($a2) # store element into vector d

addi $a2, $a2, 4 # increment to next element in vector a

lw $t0, 0($a1) # retrieve element from vector b

sw $t0, 0($a2) # store element into vector d

addi $a2, $a2, 4 # increment to next element in vector b

addi $a0,$a0, 4 # iterate to next element in vector a

addi $a1,$a1, 4 # iterate to next element in vector b

addi $t4, $t4, 1 # increment count variable by one

bne $t4, $t4, loop

mul $t3, $t3, 4

mul $t3, $t3, 2

sub $a2, $a2, $t3

lw $t0, 0($a2)

lw $t1, 4($a2)

sll $t0, $t0, 8

add $t0, $t0, $t1

sll $t0, $t0, 8

sll $t0, $t0, 8

lw $t2, 8($a2)

lw $t3, 12($a2)

sll $t2, $t2, 8

add $t2, $t2, $t3

add $t0, $t0, $t2 # first half of vector

lw $t2, 16($a2)

lw $t3, 20($a2)

sll $t2, $t2, 8

add $t2, $t2, $t3

sll $t2, $t2, 8

lw $t3, 24($a2)

add $t2, $t2, $t3

sll $t2, $t2, 8

lw $t3, 28($a2)

add $t2, $t2, $t3

exit:

ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data # [0] [1] [2] [3] [4] [5] [6] [7]

vec\_a: .word 0x5A, 0xF0, 0xA5, 0x01, 0xAB, 0x01, 0x55, 0xC3

vec\_b: .word 0XA5, 0X0F, 0X5A, 0X23, 0XCD, 0X23, 0XAA, 0X3C

vec\_d: .space 8

**9. Vector Pack**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: 9\_vector\_pack.asm

# Verson: 1.0

# Date: November 22, 2018

# Programmer: Brian Nguyen

# Description: Each high element of the result vector d is the

# truncation of the corresponding wider

# element of vector a. Each low element

# of the result is the truncation

# of the corresponding wider element of vector b

#

# vec\_pack d, a, b

#

#

#

#

#

#

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text # main (must be global)

.globl main

main:

la $a0, vec\_a # load vector a

la $a1, vec\_b # load vector b

la $a2, vec\_d # load vector d (empty vector)

li $s0, 0x10 # load the divisor

lw $t3, size # load the size of the vector

li $t4, 0 # initialize counter variable

j high\_loop

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# loop used for storing high elements into vector

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

high\_loop:

lw $t0, 0($a0) # load element from vector a

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

div $t0, $s0 # Pseudo Instruction for

mfhi $t0 # modulus: $d = $s % $t

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

sw $t0, 0($a2) # store the high result into vector d

addi $a0,$a0, 4 # iterate to next element in vector a

addi $a2,$a2, 4 # iterate to next element in vector d

addi $t4, $t4, 1 # increment loop counter

bne $t3, $t4, high\_loop # once the loop counter reaches its limit

# then exit the loop

addi $t4, $zero, 0 # reset loop counter

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# loop used for storing low elements into vector

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

low\_loop:

lw $t0, 0($a1) # load element from vector b

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

div $t0, $s0 # Pseudo Instruction for

mfhi $t0 # modulus: $d = $s % $t

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

sw $t0, 0($a2) # store the result low result into

addi $a1,$a1, 4 # iterate to next element in vector a

addi $a2,$a2, 4 # iterate to next element in vector d

addi $t4, $t4, 1 # increment loop counter

bne $t3, $t4, low\_loop # once the loop counter reaches its limit

# then exit the loop

mul $t3, $t3, 4

mul $t3, $t3, 2

sub $a2, $a2, $t3

lw $t0, 0($a2)

lw $t1, 4($a2)

sll $t0, $t0, 4

add $t0, $t0, $t1

sll $t0, $t0, 4

lw $t1, 8($a2)

add $t0,$t0, $t1

sll $t0, $t0, 4

lw $t1, 12($a2)

add $t0, $t0, $t1

sll $t0, $t0, 4

lw $t1, 16($a2)

add $t0, $t0, $t1

sll $t0, $t0, 4

lw $t1, 20($a2)

add $t0, $t0, $t1

sll $t0, $t0, 4

lw $t1, 24($a2)

add $t0, $t0, $t1

sll $t0, $t0, 4

lw $t1, 28($a2)

add $t0, $t0, $t1

lw $t2, 32($a2)

lw $t3, 36($a2)

sll $t2, $t2, 4

add $t2, $t2, $t3

sll $t2, $t2, 4

lw $t3, 40($a2)

add $t2, $t2, $t3

sll $t2, $t2, 4

lw $t3, 44($a2)

add $t2, $t2, $t3

sll $t2, $t2, 4

lw $t3, 48($a2)

add $t2,$t2, $t3

sll $t2, $t2, 4

lw $t3, 52($a2)

add $t2, $t2, $t3

sll $t2, $t2, 4

lw $t3, 56($a2)

add $t2,$t2,$t3

sll $t2, $t2, 4

lw $t3, 60($a2)

add $t2, $t2, $t3

exit:

ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data # [0] [1] [2] [3] [4] [5] [6] [7]

vec\_a: .word 0x5A, 0xFB, 0x6C, 0x1D, 0xAE, 0x5F, 0xC0, 0x41

vec\_b:.word 0X52, 0XF3, 0XA4, 0X15, 0XA6, 0X57, 0XC8, 0X49

vec\_d:.space 16

size: .word 8

**10. Vector Permute**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: 10\_vector\_permute.asm

# Verson: 1.0

# Date: November 23, 2018

# Programmer: Brian Nguyen

# Description: The "permute" instruction fills the result vector d

# with elements from either vector a or vector b,

# depending upon the

# element specifier in vector c. The vector elements

# can be specified in any order

#

# vec\_perm d, a, b, c

#

#

#

#

#

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text # main (must be global)

.globl main

main:

la $a0, vec\_a # load vector a

la $a1, vec\_b # load vector b

la $a2, vec\_c # load vector c

la $a3, vec\_d # load vector d

li $s1, 1

li $s2, 0x10 # load overflow limit

li $s4, 4

lw $t3, size # initialize the size of the vectors

li $t4, 0 # initialize counter variable

j loop

vec\_a\_permute:

div $t0, $s2

mfhi $t8 # element of vector c % 10

# (to get the last digit)

mul $t0, $t8, 4 # now multiply the mod by 4

# to get element position

add $a0, $a0, $t0

lw $t1, ($a0) # load element from vector a

sw $t1, ($a3) # store vector a element into vector

sub $a0, $a0, $t0 # go back to starting point

addi $a2, $a2, 4 # go to next position of vector c

addi $a3, $a3, 4 # go to next position of vector d

b loop # branch to loop

vec\_b\_permute:

div $t0, $s2

mfhi $t8 # element of vector c % 10

# (to get the last digit)

mul $t0, $t8, 4 # now multiply the mod by

# 4 to get element position

add $a1, $a1, $t0

lw $t1, ($a1) # load element from vector b

sw $t1, ($a3) # store vector b element into vector d

sub $a1, $a1, $t0 # go back to starting point

addi $a2, $a2, 4 # go to next position of vector c

addi $a3, $a3, 4 # go to next position of vector d

b loop # branch to loop

loop:

lw $t0, 0($a2) # load element from vector c

# (the element specifier)

div $t1, $t0, $s2 # dividing by 10

addi $t4, $t4, 1

beq $t4, $t3, combine

beq $t1, $zero, vec\_a\_permute # if the result is 0

# go to branch vec\_a\_permute

beq $t1, $s1, vec\_b\_permute # if the result is 1

# go to branch vec\_b\_permute

combine:

subi $a3, $a3, 40

lw $t0, 8($a3)

lw $t1, 12($a3)

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t1, 16($a3)

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t1, 20($a3)

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t1, 24($a3)

lw $t2, 28($a3)

sll $t1, $t1, 8

add $t1, $t1, $t2

lw $t2, 32($a3)

sll $t1, $t1, 8

add $t1, $t1, $t2

lw $t2, 36($a3)

sll $t1, $t1, 8

add $t1, $t1, $t2

exit:

ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data # [0] [1] [2] [3] [4] [5] [6] [7]

vec\_c: .word 0x04, 0x17, 0x10, 0x02, 0x13, 0x05, 0x01, 0x05

vec\_a: .word 0xA5, 0x67, 0x01, 0x3D, 0xAB, 0x45, 0x39, 0x3C

vec\_b: .word 0XEF, 0XC5, 0X4D, 0X23, 0X12, 0X77, 0XAA, 0XCD

vec\_d: .space 8

size: .word 9

**11. Vector Compare Equal-To**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: 11\_compare\_equal\_to.asm

# Verson: 1.0

# Date: November 18, 2018

# Programmer: Brian Nguyen

# Description: Each element of the result vector d is TRUE (all bits

# = 1) if the corresponding element of

# vector a is equal to the corresponding element of

# vector b. Otherwise the element of result is FALSE

# (all bits = 0)

#

#

# vec\_cmpeq d, a, b

#

#

#

#

#

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text # main (must be global)

.globl main

main:

la $a0, vec\_a # initialize vector a

la $a1, vec\_b # initialize vector b

la $a2, vec\_d # initialize vector d

li $t7, 0xFF # load 0xFF

lw $t3, size # initialize the size of the vectors

li $t4, 0 # initialize counter variable

j loop

equal:

sw $t7, 0($a2) # store 0xFF into vector d

addi $a0,$a0, 4 # iterate to next element in vector a

addi $a1,$a1, 4 # iterate to next element in vector b

addi $a2, $a2, 4 # iterate to the next element in vector d

b loop

loop:

lw $t0, 0($a0) # retrieve element from vector a

lw $t1, 0($a1) # retrieve element from vector b

beq $t1, $t0, equal # if two element are equal, then go to equal

sw $t5, ($a2)

addi $a0,$a0, 4 # iterate to next element in vector a

addi $a1,$a1, 4 # iterate to next element in vector b

addi $a2, $a2, 4 # iterate to the next element in vector d

addi $t4, $t4, 1 # increment program counter

bne $t3, $t4, loop # once program counter reaches limit, then exit

combine:

subi $a2, $a2, 40

lw $t0, 0($a2)

lw $t1, 4($a2)

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t1, 8($a2)

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t1, 12($a2)

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t1, 16($a2)

lw $t1, 20($a2)

lw $t2, 24($a2)

sll $t1, $t1, 8

add $t1, $t1, $t2

lw $t2, 28($a2)

sll $t1, $t1, 8

add $t1, $t1, $t2

lw $t2, 32($a2)

exit: ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data # [0] [1] [2] [3] [4] [5] [6] [7]

vec\_a: .word 0x5A, 0xFB, 0x6C, 0x1D, 0xA6, 0x5F, 0xC0, 0x40

vec\_b: .word 0X52, 0XFB, 0XA4, 0X15, 0XAE, 0X5F, 0XC8, 0X41

vec\_d: .space 8

size: .word 8

**12. Vector Compare Less Than**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: 12\_compare\_less\_than.asm

# Verson: 1.0

# Date: November 18, 2018

# Programmer: Brian Nguyen

# Description: Each element of the result vector d is TRUE (all bits

# = 1) if the corresponding element of

# vector a is less than to the corresponding element of

# vector b. Otherwise the element of result is FALSE

# (all bits = 0)

#

#

# vec\_cmpltu d, a, b

#

#

#

#

#

#

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text # main (must be global)

.globl main

main:

la $a0, vec\_a # initialize vector a

la $a1, vec\_b # initialize vector b

la $a2, vec\_d # initialize vector d

li $t7, 0xFF # load 0xFF

lw $t3, size # initialize the size of the vectors

li $t4, 0 # initialize counter variable

j loop

store:

sw $t7, 0($a2)

addi $a0,$a0, 4 # iterate to next element in vector a

addi $a1,$a1, 4 # iterate to next element in vector b

addi $a2, $a2, 4 # iterate to the next element in vector d

b loop

loop:

lw $t0, 0($a0) # retrieve element from vector a

lw $t1, 0($a1) # retrieve element from vector b

slt $t5, $t0, $t1 # if the element in vector a is less than the

# one in vector b, then the

# value of the current position is 0xFF

bne $t5, $zero, store

addi $a0,$a0, 4 # iterate to next element in vector a

addi $a1,$a1, 4 # iterate to next element in vector b

addi $a2, $a2, 4 # iterate to the next element in vector d

addi $t4, $t4, 1 # increment loop counter

bne $t3, $t4, loop # break when loop counter reaches limit

combine:

subi $a2, $a2, 40

lw $t0, 0($a2)

lw $t1, 4($a2)

sll $t0,$t0, 8

add $t0,$t0, $t1

lw $t1, 8($a2)

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t1, 12($a2)

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t1, 16($a2)

lw $t2, 20($a2)

sll $t1, $t1, 8

add $t1, $t1, $t2

lw $t2, 24($a2)

sll $t1, $t1, 8

add $t1, $t1, $t2

lw $t2, 28($a2)

sll $t1, $t1, 8

add $t1, $t1, $t2

exit:

ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data # [0] [1] [2] [3] [4] [5] [6] [7]

vec\_a: .word 0x5A, 0xFB, 0x6C, 0x1D, 0xA6, 0x5F, 0xC0, 0x40

vec\_b: .word 0X52, 0XFB, 0XA4, 0X15, 0XAE, 0X5F, 0XC8, 0X41

vec\_d: .space 8

size: .word 6

**Vector Swap**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: vector\_swap.asm

# Verson: 1.0

# Date: November 4, 2018

# Programmer: Brian Nguyen

# Description: All elements in vector a are placed into vector b,

# and all elements in vector b are placed into vector a

#

# vec\_swap d, a, b

#

#

#

#

#

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text # main (must be global)

.globl main

main:

la $a0, vec\_a

la $a1, vec\_b

lw $t3, size # initialize the size of the vectors

li $t4, 0 # initialize counter variable

lw $t0, 0($a0) # retrieve first element from vec\_a

addi $a0, $a0, 4

lw $t1, 0($a0) # retrieve second element from vec\_a

addi $a0, $a0, 4

lw $t2, 0($a1) # retrieve first element from vec\_b

addi $a1, $a1, 4

lw $t3, 0($a1) # retrieve second element from vec\_b

addi $a1, $a1, 4

sub $a0, $a0, 8

sub $a1, $a1, 8

sw $t0, 0($a1) # store first element from vec\_a

addi $a1, $a1, 4

sw $t1, 0($a1) # store second element from vec\_a

addi $a1, $a1, 4

sw $t2, 0($a0) # store first element from vec\_b

addi $a0, $a0, 4

sw $t3, 0($a0) # store second element from vec\_b

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data

vec\_a: .word 0xABCD, 0x1234

vec\_b:.word 0xEF01, 0x5678

vec\_d:.space 4

size: .word 2

**Vector Shift Left**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: vector\_shift\_left.asm

# Verson: 1.0

# Date: November 4, 2018

# Programmer: Brian Nguyen

# Description: The elements in a vector are shifted left once and

# placed into a new vector

#

# vec\_sll d, a

#

#

#

#

#

#

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text # main (must be global)

.globl main

main:

la $a0, vec\_a

la $a1, vec\_d

lw $t0, ($a0)

lw $t3, size

li $t4, 0

li $t5, 0xFFFF

j loop

overflow:

sw $t5, 0($a1)

addi $a0, $a0, 4

addi $a1, $a1, 4

addi $t4, $t4, 1

beq $t3, $t4, combine

loop:

lw $t0, 0($a0)

sll $t0, $t0, 1

slt $t2, $t5, $t0

bne $t2, $zero, overflow

sw $t0, 0($a1)

addi $a0, $a0, 4

addi $a1, $a1, 4

addi $t4, $t4, 1

bne $t3, $t4, loop

combine:

mul $t3, $t3, 4

sub $a1, $a1, $t3

lw $t1, 0($a1)

sll $t1, $t1, 8

sll $t1, $t1, 8

lw $t2, 4($a1)

add $t1, $t1, $t2

lw $t2, 8($a1)

lw $t3, 12($a1)

sll $t2, $t2, 8

sll $t2, $t2, 8

add $t2, $t2, $t3

xit:

ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data vec\_a: .word 0xABCD, 0x1234, 0x5678, 0x9ABC

vec\_d: .space 4

size: .word 4

**Vector Shift Right**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: vector\_shift\_right.asm

# Verson: 1.0

# Date: November 4, 2018

# Programmer: Brian Nguyen

# Description: The elements in a vector are shifted right once and

# placed into a new vector

#

# vec\_sll d, a

#

#

#

#

# Notes:

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text # main (must be global)

.globl main

main:

la $a0, vec\_a

la $a1, vec\_d

lw $t0, ($a0)

lw $t3, size

li $t4, 0

li $t5, 0xFFFF

j loop

overflow:

sw $t5, 0($a1)

addi $a0, $a0, 4

addi $a1, $a1, 4

addi $t4, $t4, 1

beq $t3, $t4, combine

loop:

lw $t0, 0($a0)

srl $t0, $t0, 1

slt $t2, $t5, $t0

bne $t2, $zero, overflow

sw $t0, 0($a1)

addi $a0, $a0, 4

addi $a1, $a1, 4

addi $t4, $t4, 1

bne $t3, $t4, loop

combine:

mul $t3, $t3, 4

sub $a1, $a1, $t3

lw $t1, 0($a1)

sll $t1, $t1, 8

sll $t1, $t1, 8

lw $t2, 4($a1)

add $t1, $t1, $t2

lw $t2, 8($a1)

lw $t3, 12($a1)

sll $t2, $t2, 8

sll $t2, $t2, 8

add $t2, $t2, $t3

exit:

ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data vec\_a: .word 0xABCD, 0x1234, 0x5678, 0x9ABC

vec\_d: .space 4

size: .word 4

**Vector Copy**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: vector\_copy.asm

# Verson: 1.0

# Date: November 4, 2018

# Programmer: Brian Nguyen

# Description: The elements in a vector are copied into another

# empty vector

#

# vec\_cpy d, a

#

#

#

#

#

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text # main (must be global)

.globl main

main:

la $a0, vec\_a

la $a1, vec\_d

lw $t3, size

li $t4, 0

j loop

loop:

lw $t0, 0($a0)

sw $t0, 0($a1)

addi $a0, $a0, 4

addi $a1, $a1, 4

addi $t4, $t4, 1

bne $t3, $t4, loop

combine:

lw $t0, 0($a0)

lw $t1, 4($a0)

sll $t0, $t0, 8

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t1, 8($a0)

lw $t2, 12($a0)

sll $t1, $t1, 8

sll $t1, $t1, 8

add $t1, $t1, $t2

exit: ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data

vec\_a: .word 0xABCD, 0x1234, 0x5678, 0x9ABC

vec\_d: .space 4

size: .word 4

**Vector Contains**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: vector\_copy.asm

# Verson: 1.0

# Date: November 4, 2018

# Programmer: Brian Nguyen

# Description: If an element in a vector contains a certain value,

# then store 0xFF in a new vector at that same index

#

# vec\_cnt d, a

#

#

#

#

# Notes:

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.text # main (must be global)

.globl main

main:

la $a0, vec\_a

la $a1, vec\_d

lw $t3, size

li $t1, 0x01

li $t2, 0xFF

li $t4, 0

j loop

store:

sw $t2, 0($a1)

addi $a0, $a0, 4

addi $a1, $a1, 4

addi $t4, $t4, 1

beq $t3, $t4, exit

b loop

loop:

lw $t0, 0($a0)

beq $t0, $t1, store

addi $a0, $a0, 4

addi $a1, $a1, 4

addi $t4, $t4, 1

bne $t3, $t4, loop

exit:

ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data

vec\_a: .word 0xAB, 0xCD, 0xEF, 0x01, 0x01, 0xF3, 0x61, 0x01

vec\_d: .space 8

size: .word 8

**Vector AND**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: vector\_and.asm

# Verson: 1.2

# Date: November 30, 2018

# Programmer: Brian Nguyen

# Description: Each element of vector a is compared with the

# corresponding element of vector b using the AND

# operator

#

#

# vec\_and d, a, b

#

#

#

#

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

main:

la $a0, a # initialize vector a

la $a1, b # initialize vector b

la $a2, d # initialize vector d

lw $t3, size # initialize the loop breaker

li $t4, 0 # initialize count variable

j loop

loop:

lw $t0, 0($a0) # load element from vector a

lw $t1, 0($a1) # load element from vector b

and $t1, $t1, $t0

sw $t1, 0($a2) # store sum in vector d

addi $a0,$a0, 4 # iterate to next element in vector a

addi $a1,$a1, 4 # iterate to next element in vector b

addi $a2, $a2, 4 # iterate to the next element in vector d

addi $t4, $t4, 1 # increment the count variable

bne $t4, $t3, loop # once the count variable reaches the

# end of the sizes

# of the vectors (8), the loop will break

combine:

mul $t3, $t3, 4

sub $a2, $a2, $t3

lw $t0, 0($a2)

lw $t1, 4($a2)

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t1, 8($a2)

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t1, 12($a2)

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t1, 16($a2)

lw $t2, 20($a2)

sll $t1, $t1, 8

add $t1, $t1, $t2

lw $t2, 24($a2)

sll $t1, $t1, 8

add $t1, $t1, $t2

lw $t2, 28($a2)

sll $t1, $t1, 8

add $t1, $t1, $t2

exit: ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

proc1: j proc1 # placeholder stub

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data # [0] [1] [2] [3] [4] [5] [6] [7]

a: .word 0x23, 0x3C, 0x47, 0x5D, 0x08, 0x7F, 0x19, 0x6F

b: .word 0X98, 0X19, 0X63, 0XC5, 0X5E, 0X80, 0XB3, 0X6E

d: .space 8

size: .word 8

**Vector OR**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: vector\_and.asm

# Verson: 1.1

# Date: November 30, 2018

# Programmer: Brian Nguyen

# Description: Each element of vector a is compared with the

# corresponding element of vector b using the OR

# operator

#

# vec\_or d, a, b

#

#

#

#

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

main:

la $a0, a # initialize vector a

la $a1, b # initialize vector b

la $a2, d # initialize vector d

lw $t3, size # initialize the loop breaker

li $t4, 0 # initialize count variable

j loop

loop:

lw $t0, 0($a0) # load element from vector a

lw $t1, 0($a1) # load element from vector b

or $t1, $t1, $t0

sw $t1, 0($a2) # store sum in vector d

addi $a0,$a0, 4 # iterate to next element in vector a

addi $a1,$a1, 4 # iterate to next element in vector b

addi $a2, $a2, 4 # iterate to the next element in vector d

addi $t4, $t4, 1 # increment the count variable

bne $t4, $t3, loop # once the count variable reaches the

# end of the sizes

# of the vectors (8), the loop will break

combine:

mul $t3, $t3, 4

sub $a2, $a2, $t3

lw $t0, 0($a2)

lw $t1, 4($a2)

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t1, 8($a2)

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t1, 12($a2)

sll $t0, $t0, 8

add $t0, $t0, $t1

lw $t1, 16($a2)

lw $t2, 20($a2)

sll $t1, $t1, 8

add $t1, $t1, $t2

lw $t2, 24($a2)

sll $t1, $t1, 8

add $t1, $t1, $t2

lw $t2, 28($a2)

sll $t1, $t1, 8

add $t1, $t1, $t2

exit: ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

proc1: j proc1 # placeholder stub

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data # [0] [1] [2] [3] [4] [5] [6] [7]

a: .word 0x23, 0x3C, 0x47, 0x5D, 0x08, 0x7F, 0x19, 0x6F

b: .word 0X98, 0X19, 0X63, 0XC5, 0X5E, 0X80, 0XB3, 0X6E

d: .space 8

size: .word 8

**Vector Subtract Unsigned**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 341 Top Level Module \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# File name: vector\_subtract.asm

# Verson: 1.2

# Date: November 23, 2018

# Programmer: Brian Nguyen

# Description: Each element of vector a is subtracted to the

# corresponding element of vector b.

# The unsigned-integer is placed into

# the corresponding element of vector d

#

# vec\_subsu d, a, b

#

#

#

#

#

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# MAIN CODESEGMENT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

main:

la $a0, a # initialize vector a

la $a1, b # initialize vector b

la $a2, d # initialize vector d

lw $t3, size # initialize the loop breaker

li $t4, 0 # initialize count variable

li $t5, 0x00 # load the limit

j loop

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Branch used to store 0x00 in an index

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

store:

sw $t5, 0($a2) # Stores 0xFF into vector d

addi $a0,$a0, 4 # iterate to next element in vector a

addi $a1,$a1, 4 # iterate to next element in vector b

addi $a2, $a2, 4 # iterate to the next element in vector d

addi $t4, $t4, 1 # increment the count variable

beq $t3, $t4, combine

b loop

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Go through each element in both

# vectors and add them

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

loop:

lw $t0, 0($a0) # load element from vector a

lw $t1, 0($a1) # load element from vector b

subu $t1, $t1, $t0 # sum of elements unsigned

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

slt $t6, $t1, $t5 # if the sum of the registers is greater than

bne $t6, $zero, store # 0xFF, then go to the "store" branch

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

sw $t1, 0($a2) # store sum in vector d

addi $a0,$a0, 4 # iterate to next element in vector a

addi $a1,$a1, 4 # iterate to next element in vector b

addi $a2, $a2, 4 # iterate to the next element in vector d

addi $t4, $t4, 1 # increment the count variable

bne $t4, $t3, loop # once the count variable reaches the end of

# the sizes of the vectors (8), the loop will

# break

combine:

mul $t3, $t3, 4

sub $a2, $a2, $t3

lw $t0, 0($a2)

lw $t1, 4($a2)

sll $t0, $t0, 8

add $t0, $t0, $t1

sll $t0, $t0, 8

addi $a2, $a2, 4

lw $t1, 4($a2)

add $t0, $t0, $t1

sll $t0, $t0, 8

lw $t1, 8($a2)

add $t0, $t0, $t1

lw $t1, 12($a2)

lw $t2, 16($a2)

sll $t1, $t1, 8

add $t1, $t1, $t2

sll $t1, $t1, 8

lw $t2, 20($a2)

add $t1, $t1, $t2

sll $t1, $t1, 8

lw $t2, 24($a2)

add $t1, $t1, $t2

exit: ori $v0, $zero, 10 # $v0 <-- function code for "exit"

syscall # Syscall to exit

proc1: j proc1 # placeholder stub

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# PROJECT RELATED DATA SECTION

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.data # [0] [1] [2] [3] [4] [5] [6] [7]

a: .word 0x23, 0x3C, 0x47, 0x5D, 0x08, 0x7F, 0x19, 0x6F

b: .word 0X98, 0X19, 0X63, 0XC5, 0X5E, 0X80, 0XB3, 0X6E

d: .space 8

size: .word 8

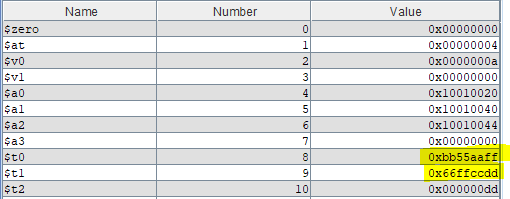
**Vec\_addsu**

Before execution:



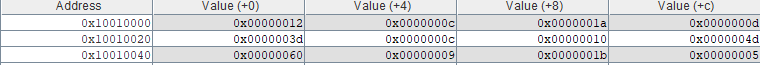


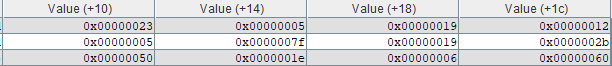
After execution



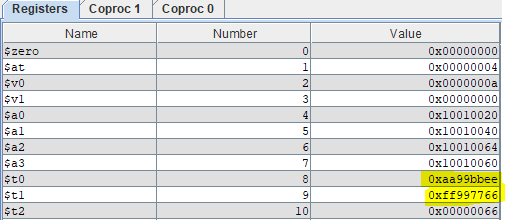
**Vec\_madd**

Before execution:





After execution



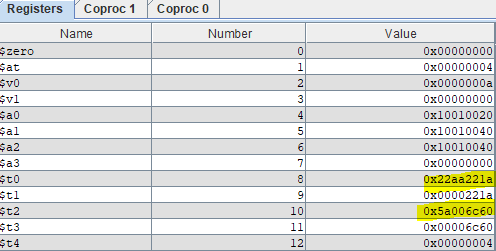
**Vec\_mule**

Before Execution





After execution



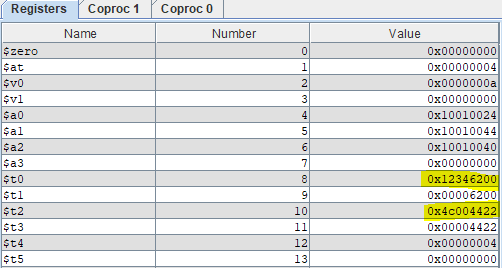
**Vec\_mulo**

Before execution



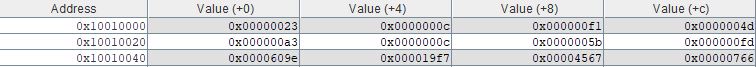


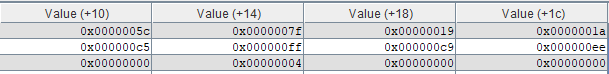
After execution



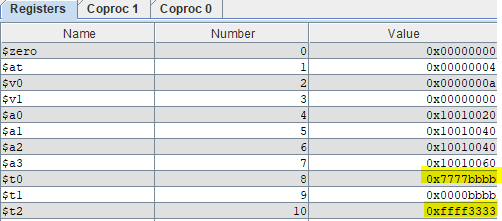
**Vec\_msums**

Before execution





After execution



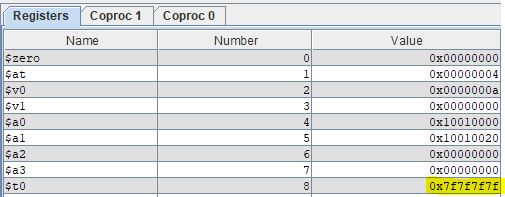
**Vec\_splat**

Before execution





After execution



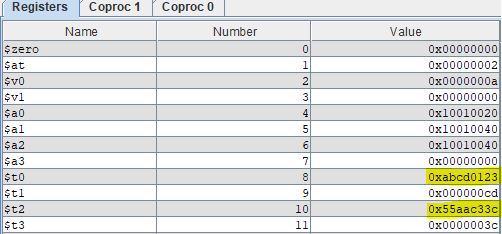
**Vec\_mergel**

Before execution





After execution



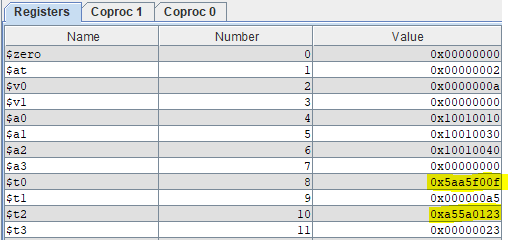
**Vec\_mergh**

Before execution

****

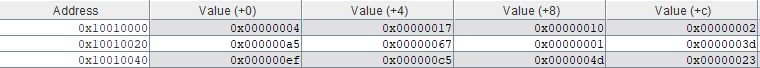
****

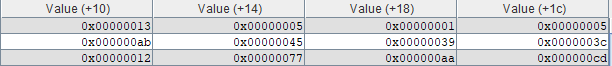
After execution



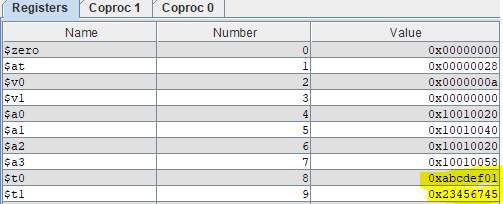
**Vec\_perm**

Before execution





After execution

****

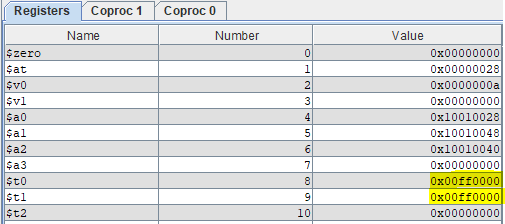
**Vec\_cmpeq**

Before execution





After execution



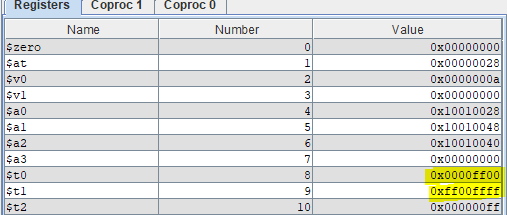
**Vec\_cmpltu**

Before execution





After execution



**Vec\_swap**

Before execution:



After execution

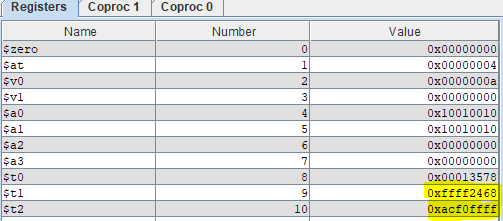


**Vec\_sll**

Before execution



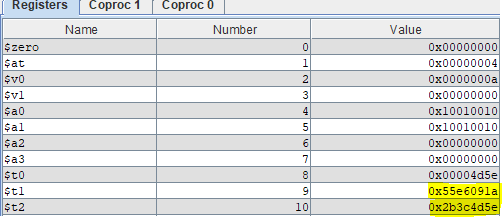
After execution

**Vec\_srl**

Before execution



After execution



**Vec\_cpy**

Before execution



After execution



**Vec\_cnt**

Before execution





After execution





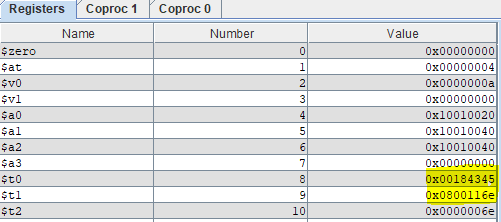
**Vec\_and**

Before execution





After execution



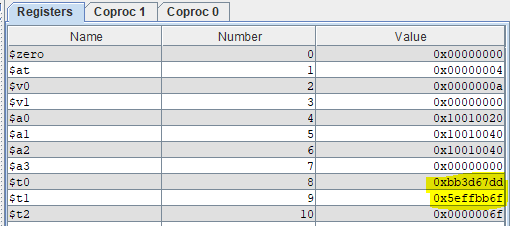
**Vec\_or**

Before execution





After execution



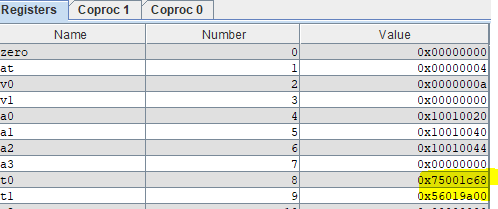
**Vec\_subu**

Before execution

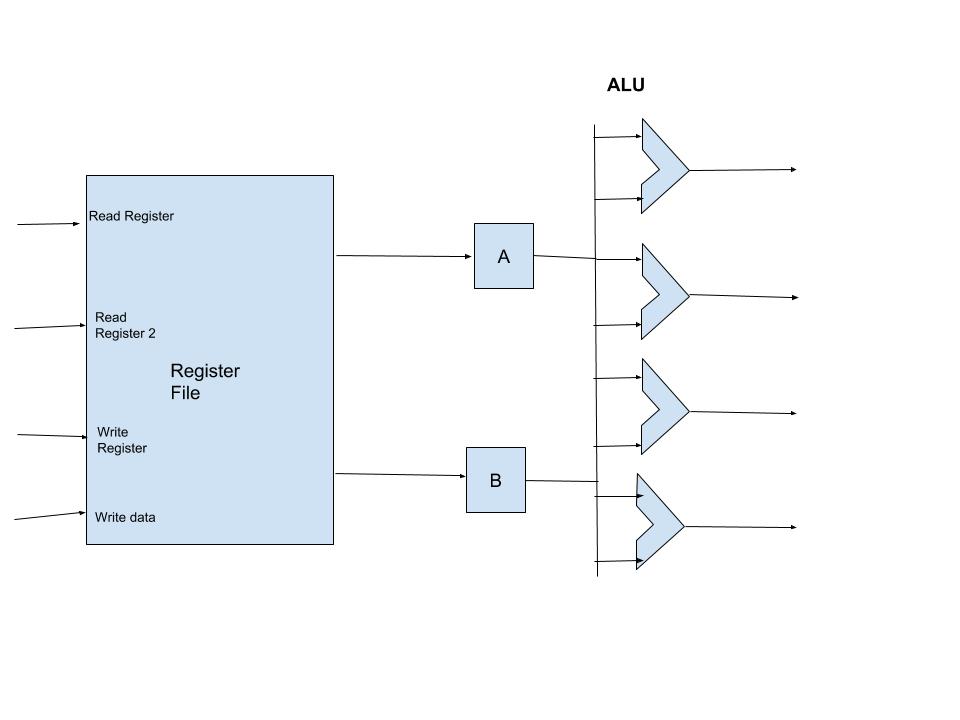
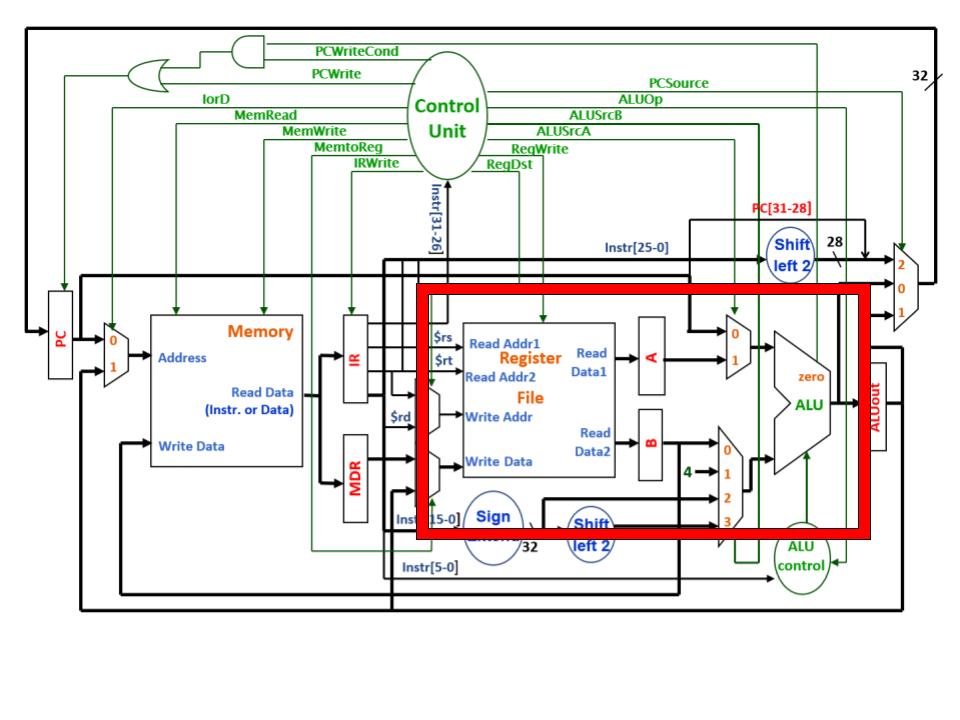




After execution



IV. Datapath Block Diagrams



V. Comments

This project is probably the most challenging one that I have ever done in my life, but also the most satisfying. The coding portion was definitely the most challenging part. I admit that was not confident with assembly programming because I didn’t have enough experience with it; however, this project definitely helped me on that aspect. At times I would get frustrated because my code didn’t work, but my tenacious attitude helped pull me through. In conclusion, I am very proud of my work and I’ve learned a lot about how assembly programming works.

* Brian Nguyen

Project 2 has been the first major term project that I have had to dealt with being a computer science major and it has been a rollercoaster of emotions. When first given this project and seeing that we had to create a one hundred page computer manual it was very intimidating even with the work being split into coding and documentation. Once we actually got into the groove of things the workload became better. Although the documentation was at times tedious working on it a little bit each day was rewarding yet it took time away from other classes. This project was a great review of MIPs but having to start this project in the middle of the semester was difficult to manage if it was given at the beginning of the semester since some parts of the documentation could have been completed in relation to each week. In the end having completed this project has allowed up to have more experience in writing and working as a team helping each other through the ends and outs but this project is very specific unless you are planning to work with processors an employer would be looking for other work a student has done.

- Randy Thiem

Bibliography

1. Robert W. Allison. PowerPoint slides and lecture notes
2. Patterson, David A., and John L. Hennessy. Computer Organization and Design: the Hardware / Software Interface. Morgan Kaufman Publishers, 2009.