



香港中文大學

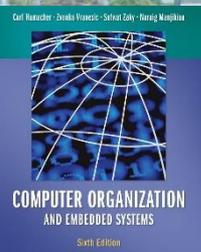
The Chinese University of Hong Kong

# CSCI2510 Computer Organization

## Lecture 04: Machine Instructions

**Ming-Chang YANG**

[mcyang@cse.cuhk.edu.hk](mailto:mcyang@cse.cuhk.edu.hk)

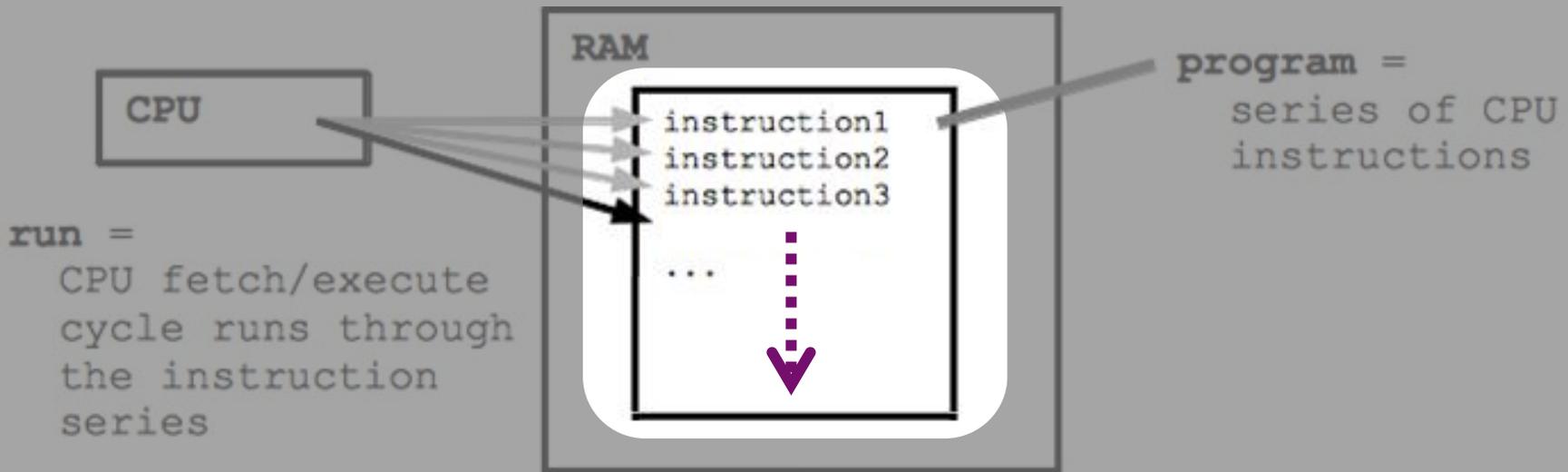


Reading: Chap. 2.3~2.4, 2.10~2.11

# Activities in a Computer: Instructions



- A computer is governed by instructions.
  - To perform a given task, a **program** consisting of a list of **machine instructions** is stored in the memory.
    - Data to be used as **operands** are also stored in the memory.
  - **Individual instructions** are brought from the memory into the processor, one after another, in a **sequential** way (normally).
  - The processor executes the specified operation/instruction.





- Machine Instruction Notations
  - Register Transfer Notation (RTN)
  - Assembly-Language Notation
- Basic Addressing Modes
  - Immediate, Register, Absolute, Register Indirect, Index, Base with Index Modes
- RISC and CISC Styles
  - RISC Instruction Sets
  - CISC Instruction Sets
    - Additional Addressing Modes

# Machine Instructions



- The tasks carried out by a computer program consist of a sequence of machine instructions.
- Machine instructions must be capable of performing the following **four** types of operations:
  - 1) **Data transfer** between memory and processor registers
  - 2) **Arithmetic and logical operations** on data in processor
  - 3) **Program sequencing and control** (e.g. branches, subroutine calls)
  - 4) **I/O transfers**
- Machine instructions are represented by **0s** and **1s**.

*To ease the discussion, we first need some **notations**.*



- Machine Instruction Notations
  - Register Transfer Notation (RTN)
  - Assembly-Language Notation
- Basic Addressing Modes
  - Immediate, Register, Absolute, Register Indirect, Index, Base with Index Modes
- RISC and CISC Styles
  - RISC Instruction Sets
  - CISC Instruction Sets
    - Additional Addressing Modes

# Register Transfer Notation (RTN)



- Register Transfer Notation (RTN) describes the data transfer from one location in computer to another.
  - Possible locations: memory locations, processor registers.
    - Locations can be identified **symbolically** with names (e.g. LOC).

**Ex.**

**R2** ← **[LOC]**

- *Transferring the contents of memory LOC into register R2.*

- ① **Contents of any location**: denoted by placing square brackets **[ ]** around its location name (e.g. **[LOC]**).
- ② **Right-hand side** of RTN: always denotes a **value**
- ③ **Left-hand side** of RTN: the name of a **location** where the value is to be placed (by overwriting the old contents)



- Machine Instruction Notations
  - Register Transfer Notation (RTN)
  - Assembly-Language Notation
- Basic Addressing Modes
  - Immediate, Register, Absolute, Register Indirect, Index, Base with Index Modes
- RISC and CISC Styles
  - RISC Instruction Sets
  - CISC Instruction Sets
    - Additional Addressing Modes

# Assembly-Language Notation



- Assembly-Language Notation is used to represent machine instructions and programs.
  - An instruction must specify an **operation** to be performed and the **operands** involved.
  - **Ex.** The instruction that causes the transfer from memory location LOC to register R2:

**Load** R2, LOC

**Load:** operation;

**LOC:** source operand;

**R2:** destination operand.

*Some machines may put  
destination last:  
operation src, dest*

- Sometimes operations are defined by using **mnemonics**.
  - **Mnemonics:** abbreviations of the words describing operations
  - E.g. **Load** can be written as **LD**, **Store** can be written as **STR** or **ST**.

# Class Exercise 4.1

Student ID: \_\_\_\_\_ Date: \_\_\_\_\_

Name: \_\_\_\_\_

- Given an **Add** instruction that
  - ① Adds the contents of registers R2 and R3, and
  - ② Places the sum into R4.
- Represent this instruction by using
  - Register Transfer Notation (RTN):
  - Answer: \_\_\_\_\_
  
  - Assembly-Language Notation:
  - Answer: \_\_\_\_\_



- Machine Instruction Notations
  - Register Transfer Notation (RTN)
  - Assembly-Language Notation
- **Basic Addressing Modes**
  - Immediate, Register, Absolute, Register Indirect, Index, Base with Index Modes
- RISC and CISC Styles
  - RISC Instruction Sets
  - CISC Instruction Sets
    - Additional Addressing Modes

# Type of Operands: Address Modes

- **Addressing Modes:** the ways for specifying the contents or locations of instruction operands.

| Address Mode         | Assembler Syntax | Addressing Function |
|----------------------|------------------|---------------------|
| 1) Immediate         | $\#Value$        | $Operand = Value$   |
| 2) Register          | $Ri$             | $EA = Ri$           |
| 3) Absolute          | $LOC$            | $EA = LOC$          |
| 4) Register indirect | $(Ri)$           | $EA = [Ri]$         |
| 5) Index             | $X(Ri)$          | $EA = [Ri] + X$     |
| 6) Base with index   | $(Ri, Rj)$       | $EA = [Ri] + [Rj]$  |

*Value*: a signed number

*EA*: the effective address of a register or a memory location

*X*: an index value

# 1) Immediate Mode



- **Immediate Mode:** the operand is given explicitly as **value** in the instruction.

**Ex.**

**Add R4 , R6 , #200**

- This instruction adds the value 200 to the contents of register R6, and places the result into register R4.
- The convention is to use the **number sign (#)** in front of the value to indicate that this value is an **immediate operand**.

- **Note:** The immediate mode
  - Does **NOT** give the operand or its address explicitly, but
  - Provides **constants** from which an effective address (EA) can be derived/calculated by the processor.
    - E.g.  $PC \leftarrow [PC] + 4$

# 2) Register Mode

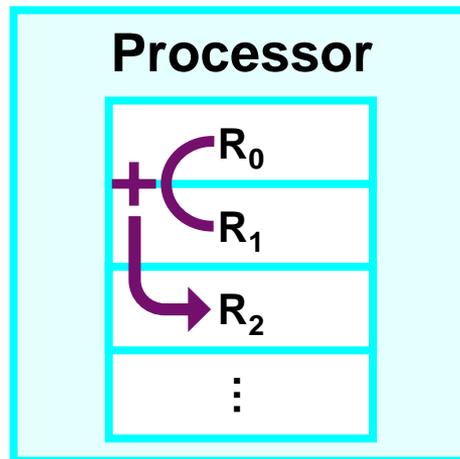


- **Register Mode:** the operand is **the contents of a processor register**; the name of the register is given.

**Ex.**

**Add R2, R0, R1**

- This instruction uses the Register mode for all 3 operands.
  - Registers R0 and R1 hold the two **source operands**, while R2 is the **destination operand**.



# 3) Absolute Mode

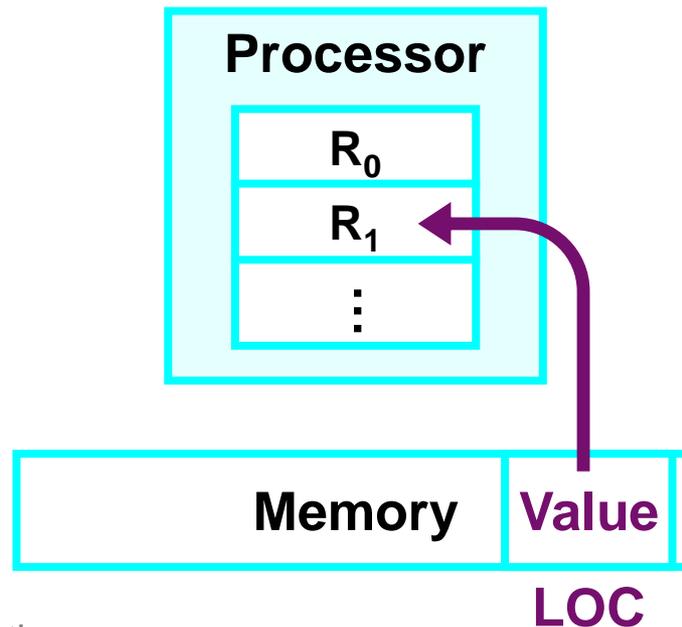


- **Absolute Mode:** the operand is **the contents of a memory location**; the address of this location is given.

**Ex.**

Load R1, LOC

- This instruction loads the value in the memory location **LOC** into register R1.



# 4) Register Indirect Mode (1/2)

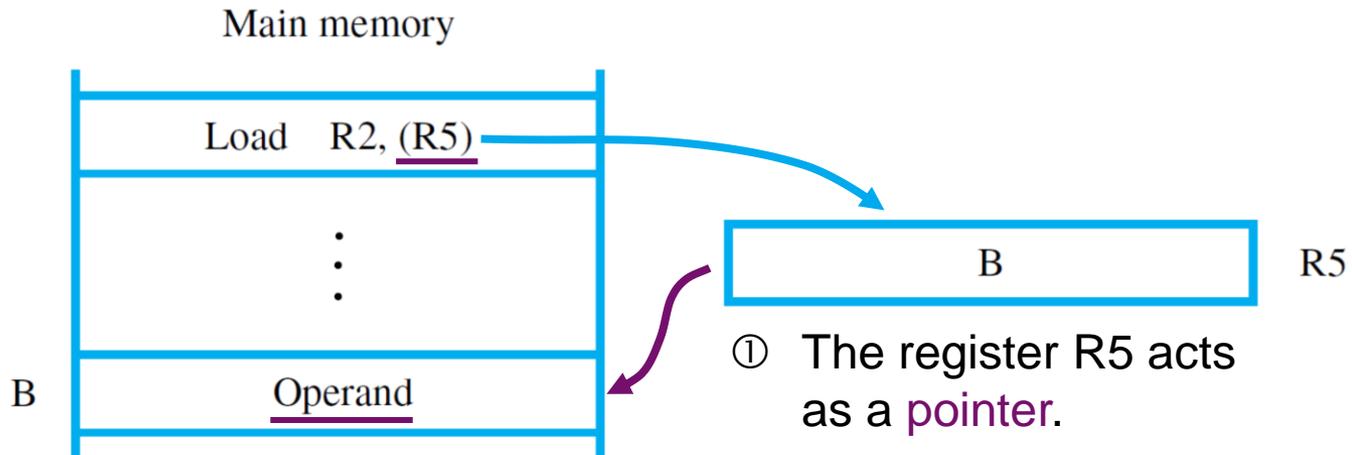


- **Register Indirect Mode:** the effective memory address of the operand is **the contents of a register**.

**Ex.**

**Load R2, (R5)**

- This instruction uses the value B, which is stored in register R5, as the effective address of the operand.
  - The indirection can be denoted by placing the name of the register given in the instruction in **parentheses ( )**.



② The memory content is accessed *indirectly* by using the content in the register.

① The register R5 acts as a **pointer**.

# 4) Register Indirect Mode (2/2)



- **Indirection** is important and powerful in programming.
  - For example, indirect addressing can be used to access successive numbers in a list.

|                             |                      |   |
|-----------------------------|----------------------|---|
| Load                        | R2, N                | Load the size of the list.                |
| Clear                       | R3                   | Initialize sum to 0.                      |
| Move                        | <b>R4, addr NUM1</b> | <b>Get address of the first number.</b>   |
| LOOP: Load                  | R5, <b>(R4)</b>      | <b>Get the next number.</b>               |
| Add                         | R3, R3, R5           | Add this number to sum.                   |
| Add                         | <b>R4, R4, #4</b>    | <b>Increment the pointer to the list.</b> |
| Subtract                    | R2, R2, #1           | Decrement the counter.                    |
| Branch_if_ <b>[R2]&gt;0</b> | LOOP                 | Branch back if not finished.              |
| Store                       | R3, SUM              | Store the final sum.                      |

- Register **R4** is used as a pointer to the numbers in the list, and the operands are accessed indirectly through R4.
- We will illustrate this code segment in Lecture 05.

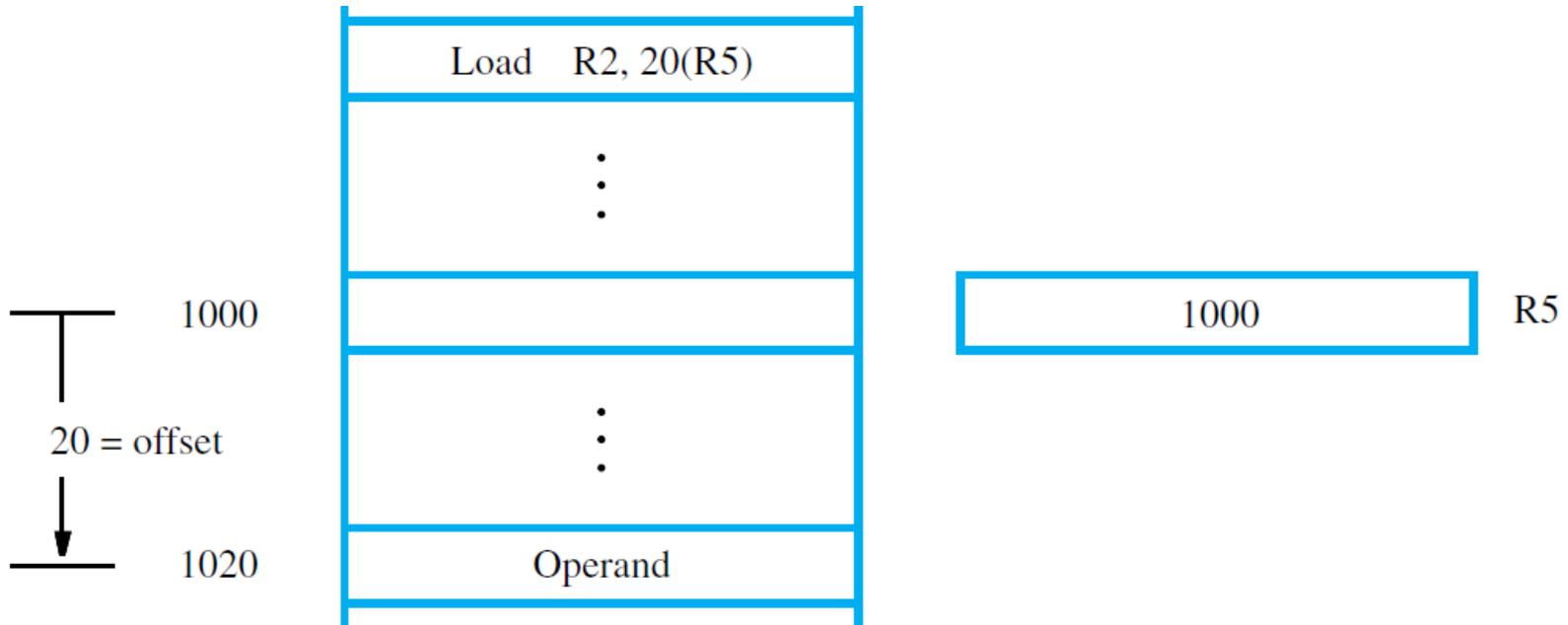
# 5) Index Mode



- **Index Mode:** the effective memory address of the operand is generated by **adding a constant index value to the contents of a register.**

**Ex.**                      **Load R2, 20(R5)**

- The index register, R5, contains the address of a memory location, and the value **20** ahead of (R5) defines an *offset*.



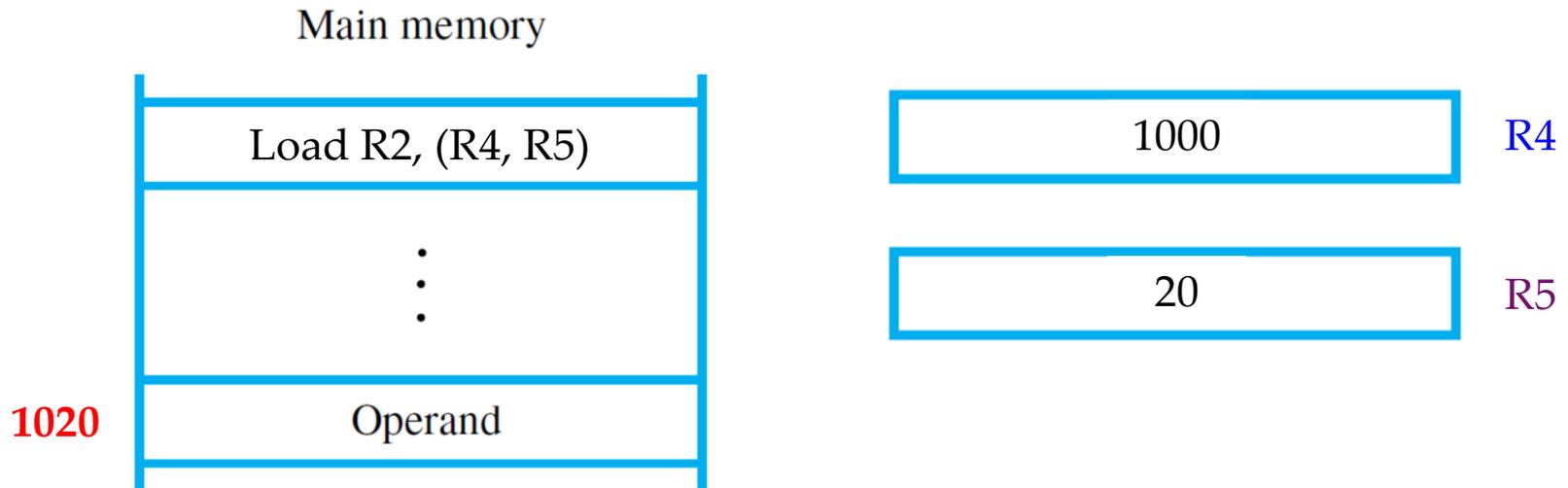
# 6) Base with Index Mode



- **Base with Index Mode:** the effective memory address of the operand is the **sum of contents of two registers** (e.g.  $R_i$  and  $R_j$ ).

**Ex.**                    **Load R2, (R4, R5)**

- The first register **R4** is usually called the **index register**.
- The second register **R5** is usually called the **base register**.



# Class Exercise 4.2



- Registers R1 and R2 of a computer contain the decimal values 1200 and 4600.
- What is the effective address (EA) for each of the following operands?
  - a) 20 (R1)  
– Answer: \_\_\_\_\_
  - b) #3000  
– Answer: \_\_\_\_\_
  - c) 30 (R1, R2)  
– Answer: \_\_\_\_\_



- Machine Instruction Notations
  - Register Transfer Notation (RTN)
  - Assembly-Language Notation
- Basic Addressing Modes
  - Immediate, Register, Absolute, Register Indirect, Index, Base with Index Modes
- **RISC and CISC Styles**
  - RISC Instruction Sets
  - CISC Instruction Sets
    - Additional Addressing Modes



- There are two fundamentally different approaches in the design of instruction sets for modern computers:

## 1) Reduced Instruction Set Computer (RISC)

- Complexity and the types of instructions can be **reduced** with the premise that **higher performance** can be achieved.
  - Each instruction occupies one word in memory.
  - Arithmetic/logic operations can be performed only on operands in the processor registers.

## 2) Complex Instruction Set Computer (CISC)

- More **complicated** or **powerful** instructions can be designed.
  - Each instruction may span more than one word in memory.
  - Arithmetic/logic operations can be performed not only on operands in the processor registers (but also operands in the memory).



- Machine Instruction Notations
  - Register Transfer Notation (RTN)
  - Assembly-Language Notation
- Basic Addressing Modes
  - Immediate, Register, Absolute, Register Indirect, Index, Base with Index Modes
- **RISC and CISC Styles**
  - RISC Instruction Sets
  - CISC Instruction Sets
    - Additional Addressing Modes



- Two key characteristics of RISC instruction sets are:
  - 1) Each instruction fits in a single word.
  - 2) A load/store architecture is used, in which
    - Memory operands are accessed only using Load and Store.  
Ex. Load/Store Ri, LOC
    - All operands involved in an arithmetic or logic operation must either be in processor registers, or  
Ex. Add R2, R0, R1
    - one of the operands is given explicitly within the word.  
Ex. Mov R0, #0

# RISC Instruction Sets Example



- Consider a typical arithmetic operation:

$$C = A + B$$

*where A, B, and C, are in distinct memory locations.*

- If we refer to the addresses of these locations as A, B, and C, respectively, this operation can be accomplished by the following **RISC instructions**:

**Load R0, A**

**Load R1, B**

**Add R2, R0, R1**

**Store R2, C**

# Class Exercise 4.3



- Question: Can we accomplish the  $C = A + B$  arithmetic operation with fewer registers using RISC instructions?
- Answer:



- Machine Instruction Notations
  - Register Transfer Notation (RTN)
  - Assembly-Language Notation
- Basic Addressing Modes
  - Immediate, Register, Absolute, Register Indirect, Index, Base with Index Modes
- **RISC and CISC Styles**
  - RISC Instruction Sets
  - **CISC Instruction Sets**
    - Additional Addressing Modes



- Two key differences between CISC and RISC:
  - 1) CISC does **NOT** have to fit into a **single word**.
  - 2) CISC is **NOT** constrained by the **load/store architecture**.
    - In RISC load/store architecture, arithmetic and logic operations can be performed only on operands that are in processor registers.
- CISC instructions typically do **NOT** use a **three-operand format**, but use the **two-operand format**:  
**operation destination, source**
  - E.g. a CISC **Add** instruction of two-address format:  
**Add B, A**
    - which performs the operation  $B \leftarrow [A] + [B]$  on memory operands.

# CISC Instruction Sets Example



- Consider the same typical arithmetic operation:

$$C = A + B$$

*where A, B, and C, are in distinct memory locations.*

- If we also refer to the addresses of these locations as A, B, and C, respectively, this operation can be accomplished by the following **CISC instructions**:

**Move C, B**

**Add C, A**

# Class Exercise 4.4



- Consider the same typical arithmetic operation:

$$C = A + B$$

*where  $A$ ,  $B$ , and  $C$ , are in distinct memory locations.*

- Question: What if a CISC processor only allows one operand to be in memory, but the other must be in register?
- Answer:



- Machine Instruction Notations
  - Register Transfer Notation (RTN)
  - Assembly-Language Notation
- Basic Addressing Modes
  - Immediate, Register, Absolute, Register Indirect, Index, Base with Index Modes
- **RISC and CISC Styles**
  - RISC Instruction Sets
  - **CISC Instruction Sets**
    - Additional Addressing Modes

# Additional Addressing Modes in CSIC



- Most **CISC** processors have all of the five basic addressing modes—Immediate, Register, Absolute, Indirect, and Index.
- **Three additional addressing modes** are often found in CISC processors:

| Address Mode      | Assembler Syntax | Addressing Function          |
|-------------------|------------------|------------------------------|
| 1*) Autoincrement | $(Ri) +$         | $EA = [Ri]$<br>$Ri = Ri + S$ |
| 2*) Autodecrement | $-(Ri)$          | $Ri = Ri - S$<br>$EA = [Ri]$ |
| 3*) Relative      | $X(PC)$          | $EA = [PC] + X$              |

*EA: effective address*

*X: index value*

*S: increment/decrement step*

# Autoincrement Mode



- **Autoincrement Mode**

- The effective address of the operand is the **contents of a register** specified in the instruction.
- After accessing the operand, the contents of register are **automatically incremented** to the next operand in memory.
  - The **increment step** is 1 for byte-sized operands, 2 for 16-bit operands, and 4 for 32-bit operands in byte-addressable memory.

- The Autoincrement mode is written as

**(Ri) +**

- Put the specified register in **parentheses**
  - To indicate the contents of the register are used as effective address.
- Followed by a **plus sign**
  - To indicate these contents are to be incremented **after the operand is accessed**.



- **Autodecrement Mode**

- The contents of a register specified in the instruction are first **automatically decremented**.
- The **contents of a register** are then used as the effective address of the operand.

- The Autoincrement mode is written as

**- (Ri)**

- Putting the specified register in **parentheses**,
- Preceded by a **minus sign**
  - To indicate the contents of the register are to be decremented **before being used as the effective address**.



- We have defined the **Index Mode** by using general-purpose processor registers.
- Some CISC processors have a version of this mode in which the **program counter (PC)** can be also used.
- **Relative Mode:** the effective address is determined by the Index mode using the **program counter (PC)** in place of the general-purpose register  $R_i$ .

**Ex.**                      **Load R2 , 20 (PC)**

- The PC contains the address of a memory location, and the value **20** ahead of (PC) defines an *offset*.
- That is, the addressed location is identified **relative to** the PC, which always indicates **the current execution point** in a program.

# RISC vs. CISC Styles



## RISC

Simple addressing modes

All instructions fitting in a single word

Fewer instructions in the instruction set, and simpler addressing modes

Arithmetic and logic operations that can be performed only on operands in processor registers

Don't allow direct transfers from one memory location to another  
Note: Such transfers must take place via a processor register.

Programs that tend to be larger in size, because more but simpler instructions are needed to perform complex tasks

Simple instructions that are conducive to fast execution by the processing unit using techniques such as pipelining (see Lec12)

## CISC

More complex addressing modes

More complex instructions, where an instruction may span multiple words

Many instructions that implement complex tasks, and complicated addressing modes

Arithmetic and logic operations that can be performed on operands in both memory and processor registers

Possible to transfer from one memory location to another by using a single Move instruction

Programs that tend to be smaller in size, because fewer but more complex instructions are needed to perform complex tasks



- Machine Instruction Notations
  - Register Transfer Notation (RTN)
  - Assembly-Language Notation
- Basic Addressing Modes
  - Immediate, Register, Absolute, Register Indirect, Index, Base with Index Modes
- RISC and CISC Styles
  - RISC Instruction Sets
  - CISC Instruction Sets
    - Additional Addressing Modes