Frost64 GPU

Andrew Clark

July 3, 2020

1 Table of Contents

Contents

2 Registers, Main Widths, etc.

- The main width of the processor, hinted at with the name, is 64-bit. Addresses are 64-bit, and some instructions only support 64-bit operations.
- Instructions LARs (ILARs)
  - There are 128 total ILARs, with 64 of them used for supervisor mode and 64 of them used for user mode.
  - The two ILARs named `i0`, which are ILAR number 0 and ILAR number 64 within the ILAR file, have their fields always set to zero. They cannot be changed.
  - ILARs are 256 bytes long, and instructions are 32-bit. This means that one ILAR holds 256 / 4 = 64 instructions.
  - Note that supervisor mode ILARs are neither readable nor writable in user mode.
- Data LARs (DLARs)
  - There are 128 total DLARs, with 64 used for supervisor mode and 64 of them used for user mode.
  - The two DLARs named `dzero`, which are DLAR number 0 and DLAR number 128 within the DLAR file, have their fields always set to zero. They cannot be changed.
  - The supervisor mode DLAR named `dira`, which is DLAR 126 within the DLAR file (index `64` when in supervisor mode), is treated specially, as its scalar data field is the interrupt return address for the `reti` instruction.
  - The supervisor mode DLAR named `dida`, which is DLAR 127 within the DLAR file (index `65` when in supervisor mode), is used as the interrupt vector.
  - DLARs are 256 bytes long.
  - DLARs can take on the following type tags (3-bit enum):
    - 8-bit, unsigned
    - 8-bit, signed
    - 16-bit, unsigned
    - 16-bit, signed
    - 32-bit, unsigned
    - 32-bit, signed
    - 64-bit, unsigned
    - 64-bit, signed
  - The base address of a DLAR is 64 - 8 = 56 bits long.
  - The scalar offset of a DLAR is 8 bits long.

- Program Counter (PC)
    - The program counter consists of an ILAR number (6-bit) and an offset into said ILAR (6-bit).
    - Two program counters exist: one for supervisor mode, and one for user mode.
- Interrupt Enable (`ie`)
    - This 1-bit register indicates whether or not an interrupt can be serviced. It can only be written to in supervisor mode, and it cannot be read at all.

3 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.

- Division by zero.
- `swi`.
  - Note that this instruction always causes an exception to occur.
- `reti` in user mode.
- `ei` in user mode.
- `di` in user mode.

4 Instructions (CPU's perspective)

4.1 Group 0 Instructions

- Encoding: `0000 aaaa aabb bbbb cccc cc00 000v oooo`
  - `a`: DLAR a
  - `b`: DLAR b
  - `c`: DLAR c
  - `v`:
    - when `0`: scalar operation
    - when `1`: vector operation
  - `o`: opcode
- Instruction List:
  1. `add dA, dB, dC`
  2. `sub dA, dB, dC`
  3. `slt dA, dB, dC`
     - Perform an unsigned set less than if `dB` is unsigned, but perform a signed set less than if `dB` is signed.
  4. `mul dA, dB, dC`
  5. `div dA, dB, dC`
     - Perform an unsigned divide if `dB` is unsigned, but perform a signed divide if `dB` is signed. This instruction causes an exception if division by zero is attempted.
  6. `and dA, dB, dC`
  7. `orr dA, dB, dC`
  8. `xor dA, dB, dC`
  9. `shl dA, dB, dC`
     - Logical shift left
  10. `shr dA, dB, dC`
      - Logical shift right if `dB` is unsigned, but arithmetic shift right if `dB` is signed.
  11. `rol dA, dB, dC`
  12. `ror dA, dB, dC`
  13. *Reserved for future expansion.*
  14. *Reserved for future expansion.*
  15. *Reserved for future expansion.*
  16. *Reserved for future expansion.*

4.2 Group 1 Instructions

- Encoding: `0001 aaaa aabb bbbb iiii iiii iiii oooo`
  - `a`: DLAR a
  - `b`: DLAR b
  - `i`: signed 12-bit immediate, `simm12`
  - `o`: opcode
- Notes: These instructions compute the address to load from or store to via the formula `dB.scalar_data + to_s64(simm12)`
- Instruction List:
  1. `ldu8 dA, [dB, simm12]`
  2. `lds8 dA, [dB, simm12]`
  3. `ldu16 dA, [dB, simm12]`
  4. `lds16 dA, [dB, simm12]`
  5. `ldu32 dA, [dB, simm12]`
  6. `lds32 dA, [dB, simm12]`
  7. `ldu64 dA, [dB, simm12]`
  8. `lds64 dA, [dB, simm12]`
  9. `stu8 dA, [dB, simm12]`
  10. `sts8 dA, [dB, simm12]`
  11. `stu16 dA, [dB, simm12]`
  12. `sts16 dA, [dB, simm12]`
  13. `stu32 dA, [dB, simm12]`
  14. `sts32 dA, [dB, simm12]`
  15. `stu64 dA, [dB, simm12]`
  16. `sts64 dA, [dB, simm12]`

4.3 Group 2 Instructions

- Encoding: `0010 aaaa aabb bbbb cccc ccii iiii oooo`
  - `a`: DLAR a
  - `b`: DLAR b
  - `c`: DLAR c
  - `i`: signed 6-bit immediate, `simm6`
  - `o`: opcode

- Notes: These instructions compute the address to load from or store to via the formula `dB.scalar_data + dC.addr + to_s64(simm12)`
- Instruction List:

1. `ldau8 dA, [dB, dC, simm6]`
2. `ldas8 dA, [dB, dC, simm6]`
3. `ldau16 dA, [dB, dC, simm6]`
4. `ldas16 dA, [dB, dC, simm6]`
5. `ldau32 dA, [dB, dC, simm6]`
6. `ldas32 dA, [dB, dC, simm6]`
7. `ldau64 dA, [dB, dC, simm6]`
8. `ldas64 dA, [dB, dC, simm6]`
9. `stau8 dA, [dB, dC, simm6]`
10. `stas8 dA, [dB, dC, simm6]`
11. `stau16 dA, [dB, dC, simm6]`
12. `stas16 dA, [dB, dC, simm6]`
13. `stau32 dA, [dB, dC, simm6]`
14. `stas32 dA, [dB, dC, simm6]`
15. `stau64 dA, [dB, dC, simm6]`
16. `stas64 dA, [dB, dC, simm6]`

4.4 Group 3 Instructions

- Encoding: `0011 aaaa aabb bbbb cccc ccii iiii jjjj`
  - `a`: ILAR a
  - `b`: DLAR b
  - `c`: ILAR c
  - `i`: signed 6-bit immediate, `isimm6`
  - `j`: unsigned 4-bit immediate, `jimm4`
- Notes:
  - This instruction computes the base address to fetch from via the formula `dB.scalar_data + iC.addr + to_s64(isimm6)`
  - This instruction fetches into up to `jimm4` consecutive ILARs, starting with `iA`, allowing a maximum 17 ILARs to be fetched into with one instruction.
  - This instruction makes use of the associativity of LARs and will only actually access memory if it has to.
- Instruction List:

1. `fetch iA, [dB, iC, isimm6], jimm4`

4.5 Group 4 Instructions

- Encoding: `0100 aaaa aabb bbbb cccc ccii iiii oooo`
  - `a`: DLAR a
  - `b`: DLAR b or ILAR b
  - `c`: ILAR c or unsigned 6-bit immediate, `cimm6`
  - `i`: unsigned 6-bit immediate, `iimm6`, or signed 6-bit immediate, `isimm6`
- Instruction List:
  1. `sel.s dA, iB, iC, isimm6`
     - This instruction jumps to `iB`, offset `imm6`, if `dA.scalar_data` is zero or `iC`, offset `imm6` if `dA.scalar_data` is non-zero.
  2. `sel.v dA, iB, iC, isimm6`
     - This instruction jumps to `iB`, offset `imm6`, if `dA.data` is zero or `iC`, offset `imm6` if `dA.data` is non-zero.
  3. `bz.s dA, iB, isimm6`
  4. `bz.v dA, iB, isimm6`
  5. `bnz.s dA, iB, isimm6`
  6. `bnz.v dA, iB, isimm6`
  7. `in.s dA, dB, cimm6, iimm6`
     - Input scalar data into `dA` from IO address in `dB.scalar_data + cat(cimm6, iimm6)`.
     - The size of the IO access is determined by `dA.type`.
  8. `in.v, dA, dB, cimm6, iimm6`
     - Input vector data into `dA` from IO address in `dB.scalar_data + cat(cimm6, iimm6)`.
  9. `out.s dA, dB, cimm6, iimm6`
     - Output scalar data in `dA` to IO address in `dB.scalar_data + cat(cimm6, iimm6)`.
     - The size of the IO access is determined by `dA.type`.
  10. `out.v, dA, dB, cimm6, iimm6`
      - Output vector data in `dA` to IO address in `dB.scalar_data + cat(cimm6, iimm6)`.
  11. `swi dA, cimm6, iimm6`
      - Perform software interrupt, where the interrupt number is stored in `dA.scalar_data + cat(cimm6, iimm6)`
      - This instruction, like a regular interrupt, stores the interrupt return address in `dira`, which is a supervisor mode DLAR.

12. `reti`

    - Switch from supervisor mode to user mode. This instruction will cause an exception if executed in user mode.

13. `ei`

    - Enable interrupts. This instruction causes an exception if executed in user mode.

14. `di`

    - Disable interrupts. This instruction causes an exception if executed in user mode.

15. *Reserved for future expansion*

16. `Reserved for future expansion.`