The RISC-V Instruction Set Manual
Volume II: Base Supervisor-Level ISA
Version 1.0

Andrew Waterman, Yunsup Lee, David Patterson, Krste Asanović
CS Division, EECS Department, University of California, Berkeley
{`waterman|yunsup|pattrsn|krste`}`@eecs.berkeley.edu`
March 28, 2012

1 Introduction

The following is a current draft of one supervisor state implementation for RISC-V. This implementation provides a paged virtual memory system and I/O mechanisms sufficient for supporting a modern operating system, such as Linux.

Two variants of the supervisor mode are described: RV32S, which supports 32-bit address spaces, and RV64S, which supports 64-bit address spaces. An implementation may provide only one or both of RV32S and RV64S. RV32S implementations support the RV32 base ISA; RV64S implementations support the RV64 base ISA but may also support RV32 user programs.

A RISC-V CPU has two operating modes: *user* mode and *supervisor* mode. The CPU normally operates in user mode until an exception forces a switch into supervisor mode. The CPU will then normally execute an exception handler in supervisor mode before executing an Exception Return (ERET) instruction to return to user mode.

2 Addressing and Memory Protection

31 22	21 12	11 0
vpn[1]	vpn[0]	pgoff
10	10	12

Figure 1: RV32S virtual address.

31 22	21 12	11 0
ppn[1]	ppn[0]	pgoff
10	10	12

Figure 2: RV32S physical address.

RV32S implementations support a 32-bit virtual address space, divided into 4 KB pages. An RV32S virtual address is partitioned into a virtual page number (VPN) and page offset, as shown in Figure 2. When virtual memory is enabled, virtual addresses are translated into physical addresses via a two-level page table. The 20-bit VPN is translated into a 20-bit physical page number (PPN), while the 12-bit page offset is untranslated.

RV64S implementations support a 43-bit virtual address space, divided into 8 KB pages. An RV64S address is partitioned as shown in Figure 4. Load and store effective addresses, which are 64 bits,

42 33	32 23	22 13	12 0
vpn[2]	vpn[1]	vpn[0]	pgoff
10	10	10	13

Figure 3: RV64S virtual address.

63 33	32 23	22 13	12 0
ppn[2]	ppn[1]	ppn[0]	pgoff
31	10	10	13

Figure 4: RV64S physical address.

must have bits 63–43 all equal to bit 42, or else an address exception will occur. The 30-bit VPN is translated into a 51-bit PPN via a three-level page table, while the 13-bit page offset is untranslated.

31 22	21 12	11 10	9	8	7	6	5	4	3	2	1	0
ppn[1]	ppn[0]	—	SR	SW	SX	UR	UW	UX	D	R	E	T
10	10	2	1	1	1	1	1	1	1	1	1	1

Figure 5: RV32S page table entry.

Page tables consist of 2^{10} page table entries (PTEs), each 4 bytes in RV32S or 8 bytes in RV64S. The PTE formats for RV32S and RV64S are shown in Figures 5 and 6, respectively. A page table is exactly the size of a page and must always be aligned to a page boundary. The physical address of the first level of the page table is stored in the `ptbr` register.

Each virtual page is configured with UNIX-style permissions. The SR, SW, and SX bits in a PTE, when set, allow the supervisor to read, write, and execute instructions, resepectively. The UR, UW, and UX bits specify these permissions for user mode.

A virtual address *va* is translated into a physical address *pa* as follows:

1. Let a be the value of the `ptbr` register, and let $i = LEVELS - 1$.
2. Let pte be the value of the PTE at address $a + va.vpn[i] \times$PTESIZE.
3. If $pte.e = 1$, determine if the requested memory access is allowed by the permission bits. If not, signal an address error. Otherwise, the translation is successful and the physical address is given as follows:
   - $pa.pgoff = va.pgoff$.
   - If $i > 0$, then this is a superpage translation and $pa.ppn[i-1:0] = va.vpn[i-1:0]$.
   - $pa.ppn[LEVELS-1:i] = pte.ppn[LEVELS-1:i]$.
4. If $pte.t = 0$ or $i = 0$, signal an address error. Otherwise, let $i = i - 1$, let $a = pte.ppn\times$ PAGESIZE, and go to step 2.

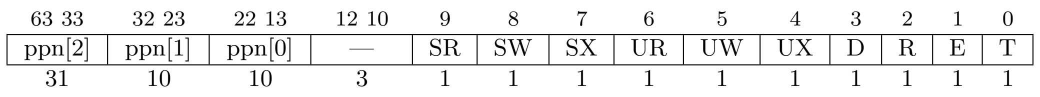

Figure 6: RV64S page table entry.

3 Privileged Control Registers

The RISC-V instruction set defines a number of privileged control registers used for system configuration, virtual memory, and exception handling. These registers are read and written with the privileged instructions `mfpcr` and `mtpcr`; they are inaccessible in user mode. These registers are listed in Table 1.

Number	Register	Description
0	`status`	Status register.
1	`epc`	Exception program counter.
2	`badvaddr`	Bad virtual address.
3	`evec`	Exception handler address.
4	`count`	Cycle counter for timer.
5	`compare`	Timer compare value.
6	`cause`	Cause of exception.
7	`ptbr`	Page table base register.
8–11		*unused.*
12	`k0`	Scratch register for exception handlers.
13	`k1`	Scratch register for exception handlers.
14–15		*unused.*
16	`tohost`	Test output register
17	`fromhost`	Test input register
18–31		*unused.*

Table 1: RISC-V privileged control registers.

4 Test Communication Registers

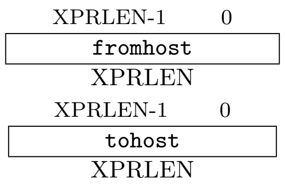

Figure 7: Fromhost and Tohost Registers.

There are two registers used for communicating and synchronizing with an external host test system. Typically, these will be accessed over a scan chain. The `fromhost` register is a XPRLEN-width read-only register that contains a value written by the host system. The `tohost` register is an `x`-register-width read/write register that contains a value that can be read back by the host system. The `tohost` register is cleared by reset to simplify synchronization with the host test rig.

5 Counter/Timer Registers

31 0
`count`
32

31 0
`compare`
32

Figure 8: Count and Compare Registers.

RISC-V includes a counter/timer facility provided by the two coprocessor 0 registers `count` and `compare`. Both registers are 32 bits wide and are both readable and writeable. Their format is shown in Figure 8. The `count` register contains a value that increments once every clock cycle. The `count` register is normally only written for initialization and test purposes. A timer interrupt is generated when the `count` register reaches the same value as the `compare` register. The interrupt will only be taken if both `IM[7]` and `ET` in the `status` register are set. The timer interrupt is cleared by writing the `compare` register.

6 Exception Processing Registers

A number of RISC-V privileged control registers are used for exception processing.

6.1 Status Register

31 17	16	15 8	7	6	5	4	3 2	1	0	
0	IM	0	VM	S64	U64	S	PS	0	EF	ET
8	8	7	1	1	1	1	1	2	1	1

Figure 9: Status Register.

The `status` register is a 32-bit read/write register formatted as shown in Figure 9. The `status` register keeps track of the processor's current operating state.

The IM field contains interrupt mask bits. Timer interrupts are disabled by clearing `IM[7]` in bit 15. The other bits within the IM field are implementation-defined. Table 3 includes a listing of interrupt bit positions and descriptions.

The VM bit enables or disables virtual memory. When VM=0, memory addresses are untranslated.

The S and PS bits form a two-level stack holding the operating mode (1 for supervisor, 0 for user). When an exception is taken, the PS bit is set to the value of the S bit. When an `eret` instruction is executed to return from the exception handler, the S bit is set to the value of the PS bit; the PS bit is unchanged.

The U64 bit indicates whether the processor executes the RV64 ISA (a value of 1) or the RV32 ISA (a value of 0) while in user mode (S = 0). An implementation might support only RV32 (RV64), in which case U64 is hard-wired to 0 (1).

The S64 bit indicates whether the processor executes the RV64S ISA (a value of 1) or the RV32S ISA (a value of 0) while in supervisor mode (S = 1). An implementation might support only RV32S (RV64S), in which case S64 is hard-wired to 0 (1). When S64 is 0, U64 is also 0 and writes to U64 are ignored.

The EF bit enables or disables floating-point instructions. An attempt to execute a floating-point instruction with EF=0 causes a floating point disabled trap. For implementations that lack a floating-point unit, EF is hard-wired to 0.

The ET bit globally enables or disables exceptions. When ET=0, interrupts are not taken, and any trap causes the processor to enter error mode. The ET bit is cleared when an exception is taken and set when an `eret` instruction is executed.

6.2 Cause Register

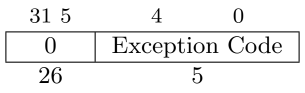

Figure 10: Cause Register.

The `cause` register is a 32-bit register formatted as shown in Figure 10. The `cause` register contains a code identifying the last exception. Writes to `cause` are ignored.

6.3 Exception Vector Register

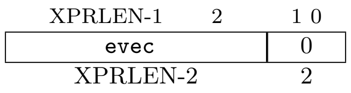

Figure 11: Exception Vector Register.

`evec` is a XPRLEN-bit register formatted as shown in Figure 11. When an exception occurs, the `pc` is set to `evec`. The `evec` register is always aligned to a 4-byte boundary.

6.4 Exception Program Counter

`epc` is a XPRLEN-bit read only register formatted as shown in Figure 12. When an exception occurs, `epc` is written with the virtual address of the instruction that caused the exception.

ExcCode	Description
0	Instruction address misaligned
1	Instruction access fault
2	Illegal instruction
3	Privileged instruction
4	Floating point disabled
5	*unused*
6	System call
7	Breakpoint
8	Load address misaligned
9	Store address misaligned
10	Load access fault
11	Store access fault
12–15	*unused*
16	External interrupt 0
17	External interrupt 1
18	External interrupt 2
19	External interrupt 3
20	External interrupt 4
21	External interrupt 5
22	External interrupt 6
23	Timer interrupt
24–31	*unused*

Table 2: Exception Types.

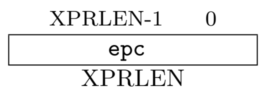

Figure 12: EPC Register.

6.5 Bad Virtual Address

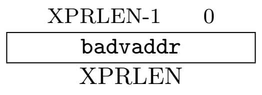

Figure 13: Badvaddr Register.

`Badvaddr` is a XPRLEN-bit read only register formatted as shown in Figure 13. When a load or store access exception or data address misaligned exception occurs, `badvaddr` is written with the faulting virtual address. The value in `badvaddr` is undefined for other exceptions.

7 Reset

Processor state is initialized by an external reset signal. Upon deassertion of external reset, the `pc` is set to 0x2000, with `status` register fields `ET`=0, `S`=0, `VM`=0, and `S64`=1 for RV64S implementations or 0 for RV32S-only implementations. The effect is to begin execution in supervisor mode with interrupts disabled and without address translation. The `tohost` register is set to zero for synchronization with the host system. All other processor state is undefined.

A typical reset sequence is shown in Figure 14.

```
reset_vector:
    mtpcr x0, cr4 # Initialize counter.
    mtpcr x0, cr5 # Initialize compare and clear pending interrupts.

    # Initialize trap table base register.
    la    x1, trap_table
    mtpcr x1, cr3

    # Initialize status register (supervisor mode, traps/interrupts on).
    li    x1, (SR_S | SR_ET | SR_IM | SR_S64 | SR_U64)
    mtpcr x1, cr0

    j     kernel_init    # Initialize kernel software.
```

Figure 14: Example reset sequence.

8 Exception Processing

Normal program flow may be disrupted by two kinds of exceptions: interrupts (asynchronous external events) and traps (synchronous events caused by program execution). Upon an exception, the processor enters supervisor mode with exceptions disabled (`status` register bits `PS`, `S`, and `ET` are set to `S`, 1, and 0, respectively). Execution resumes at the virtual address specified by the `evec` register. The PC of the interrupted or trapping instruction is stored in the `epc` register.

8.1 Interrupts

RISC-V implementations may support up to 8 interrupts. The timer uses interrupt 7. The possible interrupts in RISC-V are listed in Table 3 in decreasing order of priority.

Cause	IM index	Description
Highest Priority		
`0x10`	0	*Reserved.*
`0x11`	1	*Reserved.*
`0x12`	2	*Reserved.*
`0x13`	3	*Reserved.*
`0x14`	4	*Reserved.*
`0x15`	5	*Reserved.*
`0x16`	6	*Reserved.*
`0x17`	7	Timer interrupt.
Lowest Priority		

Table 3: RISC-V Interrupts.

All RISC-V interrupts are level triggered. For each interrupt there is an IM flag in the `status` register that enables the interrupt when set. In addition, all interrupts will be masked off when the exception enable bit `ET` is cleared. A particular interrupt can occur only if the IM bit for that interrupt is set, `ET` is set, and there are no higher priority interrupts.

8.2 Traps

Traps are listed in Table 4 in order of decreasing priority.

Cause	Description
Highest Priority	
`0x00`	Instruction address misaligned.
`0x01`	Instruction access fault.
`0x02`	Illegal instruction.
`0x03`	Privileged instruction.
`0x04`	Floating-point disabled.
`0x05`	System call.
`0x06`	Breakpoint.
`0x07`	Data address misaligned.
`0x08`	Load access fault.
`0x09`	Store access fault.
`0x0A`	*Reserved.*
`0x0B`	*Reserved.*
`0x0C`	*Reserved.*
`0x0D`	*Reserved.*
`0x0E`	*Reserved.*
`0x0F`	*Reserved.*
Lowest Priority	

Table 4: Maven Synchronous Exceptions.

If the exception was a load or store misaligned address or access fault, the `badvaddr` register is set to the faulting address. The value in `badvaddr` is undefined for other exceptions. For instruction misaligned address and access faults, the `epc` register contains the faulting address.

All unimplemented and illegal instructions cause an illegal instruction exception. Any attempt to execute a privileged instruction while not in supervisor mode causes a privileged instruction exception.