BA22

32-bit RISC Processor

Instruction Set

CAST, Inc.

CONFIDENTIAL

1 Legal Notice

PLEASE READ THE FOLLOWING TERMS AND CONDITIONS CAREFULLY.

1.1 Disclaimer and “Terms of Use”

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH BEYOND SEMICONDUCTOR'S MATERIALS. THE MATERIALS ARE PROVIDED "AS IS" WITHOUT ANY EXPRESS OR IMPLIED WARRANTY OF ANY KIND INCLUDING WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT OF INTELLECTUAL PROPERTY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT SHALL BEYOND SEMICONDUCTOR OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, LOSS OF INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE MATERIALS, EVEN IF BEYOND SEMICONDUCTOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

Beyond Semiconductor and its suppliers further do not warrant the accuracy or completeness of the information, text, graphics, links or other items contained within these materials. Beyond Semiconductor may make changes to these materials, or to the products described therein, at any time without notice. Beyond Semiconductor makes no commitment to update the Materials.

The copyright for any materials created by the Beyond Semiconductor is reserved. Any duplication or use of any material in other electronic or printed publications is not permitted without the Beyond Semiconductor's agreement.

Adherence to all applicable laws and regulations, federal and state and local, governing professional licensing, business practices, advertising and all other aspects of doing business in the US or any other jurisdiction is the sole responsibility of the user.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Beyond Semiconductor reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Any performance information, tests, ratings or specifications provided herein are measured using certain computer systems or components and only indicate the approximate performance of Beyond Semiconductor's products in accordance with those tests. Use of other computer systems or components, software, etc. may affect actual performance. Customers and prospective customers should consult with other, more reliable, sources of information before purchasing Beyond Semiconductor's products.

Proceeding beyond this disclaimer constitutes acceptance of these terms and conditions.

1.2 Trademarks

Microsoft ®, Windows ®, Windows NT ®, Windows Server ® and Windows Vista ™ are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries. Other product names used herein are for identification purposes only and might be trademarks of their respective companies. We disclaim any and all rights to those marks.

2 Table of Contents

1 Legal Notice....2
1.1 Disclaimer and "Terms of Use"....2
1.2 Trademarks....2
2 Table of Contents....3
3 Instructions....7
3.1 b.aadd....7
3.2 b.add....8
3.3 b.addc....9
3.4 b.addci....10
3.5 b.addi....11
3.6 b.addic....12
3.7 b.adds....13
3.8 b.and....14
3.9 b.andi....15
3.10 b.bc....16
3.11 b.beq....17
3.12 b.beqi....18
3.13 b.bf....19
3.14 b.bges....20
3.15 b.bgesi....21
3.16 b.bgeu....22
3.17 b.bgeui....23
3.18 b.bgts....24
3.19 b.bgtsi....25
3.20 b.bgtu....26
3.21 b.bgtui....27
3.22 b.bitrev....28
3.23 b.bles....29
3.24 b.blesi....30
3.25 b.bleu....31
3.26 b.bleui....32
3.27 b.blts....33
3.28 b.bltsi....34
3.29 b.bltu....35
3.30 b.bltui....36
3.31 b.bnc....37
3.32 b.bne....38
3.33 b.bnei....39
3.34 b.bnf....40
3.35 b.bno....41
3.36 b.bo....42

3.37 b.clz....43
3.38 b.cmov....44
3.39 b.cmpxchg....45
3.40 b.cov....46
3.41 b.di....47
3.42 b.div....48
3.43 b.divi....49
3.44 b.divu....50
3.45 b.divui....51
3.46 b.ei....52
3.47 b.entri....53
3.48 b.extbs....54
3.49 b.extbz....55
3.50 b.exths....56
3.51 b.exthz....57
3.52 b.ff1....58
3.53 b.flb....59
3.54 b.j....60
3.55 b.ja....61
3.56 b.jal....62
3.57 b.jalr....63
3.58 b.jma....64
3.59 b.jmal....65
3.60 b.jr....66
3.61 b.lbz....67
3.62 b.ld....68
3.63 b.lhz....69
3.64 b.lma....70
3.65 b.lws....71
3.66 b.lwz....72
3.67 b.mac....73
3.68 b.macs....75
3.69 b.mfspr....77
3.70 b.mlwz....78
3.71 b.mod....79
3.72 b.modu....80
3.73 b.mov....81
3.74 b.movhi....82
3.75 b.movi....83
3.76 b.msw....84
3.77 b.mtspr....85
3.78 b.mul....86
3.79 b.mulh....87
3.80 b.muli....88
3.81 b.nand....89
3.82 b.nop....90

3.83 b.not....91
3.84 b.or....92
3.85 b.ori....93
3.86 b.reti....94
3.87 b.return....95
3.88 b.rfe....96
3.89 b.ror....97
3.90 b.rori....98
3.91 b.rtnei....99
3.92 b.sb....100
3.93 b.sd....101
3.94 b.sfeq....102
3.95 b.sfeqi....103
3.96 b.sfges....104
3.97 b.sfgesi....105
3.98 b.sfgeu....106
3.99 b.sfgeui....107
3.100 b.sfgts....108
3.101 b.sfgtsi....109
3.102 b.sfgtu....110
3.103 b.sfgtui....111
3.104 b.sfles....112
3.105 b.sflesi....113
3.106 b.sfleu....114
3.107 b.sfleui....115
3.108 b.sflts....116
3.109 b.sfltsi....117
3.110 b.sfltu....118
3.111 b.sfltui....119
3.112 b.sfne....120
3.113 b.sfnei....121
3.114 b.sh....122
3.115 b.sll....123
3.116 b.slli....124
3.117 b.sma....125
3.118 b.sra....126
3.119 b.srai....127
3.120 b.srl....128
3.121 b.srli....129
3.122 b.sub....130
3.123 b.subb....131
3.124 b.subs....132
3.125 b.sw....133
3.126 b.swab....134
3.127 b.sys....135
3.128 b.trap....136

3.129 b.xor....137
3.130 b.xori....138
3.131 f.abs.s....139
3.132 f.add.s....140
3.133 f.beq.s....141
3.134 f.bge.s....142
3.135 f.bgt.s....143
3.136 f.ble.s....144
3.137 f.blt.s....145
3.138 f.bne.s....146
3.139 f.div.s....147
3.140 f.ftoi.s....148
3.141 f.itof.s....149
3.142 f.mul.s....150
3.143 f.sub.s....151

3 Instructions

3.1 b.aadd

Atomic Add

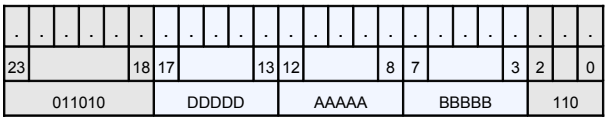
Format:

b.aadd rD,rA,rB

Description:

The value of rA is added to the memory addressd by rB. The result of the sum is also stored in rD. Operation is atomic.

bn.aadd rD,rA,rB



Effect:

```
T = MEM32[rB]
T = T + rA
MEM32[rB] = T
rD = T
```

3.2 b.add

Add Signed

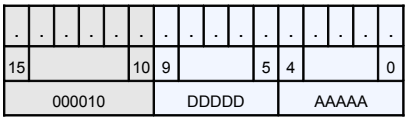
Format:

b.add rD,rA

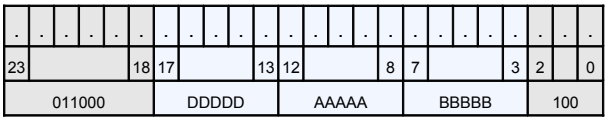
Description:

The sum of the contents of general-purpose registers rA, rD is stored into general-purpose register rD.

bt.add rD,rA



bn.add rD,rA,rB



Effect:

```
rD[31:0] <- rD[31:0] + rA[31:0]
SR[OV] <- overflow
SR[CY] <- carry
```

3.3 b.addc

Add Signed with Carry

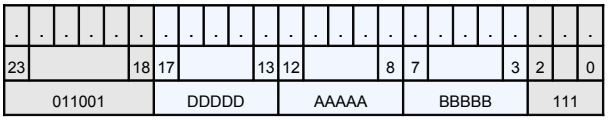
Format:

b.addc rD,rA,rB

Description:

The sum of the contents of general-purpose registers rA, rB and carry SR[CY] is stored into general-purpose register rD.

bn.addc rD,rA,rB



Effect:

```
rD[31:0] <- rA[31:0] + rB[31:0] + SR[CY]
SR[OV] <- overflow
SR[CY] <- carry
```

3.4 b.addci

Add Immediate Signed with Carry

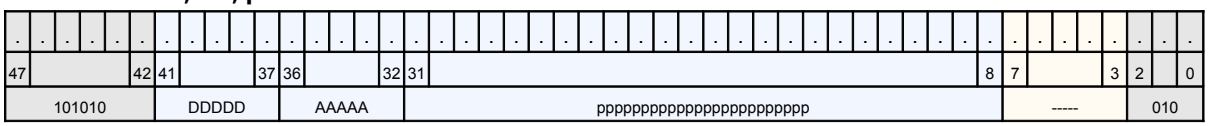
Format:

b.addci rD,rA,I

Description:

The immediate value I is added with carry to the content of general-purpose register rA to form the result, which is then placed into general-purpose register rD.

bw.addci rD,rA,p



Effect:

```
if bw.addci then I = EXTS24(p)
rD[31:0] <- rA[31:0] + I + SR[CY]
SR[OV] <- overflow
SR[CY] <- carry
```

3.5 b.addi

Add Immediate Signed

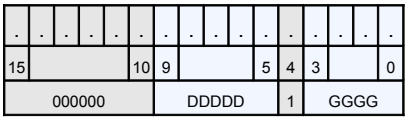
Format:

b.addi rD,rA,I

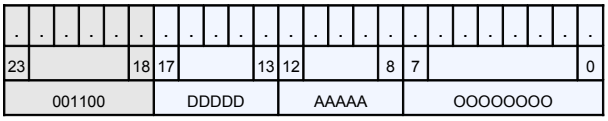
Description:

The immediate value I is added to the content of general-purpose register rA to form the result, which is then placed into general-purpose register rD.

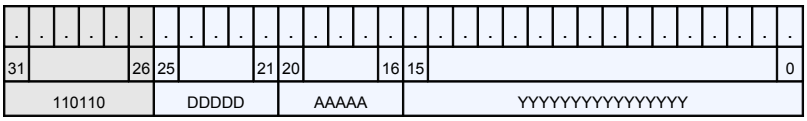
bt.addi rD,G



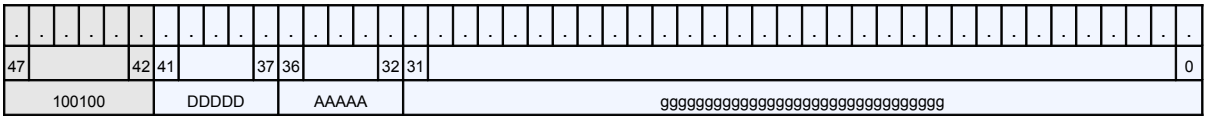
bn.addi rD,rA,O



bg.addi rD,rA,Y



bw.addi rD,rA,g



Effect:

```
if bn.addi then I = EXTS8(O)
if bg.addi then I = EXTS16(Y)
if bw.addi then I = EXTS32(g)
rD[31:0] <- rA[31:0] + I
SR[OV] <- overflow
SR[CY] <- carry
```

3.6 b.addic

Add Immediate Signed with Carry

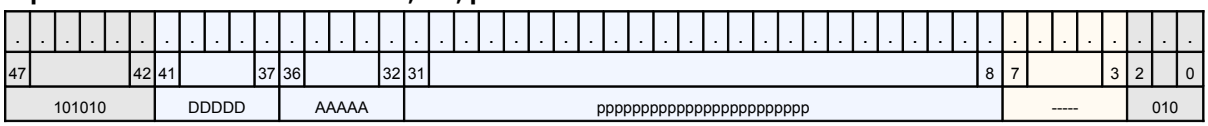
Format:

b.addic rD,rA,I

Description:

The immediate value I is added with carry to the content of general-purpose register rA to form the result, which is then placed into general-purpose register rD.

equivalent to bw.addci rD,rA,p



Effect:

```
if bw.addci then I = EXTS24(p)
rD[31:0] <- rA[31:0] + I + SR[CY] SR[OV] <- overflow SR[CY] <- carry
```

3.7 b.adds

Add Saturated

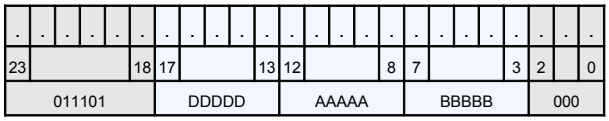
Format:

b.adds rD,rA,rB

Description:

The sum of the contents of general-purpose registers rA and rB is stored into general-purpose register rD. If the result overflows it is clamped to the most positive number INT_MAX (0x7fffffff). If the result underflows it is clamped to the most negative number INT_MIN (0x80000000). The Overflow flag is sticky.

bn.adds rD,rA,rB



Effect:

```
rD[31:0] <- rA[31:0] + rB[31:0]


if (rA + rB >= INT_MAX)
rD <- INT_MAX
else if (rA + rB =< INT_MIN)
rD <- INT_MIN
else
rD <- rA + rB
SR[OV] <- overflow
SR[CY] <- carry
```

3.8 b.and

And

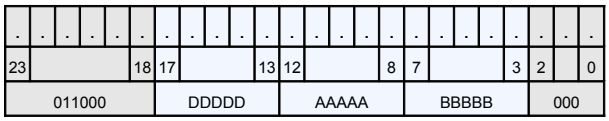
Format:

b.and rD,rA,rB

Description:

The content of general-purpose register rA is bit-wise ANDed with the content of general-purpose register rB. The result is then placed into general-purpose register rD.

bn.and rD,rA,rB



Effect:

rD[31:0] <- rA[31:0] AND rB[31:0]

3.9 b.andi

And with Immediate

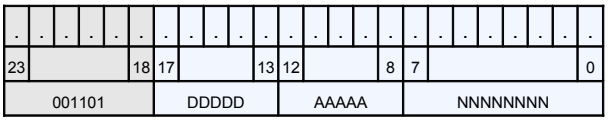
Format:

b.andi rD,rA,I

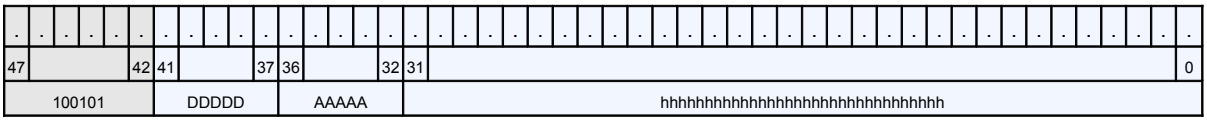
Description:

The immediate value I is shifted left [N << 0, h << 0] bits, sign-extended[h] or zero-extended[N] to program counter width. Then it is bit-wise ANDed with the content of general-purpose register rA. The result is then placed into general-purpose register rD.

bn.andi rD,rA,N



bw.andi rD,rA,h



Effect:

```
if bn.andi then I = EXTZ(N)
if bw.andi then I = EXTS32(h)
rD[31:0] <- rA[31:0] AND I
```

3.10 b.bc

Branch if Carry

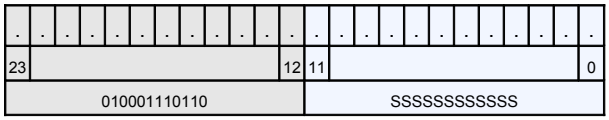
Format:

b.bc I

Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. If the carry flag is set, the program branches to EA.

bn.bc S



Effect:

```
if bn.bc then I = EXTS12(S)
EA <- I + BranchInsnAddr
PC <- EA if SR[CY] set
```

3.11 b.beq

Branch if Equal

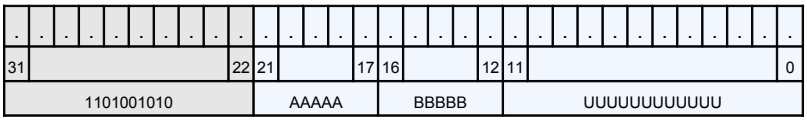
Format:

b.beq rA,rB,I

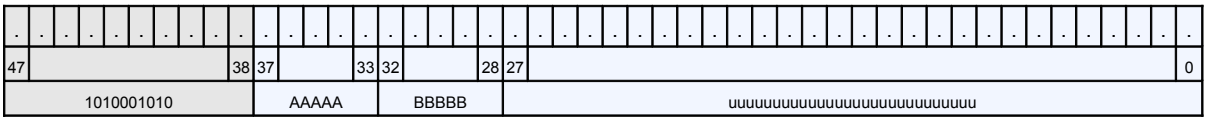
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. The contents of general-purpose register rA and rB are compared. If the two values are equal, the program branches to EA.

bg.beq rA,rB,U



bw.beq rA,rB,u



Effect:

```
if bg.beq then I = EXTS12(U)
if bw.beq then I = EXTS28(u)
EA <- I + BranchInsnAddr
PC <- EA if rA[31:0] == rB[31:0]
```

3.12 b.beqi

Branch if Equal Immediate

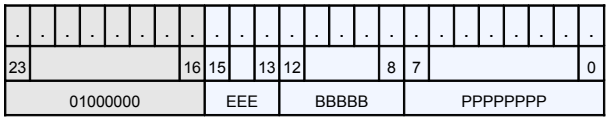
Format:

b.beqi rB,I,J

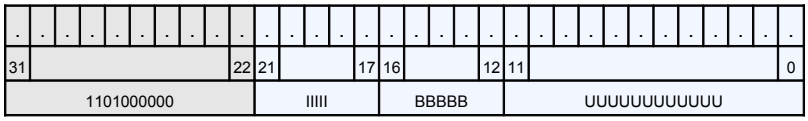
Description:

The immediate value I is shifted left [E << 0, I << 0, I << 0] bits, sign-extended[I, I] or zero-extended[E] to program counter width. Then it is added to the address of the branch instruction. The result is the effective address of the branch. The content of general-purpose register rB and the sign-extended immediate value I are compared. If the two values are equal, the program branches to EA.

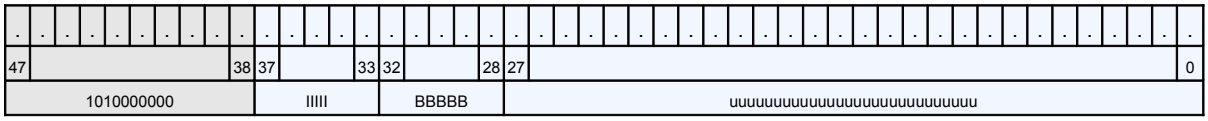
bn.beqi rB,E,P



bg.beqi rB,I,U



bw.beqi rB,I,u



Effect:

```
if bn.beqi then I = EXTZ(E)
if bg.beqi then I = EXTS5(I)
if bw.beqi then I = EXTS5(I)
if bn.beqi then J = EXTS8(P)
if bg.beqi then J = EXTS12(U)
if bw.beqi then J = EXTS28(u)
EA <- J + BranchInsnAddr
PC <- EA if rB[31:0] == I
```

3.13 b.bf

Branch if Flag

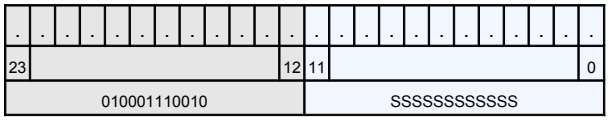
Format:

b.bf I

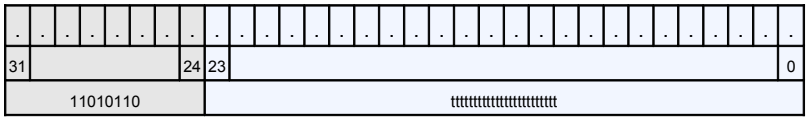
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. If the flag is set, the program branches to EA.

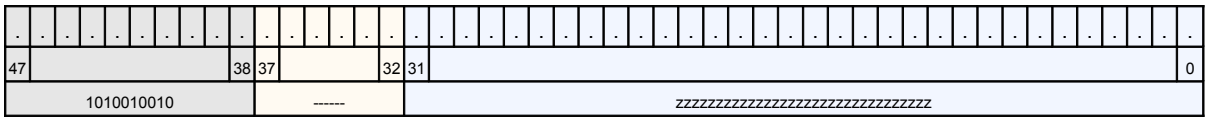
bn.bf S



bg.bf t



bw.bf z



Effect:

```
if bn.bf then I = EXTS12(S)
if bg.bf then I = EXTS24(t)
if bw.bf then I = EXTS32(z)
EA <- I + BranchInsnAddr
PC <- EA if SR[F] set
```

3.14 b.bges

Branch if Greater or Equal Than Signed

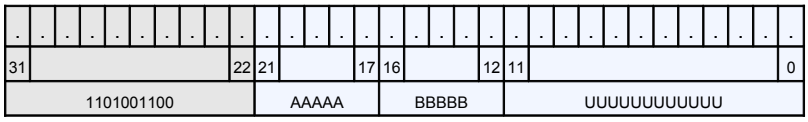
Format:

b.bges rA,rB,I

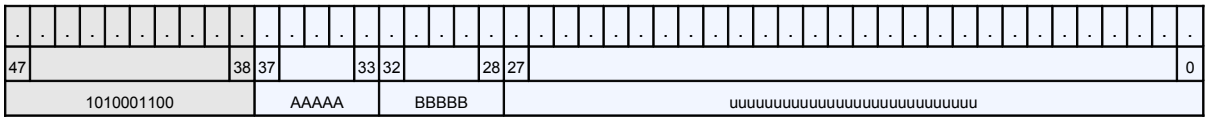
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. The contents of general-purpose register rA and rB are compared as signed integers. If the content of the first register is greater than or equal to the content of the second register, the program branches to EA.

bg.bges rA,rB,U



bw.bges rA,rB,u



Effect:

```
if bg.bges then I = EXTS12(U)
if bw.bges then I = EXTS28(u)
EA <- I + BranchInsnAddr
PC <- EA if rA[31:0] >= rB[31:0]
```

3.15 b.bgesi

Branch if Greater or Equal Immediate Signed

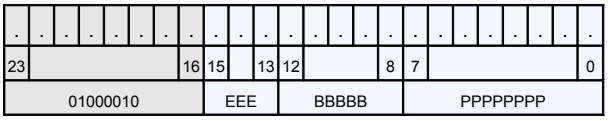
Format:

b.bgesi rB,I,J

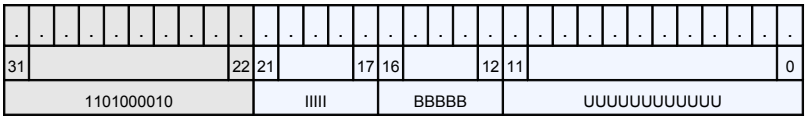
Description:

The immediate value I is shifted left [P << 0, U << 0, u << 0] bits, sign-extended[P, U, u] to program counter width. Then it is added to the address of the branch instruction. The result is the effective address of the branch. The content of general-purpose register rB and the sign-extended immediate value E are compared as signed integers. If the content of the register is greater than or equal to the immediate value E the program branches to EA.

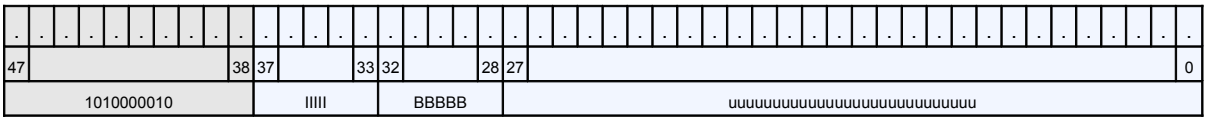
bn.bgesi rB,E,P



bg.bgesi rB,I,U



bw.bgesi rB,I,u



Effect:

```
if bn.bgesi then I = EXTZ(E)
if bg.bgesi then I = EXTS5(I)
if bw.bgesi then I = EXTS5(I)
if bn.bgesi then J = EXTS8(P)
if bg.bgesi then J = EXTS12(U)
if bw.bgesi then J = EXTS28(u)
EA <- J + BranchInsnAddr
PC <- EA if rB[31:0] >= I
```

3.16 b.bgeu

Branch if Greater or Equal Than Unsigned

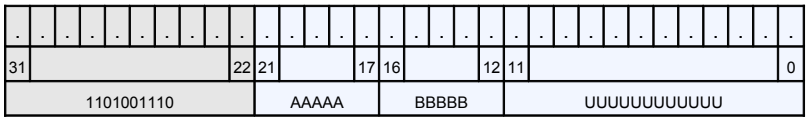
Format:

b.bgeu rA,rB,I

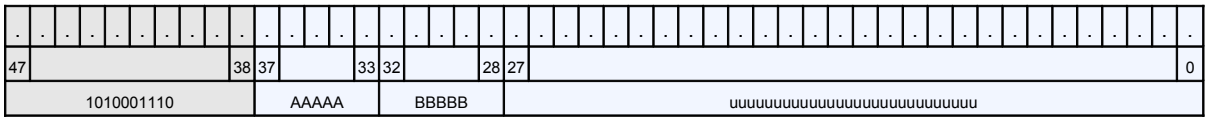
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. The contents of general-purpose register rA and rB are compared as unsigned integers. If the content of the first register is greater than or equal to the content of the second register, the program branches to EA.

bg.bgeu rA,rB,U



bw.bgeu rA,rB,u



Effect:

```
if bg.bgeu then I = EXTS12(U)
if bw.bgeu then I = EXTS28(u)
EA <- I + BranchInsnAddr
PC <- EA if rA[31:0] >= rB[31:0]
```

3.17 b.bgeui

Branch if Greater or Equal Immediate Unsigned

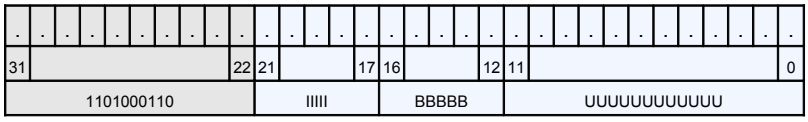
Format:

b.bgeui rB,I,J

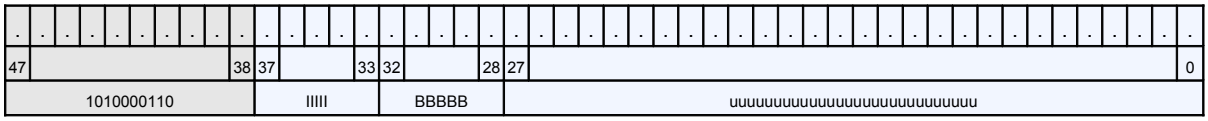
Description:

The immediate value I is shifted left [I << 0, I << 0] bits, sign-extended[I, I] to program counter width. Then it is added to the address of the branch instruction. The result is the effective address of the branch. The content of general-purpose register rB and the sign-extended immediate value I are compared as unsigned integers. If the content of the register is greater than or equal to the immediate value I the program branches to EA.

bg.bgeui rB,I,U



bw.bgeui rB,I,u



Effect:

```
if bg.bgeui then I = EXTS5(I)
if bw.bgeui then I = EXTS5(I)
if bg.bgeui then J = EXTS12(U)
if bw.bgeui then J = EXTS28(u)
EA <- J + BranchInsnAddr
PC <- EA if rB[31:0] >= I
```

3.18 b.bgts

Branch if Greater Than Signed

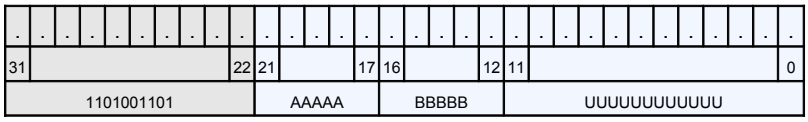
Format:

b.bgts rA,rB,I

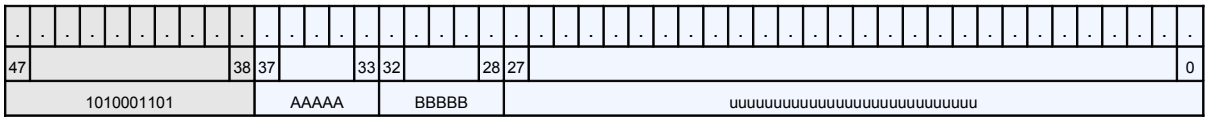
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. The contents of general-purpose register rA and rB are compared as signed integers. If the content of the first register is greater than the content of the second register, the program branches to EA.

bg.bgts rA,rB,U



bw.bgts rA,rB,u



Effect:

```
if bg.bgts then I = EXTS12(U)
if bw.bgts then I = EXTS28(u)
EA <- I + BranchInsnAddr
PC <- EA if rA[31:0] > rB[31:0]
```

3.19 b.bgtsi

Branch if Greater Than Immediate Signed

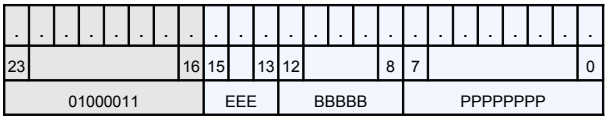
Format:

b.bgtsi rB,I,J

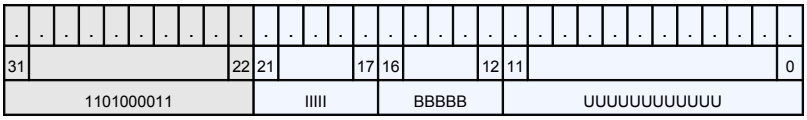
Description:

The immediate value I is shifted left [E << 0, I << 0, I << 0] bits, sign-extended[I, I] or zero-extended[E] to program counter width. Then it is added to the address of the branch instruction. The result is the effective address of the branch. The content of general-purpose register rB and the sign-extended immediate value I are compared as signed integers. If the content of the register is greater than the immediate value I the program branches to EA.

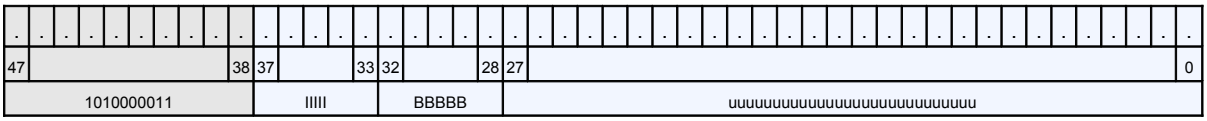
bn.bgtsi rB,E,P



bg.bgtsi rB,I,U



bw.bgtsi rB,I,u



Effect:

```
if bn.bgtsi then I = EXTZ(E)
if bg.bgtsi then I = EXTS5(I)
if bw.bgtsi then I = EXTS5(I)
if bn.bgtsi then J = EXTS8(P)
if bg.bgtsi then J = EXTS12(U)
if bw.bgtsi then J = EXTS28(u)
EA <- J + BranchInsnAddr
PC <- EA if rB[31:0] > I
```

3.20 b.bgtu

Branch if Greater Than Unsigned

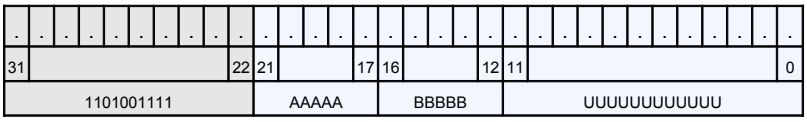
Format:

b.bgtu rA,rB,I

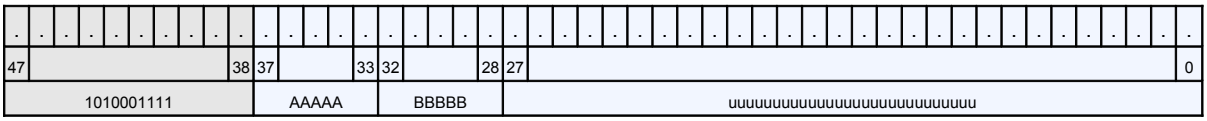
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. The contents of general-purpose register rA and rB are compared as unsigned integers. If the content of the first register is greater than the content of the second register, the program branches to EA.

bg.bgtu rA,rB,U



bw.bgtu rA,rB,u



Effect:

```
if bg.bgtu then I = EXTS12(U)
if bw.bgtu then I = EXTS28(u)
EA <- I + BranchInsnAddr
PC <- EA if rA[31:0] > rB[31:0]
```

3.21 b.bgtui

Branch if Greater Than Immediate Unsigned

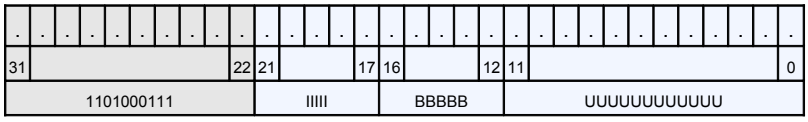
Format:

b.bgtui rB,I,J

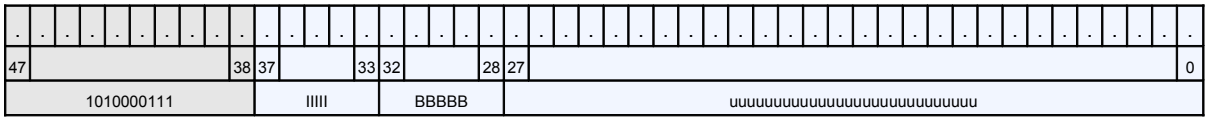
Description:

The immediate value I is shifted left [I << 0, I << 0] bits, sign-extended[I, I] to program counter width. Then it is added to the address of the branch instruction. The result is the effective address of the branch. The content of general-purpose register rB and the sign-extended immediate value I are compared as unsigned integers. If the content of the register is greater than the immediate value I the program branches to EA.

bg.bgtui rB,I,U



bw.bgtui rB,I,u



Effect:

```
if bg.bgtui then I = EXTS5(I)
if bw.bgtui then I = EXTS5(I)
if bg.bgtui then J = EXTS12(U)
if bw.bgtui then J = EXTS28(u)
EA <- J + BranchInsnAddr
PC <- EA if rB[31:0] > I
```

3.22 b.bitrev

Bit Reverse

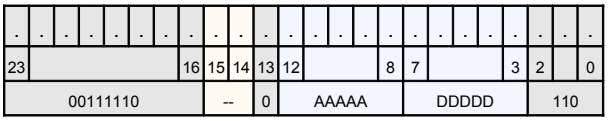
Format:

b.bitrev rD,rA

Description:

The content of general purpose register rA is bit-reversed and then placed into general-purpose register rD.

bn.bitrev rD,rA



Effect:

rD[31:0] <- rA[0:31]

3.23 b.bles

Branch if Less or Equal Than Signed

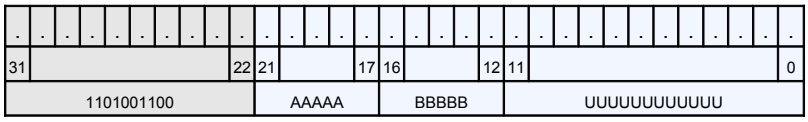
Format:

b.bles rA,rB,I

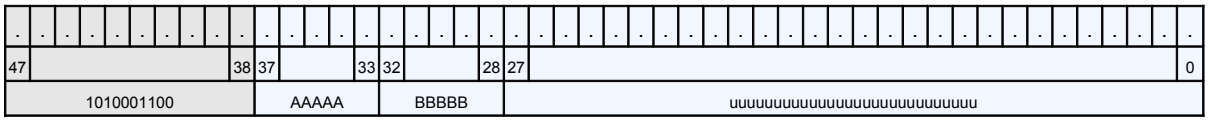
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. The contents of general-purpose register rA and rB are compared as signed integers. If the content of the first register is less than or equal to the content of the second register, the program branches to EA.

equivalent to bg.bges rA,rB,U



equivalent to bw.bges rA,rB,u



Effect:

```
if bg.bges then I = EXTS12(U)
if bw.bges then I = EXTS28(u)
EA <- I + BranchInsnAddr
PC <- EA if rA[31:0] <= rB[31:0]
```

3.24 b.blesi

Branch if Less or Equal Than Immediate Signed

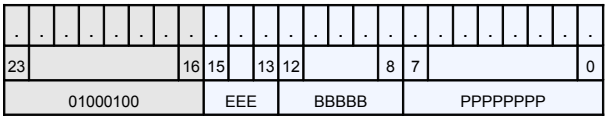
Format:

b.blesi rB,I,J

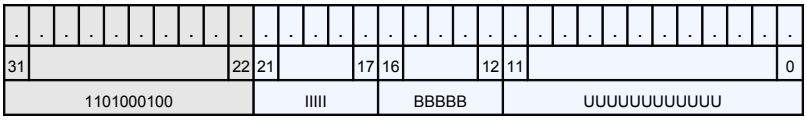
Description:

The immediate value I is shifted left [E << 0, I << 0, I << 0] bits, sign-extended[I, I] or zero-extended[E] to program counter width. Then it is added to the address of the branch instruction. The result is the effective address of the branch. The content of general-purpose register rB and the sign-extended immediate value I are compared as signed integers. If the content of the register is less than or equal to the immediate value I the program branches to EA.

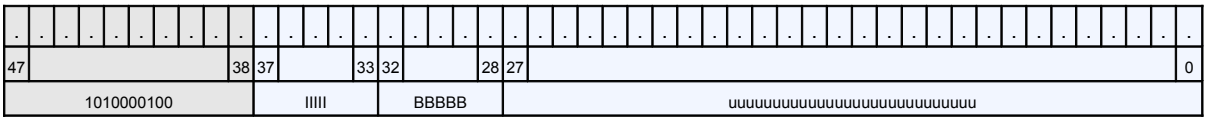
bn.blesi rB,E,P



bg.blesi rB,I,U



bw.blesi rB,I,u



Effect:

```
if bn.blesi then I = EXTZ(E)
if bg.blesi then I = EXTS5(I)
if bw.blesi then I = EXTS5(I)
if bn.blesi then J = EXTS8(P)
if bg.blesi then J = EXTS12(U)
if bw.blesi then J = EXTS28(u)
EA <- J + BranchInsnAddr
PC <- EA if rB[31:0] <= I
```

3.25 b.bleu

Branch if Less or Equal Than Unsigned

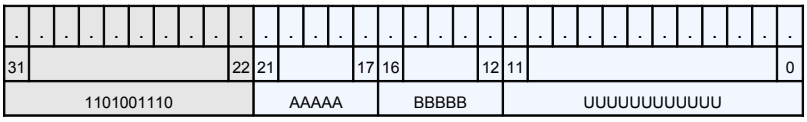
Format:

b.bleu rA,rB,I

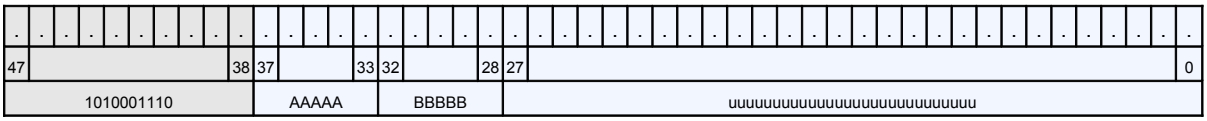
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. The contents of general-purpose register rA and rB are compared as unsigned integers. If the content of the first register is less than or equal to the content of the second register, the program branches to EA.

equivalent to bg.bgeu rA,rB,U



equivalent to bw.bgeu rA,rB,u



Effect:

```
if bg.bgeu then I = EXTS12(U)
if bw.bgeu then I = EXTS28(u)
EA <- I + BranchInsnAddr
PC <- EA if rA[31:0] <= rB[31:0]
```

3.26 b.bleui

Branch if Less or Equal Than Immediate Unsigned

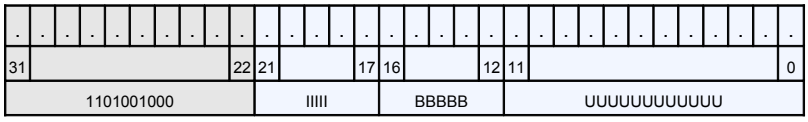
Format:

b.bleui rB,I,J

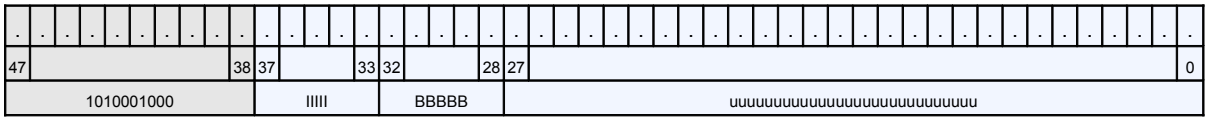
Description:

The immediate value I is shifted left [I << 0, I << 0] bits, sign-extended[I, I] to program counter width. Then it is added to the address of the branch instruction. The result is the effective address of the branch. The content of general-purpose register rB and the sign-extended immediate value I are compared as unsigned integers. If the content of the register is less than or equal to the immediate value I the program branches to EA.

bg.bleui rB,I,U



bw.bleui rB,I,u



Effect:

```
if bg.bleui then I = EXTS5(I)
if bw.bleui then I = EXTS5(I)
if bg.bleui then J = EXTS12(U)
if bw.bleui then J = EXTS28(u)
EA <- J + BranchInsnAddr
PC <- EA if rB[31:0] <= I
```

3.27 b.blts

Branch if Less Than Signed

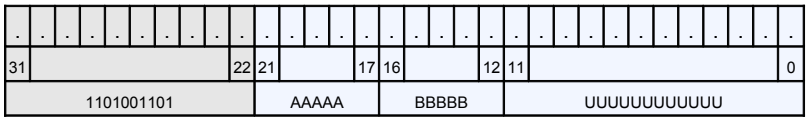
Format:

b.blts rA,rB,I

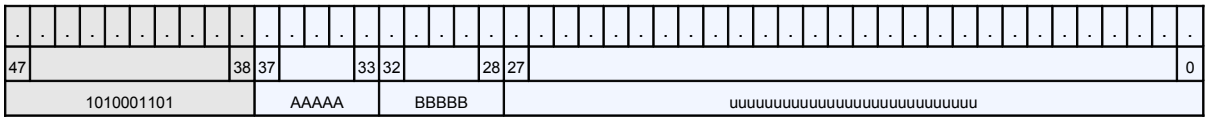
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. The contents of general-purpose register rA and rB are compared as signed integers. If the content of the first register is less than the content of the second register, the program branches to EA.

equivalent to bg.bgts rA,rB,U



equivalent to bw.bgts rA,rB,u



Effect:

```
if bg.bgts then I = EXTS12(U)
if bw.bgts then I = EXTS28(u)
EA <- I + BranchInsnAddr
PC <- EA if rA[31:0] < rB[31:0]
```

3.28 b.bltsi

Branch if Less Than Immediate Signed

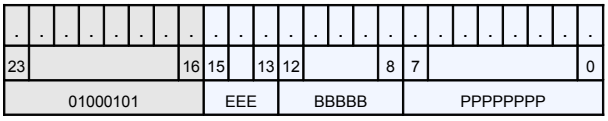
Format:

b.bltsi rB,I,J

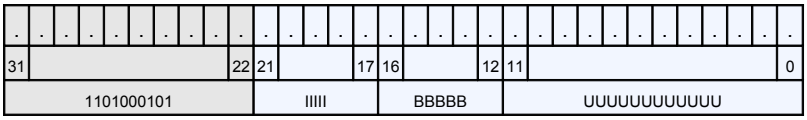
Description:

The immediate value I is shifted left [E << 0, I << 0, I << 0] bits, sign-extended[I, I] or zero-extended[E] to program counter width. Then it is added to the address of the branch instruction. The result is the effective address of the branch. The content of general-purpose register rB and the sign-extended immediate value I are compared as signed integers. If the content of the register is less than the immediate value I the program branches to EA.

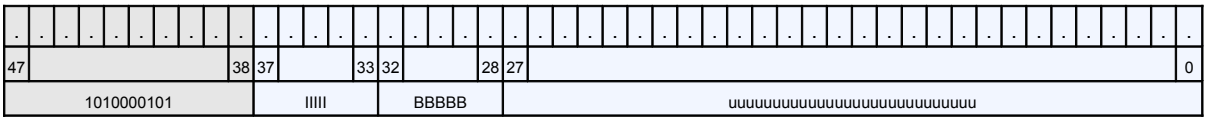
bn.bltsi rB,E,P



bg.bltsi rB,I,U



bw.bltsi rB,I,u



Effect:

```
if bn.bltsi then I = EXTZ(E)
if bg.bltsi then I = EXTS5(I)
if bw.bltsi then I = EXTS5(I)
if bn.bltsi then J = EXTS8(P)
if bg.bltsi then J = EXTS12(U)
if bw.bltsi then J = EXTS28(u)
EA <- J + BranchInsnAddr
PC <- EA if rB[31:0] < I
```

3.29 b.bltu

Branch if Less Than Unsigned

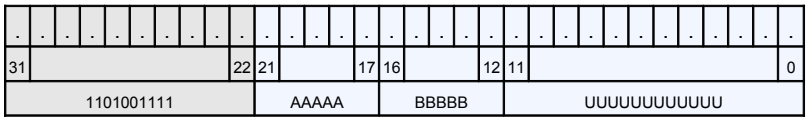
Format:

b.bltu rA,rB,I

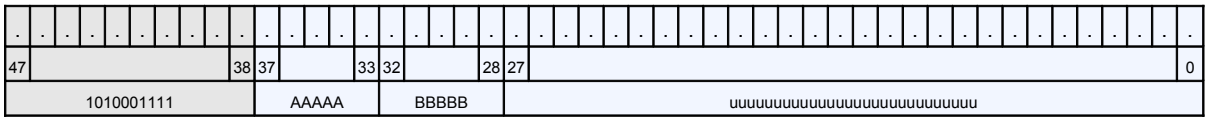
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. The contents of general-purpose register rA and rB are compared as unsigned integers. If the content of the first register is less than the content of the second register, the program branches to EA.

equivalent to bg.bgtu rA,rB,U



equivalent to bw.bgtu rA,rB,u



Effect:

```
if bg.bgtu then I = EXTS12(U)
if bw.bgtu then I = EXTS28(u)
EA <- I + BranchInsnAddr
PC <- EA if rA[31:0] < rB[31:0]
```

3.30 b.bltui

Branch if Less Than Immediate Unsigned

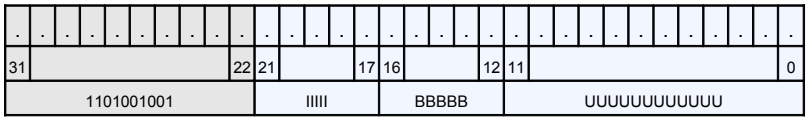
Format:

b.bltui rB,I,J

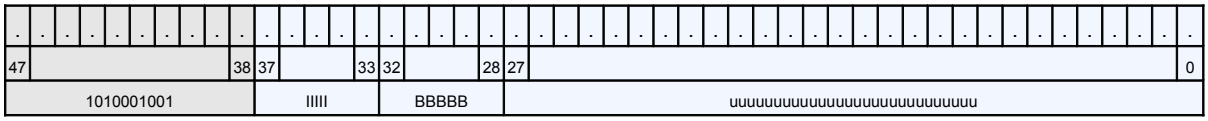
Description:

The immediate value I is shifted left [I << 0, I << 0] bits, sign-extended[I, I] to program counter width. Then it is added to the address of the branch instruction. The result is the effective address of the branch. The content of general-purpose register rB and the sign-extended immediate value I are compared as unsigned integers. If the content of the register is less than the immediate value I the program branches to EA.

bg.bltui rB,I,U



bw.bltui rB,I,u



Effect:

```
if bg.bltui then I = EXTS5(I)
if bw.bltui then I = EXTS5(I)
if bg.bltui then J = EXTS12(U)
if bw.bltui then J = EXTS28(u)
EA <- J + BranchInsnAddr
PC <- EA if rB[31:0] < I
```

3.31 b.bnc

Branch if No Carry

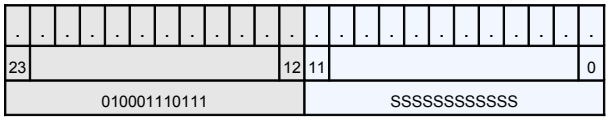
Format:

b.bnc I

Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. If the carry flag is cleared, the program branches to EA.

bn.bnc S



Effect:

```
if bn.bnc then I = EXTS12(S)
EA <- I + BranchInsnAddr
PC <- EA if SR[CY] cleared
```

3.32 b.bne

Branch if Not Equal

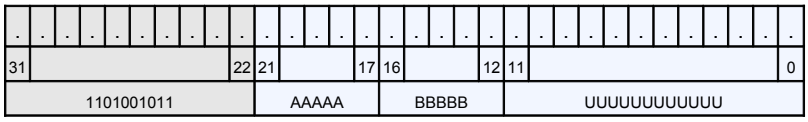
Format:

b.bne rA,rB,I

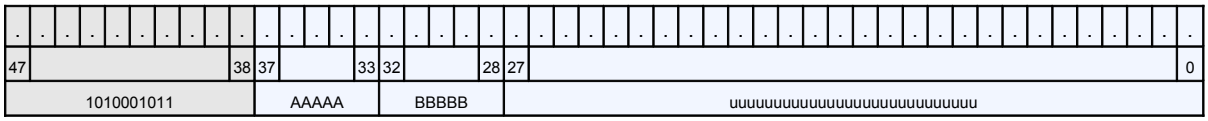
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. The contents of general-purpose register rA and rB are compared. If the two values are not equal, the program branches to EA.

bg.bne rA,rB,U



bw.bne rA,rB,u



Effect:

```
if bg.bne then I = EXTS12(U)
if bw.bne then I = EXTS28(u)
EA <- I + BranchInsnAddr
PC <- EA if rA[31:0] != rB[31:0]
```

3.33 b.bnei

Branch if Not Equal Immediate

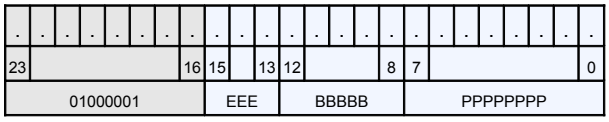
Format:

b.bnei rB,I,J

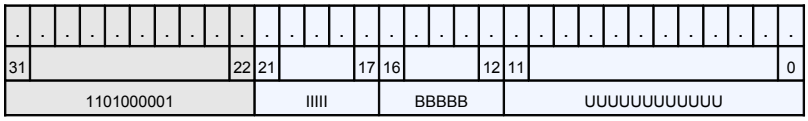
Description:

The immediate value I is shifted left [E << 0, I << 0, I << 0] bits, sign-extended[I, I] or zero-extended[E] to program counter width. Then it is added to the address of the branch instruction. The result is the effective address of the branch. The content of general-purpose register rB and the sign-extended immediate value I are compared. If the two values are not equal, the program branches to EA.

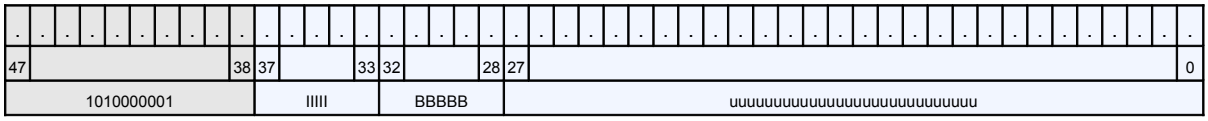
bn.bnei rB,E,P



bg.bnei rB,I,U



bw.bnei rB,I,u



Effect:

```
if bn.bnei then I = EXTZ(E)
if bg.bnei then I = EXTS5(I)
if bw.bnei then I = EXTS5(I)
if bn.bnei then J = EXTS8(P)
if bg.bnei then J = EXTS12(U)
if bw.bnei then J = EXTS28(u)
EA <- J + BranchInsnAddr
PC <- EA if rB[31:0] != I
```

3.34 b.bnf

Branch if No Flag

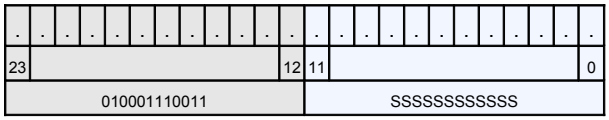
Format:

b.bnf I

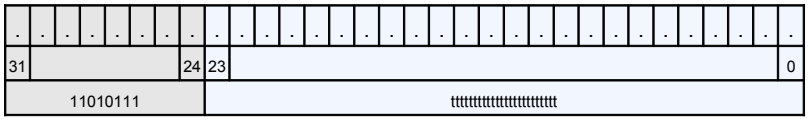
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. If the flag is cleared, the program branches to EA.

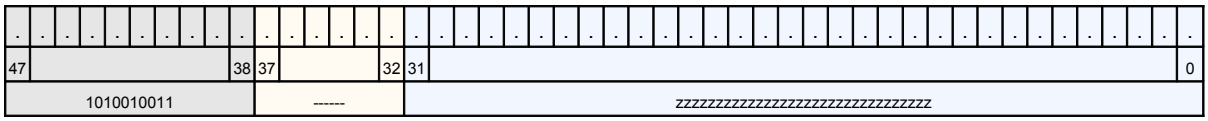
bn.bnf S



bg.bnf t



bw.bnf z



Effect:

```
if bn.bnf then I = EXTS12(S)
if bg.bnf then I = EXTS24(t)
if bw.bnf then I = EXTS32(z)
EA <- I + BranchInsnAddr
PC <- EA if SR[F] cleared
```

3.35 b.bno

Branch if No Overflow

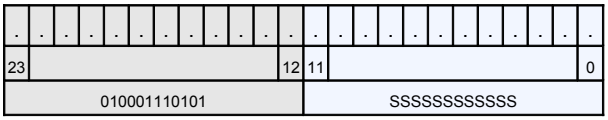
Format:

b.bno I

Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. If the overflow flag is cleared, the program branches to EA.

bn.bno S



Effect:

```
if bn.bno then I = EXTS12(S)
EA <- I + BranchInsnAddr
PC <- EA if SR[OV] cleared
```

3.36 b.bo

Branch if Overflow

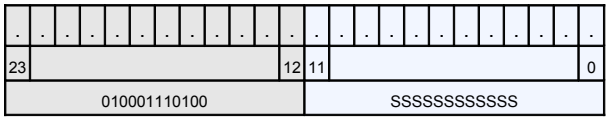
Format:

b.bo I

Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. If the overflow flag is set, the program branches to EA.

bn.bo S



Effect:

```
if bn.bo then I = EXTS12(S)
EA <- I + BranchInsnAddr
PC <- EA if SR[OV] set
```

3.37 b.clz

Count Leading Zeros

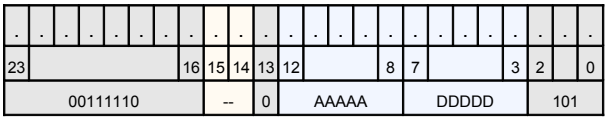
Format:

b.clz rD,rA

Description:

Number of leading zeros in general-purpose register rA is written into general-purpose register rD. The MSB is regarded as bit 32. If no bit is set, -1 is written into rD.

bn.clz rD,rA



Effect:

```
rD[31:0] <- rA[31] ? 0 : rA[30] ? 1 ... rA[0] ? 31 : -1
```

3.38 b.cmov

Conditional Move

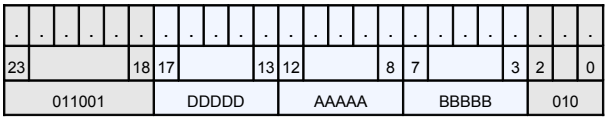
Format:

b.cmov rD,rA,rB

Description:

According to the condition flag SR[F], either rA or rB is selected to be copied to the general-purpose register rD. If flag is set, rA is selected. If flag is cleared, rB is selected.

bn.cmov rD,rA,rB



Effect:

rD[31:0] <- SR[F] ? rA[31:0] : rB[31:0]

3.39 b.cmpxchg

Compare Exchange

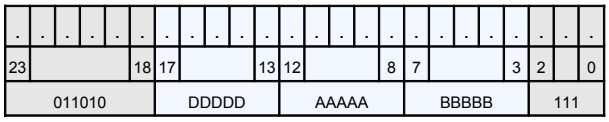
Format:

b.cmpxchg rD,rA,rB

Description:

If the memory addressed by rB is the same as the contents of rD, the value of rA is written to the memory location addressed by rB. The value of the flag reflects wether the instruction updated the memory location. Operation is atomic.

bn.cmpxchg rD,rA,rB



Effect:

```
T = MEM32[rB]
if (T == rD)
flag = 0
T = rA
else
flag = 1
MEM32[rB] = T
rD = T
```

3.40 b.cov

Clear Overflow

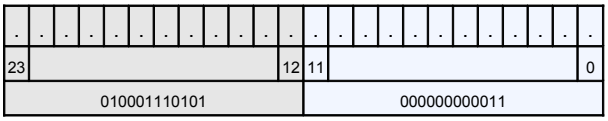
Format:

b.cov n/a

Description:

Clear overflow flag.

equivalent to bn.bno 3



Effect:

EA <- 3 + BranchInsnAddr

PC <- EA if SR[OV] <- 0

3.41 b.di

Disable Interrupts

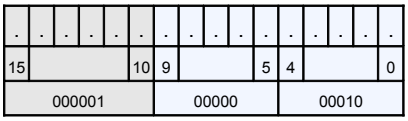
Format:

b.di n/a

Description:

Disables interrupts by setting SR[IEE] and SR[TEE] to '0'.

equivalent to bt.mov r0,r2



Effect:

SR[IEE] <- 0

SR[TEE] <- 0

3.42 b.div

Divide Signed

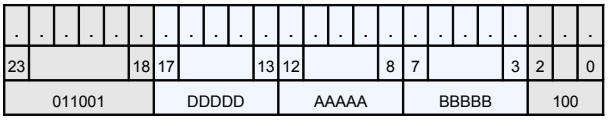
Format:

b.div rD,rA,rB

Description:

The content of general-purpose register rA is divided by the content of general-purpose register rB. The result is then placed into general-purpose register rD. Both operands are treated as signed integers. The carry flag is set when the divisor is zero (if carry SR[CY] is implemented). A corner case exits, where result is undefined. If INT_MIN (0x80000000) is divided with -1 (0xffffffff) the overflow occurs, which is not handled by the processor.

bn.div rD,rA,rB



Effect:

```
rD[31:0] <- rA[31:0] / rB[31:0]
SR[OV] <- overflow
SR[CY] <- carry
```

3.43 b.divi

Divide Signed with Immediate

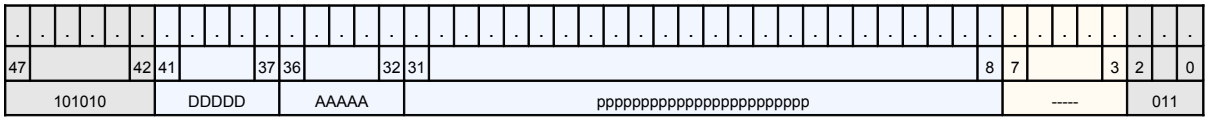
Format:

b.divi rD,rA,I

Description:

The content of general-purpose register rA is divided by the immediate value. The result is then placed into general-purpose register rD. Both operands are treated as signed integers. The carry flag is set when the divisor is zero (if carry SR[CY] is implemented). A corner case exits, where result is undefined. If MAX_INT (0x80000000) is divided with -1 (0xffffff) the overflow occurs, which is not handled by the processor.

bw.divi rD,rA,p



Effect:

```
if bw.divi then I = EXTS24(p)
rD[31:0] <- rA[31:0] / I
SR[OV] <- overflow
SR[CY] <- carry
```

3.44 b.divu

Divide Unsigned

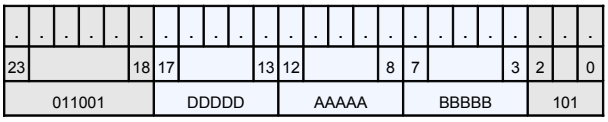
Format:

b.divu rD,rA,rB

Description:

The content of general-purpose register rA is divided by the content of general-purpose register rB. The result is then placed into general-purpose register rD. Both operands are treated as unsigned integers. The carry flag is set when the divisor is zero (if carry SR[CY] is implemented).

bn.divu rD,rA,rB



Effect:

```
rD[31:0] <- rA[31:0] / rB[31:0]
SR[OV] <- overflow
SR[CY] <- carry
```

3.45 b.divui

Divide Unsigned with Immediate

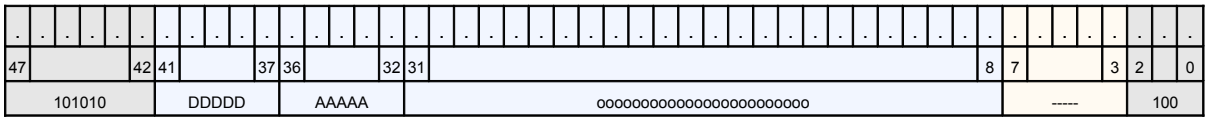
Format:

b.divui rD,rA,I

Description:

The content of general-purpose register rA is divided by the immediate value. The result is then placed into general-purpose register rD. Both operands are treated as unsigned integers. The carry flag is set when the divisor is zero (if carry SR[CY] is implemented).

bw.divui rD,rA,o



Effect:

```
if bw.divui then I = EXTZ(o)
rD[31:0] <- rA[31:0] / I
SR[OV] <- overflow
SR[CY] <- carry
```

3.46 b.ei

Enable Interrupts

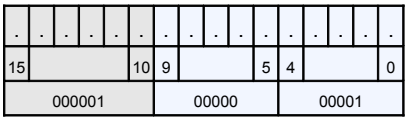
Format:

b.ei n/a

Description:

Enables interrupts by setting SR[IEE] and SR[TEE] to '1'.

equivalent to bt.mov r0,r1



Effect:

SR[IEE] <- 1

SR[TEE] <- 1

3.47 b.entri

Function Prologue Instruction

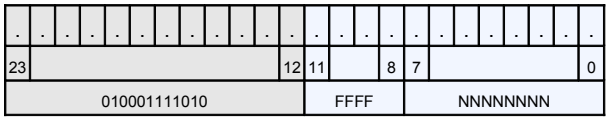
Format:

b.entri I,J

Description:

The stack pointer register r1 is decremented by 4*(I+J). The contents of I consecutive general-purpose registers, beginning with r9, are stored to the stack.

bn.entri I,J



Effect:

```
if bn.entri then I = EXTZ(F)
if bn.entri then J = EXTZ(N)
for (i=0; i<I; i++) {

EA <- -4i + r1[31:0]

(EA)[31:0] <- r(9+i)[31:0]
}
r1[31:0] <- r1[31:0] - 4I - 4J
```

3.48 b.extbs

Extend Byte with Sign

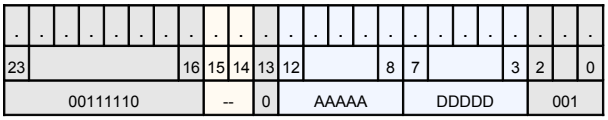
Format:

b.extbs rD,rA

Description:

The low-order 8 bits of general-purpose register rA are sign extended to 32 bits and placed into the general-purpose register rD.

bn.extbs rD,rA



Effect:

```
rD[31:8] <- {rA[7]}
rD[7:0] <- rA[7:0]
```

3.49 b.extbz

Extend Byte with Zero

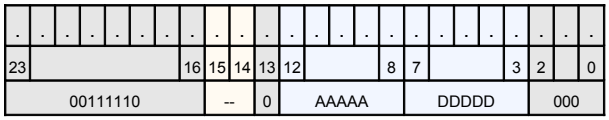
Format:

b.extbz rD,rA

Description:

The low-order 8 bits of general-purpose register rA are zero extended to 32 bits and placed into the general-purpose register rD.

bn.extbz rD,rA



Effect:

```
rD[31:8] <- 0
rD[7:0] <- rA[7:0]
```

3.50 b.exths

Extend Half Word with Sign

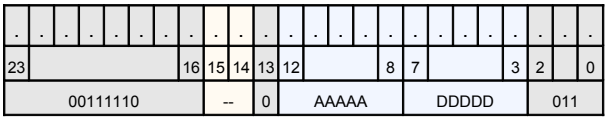
Format:

b.exths rD,rA

Description:

The low-order 16 bits of general-purpose register rA are sign extended to 32 bits and placed into the general-purpose register rD.

bn.exths rD,rA



Effect:

```
rD[31:16] <- {rA[15]}
rD[15:0] <- rA[15:0]
```

3.51 b.exthz

Extend Half Word with Zero

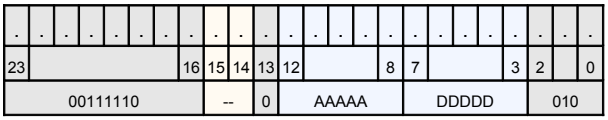
Format:

b.exthz rD,rA

Description:

The low-order 16 bits of general-purpose register rA are zero extended to 32 bits and placed into the general-purpose register rD.

bn.exthz rD,rA



Effect:

```
rD[31:16] <- {0}
rD[15:0] <- rA[15:0]
```

3.52 b.ff1

Find First 1

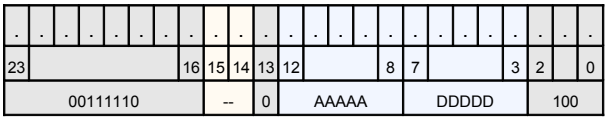
Format:

```
b.ff1 rD,rA
```

Description:

Position of the first set bit in general-purpose register rA is written into general-purpose register rD. The LSB is regarded as bit 0. If no bit is set, zero value is written into rD.

bn.ff1 rD,rA



Effect:

```
rD[31:0] <- rA[0] ? 1 : rA[1] ? 2 ... rA[31] ? 32 : 0
```

3.53 b.flb

Find Leading Byte

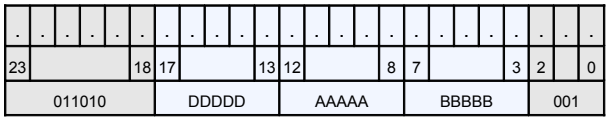
Format:

b.flb rD,rA,rB

Description:

The contents of general-purpose registers rA and rB are compared to find the position of first equal leading byte. The result is then written into general-purpose register rD. Checking for equal bytes starts with MSB (possition of byte 3/32bit 7/64bit). If equal bytes are found in MSB, 3 is written into rD and if equal bytes are found in LSB (possition of byte 0), 0 is written into rD. If there is no equal leading bytes, -1 is written in rD.

bn.flb rD,rA,rB



Effect:

32-bit Processor:

```
rD[31:0] < - (rA[31:24] == rB[31:24]) ? 3 : (rA[23:16] == rB[23:16]) ?
2 ... (rA[7:0] == rB[7:0]) ? 0 : -1
```

64-bit Processor:

```
rD[63:0] < - (rA[63:56] == rB[63:56]) ? 7 : (rA[55:48] == rB[55:48]) ?
6 ... (rA[7:0] == rB[7:0]) ? 0 : -1
```

3.54 b.j

Jump

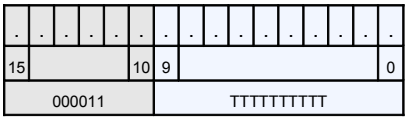
Format:

b.j I

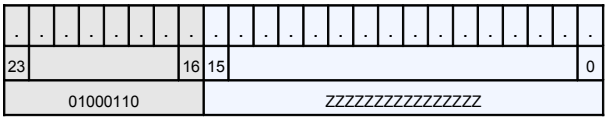
Description:

The immediate value I is added to the address of the jump instruction. The result is the effective address of the jump. The CPU unconditionally continues execution at the computed effective address of the jump.

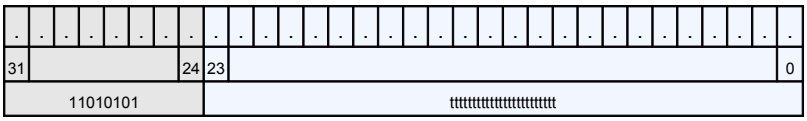
bt.j T



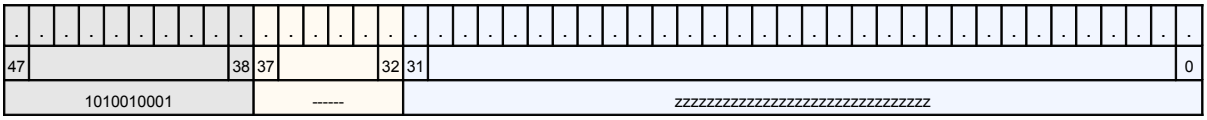
bn.j Z



bg.j t



bw.j z



Effect:

```
if bt.j then I = EXTS10(T)
if bn.j then I = EXTS16(Z)
if bg.j then I = EXTS24(t)
if bw.j then I = EXTS32(z)
PC <- I + JumpInsnAddr
```

3.55 b.ja

Jump Absolute

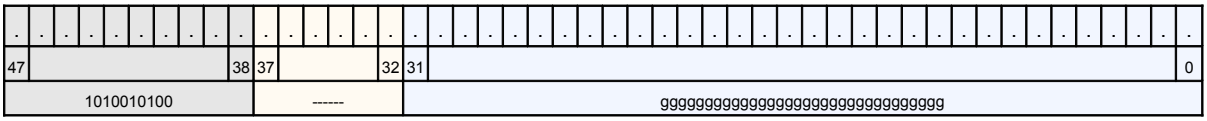
Format:

b.ja I

Description:

The CPU unconditionally continues execution at the effective address specified by the immediate I.

bw.ja g



Effect:

```
if bw.ja then I = EXTS32(g)
PC <- I
```

3.56 b.jal

Jump and Link

Format:

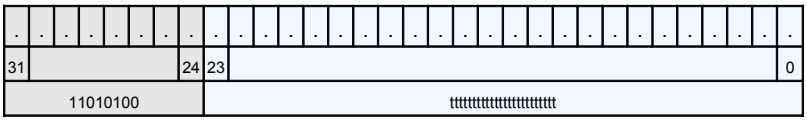
b.jal I

Description:

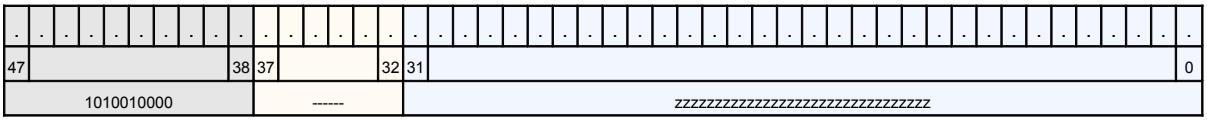
The immediate value I is added to the address of the jump instruction. The result is the effective address of the jump. The CPU unconditionally continues execution at the computed effective address of the jump. The address of the instruction after the jump is placed in the link register.

bn.jal s

bg.jal t



bw.jal z



Effect:

```
if bn.jal then I = EXTS18(s)
if bg.jal then I = EXTS24(t)
if bw.jal then I = EXTS32(z)
PC <- I + JumpInsnAddr
LR <- JumpInsnAddr + JumpInsnLen
```

3.57 b.jalr

Jump and Link Register

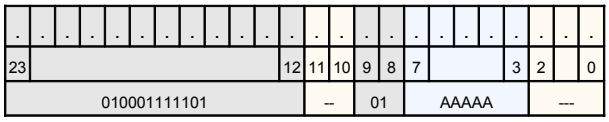
Format:

b.jalr rA

Description:

The content of general-purpose register rA is the effective address of the jump. The CPU unconditionally continues execution at the computed effective address of the jump. The address of the instruction after the jump is placed in the link register.

bn.jalr rA



Effect:

PC <- rA

LR <- JumpInsnAddr + JumpInsnLen

3.58 b.jma

Jump Memory Absolute

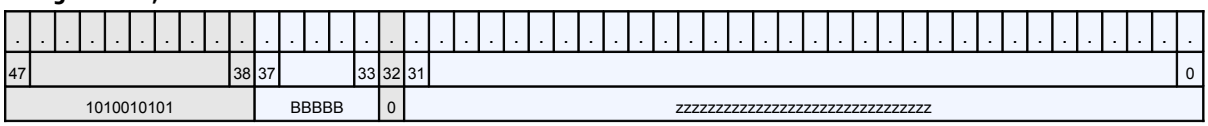
Format:

b.jma rD,I

Description:

The immediate value I is added with the program counter. The result is a pointer to the stored effective address and it is stored into rD. CPU then unconditionally continues execution at the loaded effective address.

bw.jma rD,z



Effect:

```
if bw.jma then I = EXTS32(z)
rD[31:0] <- PC + I
EA <- (PC + I)
PC <- (EA)[31:0]
```

3.59 b.jmal

Jump Memory And Link

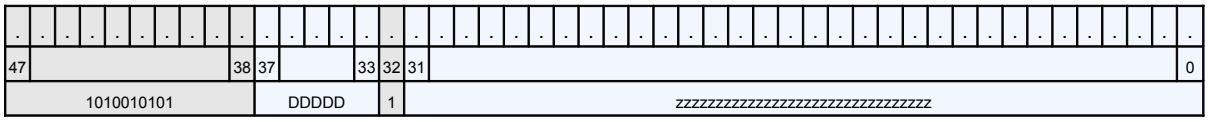
Format:

```
b.jmal rD,I
```

Description:

The immediate value I is added with the program counter. The result is a pointer to the stored effective address and it is stored into rD. CPU then unconditionally continues execution at the loaded effective address. The address of the instruction after the jump is placed in the link register.

bw.jmal rD,z



Effect:

```
if bw.jmal then I = EXTS32(z)
rD[31:0] <- PC + I
EA <- (PC + I)
PC <- (EA)[31:0]
LR <- JumpInsnAddr + JumpInsnLen
```

3.60 b.jr

Jump Register

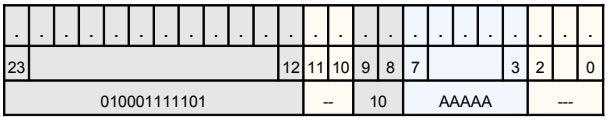
Format:

b.jr rA

Description:

The content of general-purpose register rA is the effective address of the jump. The CPU unconditionally continues execution at the computed effective address of the jump.

bn.jr rA



Effect:

PC <- rA

3.61 b.lbz

Load Byte and Extend with Zero

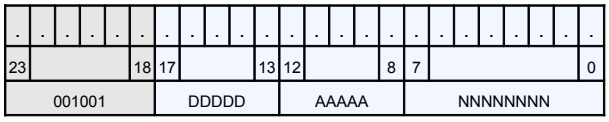
Format:

b.lbz rD,I(rA)

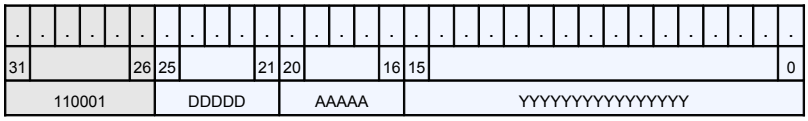
Description:

The immediate value I is shifted left [N << 0, Y << 0, h << 0] bits, sign-extended[Y, h] or zero-extended[N] to program counter width. Then it is added to the content of general-purpose register rA. The sum represents an effective address. The byte in memory addressed by EA is loaded into the low-order eight bits of general-purpose register rD. High-order bits of general-purpose register rD are replaced with zeros.

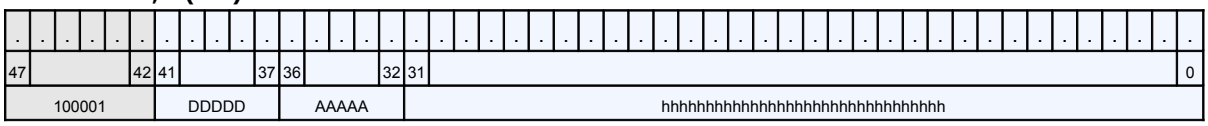
bn.lbz rD,N(rA)



bg.lbz rD,Y(rA)



bw.lbz rD,h(rA)



Effect:

```
if bn.lbz then I = EXTZ(N)
if bg.lbz then I = EXTS16(Y)
if bw.lbz then I = EXTS32(h)
EA <- I + rA[31:0]
rD[7:0] <- (EA)[7:0]
rD[31:8] <- 0
```

3.62 b.ld

Load Double Word

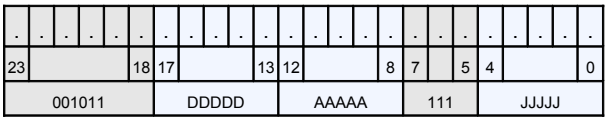
Format:

b.ld rD,I(rA)

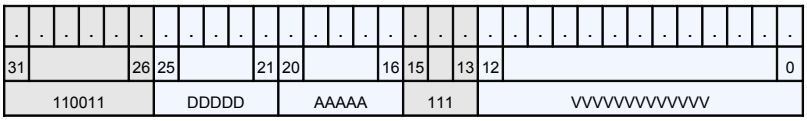
Description:

The immediate value I is shifted left [J << 3, V << 3, v << 3] bits, sign-extended[V, v] or zero-extended[J] to program counter width. Then it is added to the content of general-purpose register rA. The sum represents an effective address. The double word in memory addressed by EA is loaded into general-purpose register rD.

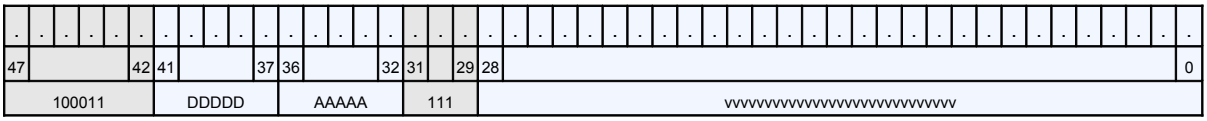
bn.ld rD,J(rA)



bg.ld rD,V(rA)



bw.ld rD,v(rA)



Effect:

```
if bn.ld then I = EXTZ(J << 3)
if bg.ld then I = EXTS16(V << 3)
if bw.ld then I = EXTS32(v << 3)
EA <- I + rA[63:0]
rD[63:0] <- (EA)[63:0]
```

3.63 b.lhz

Load Half Word and Extend with Zero

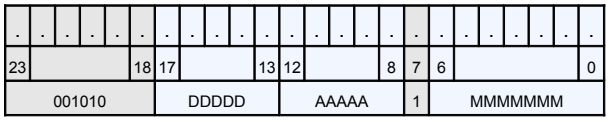
Format:

b.lhz rD,I(rA)

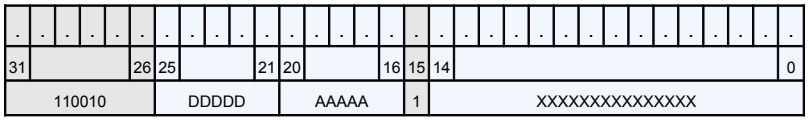
Description:

The immediate value I is shifted left [M << 1, X << 1, i << 1] bits, sign-extended[X, i] or zero-extended[M] to program counter width. Then it is added to the content of general-purpose register rA. The sum represents an effective address. The half word in memory addressed by EA is loaded into the low-order 16 bits of general-purpose register rD. High-order bits of general-purpose register rD are replaced with zeros.

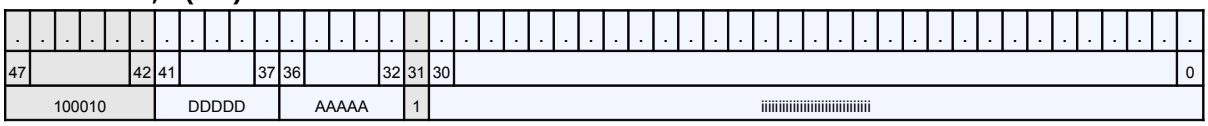
bn.lhz rD,M(rA)



bg.lhz rD,X(rA)



bw.lhz rD,i(rA)



Effect:

```
if bn.lhz then I = EXTZ(M << 1)
if bg.lhz then I = EXTS16(X << 1)
if bw.lhz then I = EXTS32(i << 1)
EA <- I + rA[31:0]
rD[15:0] <- (EA)[15:0]
rD[31:16] <- 0
```

3.64 b.lma

Load Memory Absolute

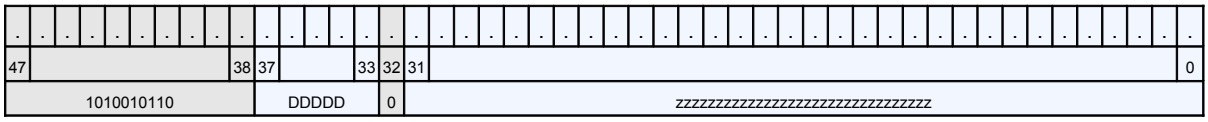
Format:

```
b.lma rD,I
```

Description:

The immediate value I is added with the program counter. The result is a pointer to the stored effective address. The single word in memory addressed by the stored EA is loaded into the low-order 32 bits of general-purpose register rD. In 64-bit implementations the high-order bits of general-purpose register rD are replaced with zeros.

bw.lma rD,z



Effect:

```
if bw.lma then I = EXTS32(z)
T <- (PC + I)
EA <- (T)[31:0]
rD <- (EA)[31:0]
```

3.65 b.lws

Load Single Word and Extend with Sign

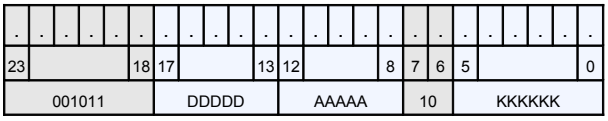
Format:

b.lws rD,I(rA)

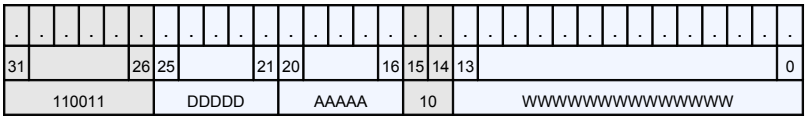
Description:

The immediate value I is shifted left [K << 2, W << 2, w << 2] bits, sign-extended[W, w] or zero-extended[K] to program counter width. Then it is added to the content of general-purpose register rA. The sum represents an effective address. The single word in memory addressed by EA is loaded into the low-order 32 bits of general-purpose register rD. In 64-bit implementations the high-order bits of general-purpose register rD are replaced with bit 31 of the loaded value.

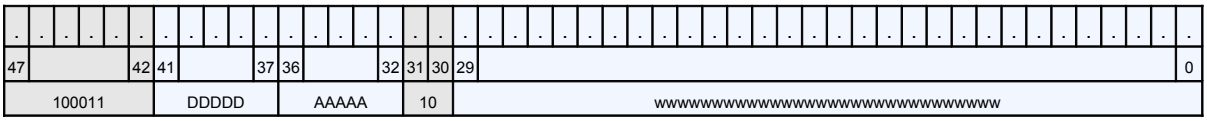
bn.lws rD,K(rA)



bg.lws rD,W(rA)



bw.lws rD,w(rA)



Effect:

```
if bn.lws then I = EXTZ(K << 2)
if bg.lws then I = EXTS16(W << 2)
if bw.lws then I = EXTS32(w << 2)
EA <- I + rA[31:0]
rD[31:0] <- (EA)[31:0]
```

3.66 b.lwz

Load Single Word and Extend with Zero

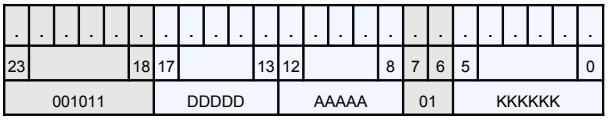
Format:

b.lwz rD,I(rA)

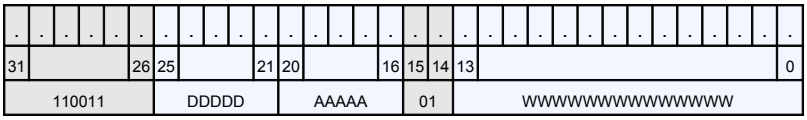
Description:

The immediate value I is shifted left [K << 2, W << 2, w << 2] bits, sign-extended[W, w] or zero-extended[K] to program counter width. Then it is added to the content of general-purpose register rA. The sum represents an effective address. The single word in memory addressed by EA is loaded into the low-order 32 bits of general-purpose register rD. In 64-bit implementations the high-order bits of general-purpose register rD are replaced with zeros.

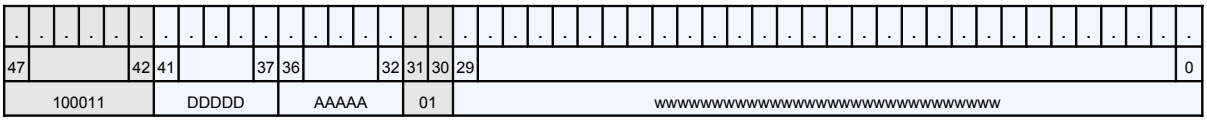
bn.lwz rD,K(rA)



bg.lwz rD,W(rA)



bw.lwz rD,w(rA)



Effect:

```
if bn.lwz then I = EXTZ(K << 2)
if bg.lwz then I = EXTS16(W << 2)
if bw.lwz then I = EXTS32(w << 2)
EA <- I + rA[31:0]
rD[31:0] <- (EA)[31:0]
```

3.67 b.mac

Multiply Signed and Accumulate

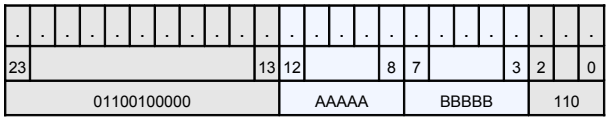
Format:

b.mac rA,rB

Description:

The content of general-purpose register rA and rB are multiplied, and then added to the special-purpose registers MACHI and MACLO, which together form a single 64-bit register MAC. UR[MACOVH] is set when the 64 bit results overflow and UR[MACOVL] is set when the results underflow. The flags are sticky. All operands are treated as signed integers.

bn.mac rA,rB



Effect:

```
mac[63:32] <- MACHI[31:0]
mac[31:0] <- MACLO[31:0]

temp[63:0] <- rA[31:0] * rB[31:0]
mac[63:0] <- temp[63:0] + mac[63:0]

MACHI[31:0] <- mac[63:32]
MACLO[31:0] <- mac[31:0]

if (temp + mac >= INT_MAX)
MACHI[31:0] <- mac[63:32]
MACLO[31:0] <- mac[31:0]
UR[MACOVH] <- 0x00000001
else if (temp + mac =< INT_MIN)
MACHI[31:0] <- mac[63:32]
MACLO[31:0] <- mac[31:0]
```

```
UR[MACOVL] <- 0x00000001
else
MACHI[31:0] <- mac[63:32]
MACLO[31:0] <- mac[31:0]

UR[MACCY] <- carry
```

3.68 b.macs

Multiply and Accumulate Saturated

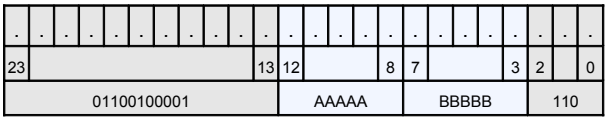
Format:

b.macs rA,rB

Description:

The content of general-purpose register rA and rB are multiplied, and then added to the special-purpose registers MACHI and MACLO, which together form a single 64-bit register MAC. When the addition overflows MAC is clamped to the most positive number INT_MAX (0x7fffffffffffffff). When the result underflows MAC is clamped to the most negative number INT_MIN (0x8000000000000000). UR[MACOVH] is set when the 64 bit results overflow and UR[MACOVL] is set when the results underflow. The flags are sticky. All operands are treated as signed integers.

bn.macs rA,rB



Effect:

```
mac[63:32] <- MACHI[31:0]
mac[31:0] <- MACLO[31:0]

temp[63:0] <- rA[31:0] * rB[31:0]
mac[63:0] <- temp[63:0] + mac[63:0]

MACHI[31:0] <- mac[63:32]
MACLO[31:0] <- mac[31:0]

if (temp + mac >= INT_MAX)
MACHI[31:0] <- 0x7fffffff
MACLO[31:0] <- 0xffffffff
mac[63:0] <- 0x7fffffffffffffff
UR[MACOVH] <- 0x00000001
else if (temp + mac =< INT_MIN)
MACHI[31:0] <- 0x80000000
```

```
MACLO[31:0] <- 0x00000000
mac[63:0] <- 0x8000000000000000
UR[MACOVL] <- 0x00000001
else
MACHI[31:0] <- mac[63:32]
MACLO[31:0] <- mac[31:0]

UR[MACCY] <- carry
```

3.69 b.mfspr

Move From Special-Purpose Register

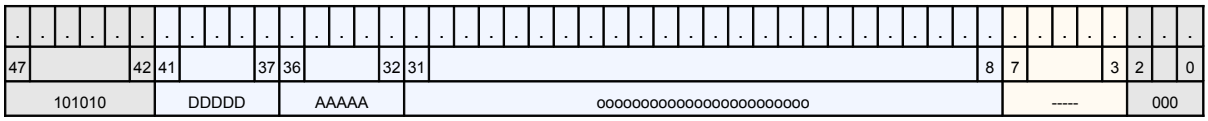
Format:

b.mfspr rD,rA,I

Description:

The content of the special-purpose register, denoted by contents of general-purpose register rA logically ORed with immediate value I, is moved into general-purpose register rD.

bw.mfspr rD,rA,o



Effect:

```
if bw.mfspr then I = EXTZ(o)
rD[31:0] <- spr(rA OR I)
```

3.70 b.mlwz

Multi Load Single Word and Extend with Zero

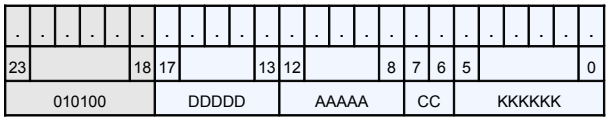
Format:

```
b.mlwz rD,I(rA),J
```

Description:

The immediate value I is shifted left [K << 2] bits, zero-extended[K] to program counter width. The immediate value I is added to the content of general-purpose register rA. The sum represents an effective address. The single word in memory addressed by EA is loaded into the low-order 32 bits of general-purpose register rD. In 64-bit implementations the high-order bits of general-purpose register rD are replaced with zeros. The instruction loads C consecutive single words into C consecutive registers starting with rD. All the actions are repeated for C consecutive single words.

bn.mlwz rD,K(rA),C



Effect:

```
if bn.mlwz then I = EXTZ(K << 2)
if bn.mlwz then J = EXTZ(C)
N={2,3,4,8}
for (i=0; i<N[C]; i++) {


EA <- I + 4i + rA[31:0]


r(D+i)[31:0] <- (EA)[31:0]
}
```

3.71 b.mod

Modulo Signed

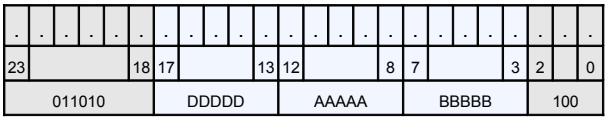
Format:

b.mod rD,rA,rB

Description:

The content of general-purpose register rA and rB are calculated with modulo operation. The result is than placed into general-purpose register rD. Both operands are treated as signed integers.

bn.mod rD,rA,rB



Effect:

rD <- rA[31:0] mod rB[31:0]

3.72 b.modu

Modulo Unsigned

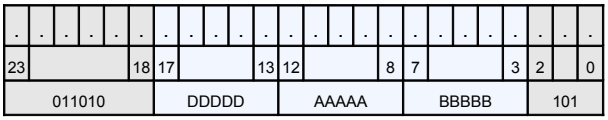
Format:

b.modu rD,rA,rB

Description:

The content of general-purpose register rA and rB are calculated with modulo operation. The result is than placed into general-purpose register rD. Both operands are treated as unsigned integers.

bn.modu rD,rA,rB



Effect:

rD <- rA[31:0] mod rB[31:0]

3.73 b.mov

Move Signed

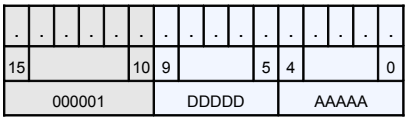
Format:

b.mov rD,rA

Description:

The content of general purpose register rA is copied to general-purpose register rD.

bt.mov rD,rA



Effect:

rD[31:0] <- rA[31:0]

3.74 b.movhi

Move Immediate High

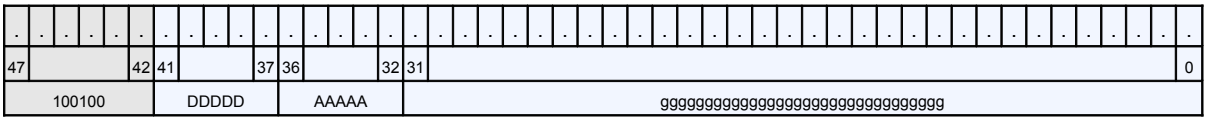
Format:

b.movhi rA,I

Description:

The immediate value I is shifted left by 16 bits, and placed into general-purpose register rA.

equivalent to bw.addi rD,rA,g



Effect:

rA[31:0] <- I << 16

3.75 b.movi

Move Immediate Signed

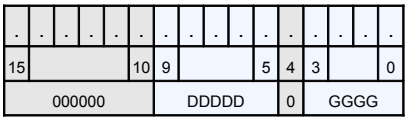
Format:

b.movi rD,I

Description:

Th immediate value I is copied to general-purpose register rD.

bt.movi rD,G



Effect:

```
if bt.movi then I = EXTS4(G)
rD[31:0] <- I
```

3.76 b.msw

Multi Store Single Word and Extend with Zero

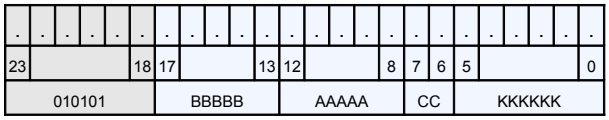
Format:

b.msw I(rA),rB,J

Description:

The immediate value I is shifted left [K << 2] bits, zero-extended[K] to program counter width. The immediate value I is added to the content of general-purpose register rA. The sum represents an effective address. The low-order 32 bits of general-purpose register rB are stored to memory location addressed by EA. The instruction stores C consecutive single words from C consecutive registers starting with rB. All the actions are repeated for C consecutive single words while the value K is.

bn.msw K(rA),rB,C



Effect:

```
if bn.msw then I = EXTZ(K << 2)
if bn.msw then J = EXTZ(C)
N={2,3,4,8}
for (i=0; i<N[C]; i++) {

EA <- I + 4i + rA[31:0]

(EA)[31:0] <- r(I+i)[31:0]
}
```

3.77 b.mtspr

Move To Special-Purpose Register

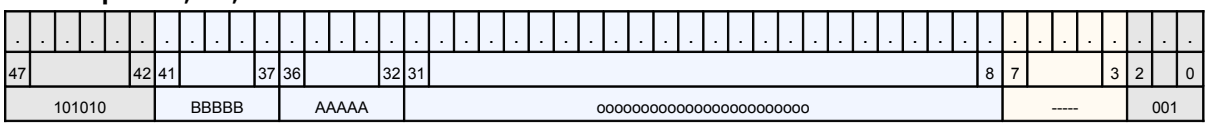
Format:

b.mtspr rA,rB,I

Description:

The content of general-purpose register rB is moved into the special-purpose register denoted by contents of general-purpose register rA logically ORed with the immediate value I.

bw.mtspr rA,rB,o



Effect:

```
if bw.mtspr then I = EXTZ(o)
spr(rA OR I) <- rB[31:0]
```

3.78 b.mul

Multiply Signed

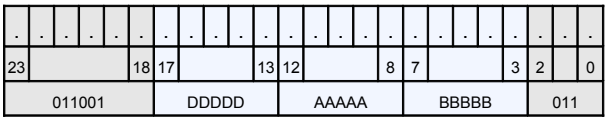
Format:

b.mul rD,rA,rB

Description:

The content of general-purpose register rA and rB are multiplied to form the result, which is then truncated to destination register width and placed into general-purpose register rD. Both operands are treated as signed integers.

bn.mul rD,rA,rB



Effect:

```
rD[31:0] <- rA[31:0] * rB[31:0]
SR[OV] <- overflow
SR[CY] <- carry
```

3.79 b.mulh

Multiply High Signed

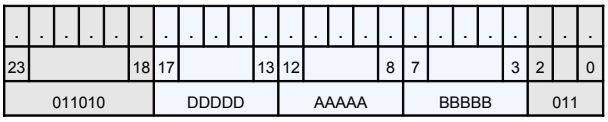
Format:

b.mulh rD,rA,rB

Description:

The content of general-purpose register rA and rB are multiplied to form the 64-bit result. The upper 32 bits of the result are then taken and placed into general-purpose register rD. Both operands are treated as signed integers.

bn.mulh rD,rA,rB



Effect:

```
r64[63:0] <- rA[31:0] * rB[31:0]
rD <- r64[63:32]
```

3.80 b.muli

Multiply Immediate Signed

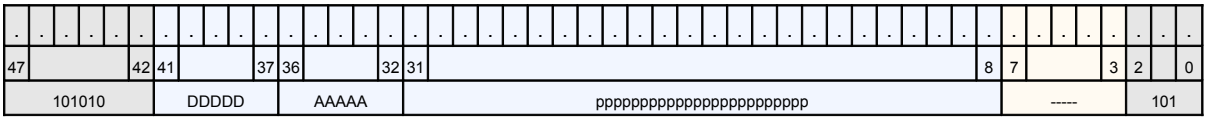
Format:

b.muli rD,rA,I

Description:

The immediate value I and the content of general-purpose register rA are multiplied to form the result, which is then truncated to destination register width and placed into general-purpose register rD.

bw.muli rD,rA,p



Effect:

```
if bw.muli then I = EXTS24(p)
rD[31:0] <- rA[31:0] * I
SR[OV] <- overflow
SR[CY] <- carry
```

3.81 b.nand

Nand

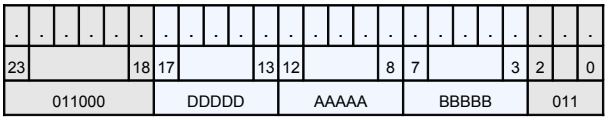
Format:

b.nand rD,rA,rB

Description:

The content of general-purpose register rA is bit-wise NANDed with the content of general-purpose register rB. The result is then placed into general-purpose register rD.

bn.nand rD,rA,rB



Effect:

rD[31:0] <- rA[31:0] NAND rB[31:0]

3.82 b.nop

No Operation

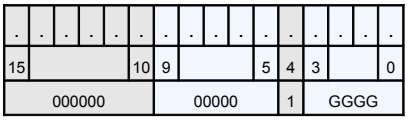
Format:

b.nop I

Description:

This is non operational instruction that it takes at least one clock cycle to complete. The immediate value I can be used for simulation purposes.

equivalent to bt.addi r0,G



Effect:

n/a

3.83 b.not

Not

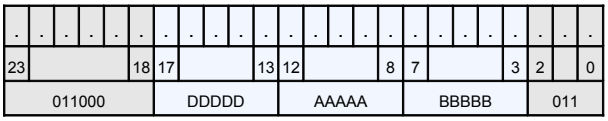
Format:

b.not rD,rA

Description:

The negated value of general-purpose register rA is placed into general-purpose register rD.

equivalent to bn.nand rD,rA,rB



Effect:

rD[31:0] <- NOT rA[31:0]

3.84 b.or

Or

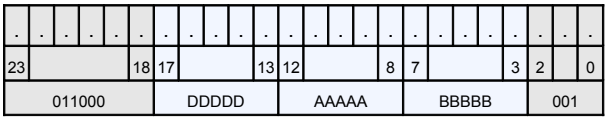
Format:

b.or rD,rA,rB

Description:

The content of general-purpose register rA is bit-wise ORed with the content of general-purpose register rB. The result is then placed into general-purpose register rD.

bn.or rD,rA,rB



Effect:

rD[31:0] <- rA[31:0] OR rB[31:0]

3.85 b.ori

Or with Immediate

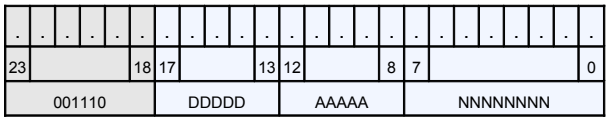
Format:

b.ori rD,rA,I

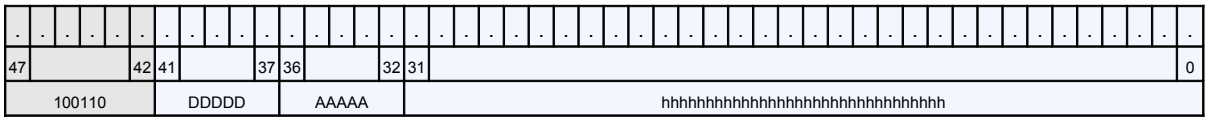
Description:

The immediate value I is shifted left [N << 0, h << 0] bits, sign-extended[h] or zero-extended[N] to program counter width. Then it is bit-wise ORed with the content of general-purpose register rA. The result is then placed into general-purpose register rD.

bn.ori rD,rA,N



bw.ori rD,rA,h



Effect:

```
if bn.ori then I = EXTZ(N)
if bw.ori then I = EXTS32(h)
rD[31:0] <- rA[31:0] OR I
```

3.86 b.reti

Function Epilogue Instruction

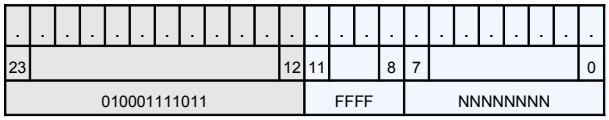
Format:

b.reti I,J

Description:

The contents of I consecutive general-purpose registers, beginning with r9, are restored from the stack. The stack pointer register r1 is incremented by 4*(I+J).

bn.reti I,J



Effect:

```
if bn.reti then I = EXTZ(F)
if bn.reti then J = EXTZ(N)
for (i=0; i<I; i++) {

EA <- 4(I-i-1+J) + r1[31:0]

r(9+i)[31:0] <- (EA)[31:0]
}
r1[31:0] <- r1[31:0] + 4I + 4J
PC <- r9
```

3.87 b.return

Return

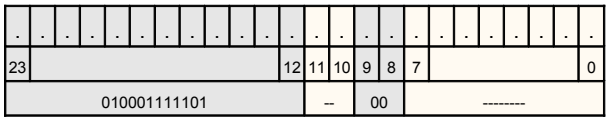
Format:

```
b.return n/a
```

Description:

The CPU unconditionally continues execution at the address pointed to by the link register. The effect is identical to instruction bn.jr r9, however it is better to use instruction b.return, so no software change will be needed if a special purpouse link register is added in the future (binary backwards compatibility).

bn.return



Effect:

```
PC <- LR
```

3.88 b.rfe

Return From Exception

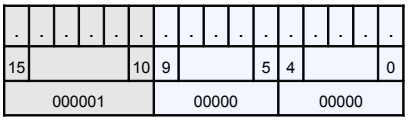
Format:

b.rfe n/a

Description:

Execution of this instruction partially restores the state of the processor prior to the exception.

equivalent to bt.mov r0,r0



Effect:

PC <- EPCR

SR <- ESR

3.89 b.ror

Rotate Right

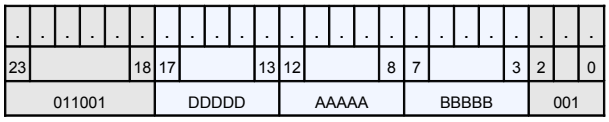
Format:

b.ror rD,rA,rB

Description:

The content of general-purpose register rA is rotated right by the number of bit positions specified in the general-purpose register rB. The result is then placed into general-purpose register rD. In 32 bit implementations 6th bit (bit 5) of rB is ignored.

bn.ror rD,rA,rB



Effect:

```
rD[31-rB[4:0]:0] <- rA[31:rB]
rD[31:32-rB[4:0] ] <- rA[rB[4:0]-1:0]
```

3.90 b.rori

Rotate Right with Immediate

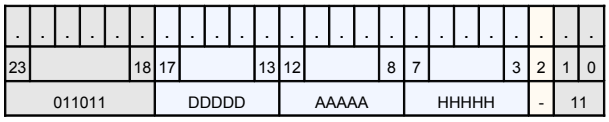
Format:

b.rori rD,rA,I

Description:

The content of general-purpose register rA is rotated right by the number of bit positions specified in the 6 bit immediate value I. The result is then placed into general-purpose register rD. In 32 bit implementations 6th bit (bit 5) of immediate is ignored.

bn.rori rD,rA,H



Effect:

```
if bn.rori then I = EXTZ(H)
rD[31-H:0] <- rA[31:H]
rD[31:32-H] <- rA[H-1:0]
```

3.91 b.rtnei

Function Epilogue Instruction

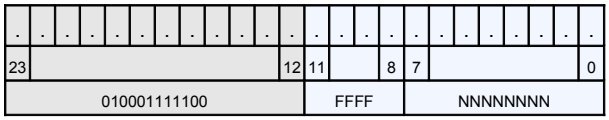
Format:

b.rtnei I,J

Description:

The contents of I consecutive general-purpose registers, beginning with r9, are restored from the stack. The stack pointer register r1 is incremented by 4*(I+J).

bn.rtnei I,J



Effect:

```
if bn.rtnei then I = EXTZ(F)
if bn.rtnei then J = EXTZ(N)
for (i=0; i<I; i++) {

EA <- 4(I-i-1+J) + r1[31:0]

r(9+i)[31:0] <- (EA)[31:0]
}
r1[31:0] <- r1[31:0] + 4I + 4J
```

3.92 b.sb

Store Byte

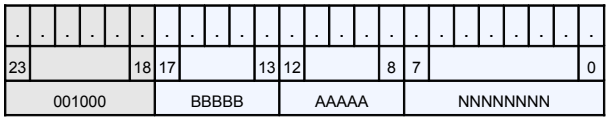
Format:

b.sb I(rA),rB

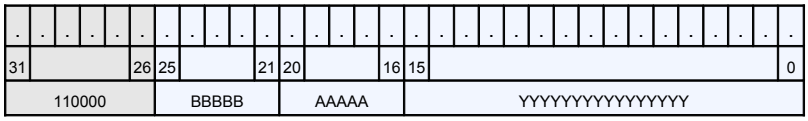
Description:

The immediate value I is shifted left [N << 0, Y << 0, h << 0] bits, sign-extended[Y, h] or zero-extended[N] to program counter width. Then it is added to the content of general-purpose register rA. The sum represents an effective address. The low-order 8 bits of general-purpose register rB are stored to memory location addressed by EA.

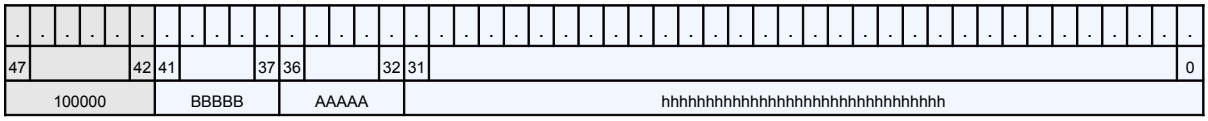
bn.sb N(rA),rB



bg.sb Y(rA),rB



bw.sb h(rA),rB



Effect:

```
if bn.sb then I = EXTZ(N)
if bg.sb then I = EXTS16(Y)
if bw.sb then I = EXTS32(h)
EA <- I + rA[31:0]
(EA)[7:0] <- rB[7:0]
```

3.93 b.sd

Store Double Word

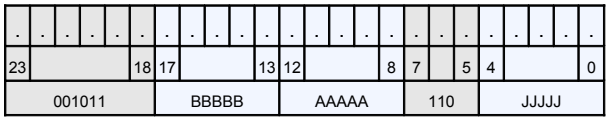
Format:

b.sd I(rA),rB

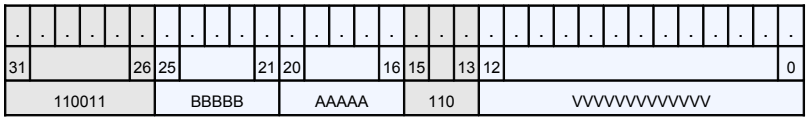
Description:

The immediate value I is shifted left [J << 3, V << 3, v << 3] bits, sign-extended[V, v] or zero-extended[J] to program counter width. Then it is sign-extended and added to the content of general-purpose register rA. The sum represents an effective address. The double word in general-purpose register rB is stored to memory location addressed by EA.

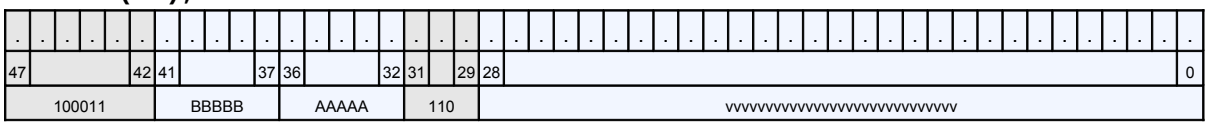
bn.sd J(rA),rB



bg.sd V(rA),rB



bw.sd v(rA),rB



Effect:

```
if bn.sd then I = EXTZ(J << 3)
if bg.sd then I = EXTS16(V << 3)
if bw.sd then I = EXTS32(v << 3)
EA <- I + rA[63:0]
(EA)[63:0] <- rB[63:0]
```

3.94 b.sfeq

Set Flag if Equal

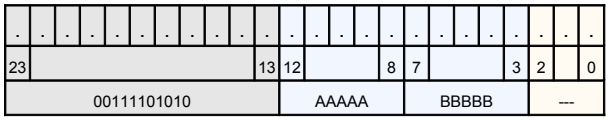
Format:

b.sfeq rA,rB

Description:

The contents of general-purpose registers rA and rB are compared. If the contents are equal, the compare flag f is set; otherwise the compare flag is cleared.

bn.sfeq rA,rB



Effect:

SR[F] <- rA[31:0] == rB[31:0]

3.95 b.sfeqi

Set Flag if Equal Immediate

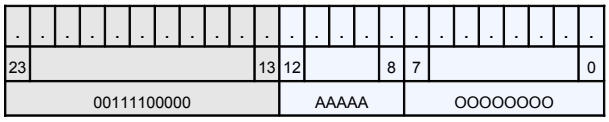
Format:

b.sfeqi rA,I

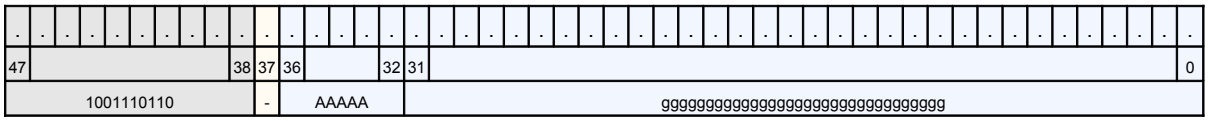
Description:

The immediate value I is shifted left [O << 0, g << 0] bits, sign-extended[O, g] to program counter width. The content of general-purpose register rA and the sign-extended immediate value are compared. If the two values are equal, the compare flag f is set; otherwise the compare flag is cleared.

bn.sfeqi rA,O



bw.sfeqi rA,g



Effect:

```
if bn.sfeqi then I = EXTS8(O)
if bw.sfeqi then I = EXTS32(g)
SR[F] <- rA[31:0] == I
```

3.96 b.sfges

Set Flag if Greater or Equal Than Signed

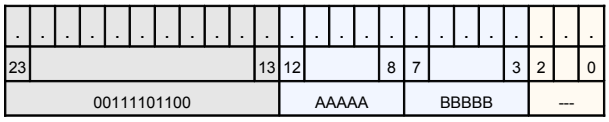
Format:

b.sfges rA,rB

Description:

The contents of general-purpose registers rA and rB are compared as signed integers. If the content of the first register is greater than or equal to the content of the second register, the compare flag f is set; otherwise the compare flag is cleared.

bn.sfges rA,rB



Effect:

SR[F] <- rA[31:0] >= rB[31:0]

3.97 b.sfgesi

Set Flag if Greater or Equal Than Immediate Signed

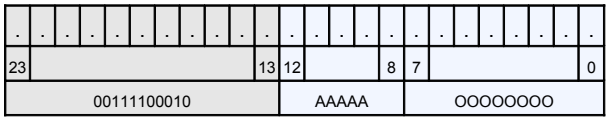
Format:

b.sfgesi rA,I

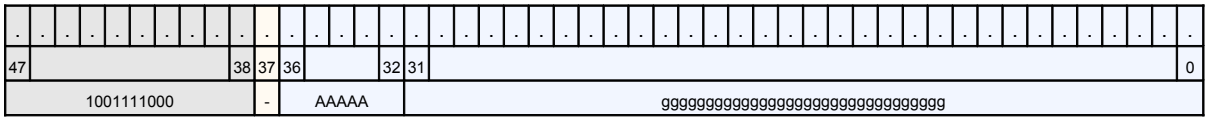
Description:

The immediate value I is shifted left [O << 0, g << 0] bits, sign-extended[O, g] to program counter width. The content of general-purpose register rA and the sign-extended immediate value are compared as signed integers. If the content of the register is greater than or equal to the immediate value the compare flag f is set; otherwise the compare flag is cleared.

bn.sfgesi rA,O



bw.sfgesi rA,g



Effect:

```
if bn.sfgesi then I = EXTS8(O)
if bw.sfgesi then I = EXTS32(g)
SR[F] <- rA[31:0] >= I
```

3.98 b.sfgeu

Set Flag if Greater or Equal Than Unsigned

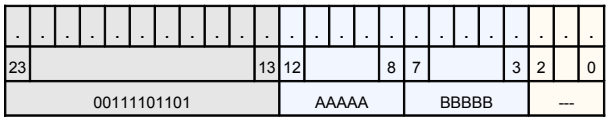
Format:

b.sfgeu rA,rB

Description:

The contents of general-purpose registers rA and rB are compared as unsigned integers. If the content of the first register is greater than or equal to the content of the second register, the compare flag f is set; otherwise the compare flag is cleared.

bn.sfgeu rA,rB



Effect:

SR[F] <- rA[31:0] >= rB[31:0]

3.99 b.sfgeui

Set Flag if Greater or Equal Than Immediate Unsigned

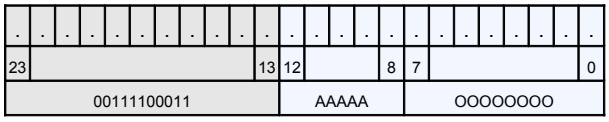
Format:

b.sfgeui rA,I

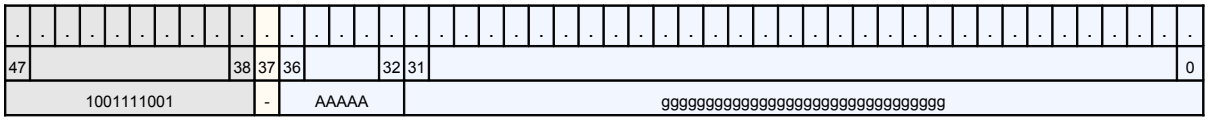
Description:

The immediate value I is shifted left [O << 0, g << 0] bits, sign-extended[O, g] to program counter width. The content of general-purpose register rA and the sign-extended immediate value are compared as unsigned integers. If the content of the register is greater than or equal to the immediate value the compare flag f is set; otherwise the compare flag is cleared.

bn.sfgeui rA,O



bw.sfgeui rA,g



Effect:

```
if bn.sfgeui then I = EXTS8(O)
if bw.sfgeui then I = EXTS32(g)
SR[F] <- rA[31:0] >= I
```

3.100 b.sfgts

Set Flag if Greater Than Signed

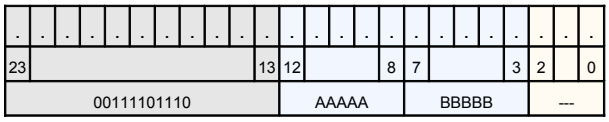
Format:

b.sfgts rA,rB

Description:

The contents of general-purpose registers rA and rB are compared as signed integers. If the content of the first register is greater than the content of the second register, the compare flag f is set; otherwise the compare flag is cleared.

bn.sfgts rA,rB



Effect:

SR[F] <- rA[31:0] > rB[31:0]

3.101 b.sfgtsi

Set Flag if Greater Than Immediate Signed

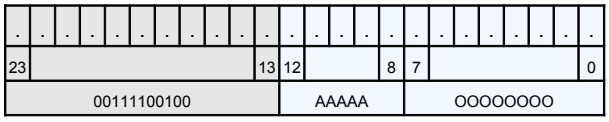
Format:

b.sfgtsi rA,I

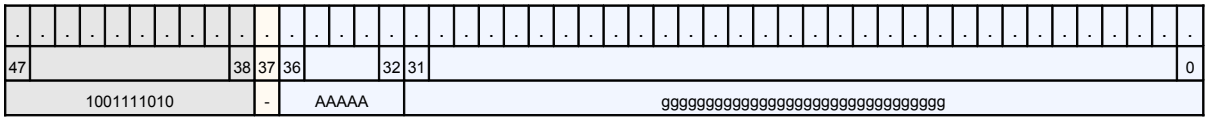
Description:

The immediate value I is shifted left [O << 0, g << 0] bits, sign-extended[O, g] to program counter width. The content of general-purpose register rA and the sign-extended immediate value are compared as signed integers. If the content of the register is greater than the immediate value the compare flag f is set; otherwise the compare flag is cleared.

bn.sfgtsi rA,O



bw.sfgtsi rA,g



Effect:

```
if bn.sfgtsi then I = EXTS8(O)
if bw.sfgtsi then I = EXTS32(g)
SR[F] <- rA[31:0] > I
```

3.102 b.sfgtu

Set Flag if Greater Than Unsigned

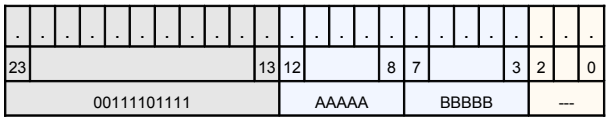
Format:

b.sfgtu rA,rB

Description:

The contents of general-purpose registers rA and rB are compared as unsigned integers. If the content of the first register is greater than the content of the second register, the compare flag f is set; otherwise the compare flag is cleared.

bn.sfgtu rA,rB



Effect:

SR[F] <- rA[31:0] > rB[31:0]

3.103 b.sfgtui

Set Flag if Greater Than Immediate Unsigned

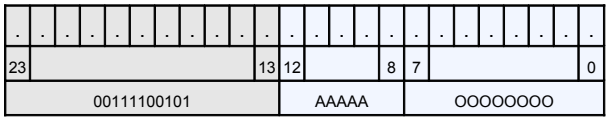
Format:

b.sfgtui rA,I

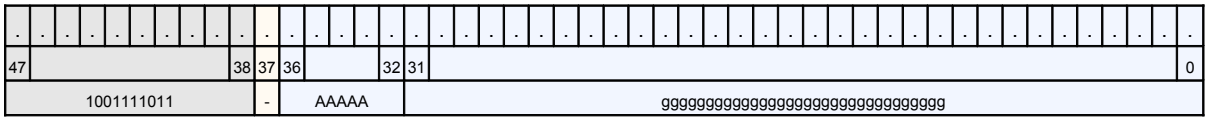
Description:

The immediate value I is shifted left [O << 0, g << 0] bits, sign-extended[O, g] to program counter width. The content of general-purpose register rA and the sign-extended immediate value are compared as unsigned integers. If the content of the register is greater than the immediate value the compare flag f is set; otherwise the compare flag is cleared.

bn.sfgtui rA,O



bw.sfgtui rA,g



Effect:

```
if bn.sfgtui then I = EXTS8(O)
if bw.sfgtui then I = EXTS32(g)
SR[F] <- rA[31:0] > I
```

3.104 b.sfles

Set Flag if Less or Equal Than Signed

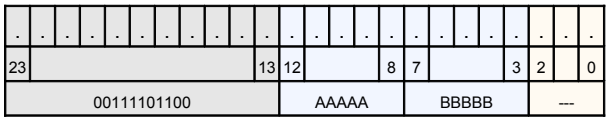
Format:

b.sfles rA,rB

Description:

The contents of general-purpose registers rA and rB are compared as signed integers. If the content of the first register is less than or equal to the content of the second register, the compare flag f is set; otherwise the compare flag is cleared.

equivalent to bn.sfges rA,rB



Effect:

SR[F] <- rA[31:0] <= rB[31:0]

3.105 b.sflesi

Set Flag if Less or Equal Than Immediate Signed

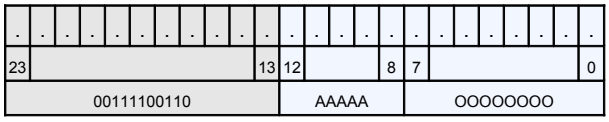
Format:

b.sflesi rA,I

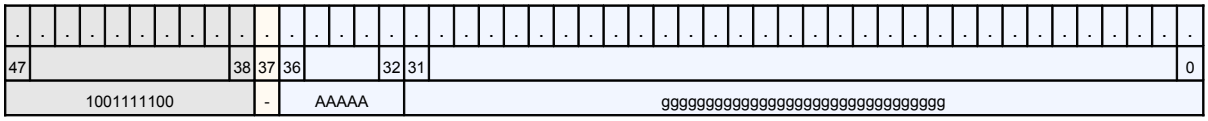
Description:

The immediate value I is shifted left [O << 0, g << 0] bits, sign-extended[O, g] to program counter width. The content of general-purpose register rA and the sign-extended immediate value are compared as signed integers. If the content of the register is less than or equal to the immediate value the compare flag f is set; otherwise the compare flag is cleared.

bn.sflesi rA,O



bw.sflesi rA,g



Effect:

```
if bn.sflesi then I = EXTS8(O)
if bw.sflesi then I = EXTS32(g)
SR[F] <- rA[31:0] <= I
```

3.106 b.sfleu

Set Flag if Less or Equal Than Unsigned

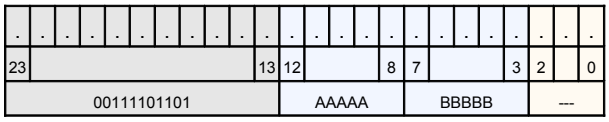
Format:

b.sfleu rA,rB

Description:

The contents of general-purpose registers rA and rB are compared as unsigned integers. If the content of the first register is less than or equal to the content of the second register, the compare flag f is set; otherwise the compare flag is cleared.

equivalent to bn.sfgeu rA,rB



Effect:

SR[F] <- rA[31:0] <= rB[31:0]

3.107 b.sfleui

Set Flag if Less or Equal Than Immediate Unsigned

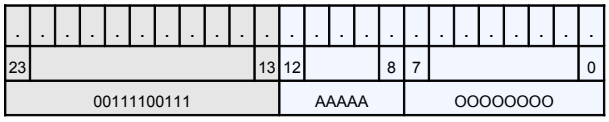
Format:

b.sfleui rA,I

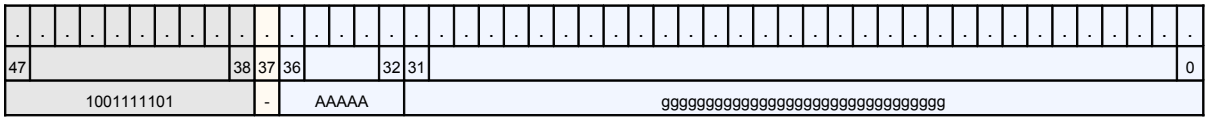
Description:

The immediate value I is shifted left [O << 0, g << 0] bits, sign-extended[O, g] to program counter width. The content of general-purpose register rA and the sign-extended immediate value are compared as unsigned integers. If the content of the register is less than or equal to the immediate value the compare flag f is set; otherwise the compare flag is cleared.

bn.sfleui rA,O



bw.sfleui rA,g



Effect:

```
if bn.sfleui then I = EXTS8(O)
if bw.sfleui then I = EXTS32(g)
SR[F] <- rA[31:0] <= I
```

3.108 b.sflts

Set Flag if Less Than Signed

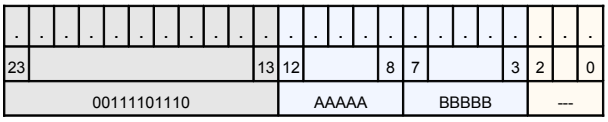
Format:

b.sflts rA,rB

Description:

The contents of general-purpose registers rA and rB are compared as signed integers. If the content of the first register is less than the content of the second register, the compare flag f is set; otherwise the compare flag is cleared.

equivalent to bn.sfgts rA,rB



Effect:

SR[F] <- rA[31:0] < rB[31:0]

3.109 b.sfltsi

Set Flag if Less Than Immediate Signed

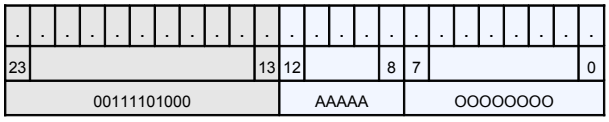
Format:

b.sfltsi rA,I

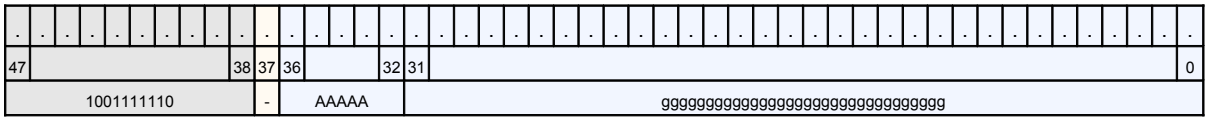
Description:

The immediate value I is shifted left [O << 0, g << 0] bits, sign-extended[O, g] to program counter width. The content of general-purpose register rA and the sign-extended immediate value are compared as signed integers. If the content of the register is less than the immediate value the compare flag f is set; otherwise the compare flag is cleared.

bn.sfltsi rA,O



bw.sfltsi rA,g



Effect:

```
if bn.sfltsi then I = EXTS8(O)
if bw.sfltsi then I = EXTS32(g)
SR[F] <- rA[31:0] < I
```

3.110 b.sfltu

Set Flag if Less Than Unsigned

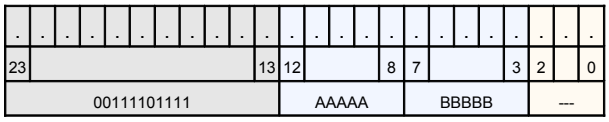
Format:

b.sfltu rA,rB

Description:

The contents of general-purpose registers rA and rB are compared as unsigned integers. If the content of the first register is less than the content of the second register, the compare flag f is set; otherwise the compare flag is cleared.

equivalent to bn.sfgtu rA,rB



Effect:

SR[F] <- rA[31:0] < rB[31:0]

3.111 b.sfltui

Set Flag if Less Than Immediate Unsigned

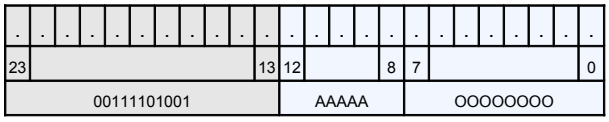
Format:

b.sfltui rA,I

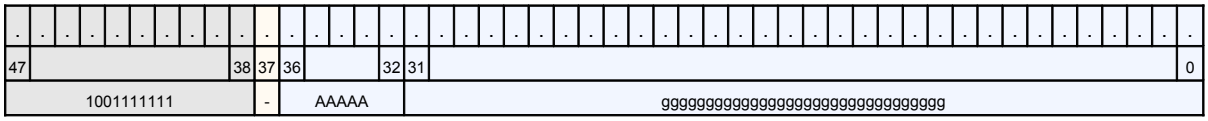
Description:

The immediate value I is shifted left [O << 0, g << 0] bits, sign-extended[O, g] to program counter width. The content of general-purpose register rA and the sign-extended immediate value are compared as unsigned integers. If the content of the register is less than the immediate value the compare flag f is set; otherwise the compare flag is cleared.

bn.sfltui rA,O



bw.sfltui rA,g



Effect:

```
if bn.sfltui then I = EXTS8(O)
if bw.sfltui then I = EXTS32(g)
SR[F] <- rA[31:0] < I
```

3.112 b.sfne

Set Flag if Not Equal

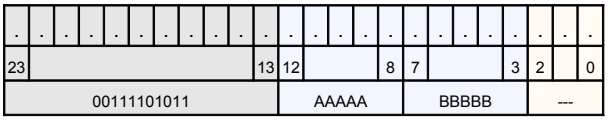
Format:

b.sfne rA,rB

Description:

The contents of general-purpose registers rA and rB are compared. If the contents are not equal, the compare flag f is set; otherwise the compare flag is cleared.

bn.sfne rA,rB



Effect:

SR[F] <- rA[31:0] != rB[31:0]

3.113 b.sfnei

Set Flag if Not Equal Immediate

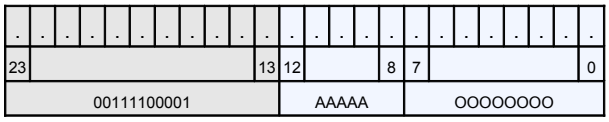
Format:

b.sfnei rA,I

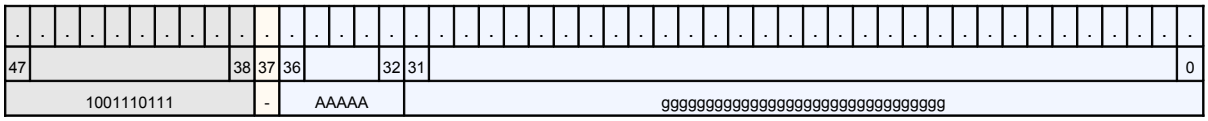
Description:

The immediate value I is shifted left [O << 0, g << 0] bits, sign-extended[O, g] to program counter width. The content of general-purpose register rA and the sign-extended immediate value are compared. If the two values are not equal, the compare flag f is set; otherwise the compare flag is cleared.

bn.sfnei rA,O



bw.sfnei rA,g



Effect:

```
if bn.sfnei then I = EXTS8(O)
if bw.sfnei then I = EXTS32(g)
SR[F] <- rA[31:0] != I
```

3.114 b.sh

Store Half Word

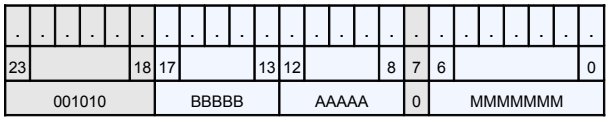
Format:

b.sh I(rA),rB

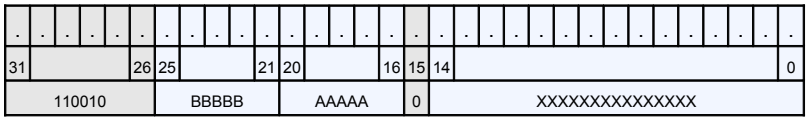
Description:

The immediate value I is shifted left [M << 1, X << 1, i << 1] bits, sign-extended[X, i] or zero-extended[M] to program counter width. Then it is added to the content of general-purpose register rA. The sum represents an effective address. The low-order 16 bits of general-purpose register rB are stored to memory location addressed by EA.

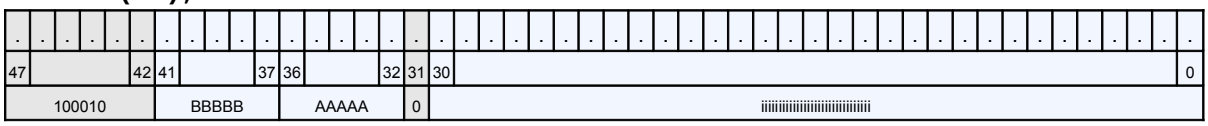
bn.sh M(rA),rB



bg.sh X(rA),rB



bw.sh i(rA),rB



Effect:

```
if bn.sh then I = EXTZ(M << 1)
if bg.sh then I = EXTS16(X << 1)
if bw.sh then I = EXTS32(i << 1)
EA <- I + rA[31:0]
(EA)[15:0] <- rB[15:0]
```

3.115 b.sll

Shift Left Logical

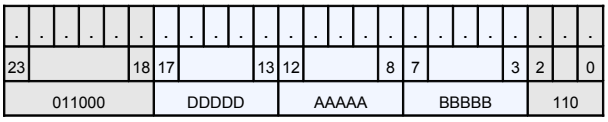
Format:

b.sll rD,rA,rB

Description:

The content of general-purpose register rA is shifted left by the number of bit positions specified in the general-purpose register rB, inserting zeros into the low-order bits. The result is then placed into general-purpose register rD. In 32 bit implementations 6th bit (bit 5) of rB is ignored.

bn.sll rD,rA,rB



Effect:

```
rD[31:rB[4:0] ] <- rA[31-rB[4:0]:0]
rD[rB[4:0]-1:0] <- 0
```

3.116 b.slli

Shift Left Logical with Immediate

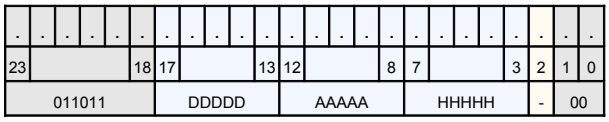
Format:

b.slli rD,rA,I

Description:

The content of general-purpose register rA is shifted left by the number of bit positions specified by the 6 bit immediate value I, inserting zeros into the low-order bits. The result is then placed into general-purpose register rD. In 32 bit implementations 6th bit (bit 5) of immediate is ignored.

bn.slli rD,rA,H



Effect:

```
if bn.slli then I = EXTZ(H)
rD[31:H] <- rA[31-H:0]
rD[H-1:0] <- 0
```

3.117 b.sma

Store Memory Absolute

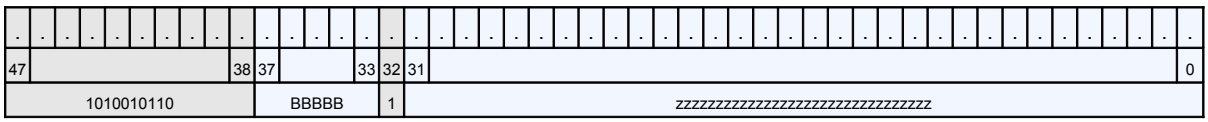
Format:

b.sma rB,I

Description:

The immediate value I is added with the program counter. The result is a pointer to the stored effective address. The low-order 32 bits of general-purpose register rB are stored to memory location addressed by the stored EA.

bw.sma rB,z



Effect:

```
if bw.sma then I = EXTS32(z)
T <- (PC + I)
EA <- (T)[31:0]
(EA)[31:0] <- rB[31:0]
```

3.118 b.sra

Shift Right Arithmetic

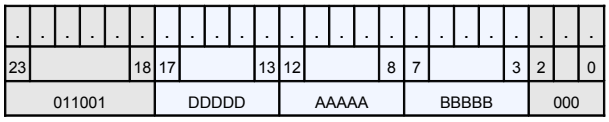
Format:

b.sra rD,rA,rB

Description:

The content of general-purpose register rA is shifted right by the number of bit positions specified in the general-purpose register rB, sign-extending the high-order bits. The result is then placed into general-purpose register rD. In 32 bit implementations 6th bit (bit 5) of rB is ignored.

bn.sra rD,rA,rB



Effect:

```
rD[31-rB[4:0]:0] <- rA[31:rB[4:0] ]
rD[31:32-rB[4:0] ] <- {rA[31]}
```

3.119 b.srai

Shift Right Arithmetic with Immediate

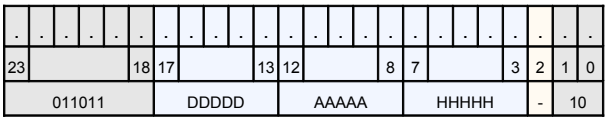
Format:

b.srai rD,rA,I

Description:

The content of general-purpose register rA is shifted right by the number of bit positions specified by the 6 bit immediate value I, sign-extending the high-order bits. The result is then placed into general-purpose register rD. In 32 bit implementations 6th bit (bit 5) of immediate is ignored.

bn.srai rD,rA,H



Effect:

```
if bn.srai then I = EXTZ(H)
rD[31-H:0] <- rA[31:H]
rD[31:32-H] <- {rA[31]}
```

3.120 b.srl

Shift Right Logical

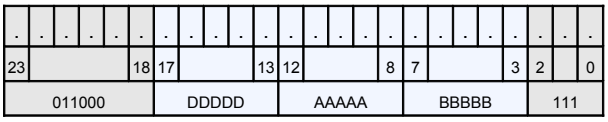
Format:

b.srl rD,rA,rB

Description:

The content of general-purpose register rA is shifted right by the number of bit positions specified in the general-purpose register rB, inserting zeros into the high-order bits. The result is then placed into general-purpose register rD. In 32 bit implementations 6th bit (bit 5) of rB is ignored.

bn.srl rD,rA,rB



Effect:

```
rD[31-rB[4:0]:0] <- rA[31:rB[4:0] ]
rD[31:32-rB[4:0] ] <- 0
```

3.121 b.srli

Shift Right Logical with Immediate

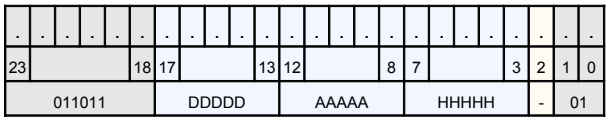
Format:

b.srli rD,rA,I

Description:

The content of general-purpose register rA is shifted right by the number of bit positions as specified by the 6 bit immediate value I, inserting zeros into the high-order bits. The result is then placed into general-purpose register rD. In 32 bit implementations 6th bit (bit 5) of immediate is ignored.

bn.srli rD,rA,H



Effect:

```
if bn.srli then I = EXTZ(H)
rD[31-H:0] <- rA[31:H]
rD[31:32-H] <- 0
```

3.122 b.sub

Subtract Signed

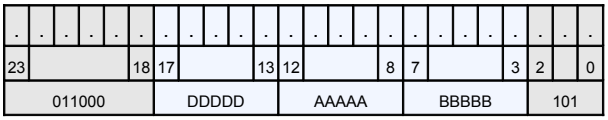
Format:

b.sub rD,rA,rB

Description:

The content of general-purpose register rB is subtracted from the content of general-purpose register rA to form the result, which is then placed into general-purpose register rD.

bn.sub rD,rA,rB



Effect:

```
rD[31:0] <- rA[31:0] - rB[31:0]
SR[OV] <- overflow
SR[CY] <- carry
```

3.123 b.subb

Subtract Signed with Borrow

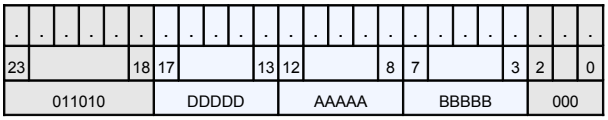
Format:

b.subb rD,rA,rB

Description:

The sum of the content of general-purpose register rB and borrow bit SR[CY] is subtracted from the content of general-purpose register rA to form the result, which is then placed into general-purpose register rD.

bn.subb rD,rA,rB



Effect:

```
rD[31:0] <- rA[31:0] - (rB[31:0] + SR[CY])
SR[OV] <- overflow
SR[CY] <- borrow
```

3.124 b.subs

Substract Saturated

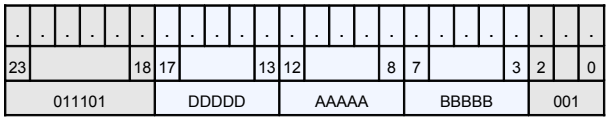
Format:

b.subs rD,rA,rB

Description:

The content of general-purpose registers rB is substracted from the content of general-purpose register rA. Result is stored into general-purpose register rD. If the result overflows it is clamped to the most positive number INT_MAX (0x7fffffff). If the result underflows it is clamped to the most negative number INT_MIN (0x80000000). The Overflow flag is sticky.

bn.subs rD,rA,rB



Effect:

```
rD[31:0] <- rA[31:0] - rB[31:0]

if (rA - rB >= INT_MAX)
rD <- INT_MAX
else if (rA - rB =< INT_MIN)
rD <- INT_MIN
else
rD <- rA - rB

SR[OV] <- overflow
SR[CY] <- carry
```

3.125 b.sw

Store Single Word

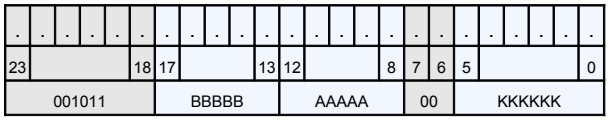
Format:

b.sw I(rA),rB

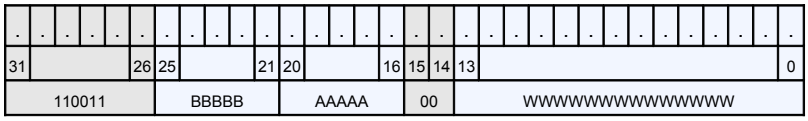
Description:

The immediate value I is shifted left [K << 2, W << 2, w << 2] bits, sign-extended[W, w] or zero-extended[K] to program counter width. Then it is added to the content of general-purpose register rA. The sum represents an effective address. The low-order 32 bits of general-purpose register rB are stored to memory location addressed by EA.

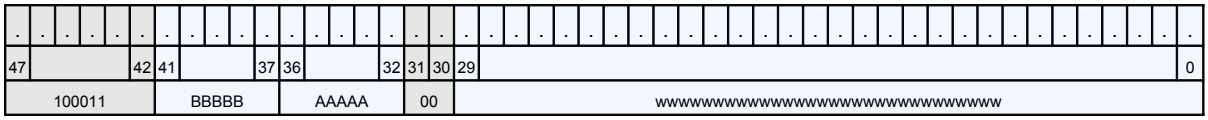
bn.sw K(rA),rB



bg.sw W(rA),rB



bw.sw w(rA),rB



Effect:

```
if bn.sw then I = EXTZ(K << 2)
if bg.sw then I = EXTS16(W << 2)
if bw.sw then I = EXTS32(w << 2)
EA <- I + rA[31:0]
(EA)[31:0] <- rB[31:0]
```

3.126 b.swab

Swap Bytes

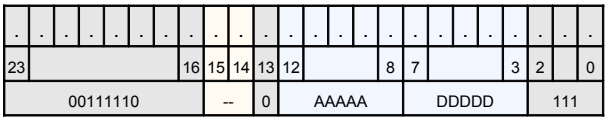
Format:

b.swab rD,rA

Description:

Bytes in general-purpose register rA are placed into general-purpose register rD in a reverse order.

bn.swab rD,rA



Effect:

```
rD[7:0]
<- rA[31:24]
rD[15:8]
<- rA[23:16]
rD[23:16] <- rA[15:8]
rD[31:24] <- rA[7:0]
```

3.127 b.sys

System Call

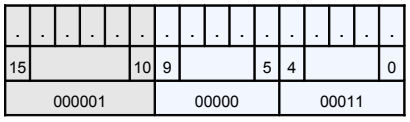
Format:

b.sys n/a

Description:

Execution of the system call instruction results in the system call exception. The system call exception is a request to the operating system to provide operating system services. The immediate value can be used to specify which system service is requested, alternatively a GPR defined by the ABI can be used to specify system service.

equivalent to bt.mov r0,r3



Effect:

system-call-exception()

3.128 b.trap

Trap

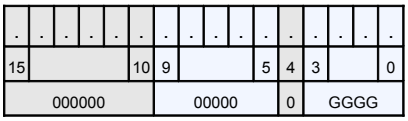
Format:

```
b.trap I
```

Description:

Execution of trap instruction results in the trap exception if specified bit in SR is set. Trap exception is a request to the operating system or to the debug facility to execute certain debug services. The immediate value is used to select which SR bit is tested by trap instruction.

equivalent to bt.movi r0,G



Effect:

```
if SR[I] = 1 then trap_exception()
```

3.129 b.xor

Exclusive Or

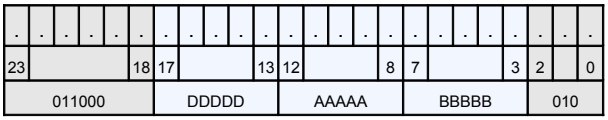
Format:

b.xor rD,rA,rB

Description:

The content of general-purpose register rA is bit-wise XORed with the content of general-purpose register rB. The result is then placed into general-purpose register rD.

bn.xor rD,rA,rB



Effect:

rD[31:0] <- rA[31:0] XOR rB[31:0]

3.130 b.xori

Exclusive Or with Immediate

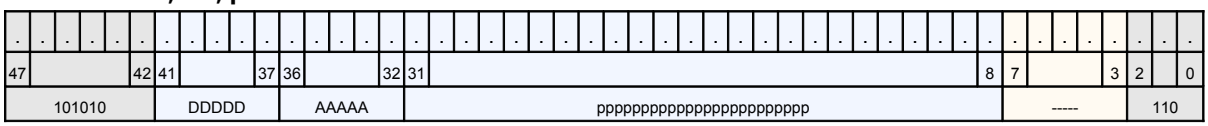
Format:

b.xori rD,rA,I

Description:

Then immediate value I is bit-wise XORed with the content of general-purpose register rA. The result is then placed into general-purpose register rD.

bw.xori rD,rA,p



Effect:

```
if bw.xori then I = EXTS24(p)
rD[31:0] <- rA[31:0] XOR I
```

3.131 f.abs.s

Floating Point Absolute (Single Precision)

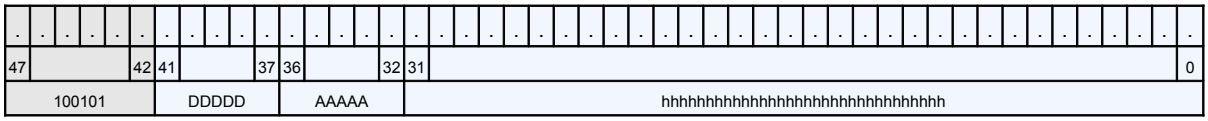
Format:

f.abs.s rD,rA

Description:

The absolute value of general-purpose register rA is placed into general-purpose register rD.

equivalent to bw.andi rD,rA,h



Effect:

rD[31:0] <- rA[31:0] AND 0x7fffffff

3.132 f.add.s

Floating Point Add (Single Precision)

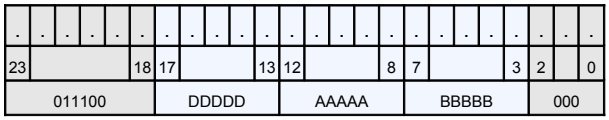
Format:

f.add.s rD,rA,rB

Description:

The sum of the contents of general-purpose registers rA and rB is stored into general-purpose register rD. Both operands are treated as floating point values.

fn.add.s rD,rA,rB



Effect:

```
rD[31:0] <- rA[31:0] + rB[31:0]
UR[FOV] <- flag overflow
UR[FUN] <- flag underflow
UR[FSN] <- flag signaling NaN
UR[FQN] <- flag quiet NaN
UR[FIX] <- flag inexact
UR[FIV] <- flag invalid
```

3.133 f.beq.s

Floating Point Branch If Equal (Single Precision)

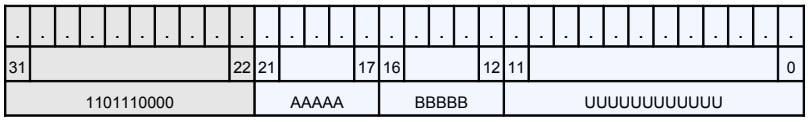
Format:

f.beq.s rA,rB,I

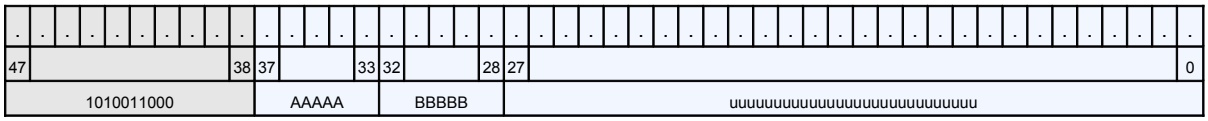
Description:

Then immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. The contents of general-purpose register rA and rB are compared as floating point values. If they are equal, the program branches to EA.

fg.beq.s rA,rB,U



fw.beq.s rA,rB,u



Effect:

```
if fg.beq.s then I = EXTS12(U)
if fw.beq.s then I = EXTS28(u)
EA <- I + BranchInsnAddr
PC <- EA if rA[31:0] == rB[31:0]
```

3.134 f.bge.s

Floating Point Branch If Greater Or Equal Than (Single Precision)

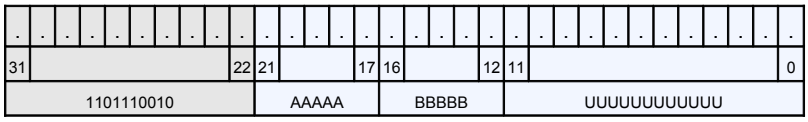
Format:

f.bge.s rA,rB,I

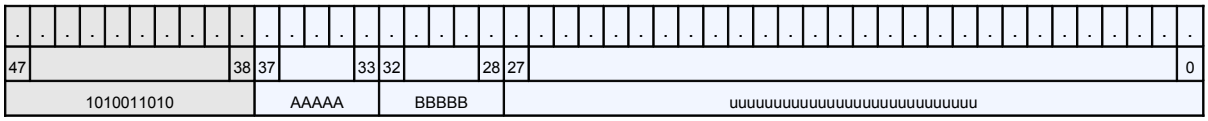
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. The contents of general-purpose register rA and rB are compared as floating point values. If the content of the first register is greater or equal than the content of the second register, the program branches to EA.

fg.bge.s rA,rB,U



fw.bge.s rA,rB,u



Effect:

```
if fg.bge.s then I = EXTS12(U)
if fw.bge.s then I = EXTS28(u)
EA <- I + BranchInsnAddr
PC <- EA if rA[31:0] >= rB[31:0]
```

3.135 f.bgt.s

Floating Point Branch If Greater Than (Single Precision)

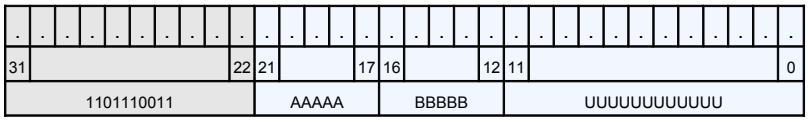
Format:

f.bgt.s rA,rB,I

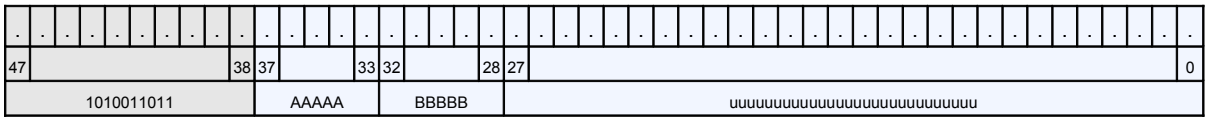
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. The contents of general-purpose register rA and rB are compared as floating point values. If the content of the first register is greater than the content of the second register, the program branches to EA.

fg.bgt.s rA,rB,U



fw.bgt.s rA,rB,u



Effect:

```
if fg.bgt.s then I = EXTS12(U)
if fw.bgt.s then I = EXTS28(u)
EA <- I + BranchInsnAddr
PC <- EA if rA[31:0] > rB[31:0]
```

3.136 f.ble.s

Floating Point Branch If Less Or Equal Than (Single Precision)

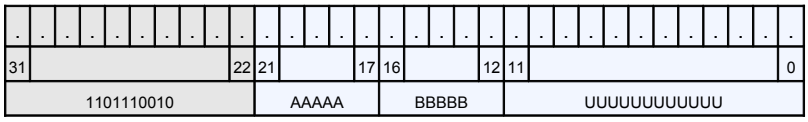
Format:

f.ble.s rA,rB,I

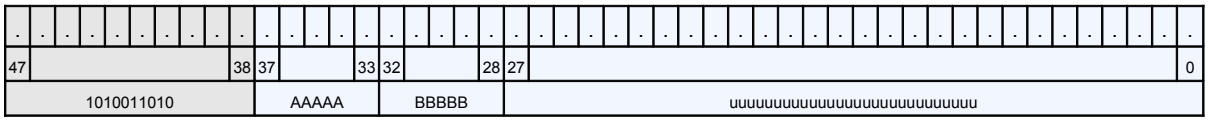
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. The contents of general-purpose register rA and rB are compared as floating point values. If the content of the first register is less or equal than the content of the second register, the program branches to EA.

equivalent to fg.bge.s rA,rB,U



equivalent to fw.bge.s rA,rB,u



Effect:

```
if fg.bge.s then I = EXTS12(U)
if fw.bge.s then I = EXTS28(u)
EA <- I + BranchInsnAddr
PC <- EA if rA[31:0] <= rB[31:0]
```

3.137 f.blt.s

Floating Point Branch If Less Than (Single Precision)

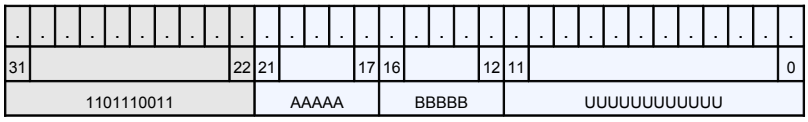
Format:

f.blt.s rA,rB,I

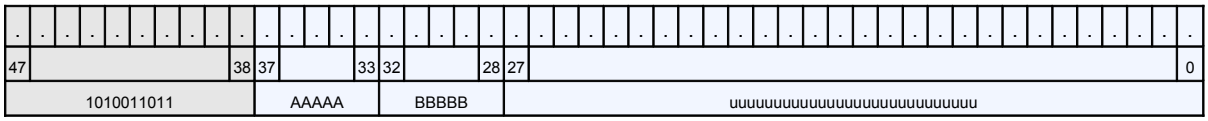
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. The contents of general-purpose register rA and rB are compared as floating point values. If the content of the first register is less than the content of the second register, the program branches to EA.

equivalent to fg.bgt.s rA,rB,U



equivalent to fw.bgt.s rA,rB,u



Effect:

```
if fg.bgt.s then I = EXTS12(U)
if fw.bgt.s then I = EXTS28(u)
EA <- I + BranchInsnAddr
PC <- EA if rA[31:0] < rB[31:0]
```

3.138 f.bne.s

Floating Point Branch If Not Equal (Single Precision)

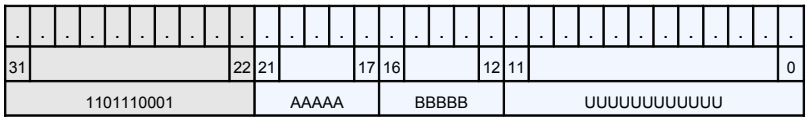
Format:

f.bne.s rA,rB,I

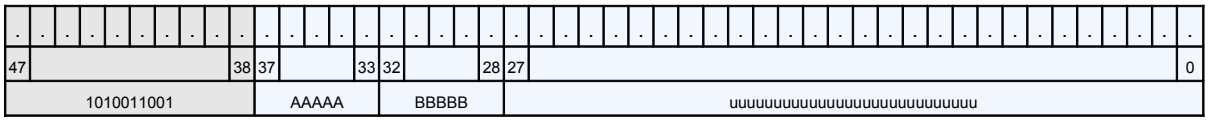
Description:

The immediate value I is added to the address of the branch instruction. The result is the effective address of the branch. The contents of general-purpose register rA and rB are compared as floating point values. If they are not equal, the program branches to EA.

fg.bne.s rA,rB,U



fw.bne.s rA,rB,u



Effect:

```
if fg.bne.s then I = EXTS12(U)
if fw.bne.s then I = EXTS28(u)
EA <- I + BranchInsnAddr
PC <- EA if rA[31:0] != rB[31:0]
```

3.139 f.div.s

Floating Point Divide (Single Precision)

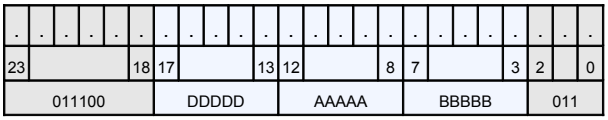
Format:

```
f.div.s rD,rA,rB
```

Description:

The contents of general-purpose register rA and rB are divided to form the result, which is then placed into general-purpose register rD. Both operands are treated as floating point values.

fn.div.s rD,rA,rB



Effect:

```
rD[31:0] <- rA[31:0] / rB[31:0]
UR[FOV] <- flag overflow
UR[FUN] <- flag underflow
UR[FSN] <- flag signaling NaN
UR[FQN] <- flag quiet NaN
UR[FIX] <- flag inexact
UR[FIV] <- flag invalid
UR[FDZ] <- flag divide by zero
```

3.140 f.ftoi.s

Floating Point Float To Integer (Single Precision)

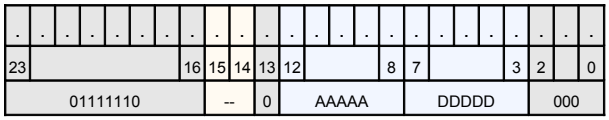
Format:

```
f.ftoi.s rD,rA
```

Description:

The contents of general-purpose register rA is treated as floating point value and is converted to signed integer, which is then placed into general-purpose register rD.

fn.ftoi.s rD,rA



Effect:

```
rD[31:0] <- ftoi(rA[31:0])
UR[FOV] <- flag overflow
UR[FUN] <- flag underflow
UR[FSN] <- flag signaling NaN
UR[FQN] <- flag quiet NaN
UR[FIX] <- flag inexact
UR[FIV] <- flag invalid
```

3.141 f.itof.s

Floating Point Integer To Float (Single Precision)

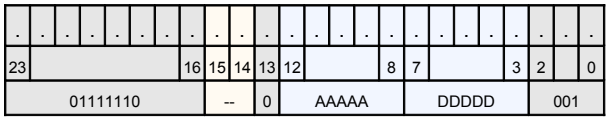
Format:

```
f.itof.s rD,rA
```

Description:

The content of general-purpose register rA is treated as signed integer and is converted to floating point value, which is then placed into general-purpose register rD.

fn.itof.s rD,rA



Effect:

```
rD[31:0] <- itof(rA[31:0])
UR[FOV] <- flag overflow
UR[FUN] <- flag underflow
UR[FSN] <- flag signaling NaN
UR[FQN] <- flag quiet NaN
UR[FIX] <- flag inexact
UR[FIV] <- flag invalid
```

3.142 f.mul.s

Floating Point Multiply (Single Precision)

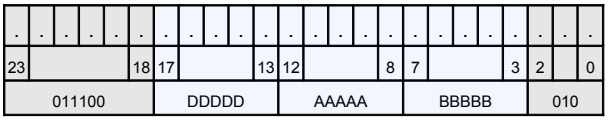
Format:

f.mul.s rD,rA,rB

Description:

The content of general-purpose register rA and rB are multiplied to form the result, which is then placed into general-purpose register rD. Both operands are treated as floating point values.

fn.mul.s rD,rA,rB



Effect:

```
rD[31:0] <- rA[31:0] * rB[31:0]
UR[FOV] <- flag overflow
UR[FUN] <- flag underflow
UR[FSN] <- flag signaling NaN
UR[FQN] <- flag quiet NaN
UR[FIX] <- flag inexact
UR[FIV] <- flag invalid
```

3.143 f.sub.s

Floating Point Subtract (Single Precision)

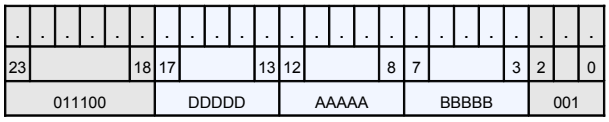
Format:

f.sub.s rD,rA,rB

Description:

The content of general-purpose register rB is subtracted from the content of general-purpose register rA to form the result, which is then placed into general-purpose register rD. Both operands are treated as floating point values.

fn.sub.s rD,rA,rB



Effect:

```
rD[31:0] <- rA[31:0] - rB[31:0]
UR[FOV] <- flag overflow
UR[FUN] <- flag underflow
UR[FSN] <- flag signaling NaN
UR[FQN] <- flag quiet NaN
UR[FIX] <- flag inexact
UR[FIV] <- flag invalid
```