NCDC

NUST CHIP DESIGN CENTRE

Computer Architecture

RISC-V – Arithmetic Logical and Shift Instructions

Lecture 2

Agenda

- Introduction to RISC-V Instruction Set
- RISC-V Assembly Language
  - Arithmetic Instructions
  - Logical and Shift Instructions
- Demo: Venus Software

Agenda

- **Introduction to RISC-V Instruction Set**
- RISC-V Assembly Language
  - Arithmetic Instructions
  - Logical and Shift Instructions
- Demo: Venus Software

What is an instruction set?

- The vocabulary of commands understood by a given architecture
  - Set of instructions a particular architecture uses
  - Like ADD A, B etc. for our simple microprocessor!
- Examples
  - MIPS (a RISC architecture) Instruction Set
    - Designed in 1980s
  - The Intel x86 (a CISC architecture) Instruction Set
    - Originated in 1970s
  - RISC-V Instruction Set
    - Developed in 2010 at UC Berkeley
    - Has much similarity with MIPS
    - Open-source and very popular

Instruction Set Architectures

- Early trend in ISA designs – more and more instructions to do elaborate operations
  - VAX architecture had an instruction to multiply polynomials!
- RISC philosophy (Cocke IBM, Patterson, Hennessy, 1980s) – Reduced Instruction Set Computing
  - Keep the instruction set small and simple
  - Let software do complicated operations by composing simpler ones
  - A simpler CPU is easier to iterate on (allowing for faster development), and can generally be made faster than a complex CPU (we're often limited by the slowest instruction we decide to implement)

Overiew of a Computing System

- A RISC-V system (or any CPU in general) is composed of two main parts:
  - The CPU, which is responsible for computing
  - Main memory, which is responsible for long-term data storage
- The CPU is designed to be extremely fast, often completing one instruction every nanosecond or faster
  - Note: Light travels 30 cm in 1 nanosecond. In other words, it takes longer for light to travel from one end of my laptop to the other, than it does for a CPU to finish one instruction.
- Going to main memory often takes hundreds or even thousands of times longer.
- The CPU can store a small amount of memory, through components called registers.

Registers in RISC-V Architecture

- 32 Registers for Integer Operands
  - Numbered x0 to x31
  - x0 is hardwired to 0 = constant 0 value!
  - Register size is 32-bit (word) in 32-bit architectures and 64-bit (doubleword) in 64-bit architectures!
- 32 Registers for Floating Point Operands
  - Numbered f0 to f31
- Different registers have different functions!
  - But most are general-purpose!
  - We will cover their specific functions in discussion about RISC-V assembly language!

Agenda

- Introduction to RISC-V Instruction Set
- **RISC-V Assembly Language**
  - Arithmetic Instructions
  - Logical and Shift Instructions
- Demo: Venus Software

Instructions

- Each line of RISC-V code is a single instruction, which executes a simple operation on registers.
- Instructions are generally written in the format
  - <instruction name> <destination register> <operands>
  - Ex. “add x5 x6 x7” means “Add the values stored in x6 and x7, and store the result in x5”
  - Commas can be added between registers (“add x5, x6, x7”), but this is optional.

Classes of Instructions

- Arithmetic
  - Example
    - add x5, x6, x7
- Logical and Shift
  - Examples
    - xor x5, x6, x7
    - srl x5, x6, x7
- Data Transfer
  - Example
    - lw x5, 40(x6) # load word
    - sw x7, 0(x2) # load doubleword (in 64-bit architecture)!
- Program Control / Branch
  - Example
    - beq x1, x2, loop
    - jal x1, target

Most Instructions have three operands / fields …

Agenda

- Introduction to RISC-V Instruction Set
- RISC-V Assembly Language
  - **Arithmetic Instructions**
  - Logical and Shift Instructions
- Demo: Venus Software

Addition and Subtraction

- Format
  - *instruction_name destination, source1, source2*
  - All operands are registers, one source can be immediate!

- Examples
  - add x5, x6, x7
    - x5 = x6 + x7
  - sub x9, x10, x11
    - x9 = x10 – x11

Exercise …

- Convert the following C-language statement into RISC-V assembly
  - f = (g + h) – (i + j)
  - Assume f, g, h, i, j are assigned to x19, x20, x21, x22 and x23
  - Use x5 and x6 as temporary registers!

- Solution

```
add x5, x20, x21        # register x5 contains g + h
add x6, x22, x23        # register x6 contains i + j
sub x19, x5, x6         # f gets x5 – x6, which is (g + h) − (i + j)
```

Comments

  - Note: We could also have written this as f = g + h - i - j; which allows us to compute this without any temporary registers. A smart compiler may write code this way instead.

Immediates

- Immediates are numerical constants
- They appear often in code, so there are special instructions for them.
- Add Immediate:
  - addi x3,x4,10 (in RISC-V)
    - x3 = x4 + 10
  - f = g + 10 (in C)
  - where RISC-V registers x3,x4 are associated with C variables f,g
- Syntax similar to add instruction, except that last argument is a number instead of a register.
- Common mistake: addi x3,x4,x5 / add x3,x4,10 are invalid!

Immediate

- Subtraction with Immediate
  - No special instruction
    - To limit types of operations to absolute minimum!
    - Is there a way to achieve this?
      - Use addi
        - addi x5, x6, –20
        - x5 = x6 – 20

- Where are immediate operands stored and how do they get to the ALU?
  - Encoded within the instruction and provided to ALU from instruction register!

Agenda

- Introduction to RISC-V Instruction Set
- RISC-V Assembly Language
  - Arithmetic Instructions
  - **Logical and Shift Instructions**
- Demo: Venus Software

Review: Bitwise Operations in C

- Bitwise AND, OR, XOR, and NOT
  - Perform the operation on the binary one bit at a time
  - Ex. 0b1001 | 0b0111 = 0b1111
  - Ex. 0b1001 & 0b0111 = 0b0001
  - Ex. 0b1001 ^ 0b0111 = 0b1110
  - Ex. ~0b1001 = 0b0110
- Left and Right Shift
  - Shift the bits of the number left/right, then fill the remaining bits
    - Ex. 0b0001 << 3 = 0b1000 = 8
    - Ex. 0b1011 >> 2 = 0b0010 = 2

Review: Bitwise Operations in C

- Shifts have numerical equivalents in math
  - Left Shift by n -> Multiply by 2^n
  - Right Shift by n -> (Floor) Divide by 2^n

- For left shifts, we always fill the new bits with 0s
- For right shifts, dividing an unsigned number should fill the bits with 0s
- But right-shifting a signed number shouldn't fill with 0s always

  - Ex. 0b1000 0010 = -126 in 8 bits. If we right-shift by 1 and fill with 0s, we get 0b0100 0001 = 65. But if we right-shift by 1 and fill with the sign bit, we get 0b1100 0001 = -63 = -126/2

Review: Bitwise Operations in C

- Logical Left Shift
  - Shift left, add zeros
  - Same as multiplication by a power of 2
- Logical Right Shift
  - Shift right, zero-extend
  - Same as dividing an unsigned number by a power of 2
- Arithmetic Right Shift
  - Shift right, sign-extend
  - Dividing a signed number by powers of 2

Logical Operations in RISC-V

- Format
  - instruction_name destination reg, source reg1, source reg2/immediate
- Examples
  - With register operands
    - and x5, x6, x7
      - x5 = x6 & x7 (bit-by-bit and operation)
  - With one immediate operand
    - xori x6, x7, 20
      - x6 = x7 ∧ 20 (bit-by-bit xor with immediate operand (signed value))
- Similar instructions: or, ori, xor, andi.

Shift Operations in RISC-V

- Format
  - instruction_name destination reg, source reg1, source reg2/immediate

- Examples
  - With register operands
    - sll x5, x6, x7
      - x5 = x6 << x7 (shift left logical x5 by value in register)
      - Logical shift left adds 0s to fill the spaces created on right!

  - With one immediate operand
    - srai x5, x6, 3
      - x5 = x6 >> 3 (arithmetic shift right by 3 bits)
      - Arithmetic shift right add sign-bit to fill the spaces created on left!

- Similar instructions: srl, sra, srli.

Pseudoinstructions

- Instructions that are more readable and assembler understands but are not part of the Instruction Set

- Examples
  - li x9, 123 # load immediate value 123 into register x9
    - li is not part of RISC-V ISA
    - Which instruction achieves the above?
      - addi x9, x0, 123 # register x9 gets register x0 + 123

  - mv x10, x11 # register x10 gets register x11
    - Equivalent Instruction?
      - add x10, x0, x11

  - nop
    - Equivalent is addi x0, x0, 0!

Practice: Bitwise Operations

- What is in register x12 after the following instructions?

```
li      x10     0x34FF
slli    x12     x10     0x10
srli    x12     x12     0x08
and     x12     x12     x10
```

Register	Value
x10	0x0000 0000
x12	0x0000 0000

Practice: Bitwise Operations

- What is in register x12 after the following instructions?

```
li      x10     0x34FF
slli    x12     x10     0x10
srli    x12     x12     0x08
and     x12     x12     x10
```

Register	Value
x10	0x0000 34FF
x12	0x0000 0000

Practice: Bitwise Operations

- What is in register x12 after the following instructions?

```
li      x10     0x34FF
slli    x12     x10     0x10
srli    x12     x12     0x08
and     x12     x12     x10
```

Register	Value
x10	0x0000 34FF
x12	0x34FF 0000

Practice: Bitwise Operations

- What is in register x12 after the following instructions?

```
li      x10     0x34FF
slli    x12     x10     0x10
srli    x12     x12     0x08
and     x12     x12     x10
```

Register	Value
x10	0x0000 34FF
x12	0x0034 FF00

Practice: Bitwise Operations

- What is in register x12 after the following instructions?

```
li      x10     0x34FF
slli    x12     x10     0x10
srli    x12     x12     0x08
and     x12     x12     x10
```

Register	Value
x10	0x0000 34FF
x12	0x0000 3400

Summary – Arithmetic, Logical and Shift

NCDC
NUST CHIP DESIGN CENTRE

- All arithmetic/logical/shift instructions in RISC-V Base ISA (RV32I)

Instruction	Description	Operation
`add rd, rs1, rs2`	add	`rd = rs1 + rs2`
`sub rd, rs1, rs2`	sub	`rd = rs1 − rs2`
`addi rd, rs1, imm`	add immediate	`rd = rs1 + SignExt(imm)`
`or rd, rs1, rs2`	or	`rd = rs1 | rs2`
`and rd, rs1, rs2`	and	`rd = rs1 & rs2`
`xor rd, rs1, rs2`	xor	`rd = rs1 ^ rs2`
`ori rd, rs1, imm`	or immediate	`rd = rs1 | SignExt(imm)`
`andi rd, rs1, imm`	and immediate	`rd = rs1 & SignExt(imm)`
`xori rd, rs1, imm`	xor immediate	`rd = rs1 ^ SignExt(imm)`
`sll rd, rs1, rs2`	shift left logical	`rd = rs1 << rs2`$_{4:0}$
`srl rd, rs1, rs2`	shift right logical	`rd = rs1 >> rs2`$_{4:0}$
`sra rd, rs1, rs2`	shift right arithmetic	`rd = rs1 >>> rs2`$_{4:0}$
`slli rd, rs1, uimm`	shift left logical immediate	`rd = rs1 << uimm`
`srli rd, rs1, uimm`	shift right logical immediate	`rd = rs1 >> uimm`
`srai rd, rs1, uimm`	shift right arithmetic imm.	`rd = rs1 >>> uimm`

Operations that are not in RISC-V!

- There's a few notable omissions in this list:
  - Bitwise NOT
    - Exists as a pseudoinstruction!
    - not x5, x5
      - Can be run by doing a bitwise XOR with 0xFFFF FFFF = -1
        - xori x5, x5, -1
  - Negate – 2's complement
    - Exists as a pseudoinstruction
    - neg x10, x10
      - Equivalent instruction?
        - sub x10, x0, x10

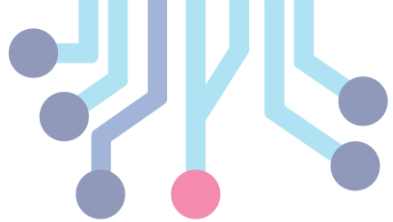
Operations that are not in RISC-V!

- Multiplication, Division, Mod
  - Circuit is significantly more complicated than bitwise ops or addition.
  - Exists as an extension of RISC-V, but is not by itself in base RISC-V
  - For multiplying/dividing by powers of 2, we can use shifts instead

- Float addition/subtraction/etc.
  - Much more complicated circuit, so also exists as an extension.

Agenda

- Introduction to RISC-V Instruction Set
- RISC-V Assembly Language
  - Arithmetic Instructions
  - Logical and Shift Instructions
- **Demo: Venus Software**



RISC-V Assembly Language Simulator

Venus: https://venus.cs61c.org/

RARS: https://github.com/TheThirdOne/rars

Assignment

- Task 1
  - Using RISC-V add / addi / sub instructions, implement 1's complement (NOT) of a value in x5 register
    - i.e., if x5 = 0xFFFFFFFF, after 1's complement x5 = 0x00000000
- Task 2
  - Write a RISC-V assembly code that creates a 32-bit constant value 0x12345678 and places it in x5
- Task 3
  - Write a RISC-V code that tests whether a number in x5 register is even or odd. The result is indicated in x10 such that x10 is 0 if the number is even and 1 if number is odd

Assignment

- Task 4
  - Write a RISC-V assembly code that toggles a specific bit of x5 with the bit number to be toggled been specified in x10
- Task 5
  - Write a RISC-V assembly code that checks if there are even number of 1's in value specified in register x5 or odd number of 1's. If there are even number of 1's in x5, set x10 to 0 otherwise set x10 to 1.
- Task 6
  - Write a RISC-V assembly code that selects one of the 4 bytes from a register x5 and places the resulting byte (with 0 extension) in x10. The byte number (0,1,2 or 3) is specified in x6.

Assignment

- Task 7
  - Write a RISC-V assembly code that selects one of the 4 bytes from a register x5 and places the resulting byte (with sign extension) in x10. The byte number (0,1, 2 or 3) is specified in x6.

- Task 8
  - Write a RISC-V assembly code that compares x5 and x6 and sets x10 to 1 if x5 is greater (signed comparison) than x6.
    - Do this without using slt (set less than) instruction which achieves the same thing!

Thank You

NCDC
NUST
CHIP
DESIGN
CENTRE