Computer Architecture, Semester 1, 2018
Assignment — RISC-V RV64I ISS — Stage 2

Your task for this assignment is to extend your RV64I ISS from Stage 1 with a subset of the RISC-V privileged architecture for handling exceptions and interrupts. The RISC-V privileged architecture is described in *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10*, available at riscv.org (also on the course web site).

The RISC-V privileged architecture specification defines three privilege levels for execution of instructions: user, supervisor, and machine modes. For this assignment you should implement features defined for user mode and machine mode, but not for supervisor mode. The specification also defines some features for debug mode, which you should also not implement.

You should implement the following machine-mode control and status registers (CSRs), listed in Table 2.4 and described in Section 3.1:

Number	Privilege	Name	Description
0xF11	MRO	mvendorid	Vendor ID. Fixed value of 0 to indicate non-commercial implementation.
0xF12	MRO	marchid	Architecture ID. Fixed value of 0 to indicated no ID implemented.
0xF13	MRO	mimpid	Implementation ID. Fixed value of 0x2018020000000000 to indicate an implementation in 2018, stage 2.
0xF14	MRO	mhartid	Hardware thread ID. Fixed value of 0, since this is the only hart in the simulated processor.
0x300	MRW	mstatus	Machine status register, formatted as shown in Figure 3.7. Only the following bits are implemented: mie, mpie, mpp. Other bits are fixed at 0, except for uxl fixed at 2.
0x301	MRW	misa	ISA and extensions register. Fixed value of 0x8000000000100100 (MXL = 2 for 64-bit XLEN, I and U bits set in Extensions field). While csr instructions can legally write to this CSR, the CSR value remains fixed.
0x304	MRW	mie	Machine interrupt enable register. Only the following bits are implemented: usie, msie, utie, mtie, ueie, meie. Other bits are fixed at 0.
0x305	MRW	mtvec	Machine trap handler base address register. The MODE field can only be 0 or 1, so bit 1 of mtvec is fixed at 0. If MODE is 1 (Vectored), then BASE must be 256-byte aligned; in that case, mtvec bits 2 to 7 are fixed at 0.
0x340	MRW	mscratch	Scratch register for machine trap handlers.
0x341	MRW	mepc	Machine exception program counter. Bits 0 and 1 are fixed at 0.
0x342	MRW	mcause	Machine trap cause. Only Interrupt bit and 4-bit Exception code field implemented. Other bits are fixed at 0.
0x343	MRW	mtval	Machine bad address or instruction. For misaligned address exceptions, mtval is written with the address. For illegal instruction exceptions, the least-significant word of mtval is written with the instruction word, and the most-significant word of mtval is set to 0. For other exceptions, mtval is set to 0.
0x344	MRW	mip	Machine interrupt pending. Only the following bits are implemented: usip, msip, utip, mtip, ueip, meip. Other bits are fixed at 0.

I will provide an extended command handler that implements the following commands, in addition to those provided for stage 1:

Command	Operation performed
csr *num*	Show the content of CSR *num* (*num* in hex). The value is displayed as 16 hex digits with leading 0s.
csr *num* = *value*	Set CSR *num* to *value* (*num* and *value* in hex).
prv	Display the current processor privilege level (0 = user, or 3 = machine)
prv = *value*	Set the current processor privilege level to *value* (0 = user, or 3 = machine)

You should implement the following instructions, previously unimplemented in stage 1:

- csrrw, csrrs, csrrc, csrrwi, csrrsi, csrrci

You should also implement the following privileged instructions defined in Section 3.2:

- ecall: environment call from user mode or from machine mode
- ebreak: breakpoint (Note that this is different from and additional to the b command breakpoint from stage 1)
- mret: return from machine trap

You should extend the way your simulated processor executes instructions to check for the exceptions in the table below. If an exception occurs during execution of an instruction, the instruction is not retired, so it is not included in the count of executed instructions.

Cause code	Exception	Instructions that cause exception
0	Instruction address misaligned	Any instruction fetch for which the PC is not a multiple of 4.
2	Illegal instruction	Any defined instruction that is not implemented. Any undefined instruction. An mret instruction executed in user mode. A csr instruction (not the csr command) that accesses an undefined or unimplemented CSR.
3	Breakpoint	ebreak
4	Load address misaligned	ld for which the effective address is not a multiple of 8. lw/lwu for which the effective address is not a multiple of 4. lh/lhu or which the effective address is not a multiple of 2.
6	Store address misaligned	sd for which the effective address is not a multiple of 8. sw for which the effective address is not a multiple of 4. sh or which the effective address is not a multiple of 2.
8	Environment call from U-mode	ecall executed in user mode.
11	Environment call from M-mode	ecall executed in machine mode.

You should also implement the following interrupts:

Cause code	Interrupt	Interrupt trigger
0	User software interrupt	(mip.usip && mie.usie) && mstatus.mie
3	Machine software interrupt	(mip.msip && mie.msie) && mstatus.mie
4	User timer interrupt	(mip.utip && mie.utie) && mstatus.mie

Cause code	Interrupt	Interrupt trigger
7	Machine timer interrupt	(mip.mtip && mie.mtie) && mstatus.mie
8	User external interrupt	(mip.ueip && mie.ueie) && mstatus.mie
11	Machine external interrupt	(mip.meip && mie.meie) && mstatus.mie

In a real processor, the mip CSR bits would be connected to hardware interrupt sources (e.g., a hardware timer for mip.mtip, or an external interrupt controller for mip.meip). In your simulator, interrupts will be made pending by setting mip CSR bits using the csr command. Your simulator should check for any enabled pending interrupts before fetching each instruction.

Your simulator should not implement nonmaskable interrupts, physical memory attributes, or physical memory protection. Your simulator should implement reset by starting in machine mode, setting mstatus.mie to 0, setting pc to 0, and setting mcause to 0.

Please keep an eye on the discussion forum on the course web site. I will answer any questions of clarification of requirements that arise there. I will also announce incremental releases of a test suite for this stage.

You must extend your program from stage 1 and check it into the 2018/s1/ca/rv64sim directory in your SVN repository. I will provide a web submission script that will check out this subdirectory, make your ISS, and run it with several test cases. Compliance with this development process will count toward the assessment of the assignment. The script will compare your output with our expected output using the "diff -iw" command (differences ignoring case and white-space).

Your work for Stage 2 will be assessed in the web submission system based on the following criteria, with points awarded out of 1500:

- Program builds and runs using web submission script — 100 points
- Correct execution, based on the number of test cases that pass — 1400 points

The points for this assignment will comprise 15% of your final assessment for the course.

The deadline for submission is **11:59pm Sunday 10 June 2018**.

Postgraduate requirements

If you are enrolled in the postgraduate course (COMP SCI 7026), you should implement the following additional requirements:

- Implement the minstret CSR to count the number of instructions retired (ie, instructions whose execution completes without exception)
- Implement the mcycle CSR to count the number of clock cycles using the cycle count values from stage 1.
- If an exception occurs in a load or store instruction, reduce the cycle count for the instruction to 1.
- If an exception or interrupt occurs, add a further 2 clock cycles for trap handling.