**Risc-V Emulator Documentation  
  
A table with numbers and letters

Description automatically generated  
instruction types**

**R-Type:** 3 register operations. No immediate.

**I-Type:** Immediate operations.

**S-Type:** store operations.

**B-Type:** conditional branch instructions

**U-Type**: upper immediate instructions, used for large immediates.

**J-Type**: general jump instructions.

**Instruction list:**   
*RV32-I extension:*

* ***Lui:***Load upper immediate to rd, like the ‘mov’ command in x86.   
  U-Type
* ***Auipc:***  
  Add upper immediate and to pc and store in rd.   
  PC does not get changed.   
  Used to build pc-relative addresses.  
  U-Type
* ***Jal:***Jump and link  
  Save pc + 4 in rd.  
  Add immediate to pc and store in pc.  
  J-Type  
  Bits are not in order to reduce number of multiplexers needed

\***multiplexer** – a device that can receive multiple input signals and synthesize a single output signal in a recoverable manner for each input signal

* ***Jalr:***

Jump and link register  
  
  
save pc + 4 in rd (should use x0 if result is not useful, like in ret from subroutine).

Add immediate to rs1, set LSB (least significant bit) to 0 add to pc.  
I-Type

* ***Cjumps:***
  + **BNE** – JNE
  + **BEQ** – JE
  + **BLT** – JL
  + **BGE** – JGE
  + **BLTU** – JB (unsigned)
  + **BGEU** – JAE (unsigned)

B-Type

* ***Alui***arithmetic and logical immediate instructions:
* **ADDI**:

Addi immediate

* **Slti**:  
  set less than immediate – places 1 in rd if rs1 < sign-extended immediate. Else 0. \*used with bne in loops, instead of blt and etc.
* **Sltiu**:  
  set less than immediate unsigned – places 1 in rd if rs1 < immediate. Else 0.  
  \*used with bne in loops, instead of blt and etc.
* **XORI**:  
  xori immediate
* **ORI**:

Ori immediate

* **Andi**:

Andi immediate

* **Slli**:

Shift left

* **Srli**:

Shift right

* **Srai**:

Arithmetic right shift

(there is not difference between arithmetic shift left and regular shift left)

I-Type

* ***ALU*** arithmetic and logical instructions:
* **ADD**:

Add

* **Slt**:  
  set less than – places 1 in rd if rs1 < rs2. Else 0.   
  \*used with bne in loops, instead of blt and etc.
* **Sltu**:  
  set less than unsigned – places 1 in rd if rs1 < rs2 (unsigned). Else 0.  
  \*used with bne in loops, instead of blt and etc.
* **XOR**:  
  xori
* **OR**:

Ori

* **And**:

And

* **Sll**:

Shift left

* **Srl**:

Shift right

* **Sra**:

Arithmetic right shift

(there is not difference between arithmetic shift left and regular shift left)

R-Type

* ***Load***  
  load from memory:  
    
  \*all extend to 32-bits
* **LB**:

Load byte

* **LBU:**  
  Load byte unsigned
* **LH**:  
  load word
* **LHU:**  
  load word unsigned
* **LW**:  
  load double word.

I-Type

* ***Store***  
  store in memory:  
  store from the lower bits of rs2 to memory
* **LB**:

Load byte

* **LBU:**  
  Load byte unsigned
* **LH**:  
  load word
* **LHU:**  
  load word unsigned
* **LW**:  
  load double word.

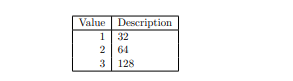
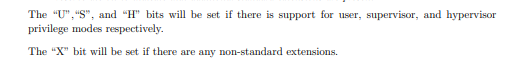
S-Type

* ***Ecall***system syscalls. Vaguely defined by RISC-V purposefully.   
  \***A7** – syscall number  
  **\*a0-a5** – args (zeroed by os if not used)  
  **\*return value – a0**  
  syscall table:  
  <https://jborza.com/post/2021-05-11-riscv-linux-syscalls/>

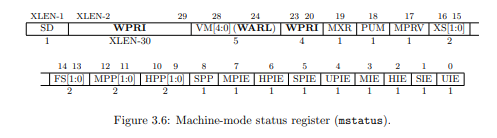
*RV32-M extension:  
notice that for signed multiplication – only the higher 32-bits differ.*

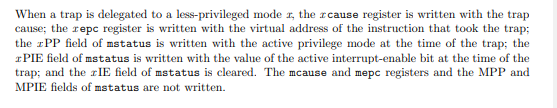
* MUL – 32bitX32bit multiplication and store the lower 32 bits in the destination register.
* MULH – same multiplication as the previous instruction but stores higher 32 bits.
* MULHSU – same multiplication as the previous instruction but with sign unsigned multiplication.
* DIV – divide 2 operands and store the result in the destination register.
* DIVU – unsigned division (same as before except for that).
* REM – the remainder of DIV
* REMU – the remainder of DIVU
* Division of signed/unsigned operands is not permitted due to the RISC-like nature of the instruction set. It might cause the instruction set to be too inefficient for the sake of robustness.
* ­**Division By Zero:**The exception could be implemented, but this would be the only arithmetic trap in the ISA and would require language implementors to interact with trap handlers. Furthermore, the current implementation requires only a single branch instruction after the division operation, whereas an exception requires an immediate control flow change.

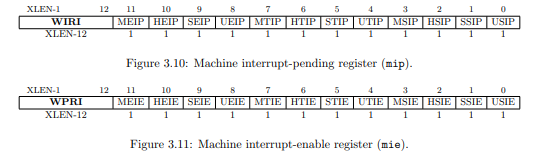
**CSR – Control and Status Registers:**divided according to privilege level – User, Supervisor, Hypervisor, Machine.   
  
*CSR listing:***misa** – Machine instruction set architecture.

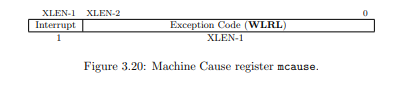
  
  
Base encodes native base integer width.   
Extensions encodes standard extensions according to alphabetical order.  


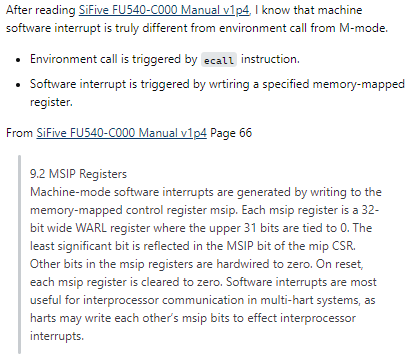
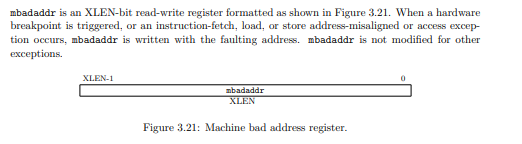
**mvendorid** – encoding the manufacturer of the part.  
  
**marched –** base microarchitecture of the hart.   
  
**mimpid –** version of the processor implementation. (architecture I believe)

**mhartid** – ID of the hardware thread (hart) running the code.   
  
**mstatus –** structured as the following:  
  
Interrupt-enable bits: MIE, HIE, SIE, UIE   
(interrupts for lower privilege modes are always disabled, and for higher ones always enabled)  
For support of nested traps:   
2 level stack of IE bits.   
previous interrupt enable bit – MPIE, HPIE, SPIE, UPIE  
previous privilege mode – MPP, HPP, SPP, UPP.  
  

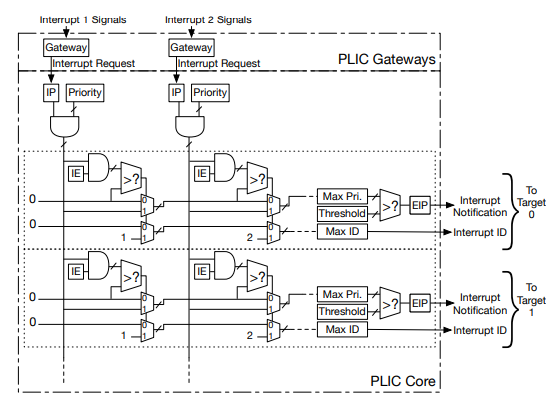
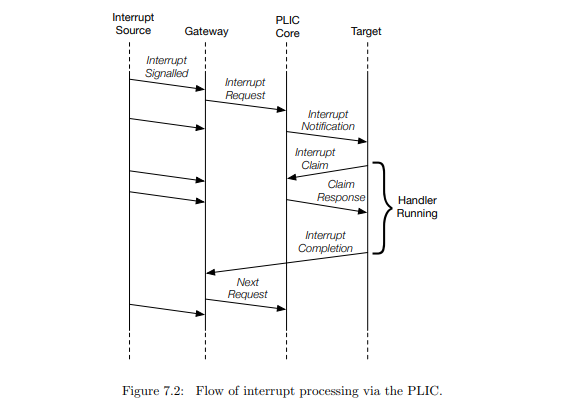

MRET, HRET, SRET, URET – return from traps in each mode respectively.   
as far as I know, they are only available for traps in their respective mode, otherwise they will raise an exception like writing iret for no reason in x86.   
  
(**find out more about them)**  
  
\*virtualization   
  
\*memory privilege   
\*f extension   
  
**mtvec –** Trap-Vector base address register.   
  
**medeleg/mideleg –** machine trap delegation registers.  
***A close-up of a document

Description automatically generated* **mip –** machine interrupt pending   
**mie** – machine interrupt enable   
**

**Mtime –** machine time register  
**mtimecmp** – machine time compare register  
mtime has 64 bit precision on all systems.   
  
**mepc**  - machine exception program counter   
**mcause** – machine cause register   
  
A screenshot of a computer program

Description automatically generated  
the difference between a software interrupt and an environment call in RISC-V:  
  
  
**mbadaddr - **  
 *RV32-Zicsr extension:*

* CSRRW – read csr (zero-extended) to rd (unless it is x0 – also find out why it is specifically mentioned). Rs1 is written to csr.
* CSRRS – read csr (zero-extended) to rd, then perform OR with rs1 on csr.
* CSRRC – read csr (zero-extended) to rd, then perform XOR with rs1 on csr.
* CSRRWI, CSRRCI, CSRRSI – same as the previous ones except that rs1 is treated as a sign extended 5-bit unsigned immediate.

**PLIC – Platform Level Interrupt Controller :** IP – interrupt pending  
IE – interrupt enable   
  
the PLIC does not support privilege-level prioritization.   
  
**Interrupt Targets -**   
usually hart contexts.   
the interrupt notifications generated by the PLIC appear in the meip bits of the mip (or other H/S/U registers accordingly).

*Interrupt table:*exceptions:  
**0. Instruction Address Misaligned –**   
occurs when jump instructions and conditional branches land at a misaligned address (4-byte alignment).   
 **1. Instruction Access Fault -**   
occurs when an application attempts to execute an invalid memory location.   
More specifically, when s->pc exceeds the allocated memory size for the program.  
  
**2. Illegal Instruction -**   
occurs when the application attempts to execute a command that is not codified within the ISA of the CPU.   
  
**3. Breakpoint -**   
to be implemented.  
  
**4. Reserved**   
  
**5. Load Access Fault -**   
occurs when an application attempts to load from an invalid memory location.   
  
**6. Store/AMO address misaligned -**   
 this exception is not implemented in this implementation. However, generally speaking, it occurs when a program attempts to store at a misaligned address (depending on the data type) to speed up performance.   
  
**7. Store/AMO access fault -**   
occurs when an application attempts to store at an invalid memory location.  
  
**8. Environment call from U-Mode -**   
syscall from User mode.   
  
**9-11 – environment call within their respective modes (S, H, M).  
  
12+ - reserved.**