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 Guide to Instruction Set

TBD

2 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.

2.1 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
```

2.2 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		0001	1			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		0001	1			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_immediate(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
```

2.3 ADD Add Register

Encoding:

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ADD		000	10				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])
```

2.4 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_immediate(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])
```

2.5   ADJW.1   Adjust Window

Encoding:

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ADJW.1			00111				000			000			11		imm3	

Assembly:

```
ADJW size
```

Purpose:

To adjust the position of register window.

Restrictions:

None.

Operation:

```
if (has_ext13)
then imm ← ext13|imm3
else imm ← sign_extend(imm3)
W ← W + imm
```

Remarks:

See also `LEAV`.

Notice:

The exact encoding of this instruction is not final.

2.6 ADJW.2 Adjust Window, Store Previous Address

Encoding:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
ADJW.2	00111				Rd			001			11		imm3		

Assembly:

```
ADJW Rd, size
```

Purpose:

To adjust the position of register window and store the previous window position to a register.

Restrictions:

None.

Operation:

```
if (has_ext13)
then imm ← ext13|imm3
else imm ← sign_extend(imm3)
tmp ← W
W ← W + imm
mem[W|Rd] ← tmp|000
```

Remarks:

See also ENTR.

Notice:

The exact encoding of this instruction is not final.

2.7 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
```

2.8 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 constant and a 16-bit integer in a register.

Restrictions:

None.

Operation:

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

2.9 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 is identical to `SUB`, with the exception that it discards the computed value.

2.10   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_immediate(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 is identical to SUBI, with the exception that it discards the computed value.

2.11 ENTR   Enter Frame

Assembly:

ENTR Rd, size

Purpose:

To set up a working area at an entry to a function.

Restrictions:

None.

Remarks:

The assembler translates **ENTR** to

    ADJW Rd, -size

See the description of ADJW.2. See also LEAV.

2.12 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 Rd, Ra, Rb
```

Purpose:

To extend the range of immediate in the next executed instruction.

Restrictions:

None.

Operation:

```
ext13 ← imm13
has_ext13 ← 1
```

Remarks:

This instruction is exclusively 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, the assembler does not accept a mnemonic for `EXTI`.

2.13 J   Jump

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
J		101		1		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		101		1		1111						off8				

Assembly:

```
J label
```

Purpose:

To unconditionally transfer control.

Restrictions:

If the long form is used, and `off8[8:3]` are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
PC ← PC + 1 + off
```

2.14 JAL — Jump And Link

Encoding (short form):

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

Assembly:

```
JAL label
```

Purpose:

To transfer control to a subroutine.

Restrictions:

If the long form is used, and **off8[8:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
mem[W|Rd] ← PC + 1
PC ← PC + 1 + off
```

2.15   JC   Jump if Carry

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JC		101		1		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		101		1		1010						off8				

Assembly:

```
JC label
```

Purpose:

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

Restrictions:

If the long form is used, and **off8[8:3]** are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
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.

2.16   JE   Jump if Equal

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JE	101			1	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	101			1	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[8:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
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`.

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JN		101		1		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		101		1		0111						off8				

Assembly:

```
JN label
```

Purpose:

To serve as a placeholder for a jump instruction.

Restrictions:

If the long form is used, and **off8[8:3]** 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 predefiend offset when the exact condition is unknown,
such as in certain self-modifying code.

2.18 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	101		1		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	101		1		0010							off8				

Assembly:

```
JNC label
```

Purpose:

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

Restrictions:

If the long form is used, and **off8[8:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
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.

2.19   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	101			1		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	101			1		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[8:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
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`.

2.20 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	101			1		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	101			1		0011						off8				

Assembly:

```
JNO label
```

Purpose:

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

Restrictions:

If the long form is used, and **off8[8:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
if (not V)
then PC ← PC + 1 + off
else PC ← PC + 1
```

2.21 JNS — Jump if Not Sign

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JNS		101		1		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		101		1		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[8:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
if (not S)
then PC ← PC + 1 + off
else PC ← PC + 1
```

2.22 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	101		1		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	101		1		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[8:3]** are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
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.

2.23   JO   Jump if Overflow

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JO		101		1		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		101		1		1011						off8				

Assembly:

```
JO label
```

Purpose:

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

Restrictions:

If the long form is used, and **off8[8:3]** are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
if (V)
then PC ← PC + 1 + off
else PC ← PC + 1
```

2.24   JR   Jump to Register

Encoding:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JR	101			11		Ra					00000000				

Assembly:

```
JR Rd
```

Purpose:

To transfer control to an absolute address contained in a register.

Restrictions:

None.

Operation:

```
addr ← mem[W|Ra]
PC ← addr
```

Notice:

The exact encoding of this instruction is not final.

2.25 JS   Jump if Sign

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JS	101			1		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	101			1		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[8:3]** are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
if (S)
then PC ← PC + 1 + off
else PC ← PC + 1
```

2.26 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	101			1		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	101			1		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[8:3] are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
if (not (S xor V))
then PC ← PC + 1 + off
else PC ← PC + 1
```

2.27 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	101		1		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	101		1		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[8:3]` are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
if (not ((S xor V) or Z))
then PC ← PC + 1 + off
else PC ← PC + 1
```

2.28   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		101		1		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		101		1		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[8:3] are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
if (((S xor V) or Z))
then PC ← PC + 1 + off
else PC ← PC + 1
```

2.29   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		101		1		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		101		1		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[8:3]` are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
if ((S xor V))
then PC ← PC + 1 + off
else PC ← PC + 1
```

2.30 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		101		1		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		101		1		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[8:3] are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
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.

2.31 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		101		1		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		101		1		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[8:3] are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
if (not ((not C) or V))
then PC ← PC + 1 + off
else PC ← PC + 1
```

2.32 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		101		1		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		101		1		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[8:3] are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
if ((not C) or V)
then PC ← PC + 1 + off
else PC ← PC + 1
```

2.33 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		101		1			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		101		1			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[8:3] are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
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.

2.34 JZ   Jump if Zero

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
JZ		101		1		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		101		1		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[8:3]** are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
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.

2.35 LD Load

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
LD		010	00			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		010	00			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[5:3]` are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off5[3:0]
else off ← sign_extend(off5)
addr ← mem[W|Ra] + off
temp ← mem[addr]
mem[W|Rd] ← temp
```

2.36   LDI   Load Indexed

Encoding (short form):

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

Assembly:

```
LDI 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[5:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
addr ← PC + mem[W|Ra] + off
temp ← mem[addr]
mem[W|Rd] ← temp
```

2.37 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[5:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off5[3:0]
else off ← sign_extend(off5)
addr ← mem[W|Ra] + off
temp ← ext[addr]
mem[W|Rd] ← temp
```

2.38 LDXZ   **Load External Absolute**

Encoding (short form):

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

Assembly:

```
LDXZ 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[8:3]** are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
temp ← ext[off]
mem[W|Rd] ← temp
```

2.39 LEAV   Leave Frame

Assembly:

```
LEAV Rd, size
```

Purpose:

To tear down a working area at an exit from a function.

Restrictions:

None.

Remarks:

The assembler translates **LEAV** to

```
ADJW size
```

See the description of `ADJW.1`. See also `ENTR`.

2.40   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

    ANDI Rd, Rs, Rs

Notice:

The exact translation of this mnemonic is not final.

2.41 MOVA Move Address PC-relative

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
MOVA		100			00		Rd					off8				

Encoding (long form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
EXTI		110							ext13							
MOVA		100			00		Rd					off8				

Assembly:

```
MOVA 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[8:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
mem[W|Rd] ← PC + 1 + off
```

2.42 MOVI   Move Immediate

Encoding (short form):

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
MOVI		100		01		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		100		01		Rd						imm8				

Assembly:

```
MOVI Rd, imm
```

Purpose:

To load a register with a constant.

Restrictions:

If the long form is used, and **off8[8:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then imm ← ext13|imm8[3:0]
else imm ← sign_extend(imm8)
mem[W|Rd] ← imm
```

2.43 OR   Bitwise OR with Register

Encoding:

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
OR		00	000				Rd			Ra		0	1		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
```

2.44 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 constant and a 16-bit integer in a register.

Restrictions:

None.

Operation:

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

2.45   ROLI   Rotate Left Immediate

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.

Remarks:

This instruction is an alias for `ROTI`.

2.46   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

```
MOV  Rd, Ra
```

and `RORI` with any other `amount` to

```
ROTI Rd, Rd, (15 - amount)
```

2.47 ROT   Rotate

Encoding:

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

Assembly:

```
ROT 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
```

2.48 ROTI   Rotate Immediate

Encoding:

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

Assembly:

```
ROTI 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]
res ← opA[15-imm3:0]|opA[16:15-imm3]
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

Remarks:

The instruction encoding allows directly representing any **amount** between 1 and 8, inclusive. The assembler translates **ROTI** with **amount** of 0 to

```
MOV  Rd, Ra
```

and **ROTI** with **amount** greater than 8 to

```
ROTI Rd, Ra, 8
ROTI Rd, Rd, (amount - 8)
```

53

2.49   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
```

2.50 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			1	1		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			1	1		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_immediate(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
```

2.51 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
```

2.52 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]
res ← opA[15-imm3:0]|0{imm3+1}
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

Remarks:

The instruction encoding allows directly representing any `amount` between 1 and 8, inclusive. The assembler translates `SLLI` with `amount` of 0 to

```
MOV  Rd, Ra
```

and `SLLI` with `amount` greater than 8 to

```
SLLI Rd, Ra, 8
SLLI Rd, Rd, (amount - 8)
```

2.53 SRA   Shift Right Arithmetical

Encoding:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
SRA	00100				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
```

2.54 SRAI   Shift Right Arithmetical Immediate

Encoding:

	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
SRAI		0	0 1 0 0				Rd			Ra		1	1		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]
opB ← mem[W|Rb]
res ← opA[15]{imm3+1}|opA[16:15-imm3]
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

Remarks:

The instruction encoding allows directly representing any `amount` between 1 and 8, inclusive. The assembler translates `SRAI` with `amount` of 0 to

```
MOV  Rd, Ra
```

and `SRAI` with `amount` greater than 8 to

```
SRAI Rd, Ra, 8
SRAI Rd, Rd, (amount - 8)
```

2.55 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
```

2.56   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]
res ← 0{imm3+1}|opA[16:15-imm3]
mem[W|Rd] ← res
Z ← res = 0
S ← res[15]
C ← UNDEFINED
V ← UNDEFINED
```

Remarks:

The instruction encoding allows directly representing any **amount** between 1 and 8, inclusive. The assembler translates **SRLI** with **amount** of 0 to

```
MOV  Rd, Ra
```

and **SRLI** with **amount** greater than 8 to

```
SRLI Rd, Ra, 8
SRLI Rd, Rd, (amount - 8)
```

2.57 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[5:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off5[3:0]
else off ← sign_extend(off5)
addr ← mem[W|Ra] + off
temp ← mem[W|Rs]
mem[addr] ← temp
```

2.58   STI   Store Indexed

Encoding (short form):

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

Assembly:

```
STI 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[5:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
addr ← PC + mem[W|Ra] + off
temp ← mem[W|Rs]
mem[addr] ← temp
```

2.59 STX   Store External with Address in Register

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[5:3]** are non-zero, the behavior is **UNPREDICTABLE**.

Operation:

```
if (has_ext13)
then off ← ext13|off5[3:0]
else off ← sign_extend(off5)
addr ← mem[W|Ra] + off
temp ← mem[W|Rs]
ext[addr] ← temp
```

2.60   STXZ   Store External Absolute

Encoding (short form):

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

Assembly:

```
STXZ 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[8:3]** are non-zero, the behavior is
UNPREDICTABLE.

Operation:

```
if (has_ext13)
then off ← ext13|off8[3:0]
else off ← sign_extend(off8)
temp ← mem[W|Rs]
ext[off] ← temp
```

2.61 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])
```

2.62 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_immediate(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])
```

2.63   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
```

2.64 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
```

2.65 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 constant and a 16-bit integer in a register.

Restrictions:

None.

Operation:

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