Gold Variable-Width Processor
Instruction Set Architecture Manual

This chapter gives an instruction set overview and provides a detailed instruction description. All processor instructions are 32-bits long. Big-Endian byte and bit labeling is used, meaning that bit/byte 0 is the most significant bit. Other conventions are listed in the table below.

Table 1: Instruction Glossary

Symbol	Meaning
A ← B	Assignment
{x, y}	Bit string concatenation
{y{x}}	x replicated y times
x[y: z]	selection of bits y to z from x
x & y	x bitwise ANDed with y
x | y	x bitwise ORed with y
x ^ y	x bitwise XORed with y
~x	bitwise inversion of x
MEM[EA]	memory contents at effective address EA
0x*value*	Hexadecimal value
0b*value*	Binary Value
(rX)	Contents of general purpose register X
nibble	4-bit value
byte	8-bit value
half-word	16-bit value
word	32-bit value
double-word	64-bit value
INT(x)	Integer value of x
x MOD y	x modulo y

The following table gives the rules of precedence and associativity for the pseudocode operators. All operators on the same line have equal precedence, and all operators on a given line have higher precedence than those on the lines below them.

Table 2: Precedence of Pseudocode Operators

Operator	Associativity
Mem[n]	Left to right
x[y:z]	Left to right
{y{x}}	Left to right
~	Left to right
×,÷	Left to right
+, -	Left to right
{x, y}	Left to right
=, !=, <, <=, >, >=	Left to right
^, &	Left to right
|	Left to right
←	none

Instruction Formats

As shown in Figure 1, most Gold processor instructions use a three-operand format (R-type) to specify two 64-bit source registers (rA and rB) and one 64-bit destination register (rD). Load and Store use a different instruction format (M-type) shown in Figure 2. Note that the above instruction format classifications are generalized, meaning that some instructions may vary from the format described above. For instance, the shift immediate instructions are classified as R-type instructions, although they specify a 5-bit immediate shift amount in place of rB. The end result is that not all R-type instructions can be decoded in exactly the same way; the rB field may be a register specifier or it may be a 5-bit immediate.

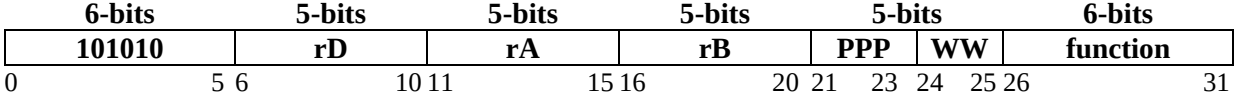


Figure 1: Format R-type for Arithmetic/Logical Operations

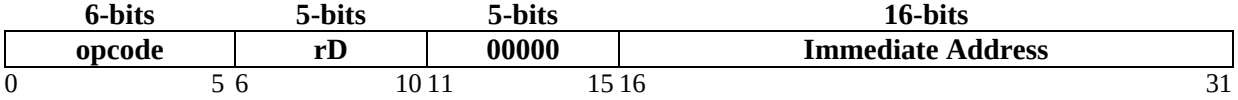


Figure 2: Format M-type for Load/Store Operations

The control field bits described as follows:

WW (word width field)
The 2-bit WW field defines the width of the operands for the R-type instructions. This affects primarily shift, and arithmetic operations. The encoding used for setting operands width is as follows:

WW value	Operand Width	Assembly Mnemonic
00	8	b
01	16	h
10	32	w
11	64	d

The following table shows the possible subfield indices for the different values of ***WW*** (recall that with Big-Endian labeling used, therefore, subfield 0 is always the most significant regardless of operand size):

WW	Width	Subfield indices within a 64-bit register for different operand widths							
		most sig.							least sig.
		Even	Odd	Even	Odd	Even	Odd	Even	Odd
00	**8**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
01	**16**	**0**		**1**		**2**		**3**	
10	**32**	**0**				**1**			
11	**64**	**0**							

For multiply and squaring instructions referring to even/odd data-operands, the exact data-operands that participate in the multiply operation are dependent on the operand width specified by the WW field. The above table may be useful for visualizing which data-operands participate based on operand width value. For instance, an "even" data-operands in MULEU instruction for WW = 01 (16-bit data) operations means that only half-words 0 and 2 are used for multiplication.

PPP (participation)
The 3-bit PPP, or participation field specifies what kind of selective execution, if any, governs the operations. Recall that under selective execution only certain bytes commit their results during the writeback stage. The bytes participating are specified by the decoding of PPP. The encoding of the PPP bits is listed in the following table:

PPP value	Participation Definition	Assembly Mnemonic
000	All bytes participate	a
001	Upper 32-bits participate	u
010	Lower 32-bits participate	d
011	Bytes with even index participate	e
100	Bytes with odd index participate	o
101	reserved	None
110	reserved	None
111	reserved	None

ALU Input:
function: 6bit= left6->2+right6->4
RA/RB = 64bit for each
RD = 5bit
PPPWW = 5bit
144bit

ALU Output:
data 64bit
PPP 3bit;
RD 5bit;
clk, rst
72bit

Preliminary Encoding of Instruction Set

Sr. No.	Instruction	Format	Encoding					
			6bits	5bits	5bits	5bits	5bits	6bits
1.	VAND	R	101010	rD	rA	rB	PPPWW	000000
2.	VOR	R	101010	rD	rA	rB	PPPWW	000001
3.	VXOR	R	101010	rD	rA	rB	PPPWW	000010
4.	VNOT	R	101010	rD	rA	00000	PPPWW	000011
5.	VMOV	R	101010	rD	rA	00000	PPPWW	000100
6.	VADD	R	101010	rD	rA	rB	PPPWW	000101
7.	VSUB	R	101010	rD	rA	rB	PPPWW	000110
8.	VMULEU	R	101010	rD	rA	rB	PPPWW	000111
9.	VMULOU	R	101010	rD	rA	rB	PPPWW	001000
10.	VRTTH	R	101010	rD	rA	00000	PPPWW	001001
11.	VSLL	R	101010	rD	rA	rB	PPPWW	001010
12.	VSLLI	R	101010	rD	rA	shift_amount	PPPWW	001011
13.	VSRL	R	101010	rD	rA	rB	PPPWW	001100
14.	VSRLI	R	101010	rD	rA	shift_amount	PPPWW	001101
15.	VSRA	R	101010	rD	rA	rB	PPPWW	001110

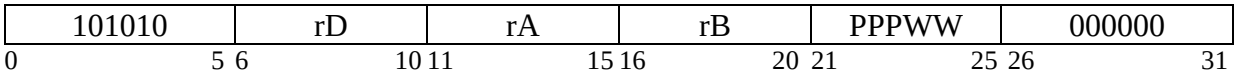
16.	**VSRAI**	R	101010	rD	rA	shift_amount	PPPWW	001111
17.	**VLD**	M	100000	rD	00000	Immediate Address		
18.	**VSD**	M	100001	rA	00000	Immediate Address		
19.	**VBEZ**	R	100010	rD	00000	Immediate Address		
20.	**VBNEZ**	R	100011	rD	00000	Immediate Address		
21.	**VNOP**	R	111100	00000	00000	00000	00000	000000

Any instruction which doesn't follow the above instruction format should be treated as illegal instruction and be executed as **No-Operation** instruction i.e. it should NOT modify contents of register-file or data-memory.

The detailed Instruction descriptions for each instruction are as follows:

vandxy – Variable width AND

vanda*w* rD, rA, rB



Variable values in the following equations are as follows:

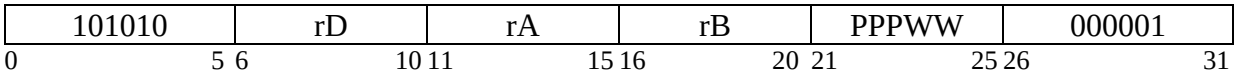
WW value	size
00	8
01	16
10	32
11	64

For i = 0 to (64 – *size*) by *size*
    if PPP bits enable writeback for this subfield
    rD[*i:(i+(size – 1))*] ← (rA)[*i:(i+(size – 1))*] & (rB)[*i:(i+(size – 1))*]

The 64-bit contents of rA are ANDed with the 64-bit contents of rB, and the result is placed into rD. The WW field determines if the 64-bit contents of rA and rB are treated as bytes/half-words/words/double-word. The WW field bits do not affect the bit-wise ANDing operation.

vorxy – Variable width OR

vora*w* rD, rA, rB



Variable values in the following equations are as follows:

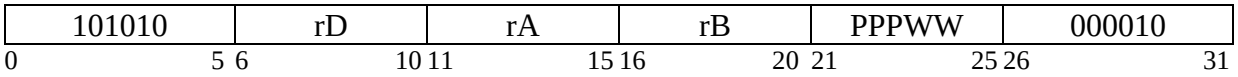
WW value	size
00	8
01	16
10	32
11	64

For i = 0 to (64 – *size*) by *size*
    if PPP bits enable writeback for this subfield
    rD[*i:(i+(size – 1))*] ← (rA)[*i:(i+(size – 1))*] | (rB)[*i:(i+(size – 1))*]

The 64-bit contents of rA are ORed with the 64-bit contents of rB, and the result is placed into rD. The WW field determines if the 64-bit contents of rA and rB are treated as bytes/half-words/words/double-word. The WW field bits do not affect the bit-wise ORing operation.

vxorxy – Variable width XOR

vxora*w* rD, rA, rB



Variable values in the following equations are as follows:

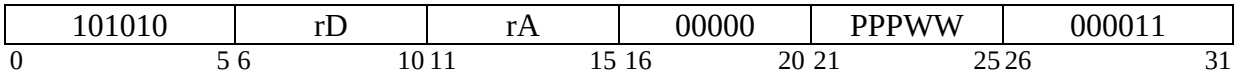
WW value	size
00	8
01	16
10	32
11	64

For i = 0 to (64 – *size*) by *size*
    if PPP bits enable writeback for this subfield
    rD[*i:(i+(size – 1))*] ← (rA)[*i:(i+(size – 1))*] ^ (rB)[*i:(i+(size – 1))*]

The 64-bit contents of rA are XOR'ed with the 64-bit contents of rB, and the result is placed into rD. The WW field determines if the 64-bit contents of rA and rB are treated as bytes/half-words/words/double-word. The WW field bits do not affect the bit-wise XORing operation.

vnotxy – Variable width NOT

vnota*w* rD, rA



Variable values in the following equations are as follows:

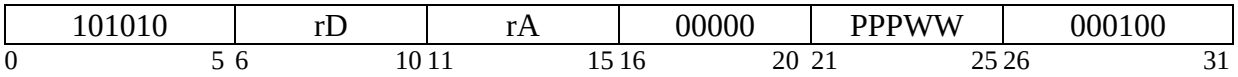
WW value	size
00	8
01	16
10	32
11	64

For i = 0 to (64 – *size*) by *size*
    if PPP bits enable writeback for this subfield
    rD [*i: (i+ (size – 1))*] ← ~ (rA) [*i: (i+ (size – 1))*]

The 64-bit contents of rA are bit-inverted, and the result is placed into rD. The WW field determines if the 64-bit contents of rA are treated as bytes/half-words/words/double-word. The WW field bits do not affect the bit-wise inversion operation.

vmovxy – Variable width Move

vmova*w* rD, rA



Variable values in the following equations are as follows:

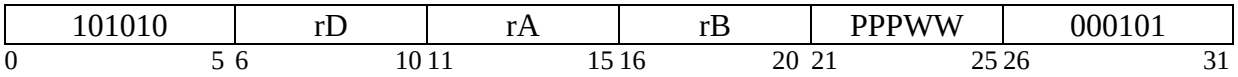
WW value	size
00	8
01	16
10	32
11	64

For i = 0 to (64 – *size*) by *size*
    if PPP bits enable writeback for this subfield
    rD [*i :(i+ (size – 1))*] ← (rA) [*i :(i+ (size – 1))*]

The entire contents of 64-bit register rA are transferred to destination register rD. The WW field determines if the 64-bit contents of rA are treated as bytes/half-words/words/double-word. The WW field bits do not affect the bit-wise MOVE operation.

vaddxy – Variable width Add

vaddu*w* rD, rA, rB



Variable values in the following equations are as follows:

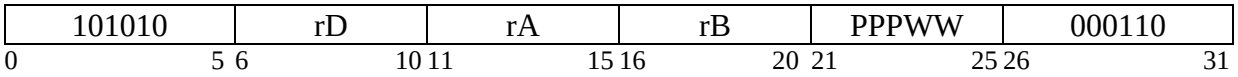
WW value	size
00	8
01	16
10	32
11	64

For i = 0 to (64 – *size*) by *size*
    if PPP bits enable writeback for this subfield
    rD[*i:(i+(size – 1))*] ← (rA)[*i:(i+(size – 1))*] + (rB)[*i:(i+(size – 1))*]

Each integer bytes/half-words/words/double-word of rA is added with the corresponding integer bytes/half-words/words/double-word of rB. The WW field determines if the 64-bit contents of rA and rB are treated as bytes/half-words/words/double-word. The resulting bytes/half-words/words/double-word sums are written into rD. We ignore the generated carry-bit for each integer addition.

vsubxy – Variable width Subtract

vsubu*w* rD, rA, rB



Variable values in the following equations are as follows:

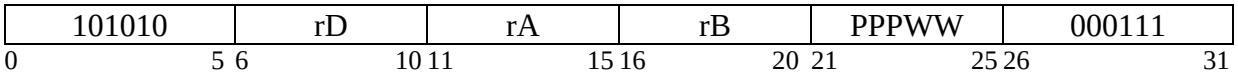
WW value	size
00	8
01	16
10	32
11	64

For i = 0 to (64 – *size*) by *size*
    if PPP bits enable writeback for this subfield
    rD[*i:(i+(size – 1))*] ← (rA)[*i:(i+(size – 1))*] + ~(rB)[*i:(i+(size – 1))*] + 1

Each integer bytes/half-words/words/double-word of rB is subtracted from the corresponding integer bytes/half-words/words/double-word of rA. The WW field determines if the 64-bit contents of rA and rB are treated as bytes/half-words/words/double-word. The resulting integer bytes/half-words/words/double-word are written into rD.
The pseudo-code presented above performs addition of contents of rA with the 2's complement value of corresponding value of rB, which is equivalent to subtracting rB from rA. We ignore any generated overflow bits.

vmuleuxy – Variable width Multiply Even Unsigned

vmuleua*w* rD, rA, rB



Variable values in the following equations are as follows:

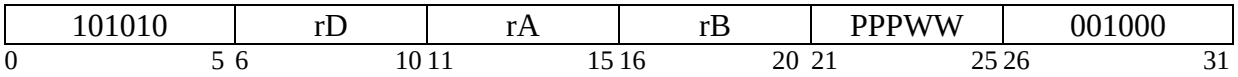
WW value	size	Output size
00	8	16
01	16	32
10	32	64

For i = 0 to (64 – *2*× *size*) by (2 × *size)*
    if PPP bits enable writeback for this subfield
    rD[*i:(i+(2*× *size – 1))*] ← (rA)[*i:(i+(size – 1))*] × (rB)[*i:(i+(size – 1))*]

Each even numbered unsigned integer bytes/half-words/words of rA is multiplied by the corresponding even numbered unsigned integer bytes/half-words/words of rB. The WW field determines if the 64-bit contents of rA and rB are treated as bytes/half-words/words. The resulting unsigned half-words/words/double-word products are written into rD. Recall, that the product in the multiplication operation has twice the width of the input operands, therefore, in a 64-bit register, we can only store complete results for up-to 32-bit operands and therefore full 64-bit x 64-bit multiplication is not supported.

vmulouxy – Variable width Multiply Odd Unsigned

vmulou*w* rD, rA, rB



Variable values in the following equations are as follows:

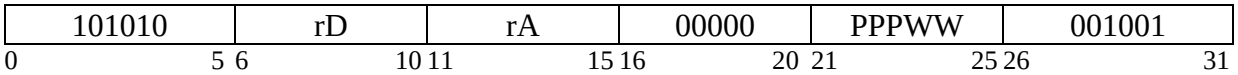
WW value	Input size	Output size
00	8	16
01	16	32
10	32	64

For i = 0 to (64 – *2× size*) by (2 × *size)*
    if PPP bits enable writeback for this subfield
    rD[*i:(i+(2×size – 1))*] ← (rA)[(*i+size):(i+(2×size – 1))*] × (rB)[(*i+size):(i+(2×size – 1))*]

Each odd numbered unsigned integer bytes/half-words/words of rA is multiplied by the corresponding odd numbered unsigned integer bytes/half-words/words of rB. The WW field determines if the 64-bit contents of rA and rB are treated as bytes/half-words/words. The resulting unsigned half-words/words/double-word products are written into rD. Recall, that the product in the multiplication operation has twice the width of the input operands, therefore, in a 64-bit register, we can only store complete results for up-to 32-bit operands and therefore full 64-bit x 64-bit multiplication is not supported.

vrtthxy – Variable width Rotate by Half

vrtthaw rD, rA



Variable values in the following equations are as follows:

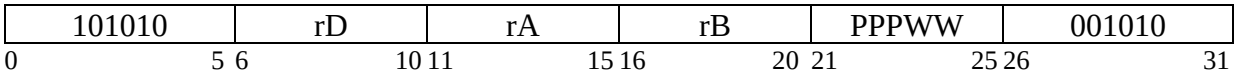
WW value	size
00	8
01	16
10	32
11	64

For i = 0 to (64 – *size*) by *size*
    if PPP bits enable writeback for this subfield
    rD[*i:(i+(size – 1))*] ← {(rA)[*i+ size/2:(i+(size – 1))*], (rA)[*i:(i+(size/2 – 1))*]}

Each bytes/half-words/words/double-word of rA are rotated right by half of the size as specified by WW field bits. The result is that lower nibbles/bytes/half-words/word is swapped with the higher nibbles/bytes/half-words/word respectively. The WW field determines if the 64-bit contents of rA are treated as bytes/half-words/words/double-word, and it also determines the size of the rotation. The resulting bytes/half-words/words/double-word are written into rD.

vsllxy – Variable width Shift Left Logical

vsllu*w* rD, rA, rB



Variable values in the following equations are as follows:

WW value	size	Bits
00	8	3
01	16	4
10	32	5
11	64	6

For i = 0 to (64 – *size*) by *size*

s ← (rB) [*(i+ size –bits) :(i + size – 1)*]

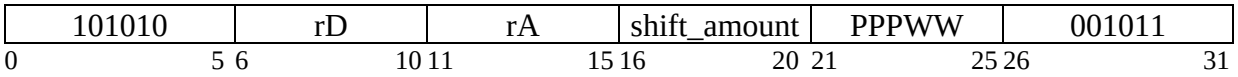
if PPP bits enable writeback for this subfield

rD[*i:(i+(size – 1))*] ← {(rA)[*(i+ s):(i+ size* – 1], s{0}}

Each bytes/half-words/words/double-word of rA are shifted left by the value given by the number of bits specified by the lower significant bits (LSB) of the corresponding data-fields contained as contents of rB, and inserting zeros into the least significant bits (LSB) of each data-field of the result. The WW field determines if the 64-bit contents of rA and rB are treated as bytes/half-words/words/double-word, and it also determines how many lower significant bits are to be considered for calculation of shift-amount. The resulting bytes/half-words/words/double-word are written into rD.

vsllixy – Variable width Shift Left Logical Immediate

vslliu*w* rD, rA, shift_amount



Variable values in the following equations are as follows:

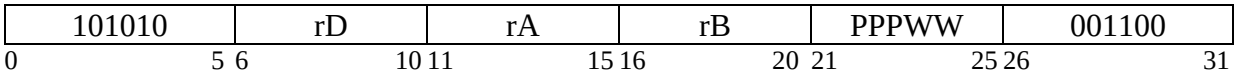
WW value	size	Bits
00	8	3
01	16	4
10	32	5
11	64	5

s ← shift_amount [*(5 –bits) :4*]
For i = 0 to (64 – *size*) by *size*
    if PPP bits enable writeback for this subfield
    rD[*i:(i+(size – 1))*] ← {(rA)[*(i+ s):(i+ size* – 1], s{0}}

Each bytes/half-words/words/double-word of rA are shifted left by the value given by the number of bits specified by the lower significant bits (LSB) of the shift_amount data-field and inserting zeros into the least significant bits (LSB) of each data-field of the result. The WW field determines if the 64-bit contents of rA are treated as bytes/half-words/words/double-word, and it also determines how many lower significant bits are to be considered for calculation of shift-amount. The resulting bytes/half-words/words/double-word are written into rD.

vsrlxy – Variable width Shift Right Logical

vsrlu*w* rD, rA, rB



Variable values in the following equations are as follows:

WW value	size	Bits
00	8	3
01	16	4
10	32	5
11	64	6

For i = 0 to (64 – *size*) by *size*

if PPP bits enable writeback for this subfield

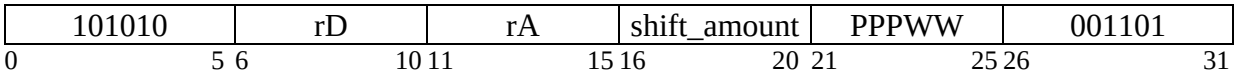
$s \leftarrow$ (rB) [*(i+ size – bits) :(i + size – 1)*]

rD [*i :(i+ (size – 1))*] $\leftarrow$ {s {0}, (rA) [*i :(i + size – s* – 1]}

Each bytes/half-words/words/double-word of rA are shifted right by the value given by the number of bits specified by the lower significant bits (LSB) of the corresponding data-fields contained as contents of rB, and inserting zeros into the most significant bits (MSB) of each data-field of the result. The WW field determines if the 64-bit contents of rA and rB are treated as bytes/half-words/words/double-word, and it also determines how many lower significant bits are to be considered for calculation of shift-amount. The resulting bytes/half-words/words/double-word are written into rD.

vsrlixy – Variable width Shift Right Logical Immediate

vsrliu*w* rD, rA, rB



Variable values in the following equations are as follows:

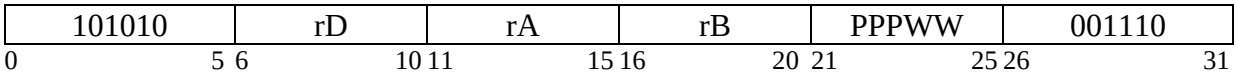
WW value	size	Bits
00	8	3
01	16	4
10	32	5
11	64	5

s ← shift_amount [*(5 – bits) :4*]
For i = 0 to (64 – *size*) by *size*
    if PPP bits enable writeback for this subfield
    rD [*i :(i+ (size – 1))*] ← {s {0}, (rA) [*i :(i + size – s* – 1]}

Each bytes/half-words/words/double-word of rA are shifted right by the value given by the number of bits specified by the lower significant bits (LSB) of the shift_amount data-fields and inserting zeros into the most significant bits (MSB) of each data-field of the result. The WW field determines if the 64-bit contents of rA are treated as bytes/half-words/words/double-word, and it also determines how many lower significant bits are to be considered for calculation of shift-amount. The resulting bytes/half-words/words/double-word are written into rD.

vsraxy – Variable width Shift Right Arithmetic

vsrau*w* rD, rA, rB



Variable values in the following equations are as follows:

WW value	size	Bits
00	8	3
01	16	4
10	32	5
11	64	6

For i = 0 to (64 – *size*) by *size*

if PPP bits enable writeback for this subfield

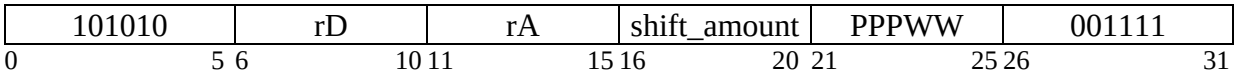
s ← (rB) [*(i+ size –bits) :(i+size – 1)*]

rD[*i:(i+(size – 1))*] ← {s{(rA)[i]}, (rA)[*i :(i + size – s* – 1]}

Each bytes/half-words/words/double-word of rA are shifted right by the value given by the number of bits specified by the lower significant bits (LSB) of the corresponding data-fields contained as contents of rB, and sign extending the most significant bit into the most significant bits (MSB) of each data-field of the result. The WW field determines if the 64-bit contents of rA and rB are treated as bytes/half-words/words/double-word, and it also determines how many lower significant bits are to be considered for calculation of shift-amount. The resulting bytes/half-words/words/double-word are written into rD.

vsraixy – Variable width Shift Right Arithmetic Immediate

vsraiu*w* rD, rA, rB



Variable values in the following equations are as follows:

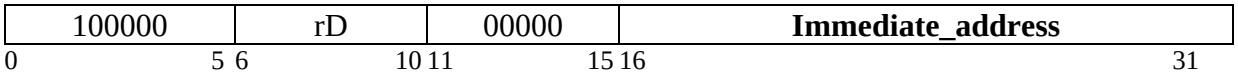
WW value	size	Bits
00	8	3
01	16	4
10	32	5
11	64	**5**

s ← shift_amount [*(5 –bits) :4*]
For i = 0 to (64 – *size*) by *size*
if PPP bits enable writeback for this subfield
rD[*i:(i+(size – 1))*] ← {s{(rA)[i]}, (rA)[*i :(i + size – s* – 1]}

Each bytes/half-words/words/double-word of rA are shifted right by the value given by the number of bits specified by the lower significant bits (LSB) of the shift_amount data-fields and sign extending the most significant bit into the most significant bits (MSB) of each data-field of the result. The WW field determines if the 64-bit contents of rA are treated as bytes/half-words/words/double-word, and it also determines how many lower significant bits are to be considered for calculation of shift-amount. The resulting bytes/half-words/words/double-word are written into rD.

vld – Load Register from data memory

vld rD, Immediate_address

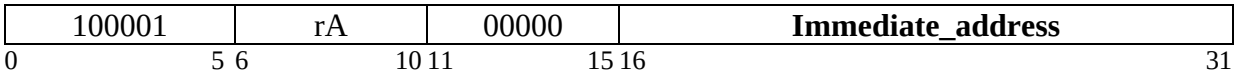


EA ← {16{0}, immediate_address}
rD ← MEM [EA]

The immediate address is assumed to be in terms of 64-bit words. The data-memory provided is only accessible in terms of 64-bit words. The 64-bit value at the memory location specified by EA is then loaded into rD.

vsd – Store Register to data memory

vsd rD, Immediate_address

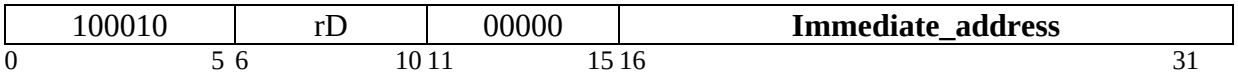


EA ← {16{0}, immediate_address}
MEM [EA] ← (rA)

The immediate address is assumed to be in terms of 64-bit wide words. The data-memory provided is only accessible in terms of 64-bit words. The 64-bit contents specified by the register rD are written to the data-memory location specified in the instruction.

vbeq – Branch if Equal to Zero

vbeq rD, Immediate_address



If (rD) == 0

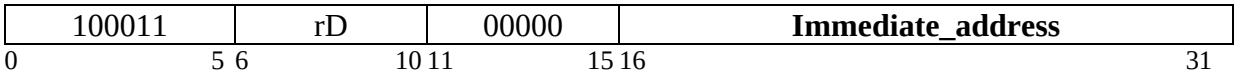
PC ← immediate_address

($0 \leq$ Immediate_address $\leq (2^{16} - 1)$)

If the contents of register rD are zero, then the branch is executed. The branch target address is the 16-bit immediate address as specified in the instruction. In this project, we are implementing absolute branch addressing scheme, therefore, with a branch-target address of 16-bits, we can jump between ($0 \leq$ Immediate_address $\leq (2^{16} - 1)$) address space.

vbneq – Branch if Not Equal to Zero

vbneq rD, Immediate_address



If (rD) != 0

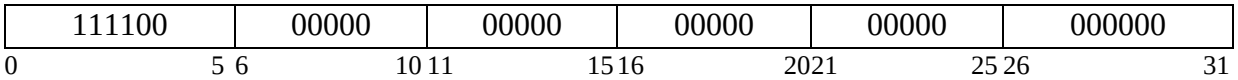
PC ← Immediate_address

$(0 \leq \text{Immediate\_address} \leq (2^{16} - 1))$

If the contents of register rD are not zero, then the branch is executed. The branch target address is the 16-bit immediate address as specified in the instruction. In this project, we are implementing absolute branch addressing scheme, therefore, with a branch-target address of 16-bits, we can jump between $(0 \leq \text{Immediate\_address} \leq (2^{16} - 1))$ address space.

vnop – No Operation

vbnop



Variable values in the following equations are as follows:

This instruction is equivalent to inserting bubble in the design. It performs no useful task. When this instruction is executed, it should not read/write to memory location and should not update register-file contents.