BANNARI AMMAN INSTITUTE OF TECHNOLOGY
An Autonomous Institution Affiliated to Anna University - Chennai • Approved by AICTE • Accredited by NAAC with "A" Grade
SATHYAMANGALAM - 638401 ERODE DISTRICT TAMILNADU INDIA
Ph : 04295-226000/221289 Fax : 04295-226666 E-mail : stayahead@bitsathy.ac.in Web : www.bitsathy.ac.in

# 22XX303 – COMPUTER ORGANIZATION AND ARCHITECTURE

# UNIT V & MODERN COMPUTER ARCHITECTURE

## 1. THE RISC-V BASE INSTRUCTION SET

- The RISC-V base instruction set is composed of just 47 instructions. Eight are system instructions that perform system calls and access performance counters. The remaining 39 instructions fall into the categories of computational instructions, control flow instructions, and memory access instructions. We will examine each of these categories in turn.

### 1.1 Computational instructions

- All the computational instructions except lui and auipc use the three-operand form.
- The first operand is the destination register, the second is a source register, and the third is either a second source register or an immediate value.
- Instruction mnemonics using an immediate value (except for auipc) end with the letter i.
- These are the instructions and their functions:
  - ✓ **add, addi, sub:** Perform addition and subtraction. The immediate value in the addi instruc- tion is a 12-bit signed value. The sub instruction subtracts the second source operand from the first. There is no subi instruction because addi can add a negative immediate value.
  - ✓ **sll, slli, srl, srli, sra, srai:** Perform logical left and right shifts (sll and srl), and arithmetic right shifts (sra). Logical shifts insert zero bits into vacated locations. Arith- metic right shifts replicate the sign bit into vacated locations. The number of bit positions to shift is taken from the lowest 5 bits of the second source register or from the 5-bit immediate value.
  - ✓ **and, andi, or, ori, xor, xori:** Perform the indicated bitwise operation on the two source operands. Immediate operands are 12 bits.
  - ✓ **slt, slti, sltu, sltui:** The seí ifi less íhan instructions set the destination

register to 1 if the first source operand is less than the second source operand. This comparison is in terms of two's complement (slt) or unsigned (sltu) operands. Immediate operand values are 12 bits.

- ✓ **lui:** Load upper immediate. This instruction loads bits 12-31 of the destination register with a 20-bit immediate value. Setting a register to an arbitrary 32-bit immediate value requires two instructions: First, lui sets bits 12-31 to the upper 20 bits of the value. Then, addi adds in the lower 12 bits to form the complete 32-bit result. lui has two operands: the destination register and the immediate value.
- ✓ **auipc:** Add upper immediate to PC and store the result in the destination register. This instruction adds a 20-bit immediate value to the upper 20 bits of the program counter. This instruction enables PC-relative addressing in RISC-V.

- ➢ To form a complete 32-bit PC-relative address, auipc forms a partial result, then an addi instruction adds in the lower 12 bits.

## 1.2 Control flow instructions

- ➢ The conditional branching instructions perform comparisons between two registers and, based on the result, may transfer control within the range of a signed 12-bit address offset from the current PC.
- ➢ Two unconditional jump instructions are available, one of which (jalr) provides access to the entire 32-bit address range:
  - ✓ **beq, bne, blt, bltu, bge, bgeu:** Branch if equal (beq), not equal (bne), less than (blt), less than unsigned (bltu), greater or equal (bge), or greater or equal, unsigned (bgeu). These instructions perform the designated comparison between two registers and, if the condition is satisfied, transfer control to the address offset provided in the 12-bit signed immediate value.
  - ✓ **jal:** Jump and link. Transfer control to the PC-relative address provided in the 20-bit signed immediate value and store the address of the next instruction (the return address) in the destination register.
  - ✓ **jalr:** Jump and link, register. Compute the target address as the sum of the source register and a signed 12-bit immediate value, then jump to that address and store the address of the next instruction in the destination register. When preceded by the auipc instruction, the jalr instruction can perform a PC-relative jump anywhere in the 32-bit address space.

## 1.3 Memory access instructions

- ➢ The memory access instructions transfer data between a register and a memory location. The first operand is the register to be loaded or stored.
- ➢ The second operand is a register containing a memory address.

- A signed 12-bit immediate value is added to the address in the register to produce the final address used for the load or store.
- The load instructions perform sign extension for signed values or zero extension for unsigned values.
- The sign or zero extension operation fills in all 32 bits in the destination register when a smaller data value (a byte or halfword) is loaded.
- Unsigned loads are specified by a trailing u in the mnemonic:
    - ✓ **lb, lbu, lh, lhu, lw:** Load an 8-bit byte (lb), a 16-bit halfword (lh), or a 32-bit word (lw) into the destination register. For byte and halfword loads, the instruction will either sign-extend (lb and lh) or zero-extend (lbu and lhu) to fill the 32-bit destination register. For example, the lw x1, 16(x2) instruction loads the word at the address (x2 + 16) into register x1.
    - ✓ **sb, sh, sw:** Store a byte (sb), halfword (sh), or word (sw) to a memory range matching the size of the data value.
    - ✓ **fence:** Enforce memory access ordering in a multithreaded context. The purpose of this instruction is to ensure a coherent view of cached data across threads. This instruction takes two operands: The first specifies the types of memory accesses that must complete prior to the fence instruction. The second specifies the types of memory accesses controlled following the fence. The operation types ordered by this instruction are memory reads and writes (r and w) and I/O device inputs and outputs (i and o). For example, the fence rw, rw instruction will guarantee that all loads and stores involving memory addresses occurring before the fence instruction will complete before any subsequent memory loads or stores take place. This instruction ensures that any values present in processor caches are properly synchronized with memory or the I/O device.
    - ✓ **fence.i:** This instruction ensures that any stores to instruction memory have complet- ed before the fence.i instruction completes. This instruction is primarily useful in the context of self-modifying code.

## 1.4 System instructions

- Of the eight system instructions, one invokes a system call, one initiates a debugger breakpoint, and the remaining six read and write system control and status registers (CSRs).
- The CSR ma- nipulation instructions read the current value of the selected CSR into a register, then update the CSR by either writing a new value, clearing selected bits, or setting selected bits.
- The source value for the CSR modification is provided in a register or as an immediate 5-bit value. CSRs are identified by a 12-bit address.

- Each CSR instruction performs the read and write of the CSR as an atomic operation:
  - ✓ **ecall:** Invoke a system call. Registers used for passing parameters into and returning from the call are defined by the ABI, not by processor hardware.
  - ✓ **ebreak:** Initiate a debugger breakpoint.
  - ✓ **csrrw, csrrwi, csrrc, csrrci, csrrs, csrrsi:** Read the specified CSR into a destination register and either write a source operand value to the register (csrrw), clear any 1 bit in the source operand in the register (csrrc), or set any 1 bit in the source operand in the register (csrrs).These instructions take three operands: the first is the destination register receiving the value read from the CSR, the second is the CSR address, and the third is a source register or a 5-bit immediate value (i suffix).
- Six CSRs are defined in the base RISC-V architecture, all read-only. To execute any of the CSR access instructions in read-only mode, the x0 register must be provided as the third operand. These registers define three 64-bit performance counters:
  - ✓ **cycle, cycleh:** The lower (cycle) and upper (cycleh) 32 bits of the 64-bit count of elapsed system clock cycles since a reference time—typically, system startup. The frequency of the system clock may vary if dynamic voltage and frequency scaling (DVFS) is active.
  - ✓ **time, timeh:** These are the lower (time) and upper (timeh) 32 bits of the 64-bit count of elapsed fixed-frequency real-time clock cycles since a reference time—typically, system startup.
  - ✓ **instret, instreth:** The lower (instret) and upper (instreth) 32 bits of the 64-bit count of processor instructions retired. Retired instructions are those that have completed execution.
- The two 32-bit halves of each performance counter cannot be read in a single atomic operation. To prevent erroneous readings, the following procedure must be used to reliably read each of the 64-bit counters:
  1. Read the upper 32 bits of the counter into a register.
  2. Read the lower 32 bits of the counter into another register.
  3. Read the upper 32 bits into yet another register.
  4. Compare the first and second reads of the upper 32 counter bits. If they differ, jump back to step 1.
- This procedure will read a valid count value, even though the counter continues to run between the reads. In general, execution of this sequence should require, at most, one backward jump in step 4.

# 2. RISC-V extensions

- The instruction set described previously is named RV32I, which stands for the RISC-V 32-bit integer instruction set.
- Although the RV32I ISA provides a complete and useful instruction set for many purposes, it lacks several functions and features available in other popular processors such as x86 and ARM.
- The RISC-V extensions provide a mechanism for adding capabilities to the base instruction set in an incremental and compatible manner.
- Implementors of RISC-V processors can selectively include extensions in their design to optimize trade-offs between chip size, system capability, and performance.
- These flexible design options are also available to developers of low-cost FPGA-based systems.
- The major extensions are named M, A, C, F, and D.

## 2.1The M extension

- The RISC-V M extension adds integer multiplication and division functionality to the base RV32I instruction set.
- The following instructions are included in this extension:
  - ✓ **mul:** Multiply two 32-bit registers and store the lower 32 bits of the result in the desti- nation register.
  - ✓ **mulh. mulhu, mulhsu:** Multiply two 32-bit registers and store the upper 32 bits of the result in the destination register. Treat the multiplicands as both signed (mulh), both unsigned (mulhu), or signed rs1 times unsigned rs2 (mulhsu). rs1 is the first source register in the instruction and rs2 is the second.
  - ✓ **div, divu:** Perform division of two 32-bit registers, rounding the result toward zero, on signed (div) or unsigned (divu) operands.
  - ✓ **rem, remu:** Return the remainder corresponding to the result of a div or divu instruction on the operands.
- Division by zero does not raise an exception.
- To detect division by zero, code should test the divisor and branch to an appropriate handler if it is zero.

## 2.2 The A extension

- The RISC-V A extension provides atomic read-modify-write operations to support multithreaded processing in shared memory.
- The load-reserved (lr.w) and store-conditional (sc.w) instructions work together to

perform a memory read followed by a write to the same location in an atomic sequence.

- The load-reserved instruction places a reservation on the memory address during the load.
- If another thread writes to the same location while the reservation is in effect, the reservation is canceled.
- When the store-conditional instruction executes, it returns a value indicating if it successfully completed the atomic operation.
- If the reservation remains valid (in other words, no intervening write occurred to the target address), the store-conditional instruction writes the register to memory and returns zero, indicating success.
- If the reservation was canceled, the store-conditional instruction does not alter the memory location and returns a nonzero value indicating that the store operation failed.
- The following instructions implement the load-reserved and store-conditional operations:
  - ✓ **lr.w:** Load a register from a memory location and place a reservation on the address.
  - ✓ **sc.w:** Store a register to a memory location conditionally. Set the destination register to zero if the operation succeeded and the memory location was written, or set the desti- nation register to a nonzero value if the reservation was canceled. If the reservation was canceled, the memory location is not modified by this instruction.
- The atomic memory operation (AMO) instructions atomically load a word from a memory loca- tion into the destination register, perform a binary operation between the value that was loaded and rs2, and store the result back to the memory address.
- The following instructions implement the AMO operations:
  - ✓ **amoswap.w:** Atomically swap rs2 into the rs1 memory location.
  - ✓ **amoadd.w:** Atomically add rs2 into the rs1 memory location.
  - ✓ **amoand.w , amoor.w, amoxor.w:** Atomically perform AND, OR, or XOR operations with rs2 into the rs1 memory location.
  - ✓ **amomin.w , amominu.w, amomax.w, amomaxu.w:** Atomically perform minimum or maximum selection of signed or unsigned (instructions with the u suffix) values with rs2 into the rs1 memory location.

## 2.3 The C extension

- The RISC-V C extension implements compressed instructions with the goals of minimizing the amount of memory consumed by instruction storage and reducing the amount of bus traffic required to fetch instructions.

- All RV32I instructions discussed previously are 32 bits in length.
- The C extension provides alternate 16-bit representations of many of the most frequently used RV32I instructions. Each compressed instruction is equivalent to one full-length instruction.
- No mode switching is necessary, mean- ing programs can freely intermix 32-bit RV32I instructions and compressed 16-bit instructions.
- In fact, assembly language programmers do not even need to take steps to specify whether an instruction should be generated in compressed form.
- The assembler and linker are capable of transparently emitting compressed instructions where possible to minimize code size, in most cases with no execution performance penalty.
- When working with processors and software development toolsets supporting the RISC-V C ex- tension, the benefits of compressed instructions are immediately available to developers working in assembly language as well as to those working with higher-level languages.

## 2.4 The F and D extensions

- The RISC-V F and D extensions provide hardware support for single-precision (F) and double-pre- cision (D) floating-point arithmetic in accordance with the IEEE 754 standard.
- The F extension adds 32 floating-point registers named f0-f31 and a control and status register named fcsr to the architecture.
- These registers are all 32 bits.
- This extension includes a set of floating-point instructions that complies with the IEEE 754-2008 single-precision standard.
- Most floating-point instructions operate on the floating-point registers.
- Data transfer instruc- tions are provided to load floating-point registers from memory, store floating-point registers to memory, and move data between floating-point registers and integer registers.
- The D extension widens f0-f31 to 64 bits. In this configuration, each f register can hold a 32-bit value or a 64-bit value.
- Double-precision floating-point instructions are added, in compliance with the IEEE 754-2008 double-precision standard.
- The D extension requires the F extension to be present.

## 2.5 Other extensions

- Several additional extensions to the RISC-V architecture, detailed in the following list, have been defined, are in development, or are at least under consideration for future development:
  - ✓ **RV32E architecture:** This is not actually an extension; rather, it is a modified architecture intended to reduce processor hardware requirements below those of the RV32I instruction set for the smallest embedded systems. The only difference between RV32I and RV32E is the reduction in the number of integer registers to 15. This change is expected to reduce processor die area and power consumption by about 25% compared to an otherwise equivalent RV23I processor. x0 remains a dedicated zero register. Halving the number of registers frees up 1 bit in each register specifier in an instruction. These bits are guaranteed to remain unused in future revisions and are thus available for use in custom instruction extensions.
  - ✓ **Q extension:** The Q extension supports 128-bit quad-precision floating-point mathematics, as defined in the IEEE 754-2008 standard.
  - ✓ **L extension:** The L extension supports decimal floating-point arithmetic, as defined in the IEEE 754-2008 standard.
  - ✓ **B extension:** The B extension supports bit manipulations such as inserting, extracting, and testing individual bits.
  - ✓ **J extension:** The J extension supports dynamically translated languages such as Java and JavaScript.
  - ✓ **T extension:** The T extension supports memory transactions composed of atomic oper- ations across multiple addresses.
  - ✓ **P extension:** The P extension provides packed single instruction, multiple data (SIMD) instructions for floating-point operations in small RISC-V systems.
  - ✓ **V extension:** The V extension supports data-parallel, or vector, operations. The V extension does not specify the lengths of data vectors; that decision is left to the implementers of a RISC-V processor design. A typical implementation of the V extension might support 512-bit data vectors, though implementations with up to 4,096-bit vector lengths are currently available.
  - ✓ **N extension:** The N extension provides support for handling interrupts and exceptions at the U privilege level.
  - ✓ **Zicsr extension:** The Zicsr extension performs atomic read-modify-write operations on the system CSRs. These instructions are described earlier in this chapter in the Sysíem instructions section.
  - ✓ **Zifencei extension:** The Zifencei extension defines the fence.i instruction, described in the Memory access instructions section.