Tikrit University
Electrical Engineering Department

EE-317
Computer Engineering
2024-2025

Instructions: Introduction

Jalal Nazar Abdulbaqi, Ph.D.

jalal.abdulbaqi@tu.edu.iq

Outline

- Introduction to Instruction set
- Arithmetic Operations
- Operands
- Signed and Unsigned Numbers

Instruction Set

- A group of instructions for computer.
  - Language of the machine.
- Different computers have different instruction sets
  - But with many aspects in common
- Early computers had very simple instruction sets
  - Simplified implementation
- Many modern computers also have simple instruction sets

Instruction Set

- Assembly language is more primitive than higher level languages
  e.g., no sophisticated control flow
- The goal of computer designers is language which makes it easy to build hardware and compiler
- Aim
  - maximize **performance**
  - minimize **cost** and **energy**.

The RISC-V Instruction Set

- Used as the example throughout this course
- Developed at UC Berkeley as open ISA
- Now managed by the RISC-V Foundation (riscv.org)
- Typical of many modern ISAs
  - See RISC-V Reference Data card
- Similar ISAs have a large share of embedded core market
  - Applications in consumer electronics, network/storage equipment, cameras, printers, …

Arithmetic Operations

- Add and subtract, three operands
  - Two sources and one destination

```
add a, b, c        // a gets b + c
```

- All arithmetic operations have this form
- ***Design Principle 1***: Simplicity favors regularity
  - Regularity makes implementation simpler
  - Simplicity enables higher performance at lower cost

Arithmetic Example

- C code:

```
f = (g + h) - (i + j);
```

- Compiled RISC-V code:

```
add t0, g, h        // temp t0 = g + h
add t1, i, j        // temp t1 = i + j
sub f, t0, t1       // f = t0 - t1
```

Register Operands

- Arithmetic instructions use register operands
- RISC-V has a **32 × 32-bit** register file
  - Use for frequently accessed data
  - 32-bit data is called a "word"
  - 64-bit data is called a "doubleword"
    - 32 x 32-bit general purpose registers x0 to x31
- ***Design Principle 2:*** Smaller is faster
  - c.f. main memory: millions of locations

RISC-V Registers

- **`x0`**: the constant value 0
- **`x1`**: return address
- **`x2`**: stack pointer
- **`x3`**: global pointer
- **`x4`**: thread pointer
- **`x5–x7,x28–x31`**: temporaries
- **`x8`**: frame pointer
- **`x9,x18–x27`**: saved registers
- **`x10–x11`**: function arguments/results
- **`x12–x17`**: function arguments

Register Operand Example

- C code:

```
f = (g + h) - (i + j);
```

  - `f, …, j in x19, x20, …, x23`

- Compiled RISC-V code:

```
add x5, x20, x21
add x6, x22, x23
sub x19, x5, x6
```

Memory Operands

- Main memory used for composite data
  - Arrays, structures, dynamic data
- To apply arithmetic operations
  - Load values from memory into registers
  - Store result from register to memory
- Memory is byte addressed
  - Each address identifies an 8-bit byte
- RISC-V does not require words to be aligned in memory
  - Unlike some other ISAs
- RISC-V is Little Endian
  - Least-significant byte at least address of a word
  - *c.f.* Big Endian: most-significant byte at least address

Memory Operand Example

- C code:

```
A[12] = h + A[8];
```

  - `h in x21, base address of A in x22`

- Compiled RISC-V code:
  - Index 8 requires offset of 32
    - **4** bytes per word

```
lw      x9, 32(x22)
add     x9, x21, x9
sw      x9, 48(x22)
```

Registers vs. Memory

- Registers are faster to access than memory
- Operating on memory data requires loads and stores
  - More instructions to be executed
- Compiler must use registers for variables as much as possible
  - Only spill to memory for less frequently used variables
  - Register optimization is important!

Immediate Operands

- Constant data specified in an instruction (12-bit)

```
addi x22, x22, 4
```

- Make the common case fast
  - Small constants are common
  - Immediate operand avoids a load instruction

Unsigned Binary Integers

- Given an n-bit number

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Range: 0 to $+2^n - 1$
- Example
  - $0000\ 0000\ \ldots\ 0000\ 1011_{two}$
    $= 0 + \ldots + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$
    $= 0 + \ldots + 8 + 0 + 2 + 1 = 11_{ten}$
- Using 32 bits: 0 to $(2^{32}-1) = +4{,}294{,}967{,}295_{ten}$

2s-Complement Signed Integers

- Given an n-bit number

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Range: -2^{n-1} to $+2^{n-1} - 1$
- Example
  - $1111\ 1111 \ldots 1111\ 1100_{two}$
    $= -1\times2^{31} + 1\times2^{30} + \ldots + 1\times2^{2} + 0\times2^{1} + 0\times2^{0}$
    $= -2{,}147{,}483{,}648 + 2{,}147{,}483{,}644 = -4_{ten}$
- Using 32 bits: -2^{32-1} = $-2.147{,}483{,}648_{ten}$
  to $+2^{32-1}-1$ = $2.147{,}483{,}647_{ten}$

2s-Complement Signed Integers

- Bit 31 is sign bit
  - 1 for negative numbers
  - 0 for non-negative numbers
- $-(-2^{n-1})$ can't be represented
- Non-negative numbers have the same unsigned and 2s-complement representation
- Some specific numbers
  - 0: 0000 0000 … 0000
  - –1: 1111 1111 … 1111
  - Most-negative: 1000 0000 … 0000
  - Most-positive: 0111 1111 … 1111

Signed Negation

- Complement and add 1
  - Complement means 1 → 0, 0 → 1

$$x + \overline{x} = 1111\ldots111_2 = -1$$

$$\overline{x} + 1 = -x$$

- Example: negate +2
  - $+2 = 0000\ 0000 \ldots 0010_{two}$
  - $-2 = 1111\ 1111 \ldots 1101_{two} + 1$
    $= 1111\ 1111 \ldots 1110_{two}$

Sign Extension

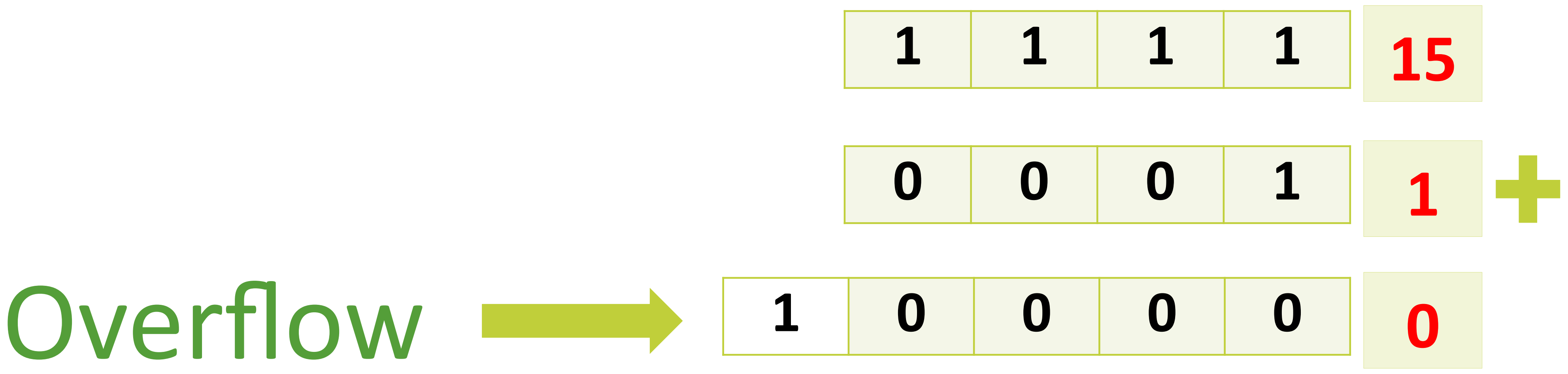
- Representing a number using more bits
  - Preserve the numeric value
- Replicate the sign bit to the left
  - c.f. unsigned values: extend with 0s
- Examples: 8-bit to 16-bit
  - +2: 0000 0010 => 0000 0000 0000 0010
  - –2: 1111 1110 => 1111 1111 1111 1110

- In RISC-V instruction set
  - **`lb`**: sign-extend loaded byte
  - **`lbu`**: zero-extend loaded byte

Overflow

- **Overflow** occurs when results of an operation larger than be in a register.

Unsigned



Overflow

- **Overflow** occurs when the leftmost retained bit of the binary bit pattern is not the same as the infinite number of digits to the left (the sign bit is incorrect): a 0 on the left of the bit pattern when the number is negative or a 1 when the number is positive.

Signed

1	0	0	0	-8
1	0	0	1	-7 +
0	0	0	1	1

0	1	1	1	7
0	0	0	1	1 +
1	0	0	0	-8

Overflow

Multiplication and Division

- There are no instructions for multiplications and division operations in RISC-V 32-bit Integer base version (RV32I).
- Instead, we can use shift instructions to multiply or divide by $\mathbf{2^i}$ i times.
- **`slli`** – Shift left logical immediate
  - Shift left and fill with 0 bits
  - **`slli`** by ***i*** bits multiplies by $\mathbf{2}^{i}$
- **`srli`** – Shift right logical immediate
  - Shift right and fill with 0 bits
  - **`srli`** by ***i*** bits divides by $\mathbf{2}^{i}$ (unsigned only)

Multiplication and Division

8	4	2	1
2^3	2^2	2^1	2^0
0	**1**	**0**	**0**

8	4	2	1
2^3	2^2	2^1	2^0
0	**0**	**1**	**0**

8	4	2	1
2^3	2^2	2^1	2^0
0	**0**	**0**	**1**

4 $\xleftarrow{\times 2}$ 2 $\xrightarrow{\div 2}$ 1

`slli // Shift to left`   `srli // Shift to right`

Multiplication and Division

8	4	2	1
2^3	2^2	2^1	2^0
1	**0**	**0**	**0**

8	4	2	1
2^3	2^2	2^1	2^0
0	**1**	**0**	**0**

8	4	2	1
2^3	2^2	2^1	2^0
0	**0**	**1**	**0**

$$8 \xleftarrow{\times 2} 4 \xleftarrow{\times 2} 2$$

```
slli x6, x6, 2     // Shift to left twice
```

Number Systems Conversion

- Binary to Decimal:

$$\mathbf{10110_{two}} = 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 1\times 2^1 + 0\times 2^0$$

$$= \mathbf{22_{ten}}$$

Number Systems Conversion

- Decimal to Binary:

$\mathbf{37_{ten} \rightarrow ?_{two}}$

37 / 2 = 18 reminder **1** LSB – Least Significant Bit
18 / 2 = 9 reminder **0**
9 / 2 = 4 reminder **1**
4 / 2 = 2 reminder **0**
2 / 2 = 1 reminder **0**
1 / 2 = 0 reminder **1** MSB – Most Significant Bit

$\mathbf{37_{ten} = 100101_{two}}$

Number Systems Conversion

- Hexadecimal to Decimal:

$$\mathbf{374F_{hex}} = 3 \times 16^3 + 7 \times 16^2 + 4 \times 16^1 + 15 \times 16^0$$
$$= \mathbf{14159_{ten}}$$

Number Systems Conversion

- Decimal to Hexadecimal :

12345_{ten} → $?_{hex}$

12345 / 16 = 771 reminder **9** LSB
771 / 16 = 48 reminder **3**
48 / 16 = 3 reminder **0**
3 / 16 = 0 reminder **3** MSB

12345_{ten} = 3039_{hex}

Number Systems Conversion

- Hexadecimal and Binary
  - Compact representation of bit strings
  - 4 bits per hex digit

0	0000	4	0100	8	1000	c	1100
1	0001	5	0101	9	1001	d	1101
2	0010	6	0110	a	1010	e	1110
3	0011	7	0111	b	1011	f	1111

Number Systems Conversion

- Binary to Hexadecimal:

$\mathbf{10110101_{two} = ?_{hex}}$ → $1011\ 0101_{two} = \mathbf{B5_{hex}}$

B 5

Number Systems Conversion

- Hexadecimal to Binary:

$\mathbf{374F_{hex} = ?_{two}} \rightarrow$ 3 7 4 F $= \mathbf{0011\ 0111\ 0100\ 1111_{two}}$

0011 0111 0100 1111

Number Systems Notion in RISC-V

- **10110101_{two} → 0b10110101**
- **$374F_{hex}$ → 0x374F**
- **12345_{ten} → 12345**
- Examples:

```
addi x5, x0, 0b10110101
addi x6, x0, 0x374F
addi x4, x0, 12345
```

Summary

- The chosen instruction set is **RISC-V** because of its **simplicity** and **generality** so that other architectures can be realized
- There are *two* arithmetic operations in RV32I: Addition - **add** and Subtraction – **sub**
- There are *three* types of operands: **registers**, **memory** and **immediate**
- Both **positive** and **negative integers** can be represented within a computer and the best way is **two's complement**