Boneless-III

Architecture Reference Manual

Notice:

This document is a work in progress and subject to change without warning. However, the parts that are *especially* subject to change carry a notice similar to this one.

Contents

1 Introduction

TBD

2 Guide to Instruction Set

TBD

3 List of Instructions

The following pages provide a detailed description of instructions, arranged in alphabetical order.

Executing any instruction with an encoding not present on the following pages has **UNPREDICTABLE** behavior.

3.1   ADC   Add Register with Carry

Encoding:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ADC		00010				Rd			Ra			01		Rb	

Assembly:

```
ADC Rd, Ra, Rb
```

Purpose:

To add 16-bit integers in registers, with carry input.

Restrictions:

None.

Operation:

```
opA ← mem[W|Ra]
opB ← mem[W|Rb]
res ← opA + opB + C
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← res[16]
V ← (opA[15] = opB[15]) and (opA[15] <> res[15])
```

Remarks:

A 32-bit addition with both operands in registers can be performed as follows:

```
; Perform (R1|R0) ← (R3|R2) + (R5|R4)
    ADD  R0, R2, R4
    ADC  R1, R3, R5
```

3.2 ADCI   Add Immediate with Carry

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ADCI		00011					Rd			Ra			01		imm3	

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110						ext13								
ADCI		00011					Rd			Ra			01		imm3	

Assembly:

```
ADCI Rd, Ra, imm
```

Purpose:

To add a constant to a 16-bit integer in a register, with carry input.

Restrictions:

None.

Operation:

```
opA ← mem[W|Ra]
if (has_ext13)
then opB ← ext13|imm3
else opB ← decode_imm_al(imm3)
res ← opA + opB + C
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← res[16]
V ← (opA[15] = opB[15]) and (opA[15] <> res[15])
```

Remarks:

A 32-bit addition with a register and an immediate operand can be performed as follows:

```
; Perform (R1|R0) ← (R3|R2) + 0x40001
    ADDI R0, R2, 1
    ADCI R1, R3, 4
```

3.3 ADD   Add Register

Encoding:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ADD	00010					Rd			Ra			00		Rb	

Assembly:

```
ADD Rd, Ra, Rb
```

Purpose:

To add 16-bit integers in registers.

Restrictions:

None.

Operation:

```
opA ← mem[W|Ra]
opB ← mem[W|Rb]
res ← opA + opB
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← res[16]
V ← (opA[15] = opB[15]) and (opA[15] <> res[15])
```

3.4 ADDI   Add Immediate

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ADDI			00011				Rd			Ra			00		imm3	

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	110			ext13												
ADDI			00011				Rd			Ra			00		imm3	

Assembly:

```
ADDI Rd, Ra, imm
```

Purpose:

To add a constant to a 16-bit integer in a register.

Restrictions:

None.

Operation:

```
opA ← mem[W|Ra]
if (has_ext13)
then opB ← ext13|imm3
else opB ← decode_imm_al(imm3)
res ← opA + opB
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← res[16]
V ← (opA[15] = opB[15]) and (opA[15] <> res[15])
```

3.5 ADJW   Adjust Window Address

Encoding (short form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ADJW		**10100**				**000**			**010**				imm5		

Encoding (long form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	**110**			ext13											
ADJW		**10100**				**000**			**010**				imm5		

Assembly:

```
ADJW imm
```

Purpose:

To increase or decrease the address of the register window.

Restrictions:

If `imm` contains a value that is not a multiple of 8, the behavior is **UNPREDICTABLE**. If the long form is used, and `imm5[4:3]` are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then imm ← ext13|imm5[2:0]
else imm ← sign_extend(imm5)
W ← W + imm
```

Remarks:

This instruction may be used in a function prologue or epilogue.

Notice:

The interpretation of the immediate field of this instruction is not final.

3.6 AND   Bitwise AND with Register

Encoding:

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
AND			00000				Rd			Ra			00		Rb	

Assembly:

```
AND Rd, Ra, Rb
```

Purpose:

To perform bitwise AND between 16-bit integers in registers.

Restrictions:

None.

Operation:

```
opA ← mem[W|Ra]
opB ← mem[W|Rb]
res ← opA and opB
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

3.7 ANDI   Bitwise AND with Immediate

Encoding (short form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ANDI	00001				Rd			Ra			00		imm3		

Encoding (long form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	110			ext13											
ANDI	00001				Rd			Ra			00		imm3		

Assembly:

```
ANDI Rd, Ra, imm
```

Purpose:

To perform bitwise AND between a 16-bit integer in a register and a constant.

Restrictions:

None.

Operation:

```
opA ← mem[W|Ra]
if (has_ext13)
then opB ← ext13|imm3
else opB ← decode_imm_al(imm3)
res ← opA and opB
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

3.8   CMP   Compare to Register

Encoding:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
CMP	00000					000			Ra			11		Rb	

Assembly:

```
CMP Rd, Ra, Rb
```

Purpose:

To compare 16-bit integers in registers.

Restrictions:

None.

Operation:

```
opA ← mem[W|Ra]
opB ← mem[W|Rb]
res ← opA - opB
Z ← res = 0
S ← res[15]
C ← not res[16]
V ← (opA[15] = not opB[15]) and (opA[15] <> res[15])
```

Remarks:

This instruction behaves identically to SUB, with the exception that it discards the computed value.

3.9 CMPI   Compare to Immediate

Encoding (short form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
CMPI		00001				000			Ra			11		imm3	

Encoding (long form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	110		ext13												
CMPI		00001				000			Ra			11		imm3	

Assembly:

```
CMPI Rd, Ra, imm
```

Purpose:

To compare a constant to a 16-bit integer in a register.

Restrictions:

None.

Operation:

```
opA ← mem[W|Ra]
if (has_ext13)
then opB ← ext13|imm3
else opB ← decode_imm_al(imm3)
res ← opA - opB
Z ← res = 0
S ← res[15]
C ← not res[16]
V ← (opA[15] = not opB[15]) and (opA[15] <> res[15])
```

Remarks:

This instruction behaves identically to SUBI, with the exception that it discards the computed value.

3.10 EXTI   Extend Immediate

Encoding:

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	110								imm13							

Assembly:

`EXTI imm`

Purpose:

To extend the range of immediate in the following instruction.

Restrictions:

None.

Operation:

```
ext13 ← imm13
has_ext13 ← 1
```

Remarks:

This instruction is automatically emitted by the assembler while translating other instructions. As it changes both the meaning of and the constraints placed on the immediate field in the following instruction, placing it manually may lead to unexpected results.

3.11   J   Jump

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
J		1011				1111						off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110						ext13								
J		1011				1111						off8				

Assembly:

```
J label
```

Purpose:

To unconditionally transfer control.

Restrictions:

If the long form is used, and `off8[7:3]` are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
PC ← PC + 1 + off
```

3.12   JAL   Jump and Link

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JAL			10101				Rd					off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110				ext13										
JAL			10101				Rd					off8				

Assembly:

```
JAL label
```

Purpose:

To transfer control to a subroutine.

Restrictions:

If the long form is used, and **off8[7:3]** are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
mem[W|Rd] ← PC + 1
PC ← PC + 1 + off
```

3.13   JC   Jump if Carry

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JC		1011				1010						off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110						ext13								
JC		1011				1010						off8				

Assembly:

```
JC label
```

Purpose:

To transfer control if an arithmetic operation resulted in unsigned overflow.

Restrictions:

If the long form is used, and **off8[7:3]** are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if (C)
then PC ← PC + 1 + off
else PC ← PC + 1
```

Remarks:

This instruction has the same encoding as JUGE.

3.14 JE   Jump if Equal

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JE		1011				1000						off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110					ext13									
JE		1011				1000						off8				

Assembly:

```
JE label
```

Purpose:

To transfer control after a CMP Ra, Rb instruction if Ra is equal to Rb.

Restrictions:

If the long form is used, and **off8[7:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if (Z)
then PC ← PC + 1 + off
else PC ← PC + 1
```

Remarks:

This instruction has the same encoding as JZ.

3.15 JN   Jump Never

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JN		1011				0111						off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110					ext13									
JN		1011				0111						off8				

Assembly:

```
JN label
```

Purpose:

To serve as a placeholder for a jump instruction.

Restrictions:

If the long form is used, and `off8[7:3]` are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
PC ← PC + 1
```

Remarks:

The `JN` instruction has no effect. It may be used as a placeholder for a different
jump instruction with a predefined offset when the exact condition is unknown,
such as in certain self-modifying code.

3.16   JNC Jump if Not Carry

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JNC		1011				0010						off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110						ext13								
JNC		1011				0010						off8				

Assembly:

```
JNC label
```

Purpose:

To transfer control if an arithmetic operation did not result in unsigned overflow.

Restrictions:

If the long form is used, and off8[7:3] are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if (not C)
then PC ← PC + 1 + off
else PC ← PC + 1
```

Remarks:

This instruction has the same encoding as JULT.

3.17 JNE — Jump if Not Equal

Encoding (short form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JNE	1011			0000				off8							

Encoding (long form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	110		ext13												
JNE	1011			0000				off8							

Assembly:

```
JNE label
```

Purpose:

To transfer control after a CMP Ra, Rb instruction if Ra is not equal to Rb.

Restrictions:

If the long form is used, and **off8[7:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if (not Z)
then PC ← PC + 1 + off
else PC ← PC + 1
```

Remarks:

This instruction has the same encoding as JNZ.

3.18   JNO   Jump if Not Overflow

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JNO		1011				0011						off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110						ext13								
JNO		1011				0011						off8				

Assembly:

```
JNO label
```

Purpose:

To transfer control if an arithmetic operation did not result in signed overflow.

Restrictions:

If the long form is used, and **off8[7:3]** are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if (not V)
then PC ← PC + 1 + off
else PC ← PC + 1
```

3.19 JNS   Jump if Not Negative

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JNS		1011				0001						off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110						ext13								
JNS		1011				0001						off8				

Assembly:

```
JNS label
```

Purpose:

To transfer control if an arithmetic or shift operation produced a non-negative result.

Restrictions:

If the long form is used, and **off8[7:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if (not S)
then PC ← PC + 1 + off
else PC ← PC + 1
```

3.20   JNZ   Jump if Not Zero

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JNZ	1011				0000				off8							

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	110			ext13												
JNZ	1011				0000				off8							

Assembly:

```
JNZ label
```

Purpose:

To transfer control if an arithmetic or shift operation produced a non-zero result.

Restrictions:

If the long form is used, and **off8[7:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if (not Z)
then PC ← PC + 1 + off
else PC ← PC + 1
```

Remarks:

This instruction has the same encoding as `JNE`.

3.21 JO   Jump if Overflow

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JO		1011				1011						off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110						ext13								
JO		1011				1011						off8				

Assembly:

```
JO label
```

Purpose:

To transfer control if an arithmetic operation resulted in signed overflow.

Restrictions:

If the long form is used, and **off8[7:3]** are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if (V)
then PC ← PC + 1 + off
else PC ← PC + 1
```

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JR			10100				Rs			100				off5		

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110						ext13								
JR			10100				Rs			100				off5		

Assembly:

```
JR Rs, off
```

Purpose:

To transfer control to a variable absolute address contained in a register, with a constant offset.

Restrictions:

If the long form is used, and `off5[4:3]` are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off5[2:0]
else off ← sign_extend(off5)
PC ← mem[W|Ra] + off
```

3.23 JRAL   Jump to Register and Link

Encoding:

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JRAL			10100				Rd			101			00		Rb	

Assembly:

```
JRAL Rd, Rb
```

Purpose:

To transfer control to a subroutine whose variable absolute address is contained in a register.

Restrictions:

None.

Operation:

```
addr ← mem[W|Rb]
mem[W|Rd] ← PC + 1
PC ← addr
```

3.24   JS   Jump if Negative

Encoding (short form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JS	1011			1001				off8							

Encoding (long form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	110		ext13												
JS	1011			1001				off8							

Assembly:

```
JS label
```

Purpose:

To transfer control if an arithmetic or shift operation produced a negative result.

Restrictions:

If the long form is used, and `off8[7:3]` are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if (S)
then PC ← PC + 1 + off
else PC ← PC + 1
```

3.25 JSGE   Jump if Signed Greater or Equal

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JSGE		1011				0101						off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110						ext13								
JSGE		1011				0101						off8				

Assembly:

```
JSGE label
```

Purpose:

To transfer control after a CMP Ra, Rb instruction if Ra is greater than or equal to Rb when interpreted as signed integer.

Restrictions:

If the long form is used, and off8[7:3] are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if (not (S xor V))
then PC ← PC + 1 + off
else PC ← PC + 1
```

3.26   JSGT   Jump if Signed Greater Than

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JSGT		1011				0110						off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110						ext13								
JSGT		1011				0110						off8				

Assembly:

```
JSGT label
```

Purpose:

To transfer control after a `CMP Ra, Rb` instruction if `Ra` is greater than to `Rb` when interpreted as signed integer.

Restrictions:

If the long form is used, and `off8[7:3]` are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if (not ((S xor V) or Z))
then PC ← PC + 1 + off
else PC ← PC + 1
```

3.27 JSLE   Jump if Signed Less or Equal

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JSLE		1011				1110						off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110						ext13								
JSLE		1011				1110						off8				

Assembly:

```
JSLE label
```

Purpose:

To transfer control after a CMP Ra, Rb instruction if Ra is less than or equal to Rb when interpreted as signed integer.

Restrictions:

If the long form is used, and off8[7:3] are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if (((S xor V) or Z))
then PC ← PC + 1 + off
else PC ← PC + 1
```

3.28   JSLT   Jump if Signed Less Than

Encoding (short form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JSLT	1011			1101				off8							

Encoding (long form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	110		ext13												
JSLT	1011			1101				off8							

Assembly:

```
JSLT label
```

Purpose:

To transfer control after a CMP Ra, Rb instruction if Ra is less than Rb when interpreted as signed integer.

Restrictions:

If the long form is used, and off8[7:3] are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if ((S xor V))
then PC ← PC + 1 + off
else PC ← PC + 1
```

3.29 JST   Jump through Switch Table

Encoding (short form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JST	10100				Rs			111			off5				

Encoding (long form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	110		ext13												
JST	10100				Rs			111			off5				

Assembly:

```
JST Rs, off
```

Purpose:

To transfer control to an address contained in a jump table at a variable offset, where the address is relative to the location of the table.

Restrictions:

If the long form is used, and `off5[4:3]` are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off5[2:0]
else off ← sign_extend(off5)
table ← PC + 1 + off
entry ← mem[W|Rs]
addr ← mem[table + entry]
PC ← table + addr
```

3.30 JUGE   Jump if Unsigned Greater or Equal

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JUGE		1011				1010						off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110						ext13								
JUGE		1011				1010						off8				

Assembly:

```
JUGE label
```

Purpose:

To transfer control after a CMP Ra, Rb instruction if Ra is greater than or equal to Rb when interpreted as unsigned integer.

Restrictions:

If the long form is used, and off8[7:3] are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if (C)
then PC ← PC + 1 + off
else PC ← PC + 1
```

Remarks:

This instruction has the same encoding as JC.

3.31 JUGT   Jump if Unsigned Greater Than

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JUGT		1011				0110						off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110						ext13								
JUGT		1011				0110						off8				

Assembly:

```
JUGT label
```

Purpose:

To transfer control after a CMP Ra, Rb instruction if Ra is greater than to Rb when interpreted as unsigned integer.

Restrictions:

If the long form is used, and off8[7:3] are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if (not ((not C) or V))
then PC ← PC + 1 + off
else PC ← PC + 1
```

3.32 JULE   Jump if Unsigned Less or Equal

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JULE		1011				1110						off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110						ext13								
JULE		1011				1110						off8				

Assembly:

```
JULE label
```

Purpose:

To transfer control after a CMP Ra, Rb instruction if Ra is less than or equal to Rb when interpreted as unsigned integer.

Restrictions:

If the long form is used, and off8[7:3] are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if ((not C) or V)
then PC ← PC + 1 + off
else PC ← PC + 1
```

3.33 JULT — Jump if Unsigned Less Than

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JULT		1011				0010							off8			

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110						ext13								
JULT		1011				0010							off8			

Assembly:

```
JULT label
```

Purpose:

To transfer control after a CMP Ra, Rb instruction if Ra is less than Rb when interpreted as unsigned integer.

Restrictions:

If the long form is used, and off8[7:3] are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if (not C)
then PC ← PC + 1 + off
else PC ← PC + 1
```

Remarks:

This instruction has the same encoding as JNC.

3.34   JVT   Jump through Virtual Table

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JVT		10100				Rs			110			off5				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110		ext13												
JVT		10100				Rs			110			off5				

Assembly:

```
JVT Rs, off
```

Purpose:

To transfer control to an address contained in a jump table at a constant offset, where the address is relative to the location of the table.

Restrictions:

If the long form is used, and `off5[4:3]` are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off5[2:0]
else off ← sign_extend(off5)
table ← mem[W|Rs]
entry ← off
addr ← mem[table + entry]
PC ← table + addr
```

3.35 JZ Jump if Zero

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JZ		1011				1000						off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110						ext13								
JZ		1011				1000						off8				

Assembly:

```
JZ label
```

Purpose:

To transfer control if an arithmetic or shift operation produced a zero result.

Restrictions:

If the long form is used, and **off8[7:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
if (Z)
then PC ← PC + 1 + off
else PC ← PC + 1
```

Remarks:

This instruction has the same encoding as JE.

3.36 LD

Load

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
LD		0	1	0	0	0		Rd			Ra			off5		

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110							ext13							
LD		0	1	0	0	0		Rd			Ra			off5		

Assembly:

```
LD Rd, Ra, off
```

Purpose:

To load a word from memory at a variable address, with a constant offset.

Restrictions:

If the long form is used, and `off5[4:3]` are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off5[2:0]
else off ← sign_extend(off5)
addr ← mem[W|Ra] + off
data ← mem[addr]
mem[W|Rd] ← data
```

3.37 LDR   Load PC-relative

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
LDR		01001					Rd			Ra			off5			

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110				ext13										
LDR		01001					Rd			Ra			off5			

Assembly:

```
LDR Rd, Ra, off
```

Purpose:

To load a word from memory at a constant PC-relative address, with a variable offset.

Restrictions:

If the long form is used, and **off5[4:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off5[2:0]
else off ← sign_extend(off5)
addr ← PC + 1 + off + mem[W|Ra]
data ← mem[addr]
mem[W|Rd] ← data
```

3.38 LDW   Adjust and Load Window Address

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
LDW		10100				Rd			011				imm5			

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110					ext13									
LDW		10100				Rd			011				imm5			

Assembly:

```
LDW Rd, imm
```

Purpose:

To increase or decrease the address of the register window, and retrieve the prior address of the register window.

Restrictions:

If `imm` contains a value that is not a multiple of 8, the behavior is **UNPREDICTABLE**. If the long form is used, and `imm5[4:3]` are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then imm ← ext13|imm5[2:0]
else imm ← sign_extend(imm5)
temp ← W
W ← W + imm
mem[W|Rd] ← temp
```

Remarks:

See also STW. This instruction may be used in a function prologue, where `Rd` is any register chosen to act as a frame pointer.

Notice:

The interpretation of the immediate field of this instruction is not final.

3.39 LDX   Load External

Encoding (short form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
LDX		01100				Rd			Ra				off5		

Encoding (long form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	110			ext13											
LDX		01100				Rd			Ra				off5		

Assembly:

```
LDX Rd, Ra, off
```

Purpose:

To complete a load cycle on external bus at a variable address, with a constant offset.

Restrictions:

If the long form is used, and `off5[4:3]` are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off5[2:0]
else off ← sign_extend(off5)
addr ← mem[W|Ra] + off
data ← ext[addr]
mem[W|Rd] ← data
```

3.40 LDXA   Load External Absolute

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
LDXA		01101				Rd					off8					

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	110			ext13												
LDXA		01101				Rd					off8					

Assembly:

```
LDXA Rd, off
```

Purpose:

To complete a load cycle on external bus at a constant absolute address.

Restrictions:

If the long form is used, and `off8[7:3]` are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
data ← ext[off]
mem[W|Rd] ← data
```

3.41   MOV   Move

Assembly:

```
MOV Rd, Rs
```

Purpose:

To move a value from register to register.

Restrictions:

None.

Remarks:

The assembler does not translate any instructions for `MOV` with identical `Rd` and `Rs`, and translates `MOV` with any other register combination to

```
AND  Rd, Rs, Rs
```

3.42 MOVI   Move Immediate

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
MOVI		10000			Rd					imm8						

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110			ext13											
MOVI		10000			Rd					imm8						

Assembly:

```
MOVI Rd, imm
```

Purpose:

To load a register with a constant.

Restrictions:

If the long form is used, and `imm8[8:3]` are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
if (has_ext13)
then imm ← ext13|imm8[2:0]
else imm ← sign_extend(imm8)
mem[W|Rd] ← imm
```

3.43 MOVR   Move PC-relative Address

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
MOVR	1	0	0	0	1			Rd				off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	1	1	0					ext13								
MOVR	1	0	0	0	1			Rd				off8				

Assembly:

```
MOVR Rd, off
```

Purpose:

To load a register with an address relative to PC with a constant offset..

Restrictions:

If the long form is used, and `off8[7:3]` are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
mem[W|Rd] ← PC + 1 + off
```

3.44 OR   Bitwise OR with Register

Encoding:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
OR	00000					Rd			Ra			01		Rb	

Assembly:

```
OR Rd, Ra, Rb
```

Purpose:

To perform bitwise OR between 16-bit integers in registers.

Restrictions:

None.

Operation:

```
opA ← mem[W|Ra]
opB ← mem[W|Rb]
res ← opA or opB
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

3.45 ORI   Bitwise OR with Immediate

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ORI		**00001**				Rd			Ra			**01**		imm3		

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	**110**			ext13												
ORI		**00001**				Rd			Ra			**01**		imm3		

Assembly:

```
ORI Rd, Ra, imm
```

Purpose:

To perform bitwise OR between a 16-bit integer in a register and a constant.

Restrictions:

None.

Operation:

```
opA ← mem[W|Ra]
if (has_ext13)
then opB ← ext13|imm3
else opB ← decode_imm_al(imm3)
res ← opA or opB
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

3.46   ROL   Rotate Left

Encoding:

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ROL			00100				Rd			Ra			01		Rb	

Assembly:

```
ROL Rd, Ra, Rb
```

Purpose:

To perform a left rotate of a 16-bit integer in a register by a variable bit amount.

Restrictions:

If `Rb` contains a value greater than 15, the behavior is **UNPREDICTABLE**.

Operation:

```
opA ← mem[W|Ra]
opB ← mem[W|Rb]
res ← opA[16-opB:0]|opA[16:16-opB]
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

3.47   ROLI   Rotate Left Immediate

Encoding:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ROLI		00101				Rd			Ra			01		imm3	

Assembly:

```
ROLI Rd, Ra, amount
```

Purpose:

To perform a left rotate of a 16-bit integer in a register by a constant bit amount.

Restrictions:

The **amount** may be between 0 and 15, inclusive.

Operation:

```
opA ← mem[W|Ra]
if (has_ext13)
then opB ← ext13|imm3
else opB ← decode_imm_sr(imm3)
res ← opA[15-imm3:0]|opA[16:15-imm3]
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

3.48 RORI   Rotate Right Immediate

Assembly:

```
RORI Rd, Ra, amount
```

Purpose:

To perform a right rotate of a 16-bit integer in a register by a constant bit amount.

Restrictions:

The `amount` may be between 0 and 15, inclusive.

Remarks:

The assembler translates `RORI` with `amount` of 0 to

```
ROLI Rd, Ra, 0
```

and `RORI` with any other `amount` to

```
ROLI Rd, Ra, (16 - amount)
```

3.49   SBB   Subtract Register with Borrow

Encoding:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
SBB	00010					Rd			Ra			11		Rb	

Assembly:

```
SBB Rd, Ra, Rb
```

Purpose:

To subtract 16-bit integers in registers, with borrow input.

Restrictions:

None.

Operation:

```
opA ← mem[W|Ra]
opB ← mem[W|Rb]
res ← opA - opB - not C
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← not res[16]
V ← (opA[15] = not opB[15]) and (opA[15] <> res[15])
```

Remarks:

A 32-bit subtraction with both operands in registers can be performed as follows:

```
; Perform (R1|R0) ← (R3|R2) - (R5|R4)
    SUB  R0, R2, R4
    SBB  R1, R3, R5
```

3.50 SBBI   Subtract Immediate with Borrow

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
SBBI		000	11			Rd			Ra			11			imm3	

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110							ext13							
SBBI		000	11			Rd			Ra			11			imm3	

Assembly:

```
SBBI Rd, Ra, imm
```

Purpose:

To subtract a constant from a 16-bit integer in a register, with borrow input.

Restrictions:

None.

Operation:

```
opA ← mem[W|Ra]
if (has_ext13)
then opB ← ext13|imm3
else opB ← decode_imm_al(imm3)
res ← opA - opB - not C
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← not res[16]
V ← (opA[15] = not opB[15]) and (opA[15] <> res[15])
```

Remarks:

A 32-bit subtraction with a register and an immediate operand can be performed as follows:

```
; Perform (R1|R0) ← (R3|R2) - 0x40001
    SUBI R0, R2, 1
    SBBI R1, R3, 4
```

3.51 SLL   Shift Left Logical

Encoding:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
SLL		00100				Rd			Ra			00		Rb	

Assembly:

```
SLL Rd, Ra, Rb
```

Purpose:

To perform a left logical shift of a 16-bit integer in a register by a variable bit amount.

Restrictions:

If **Rb** contains a value greater than 15, the behavior is **UNPREDICTABLE**.

Operation:

```
opA ← mem[W|Ra]
opB ← mem[W|Rb]
res ← opA[16-opB:0]|0{opB}
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

3.52 SLLI   Shift Left Logical Immediate

Encoding:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
SLLI	00101				Rd			Ra			00		imm3		

Assembly:

```
SLLI Rd, Ra, amount
```

Purpose:

To perform a left logical shift of a 16-bit integer in a register by a constant bit amount.

Restrictions:

The `amount` may be between 0 and 15, inclusive.

Operation:

```
opA ← mem[W|Ra]
if (has_ext13)
then opB ← ext13|imm3
else opB ← decode_imm_sr(imm3)
res ← opA[15-imm3:0]|0{imm3+1}
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

3.53 SRA   Shift Right Arithmetical

Encoding:

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
SRA		0	0	1	0	0		Rd			Ra			11		Rb

Assembly:

SRA Rd, Ra, Rb

Purpose:

To perform a right arithmetical shift of a 16-bit integer in a register by a variable bit amount.

Restrictions:

If **Rb** contains a value greater than 15, the behavior is **UNPREDICTABLE**.

Operation:

```
opA ← mem[W|Ra]
opB ← mem[W|Rb]
res ← opA[15]{opB}|opA[16:16-opB]
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

3.54 SRAI Shift Right Arithmetical Immediate

Encoding:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
SRAI	00100				Rd			Ra			11		imm3		

Assembly:

```
SRAI Rd, Ra, amount
```

Purpose:

To perform a right arithmetical shift of a 16-bit integer in a register by a constant bit amount.

Restrictions:

The `amount` may be between 0 and 15, inclusive.

Operation:

```
opA ← mem[W|Ra]
if (has_ext13)
then opB ← ext13|imm3
else opB ← decode_imm_sr(imm3)
res ← opA[15]{imm3+1}|opA[16:15-imm3]
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

3.55 SRL   Shift Right Logical

Encoding:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
SRL	00100				Rd			Ra			10		Rb		

Assembly:

```
SRL Rd, Ra, Rb
```

Purpose:

To perform a right logical shift of a 16-bit integer in a register by a variable bit amount.

Restrictions:

If **Rb** contains a value greater than 15, the behavior is **UNPREDICTABLE**.

Operation:

```
opA ← mem[W|Ra]
opB ← mem[W|Rb]
res ← 0{opB}|opA[16:16-opB]
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

3.56 SRLI   Shift Right Logical Immediate

Encoding:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
SRLI		00101				Rd			Ra			10		imm3	

Assembly:

```
SRLI Rd, Ra, amount
```

Purpose:

To perform a right logical shift of a 16-bit integer in a register by a constant bit amount.

Restrictions:

The `amount` may be between 0 and 15, inclusive.

Operation:

```
opA ← mem[W|Ra]
if (has_ext13)
then opB ← ext13|imm3
else opB ← decode_imm_sr(imm3)
res ← 0{imm3+1}|opA[16:15-imm3]
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

3.57 ST   Store

Encoding (short form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ST		01010				Rs			Ra				off5		

Encoding (long form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	110			ext13											
ST		01010				Rs			Ra				off5		

Assembly:

```
ST Rs, Ra, off
```

Purpose:

To store a word to memory at a variable address, with a constant offset.

Restrictions:

If the long form is used, and `off5[4:3]` are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off5[2:0]
else off ← sign_extend(off5)
addr ← mem[W|Ra] + off
data ← mem[W|Rs]
mem[addr] ← data
```

3.58 STR Store PC-relative

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
STR		01011					Rs			Ra			off5			

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110			ext13											
STR		01011					Rs			Ra			off5			

Assembly:

```
STR Rs, Ra, off
```

Purpose:

To store a word to memory at a constant PC-relative address, with a variable offset.

Restrictions:

If the long form is used, and **off5[4:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off5[2:0]
else off ← sign_extend(off5)
addr ← PC + 1 + off + mem[W|Ra]
data ← mem[W|Rs]
mem[addr] ← data
```

3.59 STW   Store to Window Address

Encoding:

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
STW			10100				000			000			00		Rb	

Assembly:

```
STW Rb
```

Purpose:

To arbitrarily change the address of the register window.

Restrictions:

If `Rb` contains a value that is not a multiple of 8, the behavior is
UNPREDICTABLE.

Operation:

```
W ← mem[W|Rb]
```

Remarks:

See also LDW. This instruction may be used in a function epilogue, where `Rb` is any
register chosen to act as a frame pointer.

3.60 STX   Store External

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
STX		011	10				Rs			Ra				off5		

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	110								ext13							
STX		011	10				Rs			Ra				off5		

Assembly:

```
STX Rs, Ra, off
```

Purpose:

To complete a store cycle on external bus at a variable address, with a constant offset.

Restrictions:

If the long form is used, and **off5[4:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off5[2:0]
else off ← sign_extend(off5)
addr ← mem[W|Ra] + off
data ← mem[W|Rs]
ext[addr] ← data
```

3.61 STXA   Store External Absolute

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
STXA	0	1	1	1	1		Rs					off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	1	1	0					ext13								
STXA	0	1	1	1	1		Rs					off8				

Assembly:

```
STXA Rs, off
```

Purpose:

To complete a store cycle on external bus at a constant absolute address.

Restrictions:

If the long form is used, and `off8[7:3]` are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
if (has_ext13)
then off ← ext13|off8[2:0]
else off ← sign_extend(off8)
data ← mem[W|Rs]
ext[off] ← data
```

3.62   SUB   Subtract Register

Encoding:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
SUB	00010				Rd			Ra			10		Rb		

Assembly:

```
SUB Rd, Ra, Rb
```

Purpose:

To subtract 16-bit integers in registers.

Restrictions:

None.

Operation:

```
opA ← mem[W|Ra]
opB ← mem[W|Rb]
res ← opA - opB
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← not res[16]
V ← (opA[15] = not opB[15]) and (opA[15] <> res[15])
```

3.63 SUBI   Subtract Immediate

Encoding (short form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
SUBI		00011				Rd			Ra			10		imm3	

Encoding (long form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	110							ext13							
SUBI		00011				Rd			Ra			10		imm3	

Assembly:

SUBI Rd, Ra, imm

Purpose:

To subtract a constant from a 16-bit integer in a register.

Restrictions:

None.

Operation:

```
opA ← mem[W|Ra]
if (has_ext13)
then opB ← ext13|imm3
else opB ← decode_imm_al(imm3)
res ← opA - opB
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← not res[16]
V ← (opA[15] = not opB[15]) and (opA[15] <> res[15])
```

3.64   XCHG   Exchange Registers

Assembly:

  XCHG Ra, Rb

Purpose:

  To exchange the values of two registers.

Restrictions:

  None.

Remarks:

  The assembler does not translate any instructions for XCHG with identical Ra and Rb, and translates XCHG with any other register combination to

```
XOR  Ra, Ra, Rb
XOR  Rb, Rb, Ra
XOR  Ra, Ra, Rb
```

3.65   XCHW   **Exchange Window Address**

Encoding:

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
XCHW	10100					Rd			001			00		Rb		

Assembly:

```
XCHW Rd, Rb
```

Purpose:

To exchange the address of the register window with a register.

Restrictions:

If **Rb** contains a value that is not a multiple of 8, the behavior is
UNPREDICTABLE.

Operation:

```
temp ← W
W ← mem[W|Rb]
mem[W|Rd] ← temp
```

Remarks:

This instruction may be used in a context switch routine. For example, if multiple
register windows are set up such that each contains the address of the next one in
R7, the following code may be used to switch contexts:

```
yield:
    XCHW R7, R7
    JR   R0
; Elsewhere:
    JALR R0, yield
```

3.66 XOR   **Bitwise XOR with Register**

Encoding:

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
XOR			00000				Rd			Ra			10		Rb	

Assembly:

```
XOR Rd, Ra, Rb
```

Purpose:

To perform bitwise XOR between 16-bit integers in registers.

Restrictions:

None.

Operation:

```
opA ← mem[W|Ra]
opB ← mem[W|Rb]
res ← opA xor opB
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

3.67 XORI   Bitwise XOR with Immediate

Encoding (short form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
XORI		00001				Rd			Ra			10		imm3	

Encoding (long form):

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI	110						ext13								
XORI		00001				Rd			Ra			10		imm3	

Assembly:

```
XORI Rd, Ra, imm
```

Purpose:

To perform bitwise XOR between a 16-bit integer in a register and a constant.

Restrictions:

None.

Operation:

```
opA ← mem[W|Ra]
if (has_ext13)
then opB ← ext13|imm3
else opB ← decode_imm_al(imm3)
res ← opA xor opB
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

4 List of Assembly Directives

TBD

5 Function Calling Sequence

TBD