Volt32 CPU

Andrew Clark

March 9, 2021

Table of Contents

Registers, Main Widths, etc.

- The main width of the processor is 32-bit, and addresses are 32-bit. Some 64-bit operations exist.
- The machine is an implementation of Line Associative Registers (LARs). Both instruction LARs (ILARs) and data LARs (DLARs) are included in the design. There are a grand total of 128 ILARs and 128 DLARs, but they are split between the LARs owned by the supervisor mode and the LARs owned by the user mode. There are 64 supervisor mode ILARs, 64 supervisor mode DLARs, 64 user mode ILARs, and 64 user mode DLARs.
- The machine boots in supervisor mode. The processor jumps to address `0x0` when it enters supervisor mode, which includes when the machine boots.
- ILARs
  - In user mode, ILARs 0 to 63 are referred to as `i0`, `i1`, `i2`, ..., `i61`, `i62`, `ipc`.
  - In supervisor mode, ILARs 64 to 127 are referred to as `i0`, `i1`, `i2`, ..., `i61`, `i62`, `ipc`.
  - The two ILARs called "`i0`" have all their fields set to zero, and when written to, the contents of the ILAR does not change.
  - The two ILARs called "`ipc`" are the program counters for the two operating modes of the processor.
  - An ILAR's data field is 128 bytes long. It is composed of 32-bit instructions aligned to 32 bits.
  - An ILAR's scalar offset field is 5-bit due to instructions being 32-bit and the data field being 128 bytes long.
  - The base address field of an ILAR is (32 - 7 = 25)-bit.
  - An ILAR's tag field is 6-bit.
- DLARs
  - In user mode, DLARs 0 to 63 are referred to as `d0`, `d1`, `d2`, ..., `d59`, `d60`, `dcp`, `dfp`, `dsp`.
  - In supervisor mode, DLARs 64 to 127 are referred to as `d0`, `d1`, `d2`, ..., `d59`, `d60`, `dcp`, `dfp`, `dsp`.
  - The two DLARs called "`d0`" have all their fields set to zero, and when written to, the contents of the DLAR does not change.
  - A DLAR's data field is 128 bytes long. It is composed of the scalar data elements of the 128 byte vectors, where the type of the scalar data elements is determined by the type tag field of the DLAR.
  - A DLAR's scalar offset field is 7-bit due to the data field being 128 bytes long.

  - The base address field of a DLAR is (32 - 7 = 25)-bit.
  - DLARs can take on the following types (3-bit enum):
    * 8-bit, unsigned (`u8`)
    * 8-bit, signed (`i8`)
    * 16-bit, unsigned (`u16`)
    * 16-bit, signed (`i16`)
    * 32-bit, unsigned (`u32`)
    * 32-bit, signed (`i32`)
    * 64-bit, unsigned (`u64`); only usable for some operations
    * 64-bit, signed (`i64`); only usable for some operations
  - A DLAR's tag field is 6-bit.

- The `ie` register
  - This register is 1-bit.
  - This register is a flag indicating whether or not an interrupt can be serviced. It can be written to with the `cpy` instruction (with `ie` as the destination operand), and it can be read from with the `cpy` instruction (with `ie` as the source operand).
  - The `reti` instruction sets `ie` to `0b1` and returns to user mode from supervisor mode.

- The `xct` register
  - This register is 32-bit.
  - This register gets set to one of the following values upon the machine entering supervisor mode. It can be written to using the `cpy` instruction (with `xct` as the destination operand), and it can be read from using the `cpy` instruction (with `xct` as the source operand).

Exceptions

Some instructions may cause an exception to occur, putting the processor in supervisor mode.

The following exceptions may occur during normal execution of a program.

- Taking an interrupt, which also sets `ie` to `0b0`.
- Division by zero.
- Undefined instruction.
- Instructions where 64-bit ops are not defined.
- `swi`.
  - Note that this instruction always causes an exception to occur.
- `reti` in user mode.
- `cpy` that reads from/writes to `ie` when in user mode.
- `cpy` that reads from/writes to `xct` when in user mode.
- Instructions that read from/write to supervisor mode ILARs or DLARs when in user mode.

Instructions (CPU's perspective)

4.1 Group 0 Instructions

- Encoding: `0000 aaaa aabb bbbb cccc ccdd dddv oooo`
  - `a`: DLAR a
  - `b`: DLAR b
  - `c`: DLAR c
  - `d`: DLAR d (`d32` to `d63`)
  - `v`:
    - when `0b0`: scalar operation
    - when `0b1`: vector operation
  - `o`: Opcode
- Instruction List:
  1. `add dA, dB, dC`
     - This instruction causes an exception if `dA` is of the following types: `u64`, `i64`.
  2. `sub dA, dB, dC`
     - This instruction causes an exception if `dA` is of the following types: `u64`, `i64`.
  3. `slt dA, dB, dC`
     - This instruction causes an exception if `dA` is of the following types: `u64`, `i64`.
  4. `mul dA, dB, dC`
     - This instruction causes an exception if `dB` or `dC` is of the following types: `u64`, `i64`.
  5. `div dA, dB, dC, dD`
     - This instruction causes an exception if `dB`, `dC`, or `dD` is of the following types: `u64`, `i64`.
     - This instruction writes the quotient into `dA`, and the remainder into `dD`.
  6. `and dA, dB, dC`
  7. `or dA, dB, dC`
  8. `xor dA, dB, dC`
  9. `shl dA, dB, dC`
     - Logical shift left.
     - This instruction causes an exception if `dA`, `dB`, or `dC` is of the following types: `u64`, `i64`.

- This instruction casts a temporary copy of `dC` to the unsigned type that is the same size as `dA`'s type and uses that instead of `dC`.

10. `shr dA, dB, dC`
    - Logical shift right if `dA` is unsigned, or arithmetic shift right if `dA` is signed.
    - This instruction causes an exception if `dA`, `dB`, or `dC` is of the following types: `u64`, `i64`.
    - This instruction casts a temporary copy of `dC` to the unsigned type that is the same size as `dA`'s type and uses that instead of `dC`.
11. `rol dA, dB, dC`
    - Rotate left.
    - This instruction causes an exception if `dA`, `dB`, or `dC` is of the following types: `u64`, `i64`.
    - This instruction casts a temporary copy of `dC` to the unsigned type that is the same size as `dA`'s type and uses that instead of `dC`.
12. `ror dA, dB, dC`
    - Rotate right.
    - This instruction causes an exception if `dA`, `dB`, or `dC` is of the following types: `u64`, `i64`.
    - This instruction casts a temporary copy of `dC` to the unsigned type that is the same size as `dA`'s type and uses that instead of `dC`.
13. `add dA, dB.addr, dC`
    - This instruction causes an exception if `dA` or `dC` is of the following types: `u64`, `i64`.
    - For `add.v dA, dB.addr, dC`, this instruction duplicates `cast(dA.type, dB.addr)` (a scalar) into a temporary (128 bytes long) vector of element type `dA.type`.
14. `shl dA, dB.addr, dC`
    - This instruction causes an exception if `dA` or `dC` is of the following types: `u64`, `i64`.
    - For `shl.v dA, dB.addr, dC`, this instruction duplicates `cast(dA.type, dB.addr)` (a scalar) into a temporary (128 bytes long) vector of element type `dA.type`.
    - This instruction casts a temporary copy of `dC` to the unsigned type that is the same size as `dA`'s type and uses that instead of `dC`.
15. *Reserved for future expansion.*
16. *Reserved for future expansion.*

4.2 Group 1 Instructions

- Encoding: `0001 aaaa aabb bbbb cccc ccii iiii iooo`
  - `a`: DLAR a
  - `b`: DLAR b
  - `c`: DLAR c
  - `i`: `simm7` (sign-extended 7-bit immediate)
  - `o`: Opcode
- For these instructions, the `dB` register's scalar data field (temporarily casted to the `u32` type) and the `dC` register's address field are used.
- Instruction List:
  1. `ldu8 dA, dB, dC, simm7`
  2. `ldi8 dA, dB, dC, simm7`
  3. `ldu16 dA, dB, dC, simm7`
  4. `ldi16 dA, dB, dC, simm7`
  5. `ldu32 dA, dB, dC, simm7`
  6. `ldi32 dA, dB, dC, simm7`
  7. `ldu64 dA, dB, dC, simm7`
  8. `ldi64 dA, dB, dC, simm7`

4.3 Group 2 Instructions

- Encoding: `0010 aaaa aabb bbbb cccc ccii iiii iooo`
  - `a`: DLAR a
  - `b`: DLAR b
  - `c`: DLAR c
  - `i`: `simm7` (sign-extended 7-bit immediate)
  - `o`: Opcode
- For these instructions, the `dB` register's scalar data field (temporarily casted to the `u32` type) and the `dC` register's address field are used.
- Instruction List:
  1. `stu8 dA, dB, dC, simm7`
  2. `sti8 dA, dB, dC, simm7`
  3. `stu16 dA, dB, dC, simm7`
  4. `sti16 dA, dB, dC, simm7`
  5. `stu32 dA, dB, dC, simm7`
  6. `sti32 dA, dB, dC, simm7`
  7. `stu64 dA, dB, dC, simm7`
  8. `sti64 dA, dB, dC, simm7`

4.4 Group 3 Instructions

- Encoding: `0011 aaaa aabb bbbb 0000 0000 0000 0ooo`
  - `a`: DLAR a
  - `b`: DLAR b
  - `o`: Opcode
- For these instructions, the `dB` register's scalar data field is used.
- Instruction List:
  1. `dpu8 dA, dB`
     - This instruction casts (a temporary copy of) the scalar data of `dB` to the `u8` type. The casted scalar data is then stored into every `u8` vector element of `dA`. The type of `dA` is then changed to `u8`.
  2. `dpi8 dA, dB`
     - This instruction casts (a temporary copy of) the scalar data of `dB` to the `i8` type. The casted scalar data is then stored into every `i8` vector element of `dA`. The type of `dA` is then changed to `i8`.
  3. `dpu16 dA, dB`
     - This instruction casts (a temporary copy of) the scalar data of `dB` to the `u16` type. The casted scalar data is then stored into every `u16` vector element of `dA`. The type of `dA` is then changed to `u16`.
  4. `dpi16 dA, dB`
     - This instruction casts (a temporary copy of) the scalar data of `dB` to the `i16` type. The casted scalar data is then stored into every `i16` vector element of `dA`. The type of `dA` is then changed to `i16`.
  5. `dpu32 dA, dB`
     - This instruction casts (a temporary copy of) the scalar data of `dB` to the `u32` type. The casted scalar data is then stored into every `u32` vector element of `dA`. The type of `dA` is then changed to `u32`.
  6. `dpi32 dA, dB`
     - This instruction casts (a temporary copy of) the scalar data of `dB` to the `i32` type. The casted scalar data is then stored into every `i32` vector element of `dA`. The type of `dA` is then changed to `i32`.
  7. `dpu64 dA, dB`
     - This instruction casts (a temporary copy of) the scalar data of `dB` to the `u64` type. The casted scalar data is then stored into every `u64` vector element of `dA`. The type of `dA` is then changed to `u64`.
  8. `dpi64 dA, dB`
     - This instruction casts (a temporary copy of) the scalar data of `dB` to the `i64` type. The casted scalar data is then stored into every `i64` vector element of `dA`. The type of `dA` is then changed to `i64`.

4.5 Group 4 Instructions

- Encoding: `0100 aaaa aabb bbbb cccc ccii iiii iiii`
  - `a`: ILAR a
  - `b`: ILAR b
  - `c`: DLAR c
  - `i`: `simm10` (sign-extended 10-bit immediate)
- This instruction uses the address field of `iB` and the scalar data field (temporarily casted to the `u32` type) of the `dC` register.
- Instruction List:
  1. `fetch iA, iB, dC, simm10`

4.6 Group 5 Instructions

- Encoding: `0101 aaaa aabb bbbb iiii ijjj jj0v oooo`
  - `a`: DLAR a
  - `b`: ILAR b
  - `i`: `iimm5` (zero-extended 5-bit immediate i)
  - `j`: `jimm5` (zero-extended 5-bit immediate j)
  - `v`:
    - when `0b0`: scalar operation (uses the scalar data of `dA`)
    - when `0b1`: vector operation (uses the vector data of `dA`)
  - `o`: Opcode
- These instructions use the scalar or vector data field of `dA`.
- Instruction List:
  1. `sel dA, iB, iimm5, jimm5`
     - This instruction jumps to `iB[iimm5 << 2]` if the particular data field of `dA` is non-zero, otherwise to the address `iB[jimm5 << 2]`.
  2. `jz dA, iB, iimm5`
     - This instruction jumps to `iB[iimm5 << 2]` if the particular data field of `dA` is zero.
  3. `jnz dA, iB, iimm5`
     - This instruction jumps to `iB[iimm5 << 2]` if the particular data field of `dA` is non-zero.
  4. `reti dA`
     - This instruction returns from an interrupt if in supervisor mode and `dA` is non-zero, setting `ie`
  5. *Reserved for future expansion.*
  6. *Reserved for future expansion.*
  7. *Reserved for future expansion.*
  8. *Reserved for future expansion.*
  9. *Reserved for future expansion.*
  10. *Reserved for future expansion.*
  11. *Reserved for future expansion.*
  12. *Reserved for future expansion.*
  13. *Reserved for future expansion.*
  14. *Reserved for future expansion.*
  15. *Reserved for future expansion.*
  16. *Reserved for future expansion.*