RFC ls007 Ternary/Binary GPR and CR Field bit-operations </>

- Funded by NLnet under the Privacy and Enhanced Trust Programme, EU Horizon2020 Grant 825310, and NGI0 Entrust No 101069594

- https://libre-soc.org/openpower/sv/bitmanip/
- https://libre-soc.org/openpower/sv/rfc/ls007/
- https://bugs.libre-soc.org/show_bug.cgi?id=1017
- https://git.openpower.foundation/isa/PowerISA/issues/117

Severity: Major

Status: new

Date: 20 Oct 2022, 1st draft submitted 2023mar22

Target: v3.2B

Source: v3.1B

Books and Section affected: **UPDATE**

- Book I 2.5.1 Condition Register Logical Instructions
- Book I 3.3.13 Fixed-Point Logical Instructions
- Appendix E Power ISA sorted by opcode
- Appendix F Power ISA sorted by version
- Appendix G Power ISA sorted by Compliancy Subset
- Appendix H Power ISA sorted by mnemonic

Summary

Instructions added

- `ternlogi` – GPR Ternary Logic Immediate
- `crternlogi` – Condition Register Field Ternary Logic Immediate
- `binlog` – GPR Dynamic Binary Logic
- `crbinlog` – Condition Register Field Dynamic Binary Logic

Submitter: Luke Leighton (Libre-SOC)

Requester: Libre-SOC

Impact on processor:

- Addition of two new GPR-based instructions
- Addition of two new CR-field-based instructions

Impact on software:

- Requires support for new instructions in assembler, debuggers, and related tools.

Keywords:

`GPR, CR-Field, bit-manipulation, ternary, binary, dynamic, look-up-table (LUT), FPGA, JIT`

Motivation

- `ternlogi` is similar to existing `and`/`or`/`xor`/etc. instructions, but allows any arbitrary 3-input 1-output bitwise operation. This can be used to combine several instructions into one. E.g. `A ^ (~B & (C | A))` can become one instruction. This can also be used to have one instruction for bitwise MUX `(A & B) | (~A & C)`.
- `binlog` is like `ternlogi` except it supports any arbitrary 2-input 1-output bitwise operation, where the operation can be selected dynamically at runtime. This operates similarly to a LUT in a FPGA.
- `crternlogi` is like `ternlogi` except it works with CRs instead of GPRs.
- `crbinlog` is like `binlog` except it works with CRs instead of GPRs. Likewise it is similar to a LUT in an FPGA.
- Combined these instructions save on instruction count and also help accelerate AI and JIT runtimes.

Notes and Observations:

- `ternlogi` is like the existing `xxeval` instruction, except operates on GPRs instead of VSRs and does not require VSX/VMX. SFS and SFFS are comparatively compromised.
- SVP64/VSX may have different semantics from SVP64/SFFS. SVP64 orthogonality is compromised by a *non*-Vector-considerate argument that if equivalent instructions are in VSX they do not have to be added to SFFS: they do.
- `crternlogi` is similar to the group of CR Operations (crand, cror etc) which have been identified as a Binary Lookup Group, except an 8-bit immediate is used instead of a 4-bit one, and up to 4 bits of a CR Field may be computed at once, saving at least 3 groups of CR operations.
- `crbinlut` is similar to the Binary Lookup Group of CR Operations except that the 4-bit lookup table comes from a CR Field instead of from an Immediate. Also like `crternlogi` up to 4 bits may be computed at once.

Changes

Add the following entries to:

- Book I 2.5.1 Condition Register Logical Instructions
- Book I 3.3.13 Fixed-Point Logical Instructions
- Book I 1.6.1 and 1.6.2

GPR Ternary Logic Immediate </>

Add this section to Book I 3.3.13

TLI-form

0-5	6-10	11-15	16-20	21-28	29-30	31	Form
PO	RT	RA	RB	TLI	XO	Rc	TLI-Form

- `ternlogi RT, RA, RB, TLI` (`Rc=0`)
- `ternlogi. RT, RA, RB, TLI` (`Rc=1`)

Pseudocode:

```
result <- (~RT & ~RA & ~RB & TLI[0]*64) | # 64 copies of TLI[0]
          (~RT & ~RA &  RB & TLI[1]*64) | # ...
          (~RT &  RA & ~RB & TLI[2]*64) |
          (~RT &  RA &  RB & TLI[3]*64) |
          ( RT & ~RA & ~RB & TLI[4]*64) |
          ( RT & ~RA &  RB & TLI[5]*64) |
          ( RT &  RA & ~RB & TLI[6]*64) | # ...
          ( RT &  RA &  RB & TLI[7]*64)   # 64 copies of TLI[7]
RT <- result
```

For each integer value i, 0 to 63, do the following.

```
Let j be the value of the concatenation of the
contents of bit i of RT, bit i of RB, bit i of RT.
The value of bit j of TLI is placed into bit i of RT.

See Table 145, "xxeval(A, B, C, TLI) Equivalent
Functions," on page 968 for the equivalent function
evaluated by this instruction for any given value of TLI.
```

Programmer's Note: this is a Read-Modify-Write instruction on RT. A simple copy instruction may be used to achieve the effect of 3-in 1-out. The copy instruction should come immediately before ***`ternlogi`*** *so that hardware may optionally Macro-Op Fuse them*

Programmer's note: This instruction is useful when combined with Matrix REMAP in "Inner Product" Mode, creating Warshall Transitive Closure that has many applications in Computer Science.

Special registers altered:

```
CR0                     (if Rc=1)
```

Condition Register Ternary Logic Immediate </>

Add this section to Book I 2.5.1

CRB-form

0.5	6.8	9.10	11.13	14.15	16.18	19.25	26.30	31	Form
PO	BF	msk	BFA	msk	BFB	TLI	XO	TLI	CRB-Form

- `crternlogi BF, BFA, BFB, BFC, TLI, msk`

Pseudocode:

```
a <- CR[4*BF+32:4*BF+35]
b <- CR[4*BFA+32:4*BFA+35]
c <- CR[4*BFB+32:4*BFB+35]
ternary <- (~a & ~b & ~c & TLI[0]*4) | # 4 copies of TLI[0]
           (~a & ~b &  c & TLI[1]*4) | # 4 copies of TLI[1]
           (~a &  b & ~c & TLI[2]*4) | # ...
           (~a &  b &  c & TLI[3]*4) |
           ( a & ~b & ~c & TLI[4]*4) |
           ( a & ~b &  c & TLI[5]*4) |
           ( a &  b & ~c & TLI[6]*4) | # ...
           ( a &  b &  c & TLI[7]*4))  # 4 copies of TLI[7]
do i = 0 to 3
    if msk[i] = 1 then
        CR[4*BF+32+i] <- ternary[i]
```

For each integer value i, 0 to 3, do the following.

```
Let j be the value of the concatenation of the
contents of bit i of CR Field BF, bit i of CR Field BFA,
bit i of CR Field BFB.

If bit i of msk is set to 1 then the value of bit j of TLI
is placed into bit i of CR Field BF.

Otherwise, if bit i of msk is a zero then bit i of
CR Field BF is unchanged.

See Table 145, "xxeval(A, B, C, TLI) Equivalent
Functions," on page 968 for the equivalent function
evaluated by this instruction for any given value of TLI.
```

If `msk` is zero an Illegal Instruction trap is raised.

Programmer's Note: this instruction is a "masked" overwrite on CR Field BF. For each bit set in msk a Write is performed but for each bit clear in msk the corresponding bit of BF is preserved. Overall this makes crbinlog a conditionally Read-Modify-Write instruction on CR Field BF. A simple copy instruction may be used to achieve the effect of 3-in 1-out. The copy instruction should come immediately before `crternlogi` so that hardware may optionally Macro-Op Fuse them

Special registers altered:

```
CR field BF
```

GPR Dynamic Binary Logic </>

Add this section to Book I 3.3.13

VA-form

0-5	6-10	11-15	16-20	21-25	26	27-31	Form
PO	RT	RA	RB	RC	nh	XO	VA-Form

- `binlog RT, RA, RB, RC, nh`

Pseudocode:

```
if nh = 1 then lut <- (RC)[56:59]
else           lut <- (RC)[60:63]
result <- (~RA & ~RB & lut[0]*64) |
          (~RA &  RB & lut[1]*64) |
          ( RA & ~RB & lut[2]*64) |
          ( RA &  RB & lut[3]*64))
RT <- result
```

For each integer value i, 0 to 63, do the following.

```
If nh contains a 0, let lut be the four LSBs of RC
(bits 60 to 63).  Otherwise let lut be the next
four LSBs of RC (bits 56 to 59).

Let j be the value of the concatenation of the
contents of bit i of RT with bit i of RB.

The value of bit j of lut is placed into bit i of RT.
```

Special registers altered:

```
None
```

Programmer's Note:

Dynamic (non-immediate-based) Ternary Logic, suitable for FPGA-style LUT3 dynamic lookups and for JIT runtime acceleration, may be emulated by appropriate combination of `binlog` and `ternlogi`, using the `nh` (next half) operand to select first and second nibble:

```
# compute r3 = ternlog(r4, r5, r6, table=r7)
# compute the values for when r6[i] = 0:
binlog r3, r4, r5, r7, 0  # takes look-up-table from LSB 4 bits
# compute the values for when r6[i] = 1:
binlog r4, r4, r5, r7, 1  # takes look-up-table from second-to-LSB 4 bits
# mux the two results together: r3 = (r3 & ~r6) | (r4 & r6)
ternlogi r3, r4, r6, 0b11011000
```

Condition Register Field Dynamic Binary Logic </>

Add this section to Book I 2.5.1

CRB-form

0.5	6.8	9.10	11.13	14.15	16.18	19.25	26.30	31	Form
PO	BF	msk	BFA	msk	BFB	//	XO	//	CRB-Form

- `crbinlog BF, BFA, BFB, msk`

Pseudocode:

```
a <- CR[4*BF+32:4*BFA+35]
b <- CR[4*BFA+32:4*BFA+35]
lut <- CR[4*BFB+32:4*BFB+35]
binary <- (~a & ~b & lut[0]*4) |
          (~a &  b & lut[1]*4) |
          ( a & ~b & lut[2]*4) |
          ( a &  b & lut[3]*4))
do i = 0 to 3
    if msk[i] = 1 then
        CR[4*BF+32+i] <- binary[i]
```

For each integer value i, 0 to 3, do the following.

```
Let j be the value of the concatenation of the
contents of bit i of CR Field BF with bit i of CR Field BFA.

If bit i of msk is set to 1 then the value of bit j of
CR Field BFB is placed into bit i of CR Field BF.

Otherwise, if bit i of msk is a zero then bit i of
CR Field BF is unchanged.
```

If `msk` is zero an Illegal Instruction trap is raised.

Special registers altered:

```
CR field BF
```

Programmer's Note: just as with binlut and ternlogi, a pair of crbinlog instructions followed by a merging crternlogi may be deployed to synthesise dynamic ternary (LUT3) CR Field manipulation

Programmer's Note: this instruction is a "masked" overwrite on CR Field BF. For each bit set in `msk` *a Write is performed but for each bit clear in* `msk` *the corresponding bit of BF is preserved. Overall this makes* `crbinlog` *a conditionally Read-Modify-Write instruction on CR Field BF. A simple copy instruction may be used to achieve the effect of 3-in 1-out. The copy instruction should come immediately before* `crternlogi` *so that hardware may optionally Macro-Op Fuse them*

[[!tag standards]]

Forms </>

CRB-FORM </>

Add the following section to Book I 1.6.1

```
    |0   |6   |9    |11   |14   |16   |19   |26  |31   |
    | PO | BF | msk | BFA | msk | BFB | TLI | XO | TLI |
    | PO | BF | msk | BFA | msk | BFB | //  | XO | /   |
```

TLI-FORM </>

Add the following section to Book I 1.6.1

```
    |0   |6   |11  |16  |21   |29  |31  |
    | PO | RT | RA | RB | TLI | XO | Rc |
```

VA-FORM </>

Add the following entry to VA-FORM in Book I 1.6.1.12

```
    |0      |6     |11     |16     |21     |26|27   |
    | PO    |  RT  |   RA  |   RB  |    RC |nh| XO  |
```

Word Instruction Fields </>

Add the following to Book I 1.6.2

```
msk (9:10,14:15)
    Field used by crternlogi and crbinlut to decide which CR Field bits to
    modify.
    Formats: CRB
nh (26)
    Nibble High. Field used by binlog to decide if the look-up-table should
    be taken from bits 60:63 (nh=0) or 56:59 (nh=1) of RC.
    Formats: VA
TLI (21:28)
    Field used by the ternlogi instruction as the
    look-up table.
    Formats: TLI
TLI (21:25,19:20,31)
     Field used by the crternlogi instruction as the
     look-up table.
     Formats: CRB
```

- Add `TLI` to the `Formats:` list of all of `RA`, `RB`, `RT`, and `Rc`.
- Add `CRB` to the `Formats:` list of all of `BF`, `BFA`, `BFB`, and `BFC`.
- Add `TLI` to the `Formats:` list of `XO (29:30)`.
- Add `CRB` to the `Formats:` list of `XO (26:31)`.
- Add `VA` to the `Formats:` list of `XO (27:31)`.

Appendices </>

```
Appendix E Power ISA sorted by opcode
Appendix F Power ISA sorted by version
Appendix G Power ISA sorted by Compliancy Subset
Appendix H Power ISA sorted by mnemonic
```

Form	Book	Page	Version	mnemonic	Description
TLI	I	#	3.2B	ternlogi	GPR Ternary Logic Immediate
VA	I	#	3.2B	binlog	GPR Binary Logic
CRB	I	#	3.2B	crternlogi	CR Field Ternary Logic Immediate
CRB	I	#	3.2B	crbinlog	CR Field Binary Logic

[[!tag opf_rfc]]
