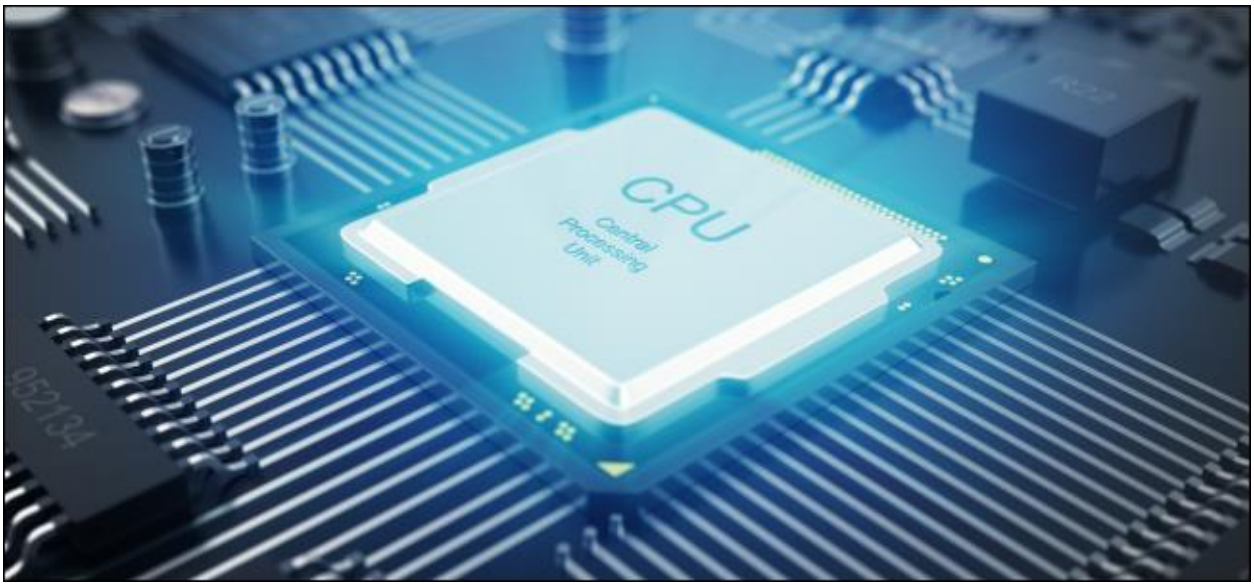
SIMD Enhanced MIPS Instructions



Brian Nguyen - 015188925
Randy Thiem - 015432506
CECS 341
MW 8:00 AM - 10:15 AM
R.W. Allison

Table Of Contents

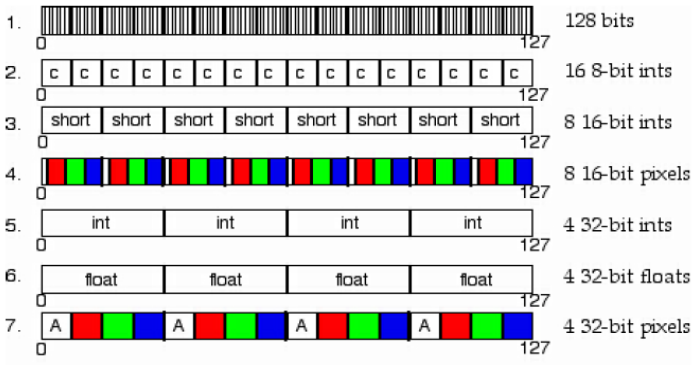
I. Purpose

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

A. Machine Register Set

Name	Number	Purpose
$zero	0	Value 0
$at	1	Assembler Temp
$v0	2	Function Results
$v1	3	Expression Evaluation
$a0	4	Argument
$a1	5	
$a2	6	
$a3	7	
$t0	8	
$t1	9	Temporary
$t2	10	
$t3	11	
$t4	12	
$t5	13	
$t6	14	
$t7	15	
$s0	16	Saved Temp
$s1	17	
$s2	18	
$s3	19	
$s4	20	
$s5	21	
$s6	22	
$s7	23	
$t8	24	Temporary
$t9	25	
$k0	26	
$k1	27	
$gp	28	Global Pointer
$sp	29	Stack Pointer
$fp	30	Frame Pointer
$ra	31	Return Address
Hi		
Lo		

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.

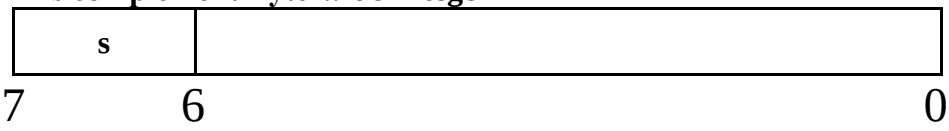
Registers

- $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

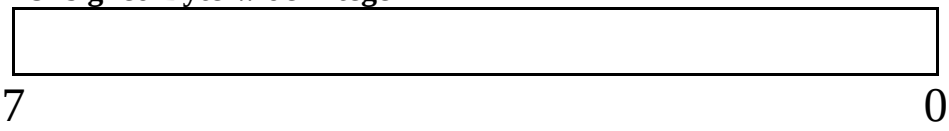
B. Data Types

8-bit integer

2’s complement Byte-wide Integer

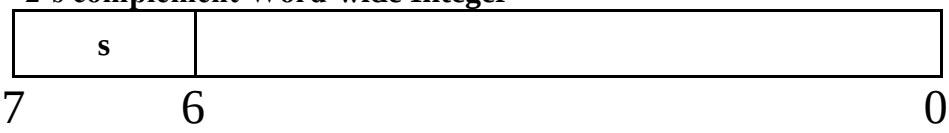


Unsigned Byte-wide Integer

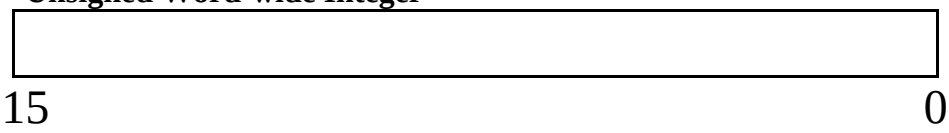


16-bit integer

2’s complement Word-wide Integer

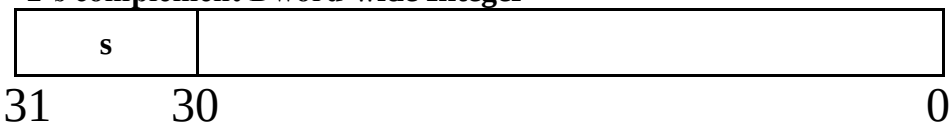


Unsigned Word-wide Integer

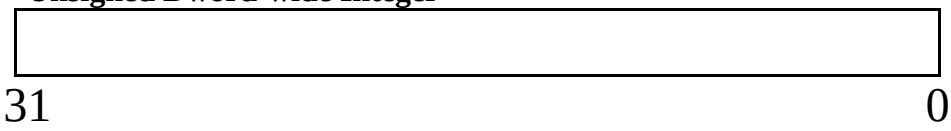


32-bit integer

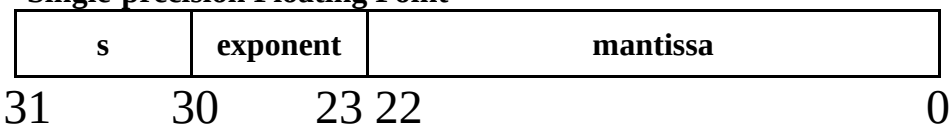
2’s complement Dword-wide Integer



Unsigned Dword-wide Integer

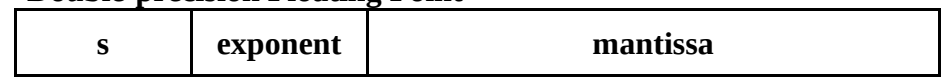


Single-precision Floating Point



64-bit integer

Double-precision Floating Point



C. Addressing Modes

1. Immediate mode

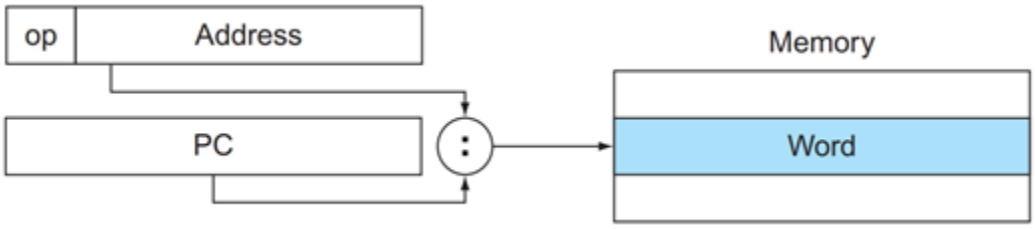
The operand itself is part of the instruction. The program recognizes operand as an immediate mode when it decodes the addressing mode.

2. Register mode

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

3. 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

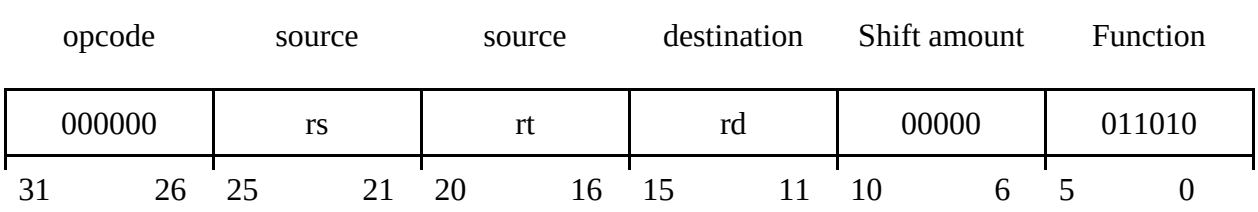
Divide

Description

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

Instruction Format

R - type



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

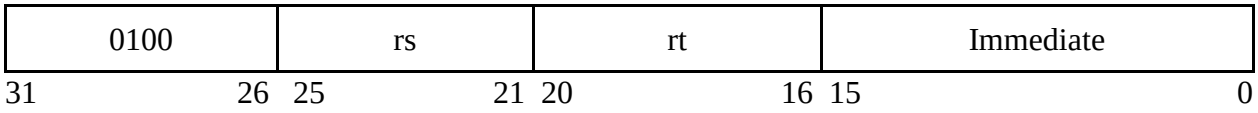
Branch On Equal

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



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.

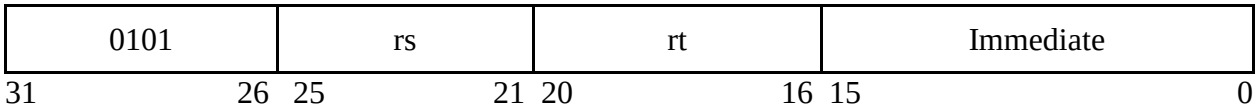
Branch On Not Equal

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



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

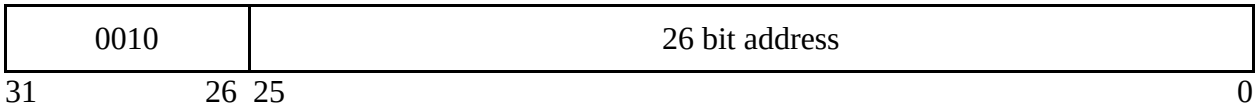
Jump

Description

Jump is used to go to a desired instruction.

Instruction Format

J - type



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

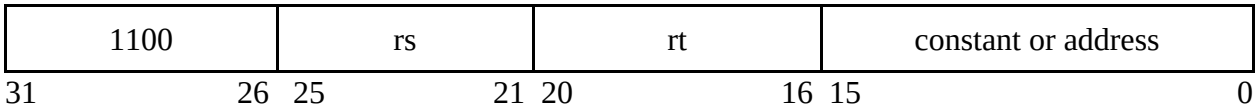
Add Immediate

Description

Add immediate is used to add constants to registers.

Instruction Format

I - type



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

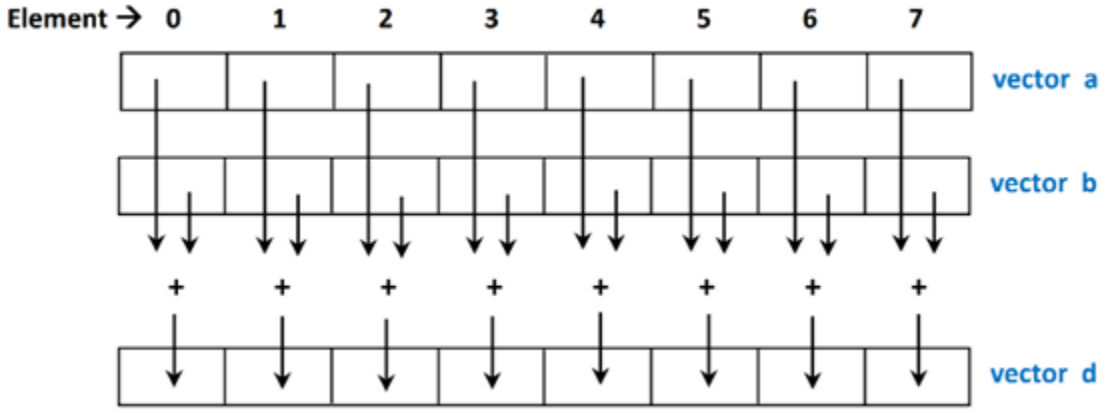
opcode	function	destination	source	source2	source3
0111	0000	vd	vs	vt	00000
31 28	27 24	23 18	17 12	11 6	5 0

MIPS Format

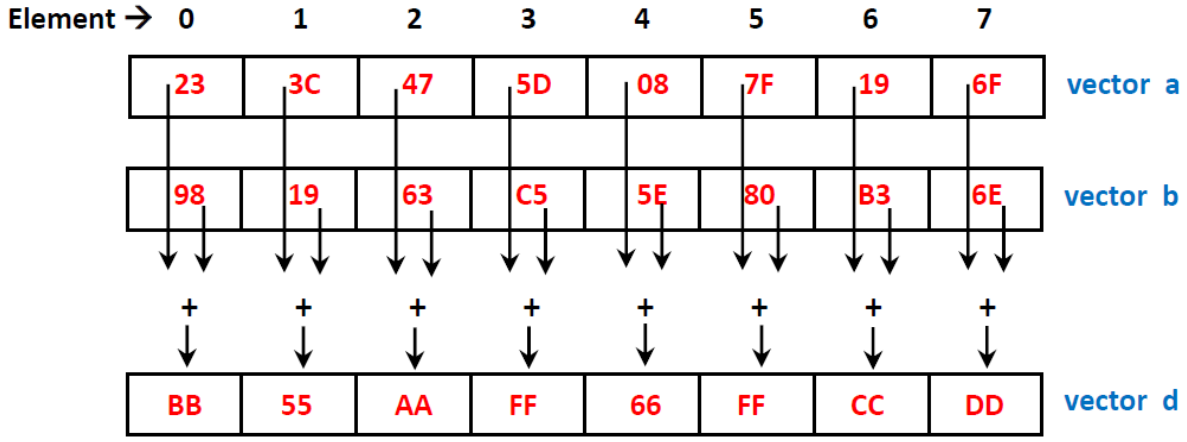
vec_addsu d, a, b

Operation

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



Example


Element → 0 1 2 3 4 5 6 7
23 3C 47 5D 08 7F 19 6F
vector a
98 19 63 C5 5E 80 B3 6E
vector b
+ + + + + + + +
BB 55 AA FF 66 FF CC DD
vector d

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

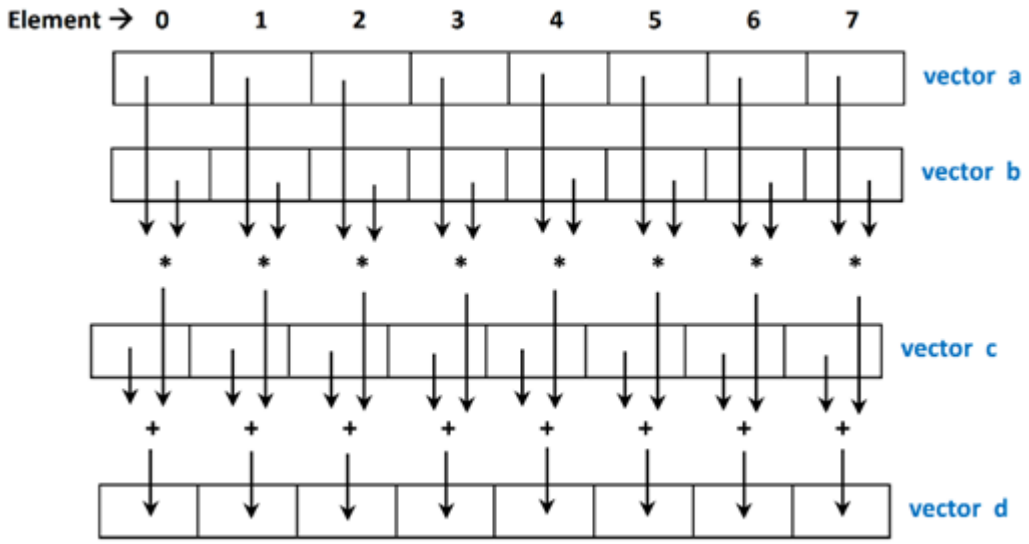
opcode	function	destination	source	source2	source3
0111	0001	vd	vs	vt	vu
31 28	27 24	23 18	17 12	11 6	5 0

MIPS Format

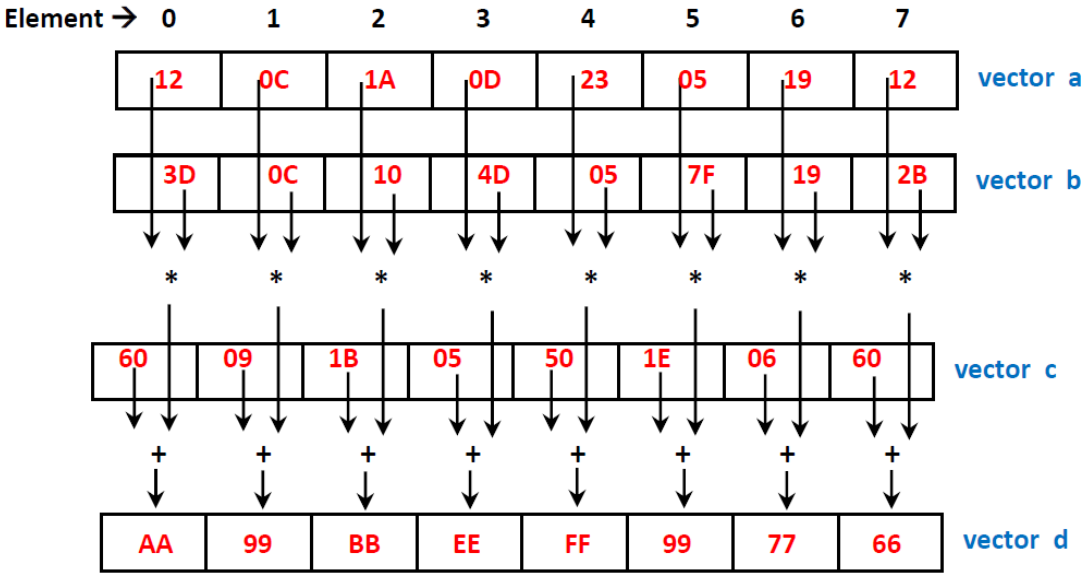
vec_madd d, a, b, c

Operation

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



Example


Element → 0 1 2 3 4 5 6 7
12 0C 1A 0D 23 05 19 12
vector a
3D 0C 10 4D 05 7F 19 2B
vector b
* * * * * * * *
60 09 1B 05 50 1E 06 60
vector c
+ + + + + + + +
AA 99 BB EE FF 99 77 66
vector d

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

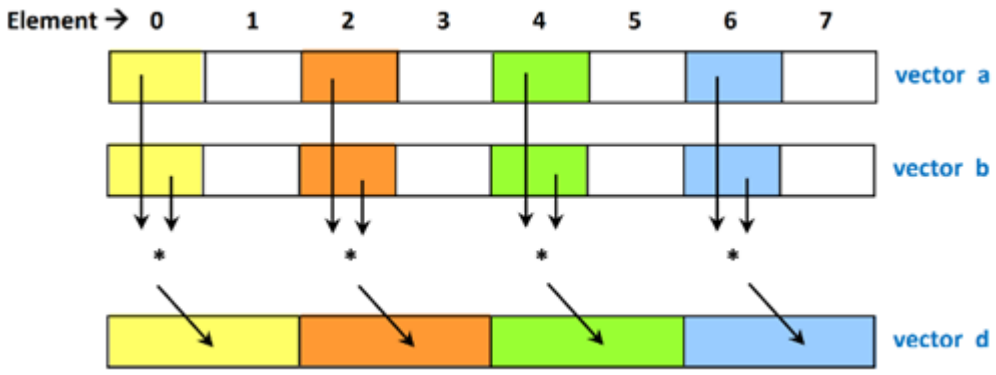
opcode	function	destination	source	source2	source3
0111	0010	vd	vs	vt	00000
31 28	27 24	23 18	17 12	11 6	5 0

MIPS Format

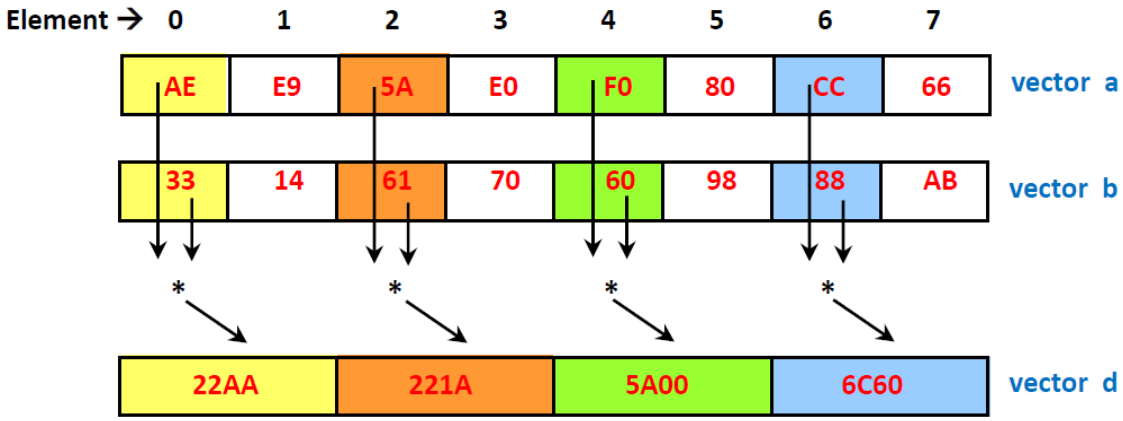
vec_mule d, a, b

Operation

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



Example


Element → 0 1 2 3 4 5 6 7
AE E9 5A E0 F0 80 CC 66
vector a
33 14 61 70 60 98 88 AB
vector b
*
*
*
*
22AA 221A 5A00 6C60
vector d

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

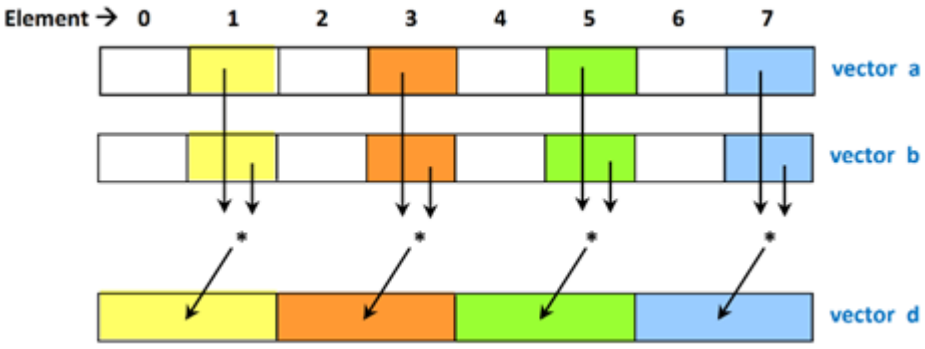
opcode	function	destination	source	source2	source3
0111	0011	vd	vs	vt	00000
31 28	27 24	23 18	17 12	11 6	5 0

MIPS Format

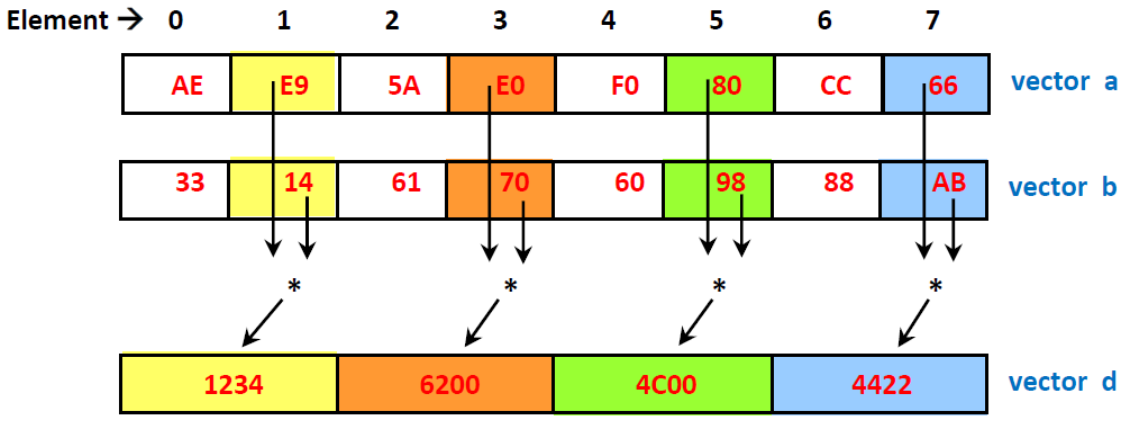
vec_mulo d, a, b

Operation

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



Example


Element → 0 1 2 3 4 5 6 7
AE E9 5A E0 F0 80 CC 66
vector a
33 14 61 70 60 98 88 AB
vector b
*
1234 6200 4C00 4422
vector d

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

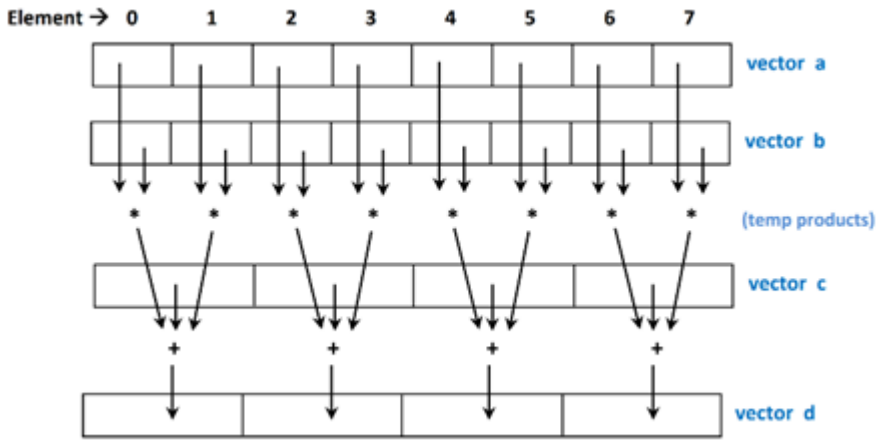
opcode	function	destination	source	source2	source3
0111	0100	vd	vs	vt	vu
31 28	27 24	23 18	17 12	11 6	5 0

MIPS Format

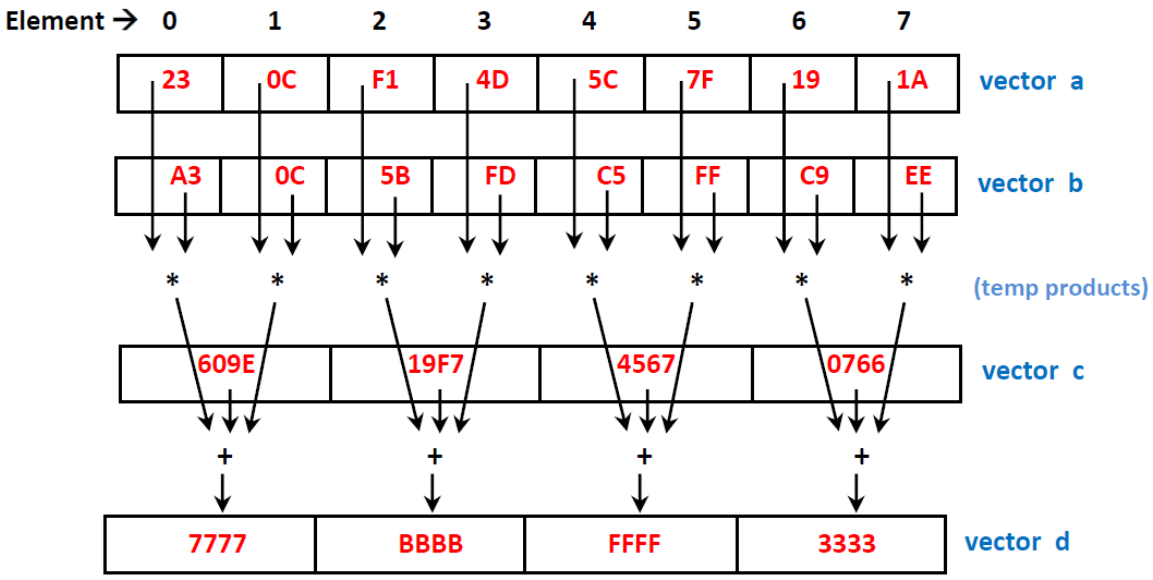
vec_msums d, a, b, c

Operation

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



Example


Element → 0 1 2 3 4 5 6 7
23 0C F1 4D 5C 7F 19 1A
vector a
A3 0C 5B FD C5 FF C9 EE
vector b
* * * * * * * *
(temp products)
609E 19F7 4567 0766
vector c
+ + + +
7777 BBBB FFFF 3333
vector d

Vector Splat

Description

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

Instruction Format

VI - Type

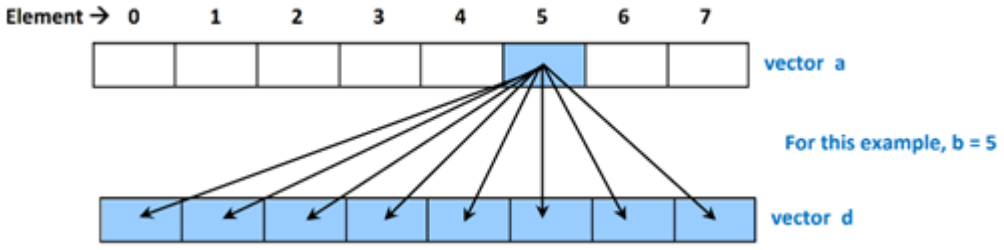
opcode	function	destination	source	11-bit immediate value
0111	10000	vd	vs	immediate
31 28	27 23	22 17	16 11	10 0

MIPS Format

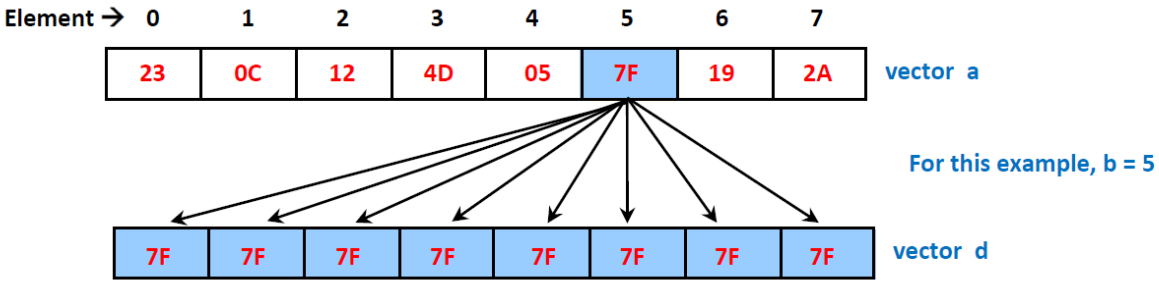
Vec_splat d, a, b

Operation

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



Example


Element → 0 1 2 3 4 5 6 7
23 0C 12 4D 05 7F 19 2A
vector a
For this example, b = 5
7F 7F 7F 7F 7F 7F 7F 7F
vector d

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

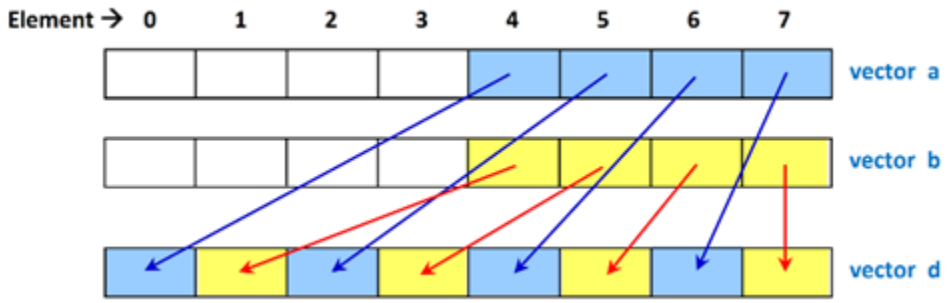
opcode	function	destination	source	source2	source3
0111	0101	vd	vs	vt	000000
31 28	27 24	23 18	17 12	11 6	5 0

MIPS Format

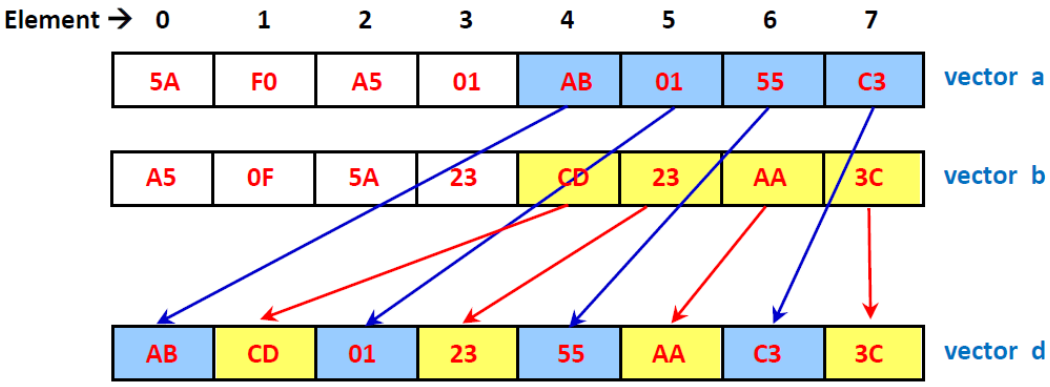
vec_mergel d, a, b

Operation

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



Example


Element → 0 1 2 3 4 5 6 7
5A F0 A5 01 AB 01 55 C3
vector a
A5 0F 5A 23 CD 23 AA 3C
vector b
AB CD 01 23 55 AA C3 3C
vector d

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

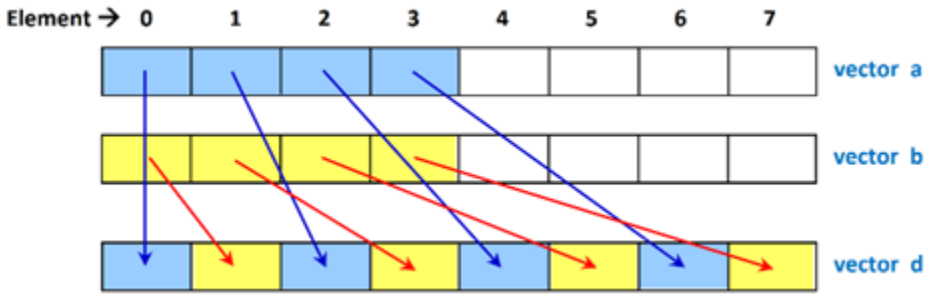
opcode	function	destination	source	source2	source3
0111	0110	vd	vs	vt	000000
31 28	27 24	23 18	17 12	11 6	5 0

MIPS Format

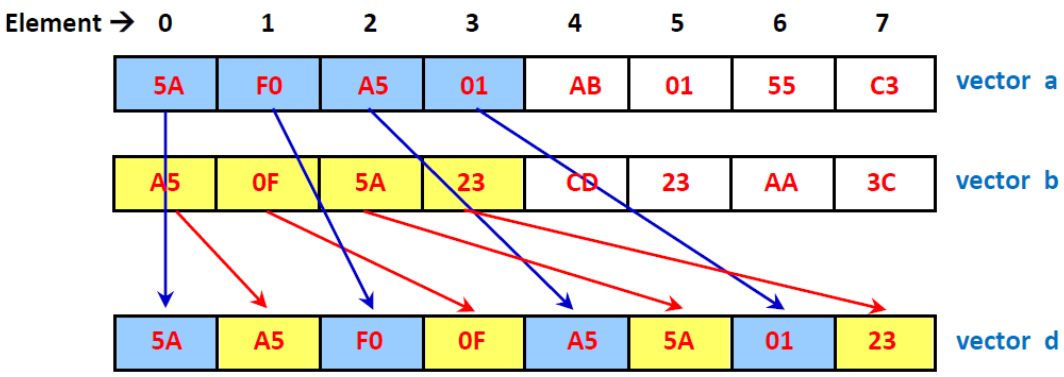
vec_mergeh d, a, b

Operation

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



Example


Element → 0 1 2 3 4 5 6 7
5A F0 A5 01 AB 01 55 C3
vector a
A5 0F 5A 23 CD 23 AA 3C
vector b
5A A5 F0 0F A5 5A 01 23
vector d

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

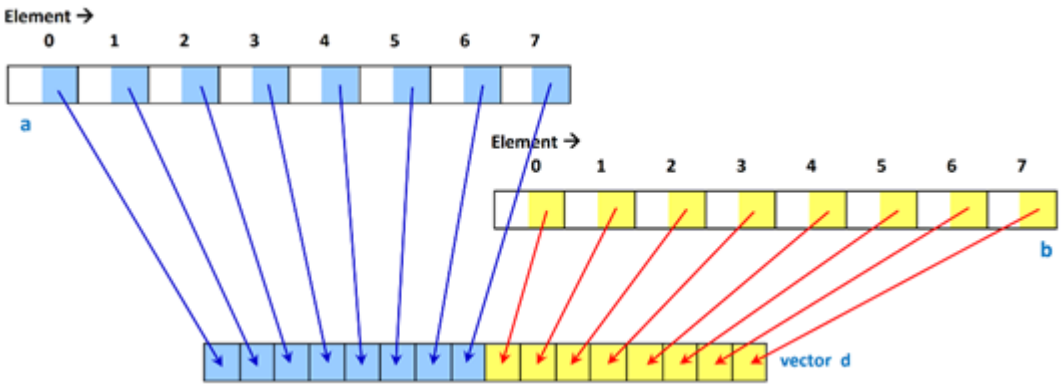
opcode	function	destination	source	source2	source3
0111	0111	vd	vs	vt	000000
31 28	27 24	23 18	17 12	11 6	5 0

MIPS Format

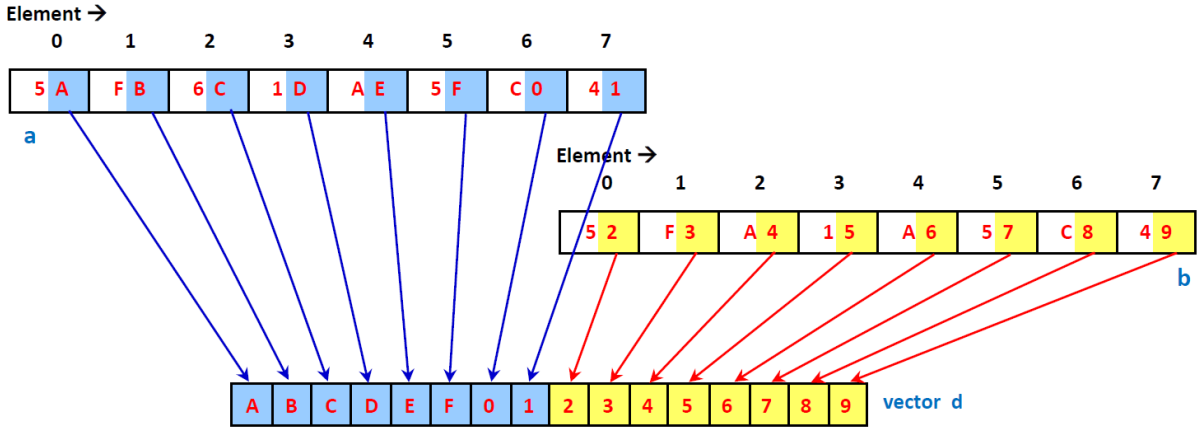
vec_pack d, a, b

Operation

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



Example


Element →
0 1 2 3 4 5 6 7
5 A F B 6 C 1 D A E 5 F C 0 4 1
a
Element →
0 1 2 3 4 5 6 7
5 2 F 3 A 4 1 5 A 6 5 7 C 8 4 9
b
A B C D E F 0 1 2 3 4 5 6 7 8 9
vector d

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

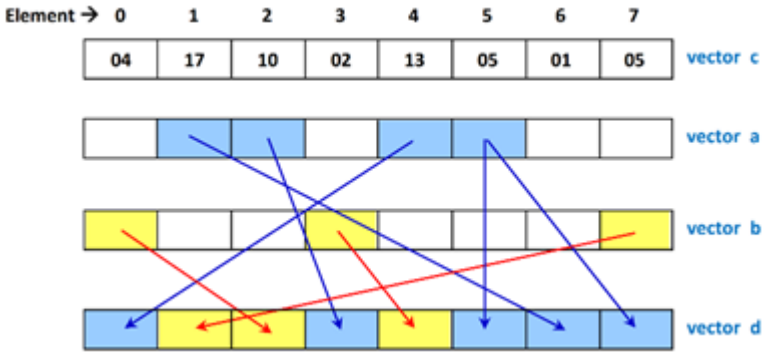
opcode	function	destination	source	source2	source3
0111	1000	vd	vs	vt	vu
31 28	27 24	23 18	17 12	11 6	5 0

MIPS Format

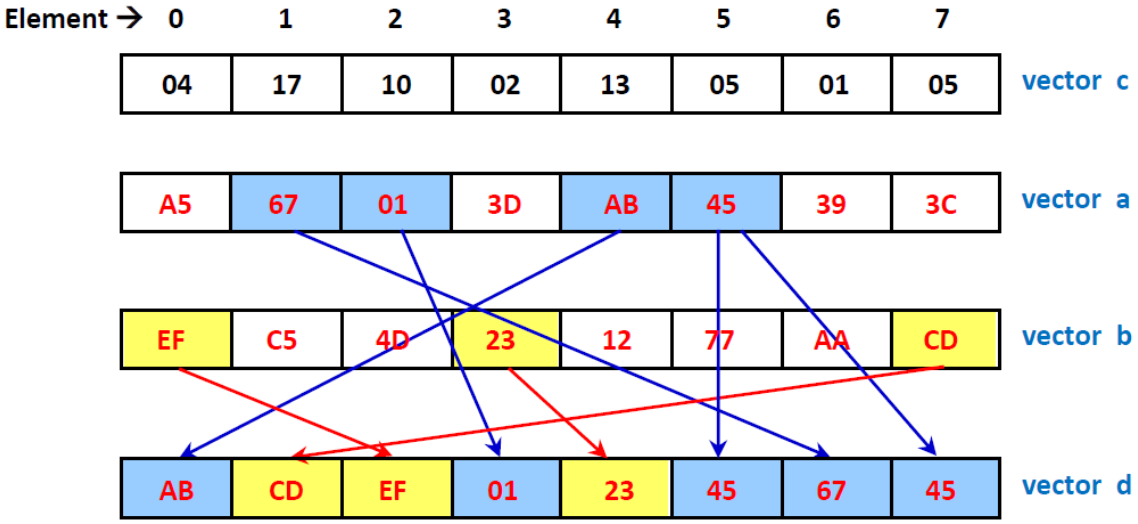
vec_perm d, a, b, c

Operation

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



Example


Element → 0 1 2 3 4 5 6 7
04 17 10 02 13 05 01 05
vector c
A5 67 01 3D AB 45 39 3C
vector a
EF C5 4D 23 12 77 AA CD
vector b
AB CD EF 01 23 45 67 45
vector d

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

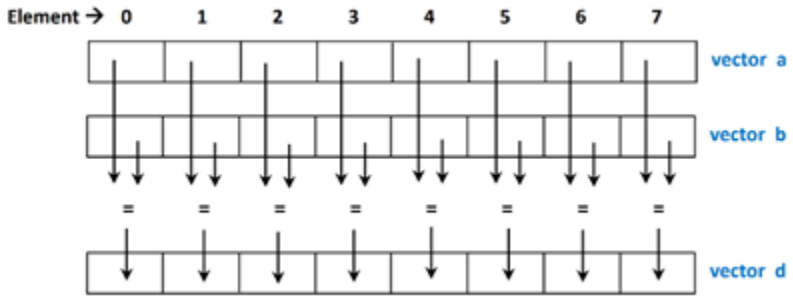
opcode	function	destination	source	source2	source3
0111	1001	vd	vs	vt	000000
31 28	27 24	23 18	17 12	11 6	5 0

MIPS Format

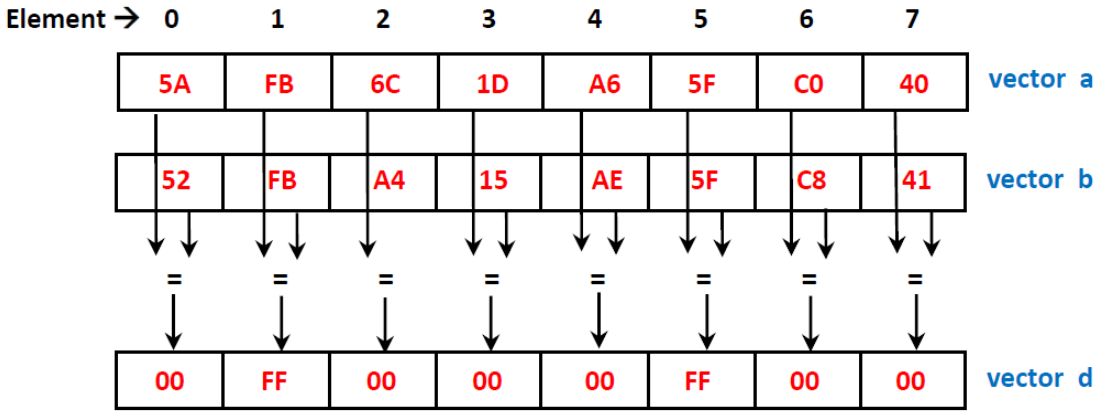
vec_cmpeq d, a, b

Operation

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



Example


Element → 0 1 2 3 4 5 6 7
5A FB 6C 1D A6 5F C0 40
vector a
52 FB A4 15 AE 5F C8 41
vector b
= = = = = = = =
00 FF 00 00 00 FF 00 00
vector d

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

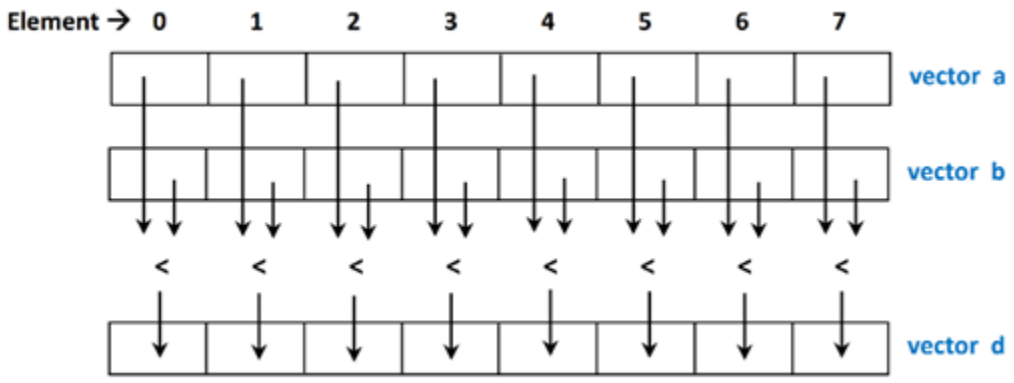
opcode	function	destination	source	source2	source3
0111	1010	vd	vs	vt	000000
31 28	27 24	23 18	17 12	11 6	5 0

MIPS Format

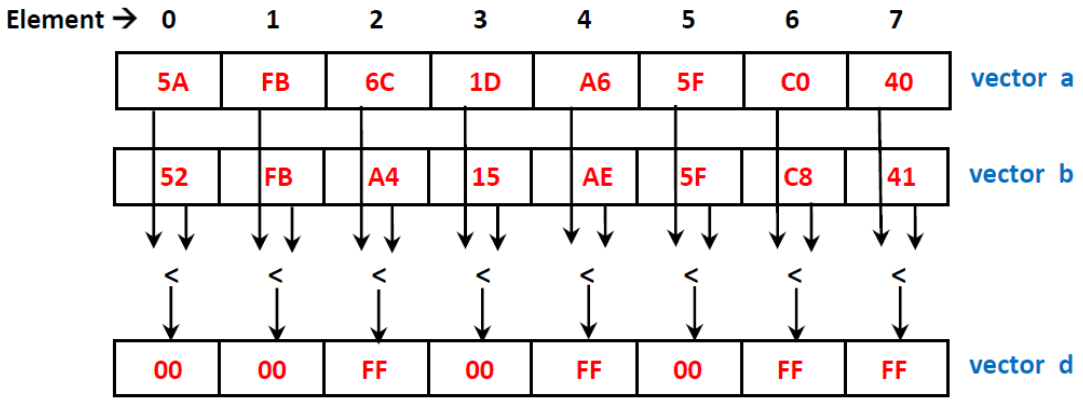
Vec_cmpltu d, a, b

Operation

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



Example


Element → 0 1 2 3 4 5 6 7
5A FB 6C 1D A6 5F C0 40
vector a
52 FB A4 15 AE 5F C8 41
vector b
< < < < < < < <
00 00 FF 00 FF 00 FF FF
vector d

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

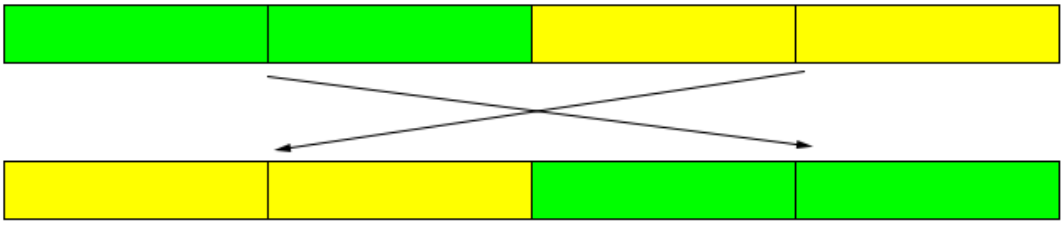
opcode	function	destination	source	11-bit immediate
0111	1010	vd	vs	0000000000
31 28	27 23	22 17	16 11	10 0

MIPS Format

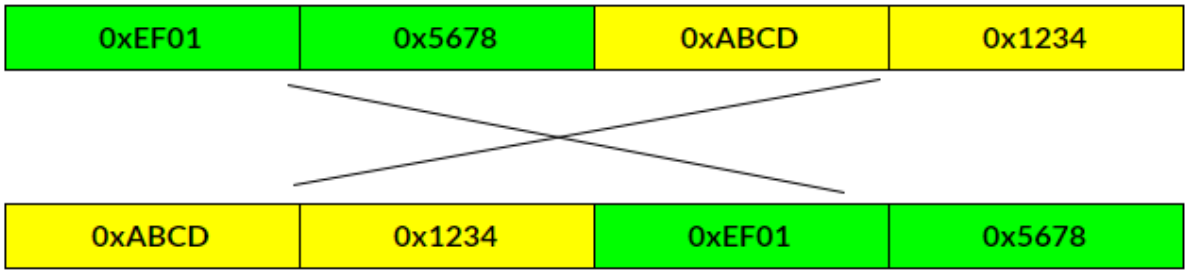
Vec_swap $vd, $vs

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

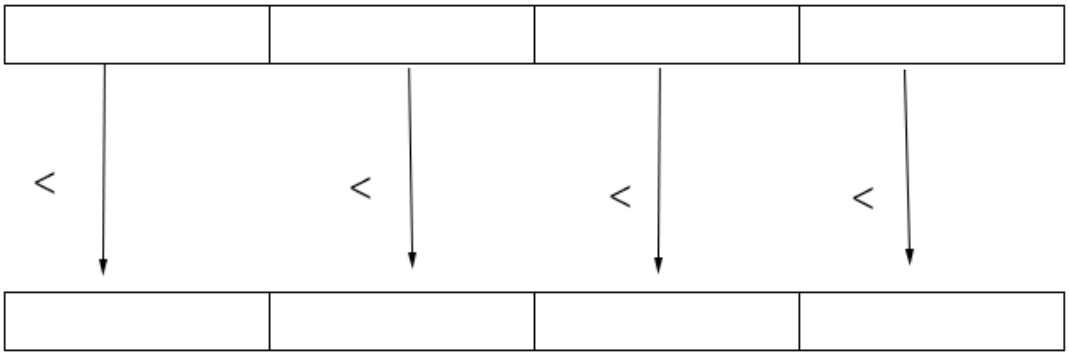
opcode	function	destination	source	11-bit immediate
0111	1010	vd	vs	0000000000
31 28	27 23	22 17	16 11	10 0

MIPS Format

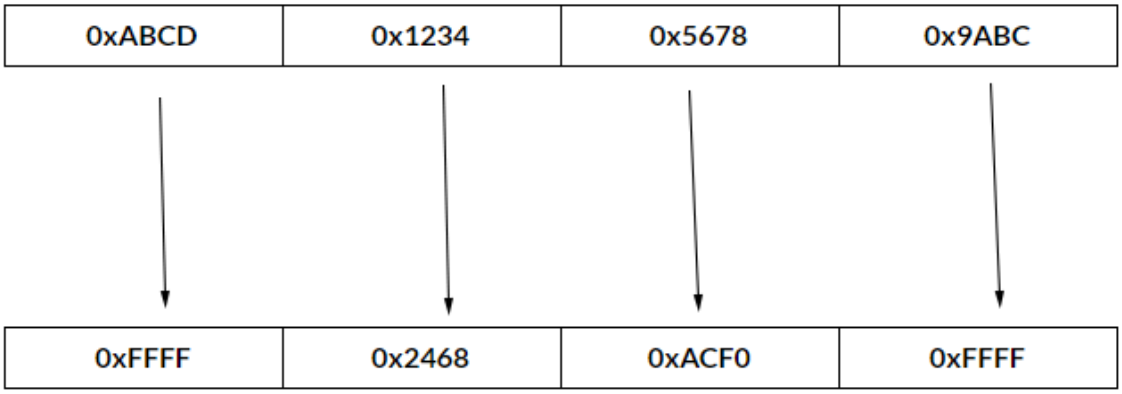
Vec_sll $vd, $vs

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

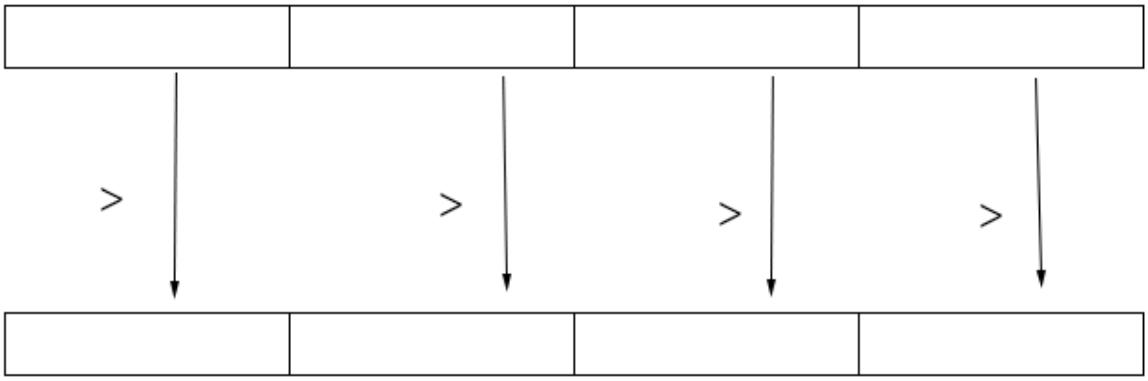
opcode	function	destination	source	11-bit immediate
0111	1010	vd	vs	0000000000
31 28	27 23	22 17	16 11	10 0

MIPS Format

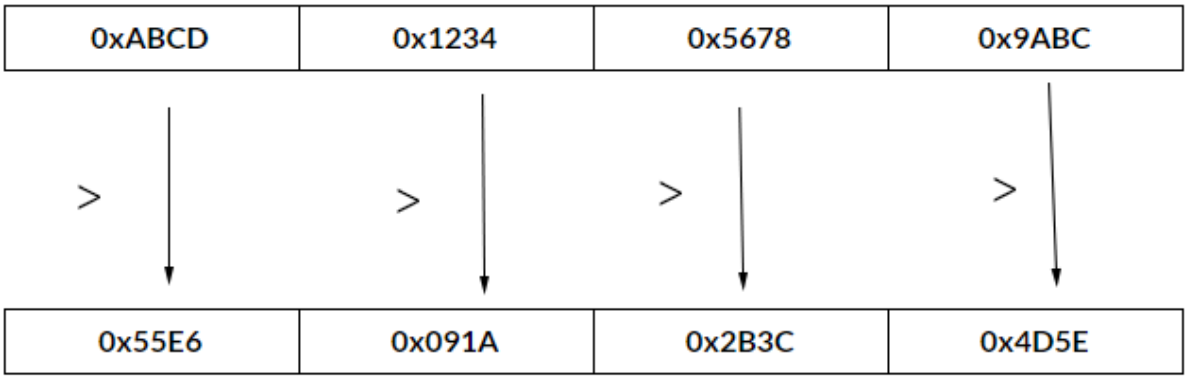
Vec_srl $vd, $vs

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

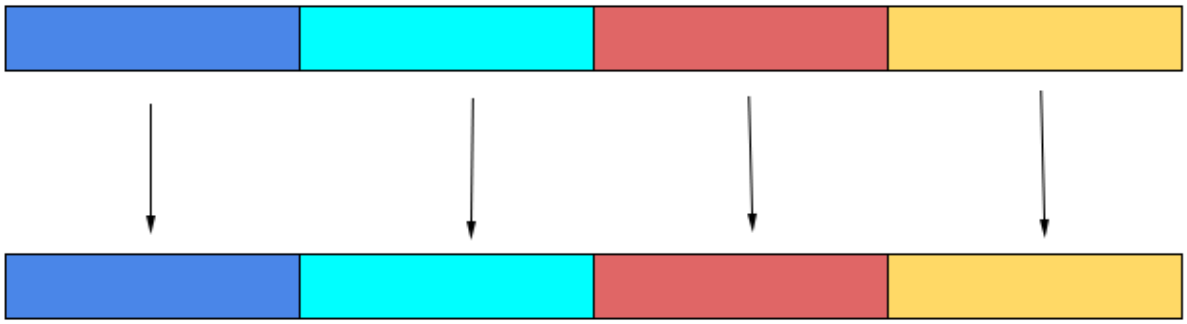
opcode	function	destination	source	11-bit immediate
0111	1010	vd	vs	0000000000
31 28	27 23	22 17	16 11	10 0

MIPS Format

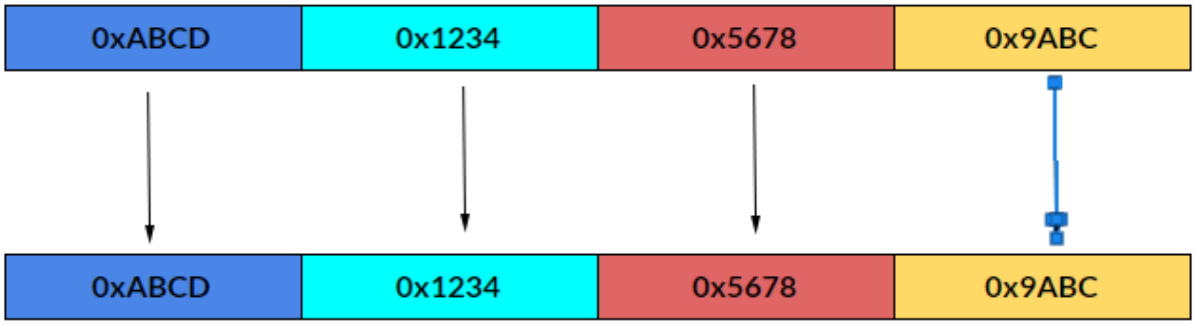
Vec_cpy $vd, $vs

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

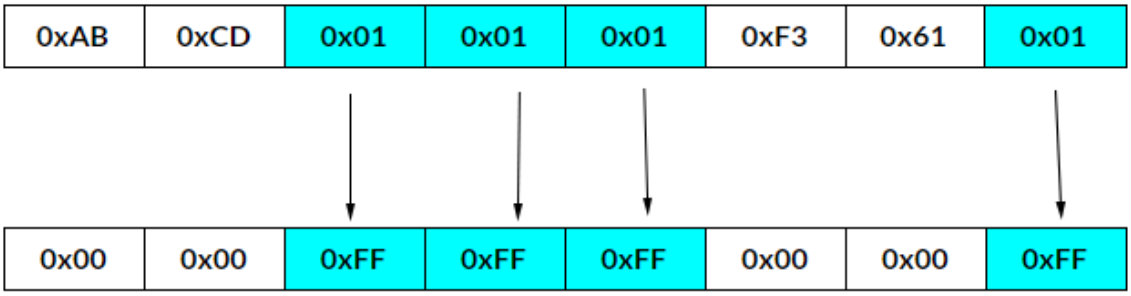
Vec_cnt $vd, $vs, b

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

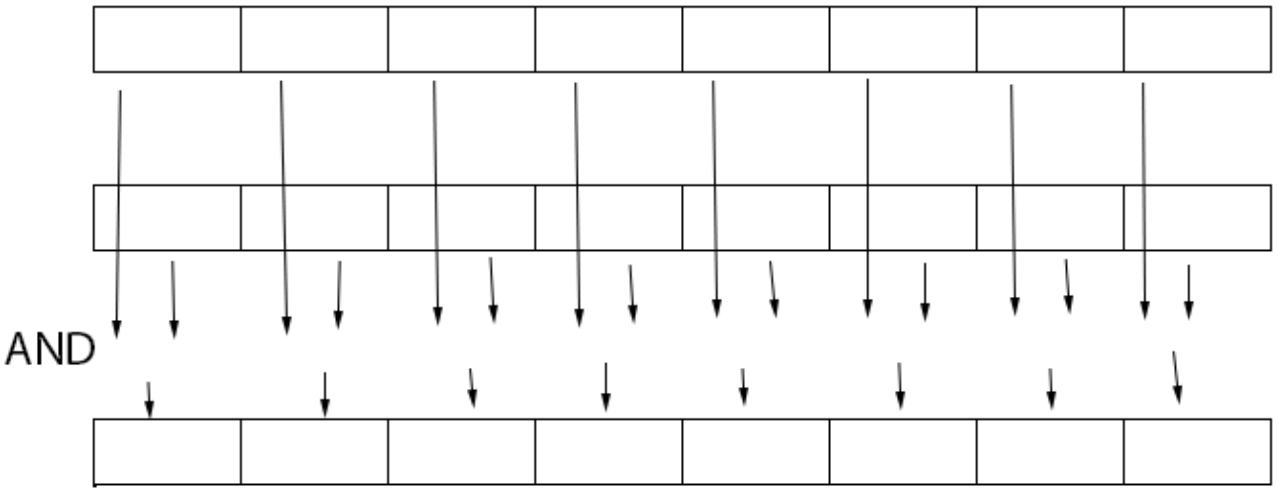
opcode	function	destination	source	source2	source3
0111	1010	vd	vs	vt	000000
31 28	27 24	23 18	17 12	11 6	5 0

MIPS Format

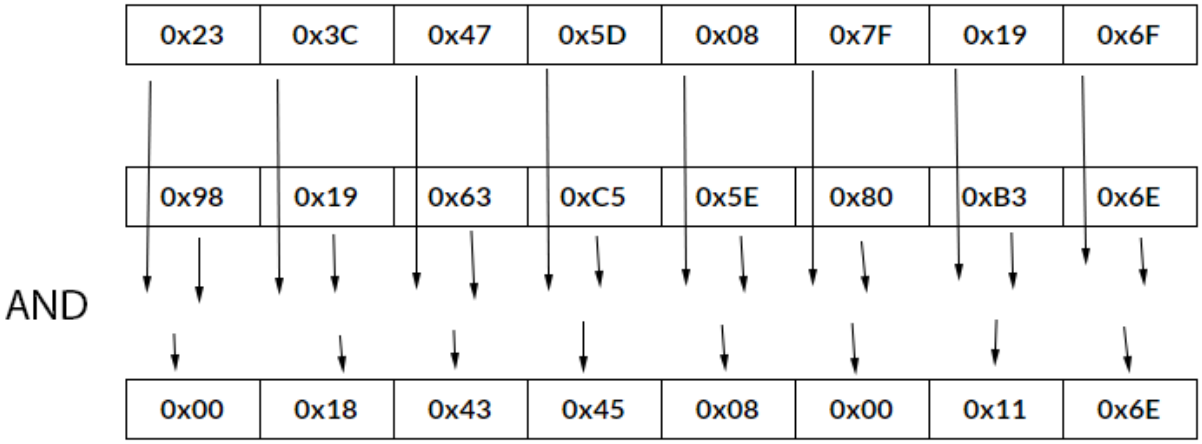
Vec_and $vd, $vs, $vt

Operation

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



Example


0x23
0x3C
0x47
0x5D
0x08
0x7F
0x19
0x6F
0x98
0x19
0x63
0xC5
0x5E
0x80
0xB3
0x6E
AND
0x00
0x18
0x43
0x45
0x08
0x00
0x11
0x6E

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

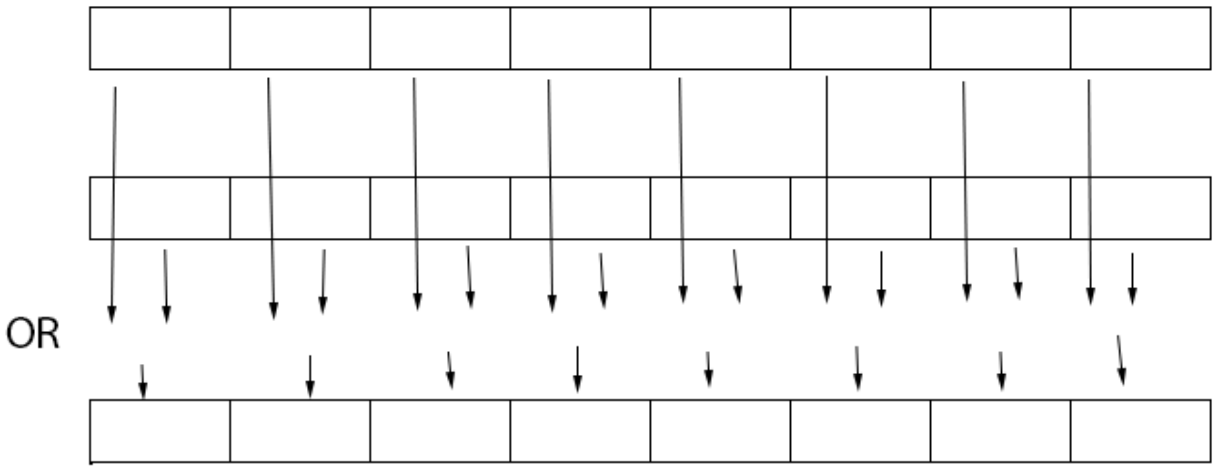
opcode	function	destination	source	source2	source3
0111	1010	vd	vs	vt	000000
31 28	27 24	23 18	17 12	11 6	5 0

MIPS Format

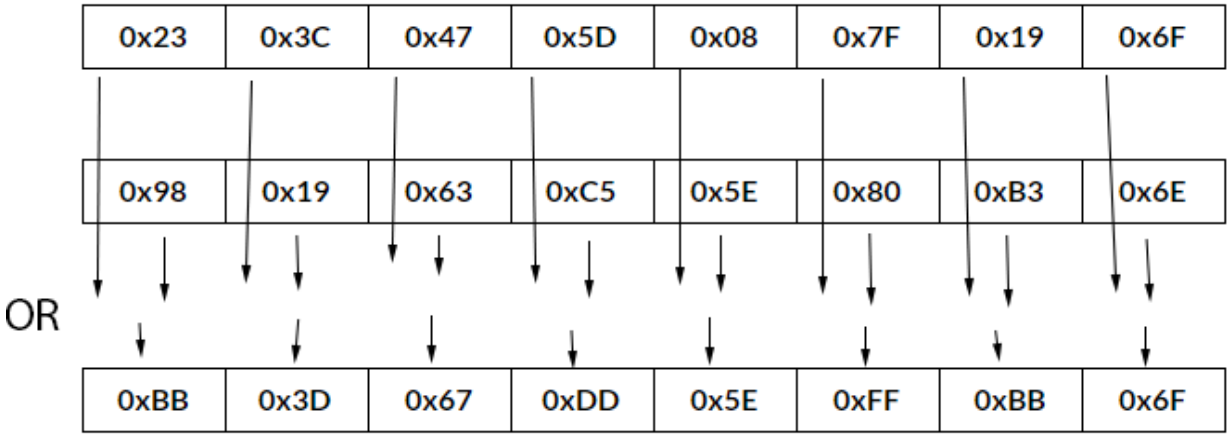
Vec_or $vd, $vs, $vt

Operation

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



Example


0x23
0x3C
0x47
0x5D
0x08
0x7F
0x19
0x6F
0x98
0x19
0x63
0xC5
0x5E
0x80
0xB3
0x6E
OR
0xBB
0x3D
0x67
0xDD
0x5E
0xFF
0xBB
0x6F

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

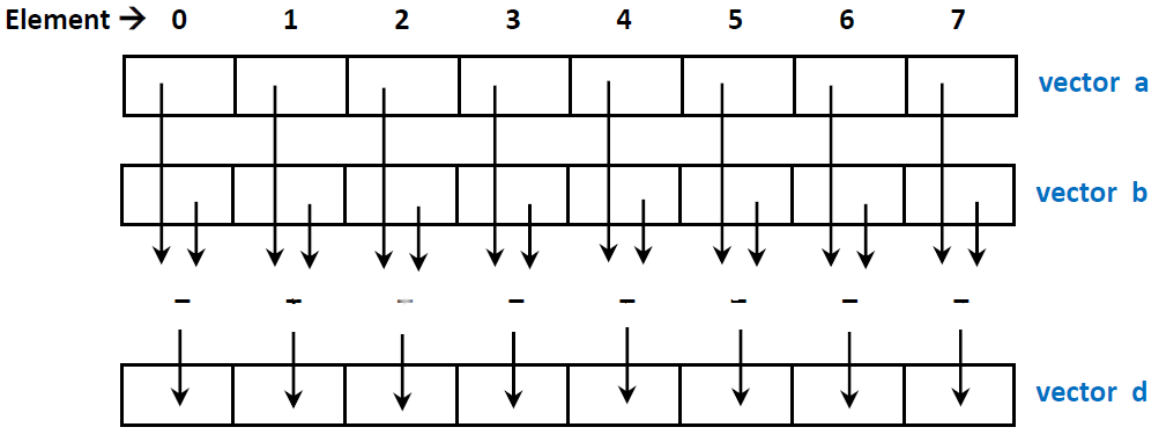
opcode	function	destination	source	source2	source3
0111	0000	vd	vs	vt	00000
31 28	27 24	23 18	17 12	11 6	5 0

MIPS Format

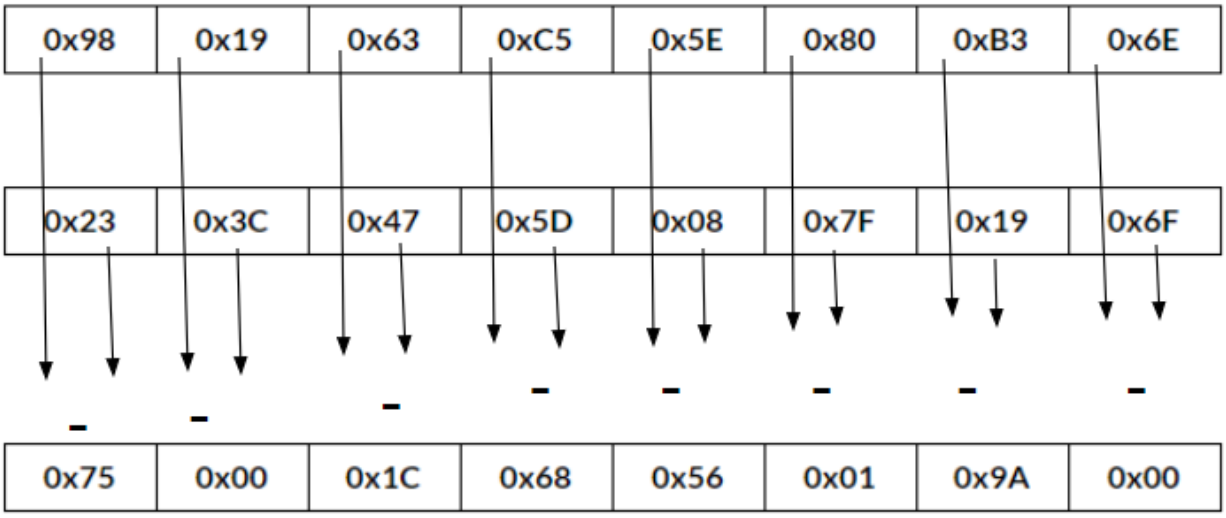
vec_subu d, a, b

Operation

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



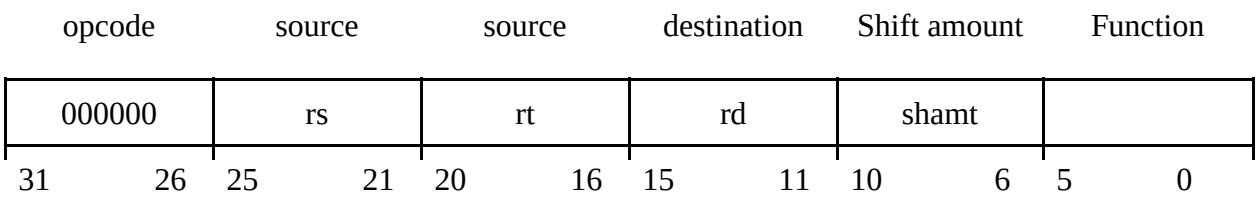
Example


0x98
0x19
0x63
0xC5
0x5E
0x80
0xB3
0x6E
0x23
0x3C
0x47
0x5D
0x08
0x7F
0x19
0x6F
0x75
0x00
0x1C
0x68
0x56
0x01
0x9A
0x00

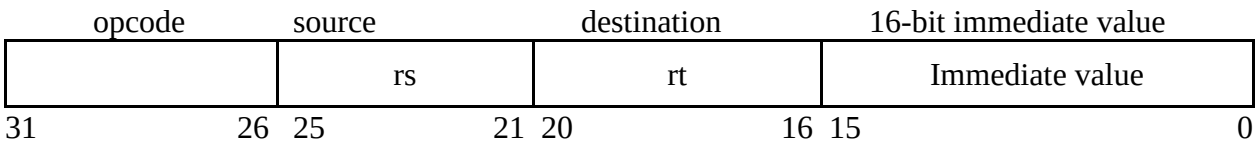
E. Summary of the Instruction Formats of “Baseline SIMD Enhancements” and “Application Specific” enhancements

Basic Formats

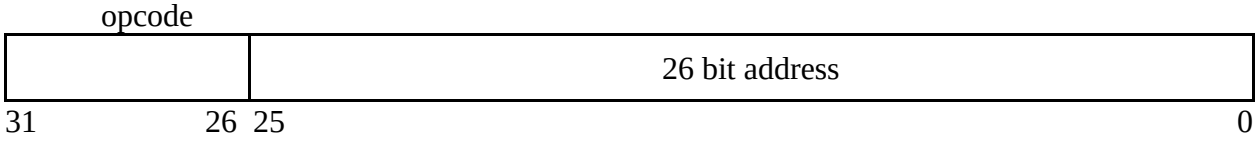
R-Type



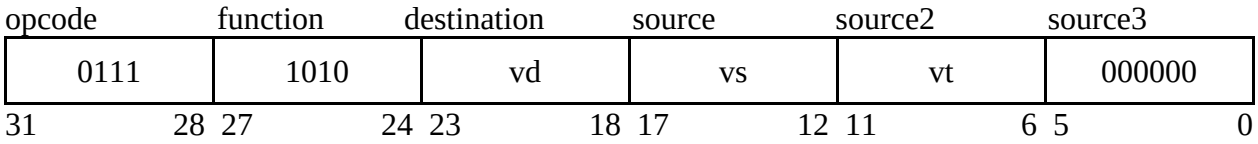
I-Type



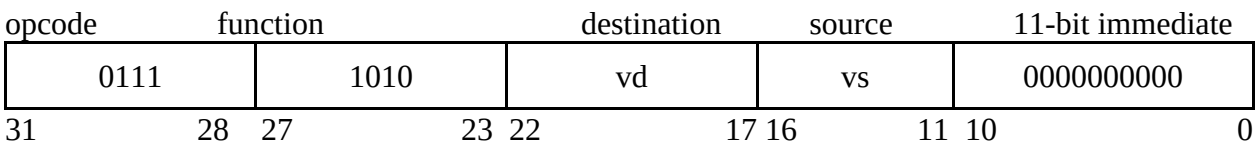
J-Type



V-type



VI-type



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:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x00000023	0x0000003c	0x00000047	0x0000005d
0x10010020	0x00000098	0x00000019	0x00000063	0x000000c5

Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000008	0x0000007f	0x00000019	0x0000006f
0x0000005e	0x00000080	0x000000b3	0x0000006e

After execution

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000004
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010020
$a1	5	0x10010040
$a2	6	0x10010044
$a3	7	0x00000000
$t0	8	0xbb55aaff
$t1	9	0x66ffccdd
$t2	10	0x000000dd

Vec_madd

Before execution:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000012	0x0000000c	0x0000001a	0x0000000d	0x00000023	0x00000005	0x00000019	0x00000012
0x10010020	0x0000003d	0x0000000c	0x00000010	0x0000004d	0x00000005	0x0000007f	0x00000019	0x0000002b
0x10010040	0x00000060	0x00000009	0x0000001b	0x00000005	0x00000050	0x0000001e	0x00000006	0x00000060

After execution

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000004
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010020
$a1	5	0x10010040
$a2	6	0x10010064
$a3	7	0x10010060
$t0	8	0xaa99bbee
$t1	9	0xff997766
$t2	10	0x00000066

Vec_mule

Before Execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x000000ae	0x000000e9	0x0000005a	0x00000050
0x10010020	0x00000033	0x00000014	0x00000061	0x00000070

Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x000000f0	0x00000080	0x000000cc	0x00000066
0x00000060	0x00000098	0x00000088	0x000000ab

After execution

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000004
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010020
$a1	5	0x10010040
$a2	6	0x10010040
$a3	7	0x00000000
$t0	8	0x22aa221a
$t1	9	0x0000221a
$t2	10	0x5a006c60
$t3	11	0x00006c60
$t4	12	0x00000004

Vec_mulo

Before execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x000000ae	0x000000e9	0x0000005a	0x000000e0
0x10010020	0x00000033	0x00000014	0x00000061	0x00000070

Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x000000f0	0x00000080	0x000000cc	0x00000066
0x00000060	0x00000098	0x00000088	0x000000ab

After execution

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000004
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010024
$a1	5	0x10010044
$a2	6	0x10010040
$a3	7	0x00000000
$t0	8	0x12346200
$t1	9	0x00006200
$t2	10	0x4c004422
$t3	11	0x00004422
$t4	12	0x00000004
$t5	13	0x00000000

Vec_msums

Before execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x00000023	0x0000000c	0x000000f1	0x0000004d
0x10010020	0x000000a3	0x0000000c	0x0000005b	0x000000fd
0x10010040	0x0000609e	0x000019f7	0x00004567	0x00000766

Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x0000005c	0x0000007f	0x00000019	0x0000001a
0x000000c5	0x000000ff	0x000000c9	0x000000ee
0x00000000	0x00000004	0x00000000	0x00000000

After execution

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000004
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010020
$a1	5	0x10010040
$a2	6	0x10010040
$a3	7	0x10010060
$t0	8	0x7777bbbb
$t1	9	0x0000bbbb
$t2	10	0xffff3333

Vec_splat

Before execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x00000023	0x0000000c	0x00000012	0x0000004d

Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000005	0x0000007f	0x00000019	0x0000002a

After execution

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000004
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010000
$a1	5	0x10010020
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x7f7f7f7f

Vec_mergel

Before execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x0000005a	0x000000f0	0x000000a5	0x00000001
0x10010020	0x000000a5	0x0000000f	0x0000005a	0x00000023

Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x000000ab	0x00000001	0x00000055	0x000000c3
0x000000cd	0x00000023	0x000000aa	0x0000003c

After execution

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000002
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010020
$a1	5	0x10010040
$a2	6	0x10010040
$a3	7	0x00000000
$t0	8	0xabcd0123
$t1	9	0x000000cd
$t2	10	0x55aac33c
$t3	11	0x0000003c

Vec_mergh

Before execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x0000005a	0x000000f0	0x000000a5	0x00000001
0x10010020	0x000000a5	0x0000000f	0x0000005a	0x00000023

Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x000000ab	0x00000001	0x00000055	0x000000c3
0x000000cd	0x00000023	0x000000aa	0x0000003c

After execution

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000002
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010010
$a1	5	0x10010030
$a2	6	0x10010040
$a3	7	0x00000000
$t0	8	0x5aa5f00f
$t1	9	0x000000a5
$t2	10	0xa55a0123
$t3	11	0x00000023

Vec_perm

Before execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x00000004	0x00000017	0x00000010	0x00000002
0x10010020	0x000000a5	0x00000067	0x00000001	0x0000003d
0x10010040	0x000000ef	0x000000c5	0x0000004d	0x00000023

Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000013	0x00000005	0x00000001	0x00000005
0x000000ab	0x00000045	0x00000039	0x0000003c
0x00000012	0x00000077	0x000000aa	0x000000cd

After execution

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000028
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010020
$a1	5	0x10010040
$a2	6	0x10010020
$a3	7	0x10010058
$t0	8	0xabcdef01
$t1	9	0x23456745

Vec_cmpeq

Before execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x0000005a	0x000000fb	0x0000006c	0x0000001d	0x000000a6	0x0000005f	0x000000c0	0x00000040
0x10010020	0x00000052	0x000000fb	0x000000a4	0x00000015	0x000000ae	0x0000005f	0x000000c8	0x00000041

After execution

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000028
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010028
$a1	5	0x10010048
$a2	6	0x10010040
$a3	7	0x00000000
$t0	8	0x00ff0000
$t1	9	0x00ff0000
$t2	10	0x00000000

Vec_cmpltu

Before execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x0000005a	0x000000fb	0x0000006c	0x0000001d
0x10010020	0x00000052	0x000000fb	0x000000a4	0x00000015

Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x000000a6	0x0000005f	0x000000c0	0x00000040
0x000000ae	0x0000005f	0x000000c8	0x00000041

After execution

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000028
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010028
$a1	5	0x10010048
$a2	6	0x10010040
$a3	7	0x00000000
$t0	8	0x0000ff00
$t1	9	0xff00ffff
$t2	10	0x000000ff

Vec_swap

Before execution:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x0000abcd	0x00001234	0x0000ef01	0x00005678

After execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x0000ef01	0x00005678	0x0000abcd	0x00001234

Vec_sll

Before execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x0000abcd	0x00001234	0x00005678	0x00009abc

After execution

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000004
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010010
$a1	5	0x10010010
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00013578
$t1	9	0xffff2468
$t2	10	0xacf0ffff

Vec_srl

Before execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x0000abcd	0x00001234	0x00005678	0x00009abc

After execution

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000004
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010010
$a1	5	0x10010010
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00004d5e
$t1	9	0x55e6091a
$t2	10	0x2b3c4d5e

Vec_cpy

Before execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x0000abcd	0x00001234	0x00005678	0x00009abc

After execution

Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x0000abcd	0x00001234	0x00005678	0x00009abc

Vec_cnt

Before execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x000000ab	0x000000cd	0x00000001	0x00000001

Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000001	0x000000f3	0x00000061	0x00000001

After execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x000000ab	0x000000cd	0x00000001	0x00000001
0x10010020	0x00000000	0x00000000	0x000000ff	0x000000ff

Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000001	0x000000f3	0x00000061	0x00000001
0x000000ff	0x00000000	0x00000000	0x000000ff

Vec_and

Before execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x00000023	0x0000003c	0x00000047	0x0000005d
0x10010020	0x00000098	0x00000019	0x00000063	0x000000c5

Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000008	0x0000007f	0x00000019	0x0000006f
0x0000005e	0x00000080	0x000000b3	0x0000006e

After execution

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000004
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010020
$a1	5	0x10010040
$a2	6	0x10010040
$a3	7	0x00000000
$t0	8	0x00184345
$t1	9	0x0800116e
$t2	10	0x0000006e

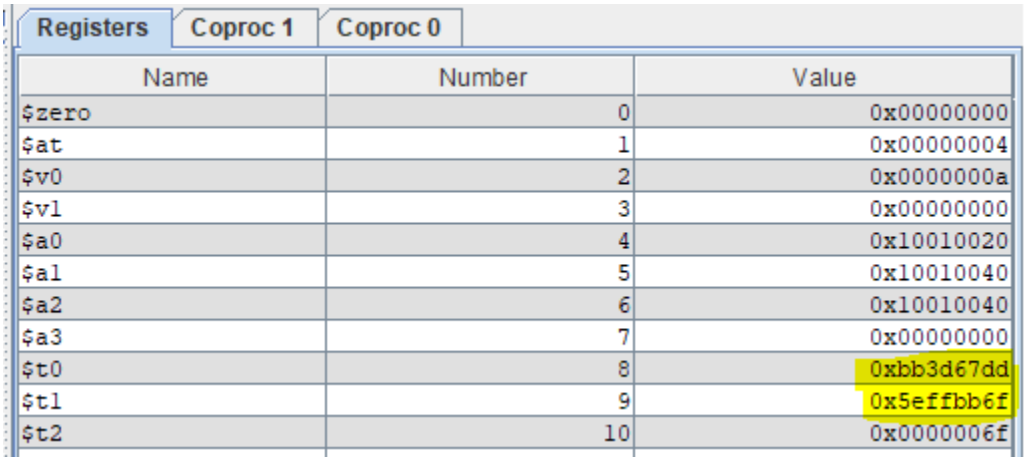
Vec_or

Before execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x00000023	0x0000003c	0x00000047	0x0000005d
0x10010020	0x00000098	0x00000019	0x00000063	0x000000c5

Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000008	0x0000007f	0x00000019	0x0000006f
0x0000005e	0x00000080	0x000000b3	0x0000006e

After execution



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000004
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10010020
$a1	5	0x10010040
$a2	6	0x10010040
$a3	7	0x00000000
$t0	8	0xbb3d67dd
$t1	9	0x5effbb6f
$t2	10	0x0000006f

Vec_subu

Before execution

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x00000023	0x0000003c	0x00000047	0x0000005d
0x10010020	0x00000098	0x00000019	0x00000063	0x000000c5

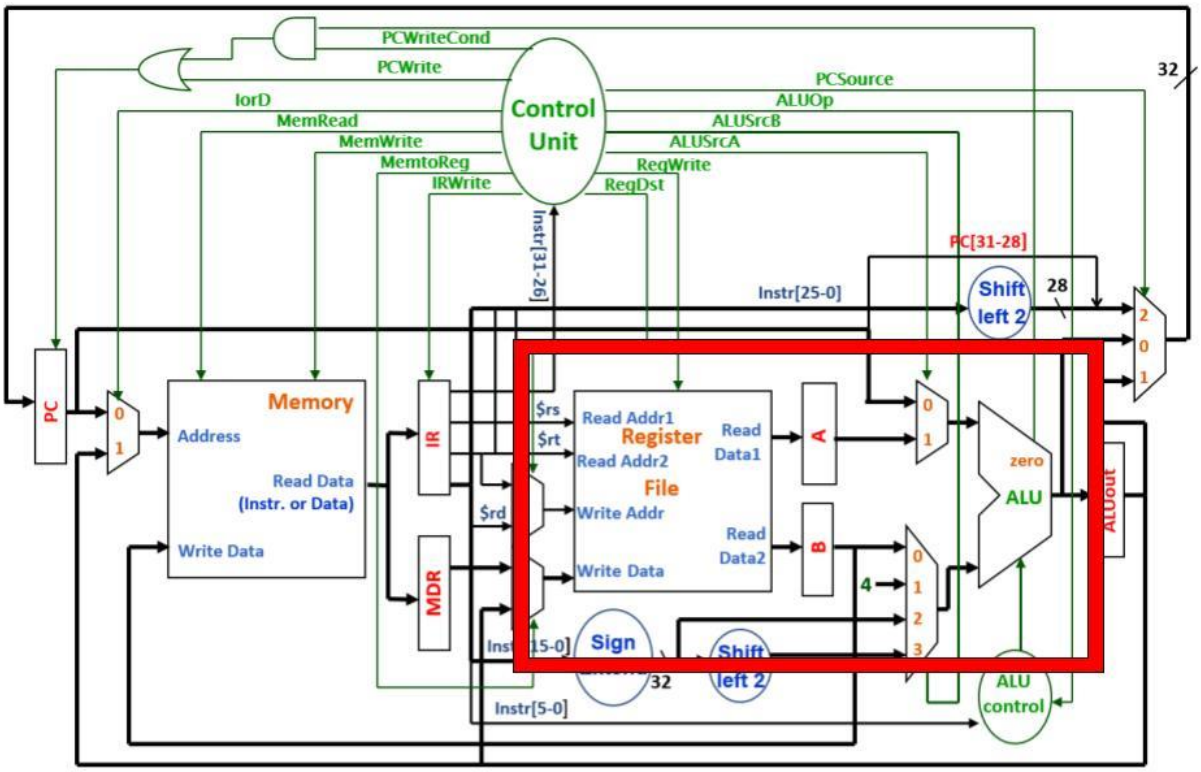
Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000008	0x0000007f	0x00000019	0x0000006f
0x0000005e	0x00000080	0x000000b3	0x0000006e

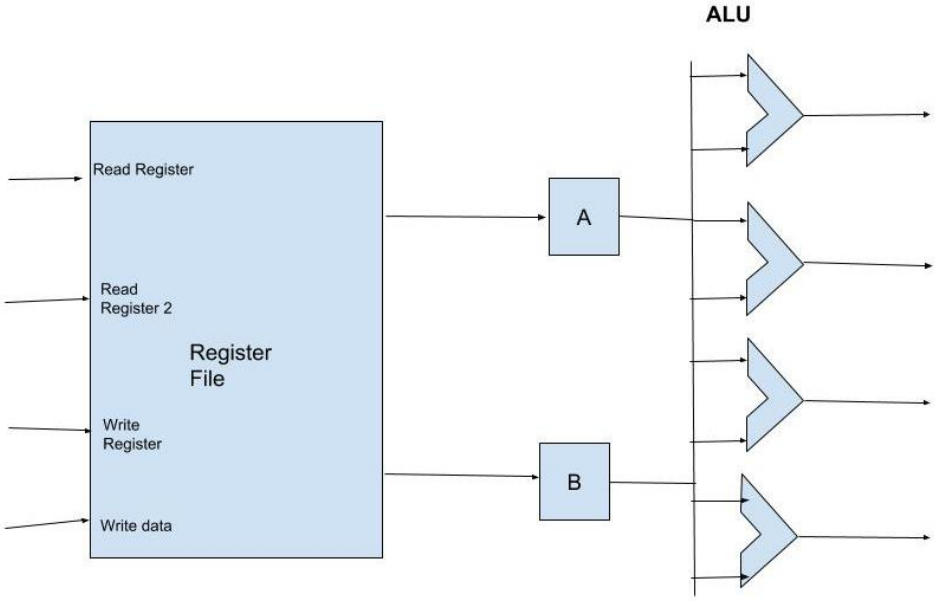
After execution

Registers | Coproc 1 | Coproc 0

Name	Number	Value
zero	0	0x00000000
at	1	0x00000004
v0	2	0x0000000a
v1	3	0x00000000
a0	4	0x10010020
a1	5	0x10010040
a2	6	0x10010044
a3	7	0x00000000
t0	8	0x75001c68
t1	9	0x56019a00

IV. Datapath Block Diagrams


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Control Unit
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
32
Instr[31-26]
PC[31-28]
Instr[25-0]
Shift left 2
28
PC
Memory
Address
Read Data (Instr. or Data)
Write Data
IR
MDR
$rs
$rt
$rd
Read Addr1
Read Addr2
Register File
Write Addr
Write Data
Read Data1
Read Data2
A
B
4
zero
ALU
ALUout
Sign
Shift left 2
32
Instr[5-0]
ALU control


ALU
Read Register
Read Register 2
Register File
Write Register
Write data
A
B

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.