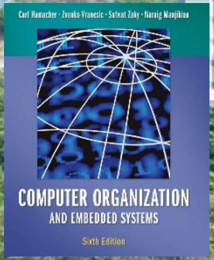
香港中文大學
The Chinese University of Hong Kong

*CSCI2510 Computer Organization*

# Lecture 04: Machine Instructions

**Ming-Chang YANG**

*mcyang@cse.cuhk.edu.hk*


Carl Hamacher · Zvonko Vranesic · Safwat Zaky · Naraig Manjikian

COMPUTER ORGANIZATION
AND EMBEDDED SYSTEMS

Sixth Edition

*Reading: Chap. 2.3~2.4, 2.10~2.11*

## Example of Language Translation

High-level Language

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
TEMP = V(k);
V(k) = V(k+1);
V(k+1) = TEMP;
```

C/Java Compiler

Fortran Compiler

Assembly Language

`lw:` loads a word from **memory** into a register
`sw:` saves a word from a register into **RAM**
`0($2):` treats the value of register $2 + 0 bytes as a location
`4($2):` treats the value of register $2 + 4 bytes as a location

```
lw $t0, 0($2)
lw $t1, 4($2)
sw $t1, 0($2)
Sw $t0, 4($2)
```

MIPS Assembler

Machine Language

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

https://gerardnico.com/code/lang/machine
https://clip2art.com/explore/Boy%20clipart%20teacher/

## Outline

- 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 can perform the following **four** types of operations to govern a computer:
  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**.*

## Outline

- 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.**

$$R2 \leftarrow [LOC]$$

– *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)

## Outline

- 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: ______________________________

## Outline

- 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: Addressing Modes

- **Addressing Modes**: the ways for specifying the **effective address (EA)** of an instruction operand.

| 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*

**Different assembly language (e.g., MASM) may have different syntax.**

## Addressing Mode 1) Immediate

- **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.

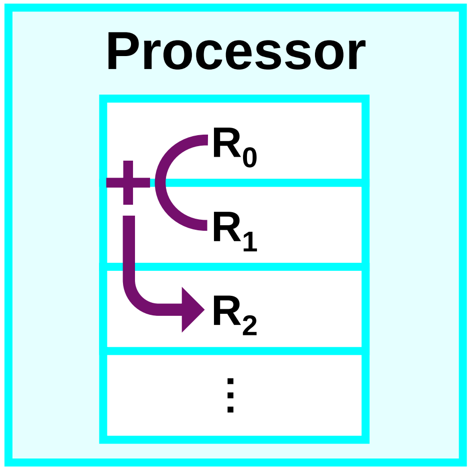
- Note: The immediate mode
  – Does NOT give the operand or its address explicitly, but
  – Provides constants from which an effective address (EA) can be derived/calculated by the processor.
    - E.g. PC ← [PC] + 4

# Addressing Mode 2) Register

- **Register Mode**: the operand is the content of a processor register.
  - That is, the name of the register is explicitly specified as the **effective address** of the operand.

  **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.

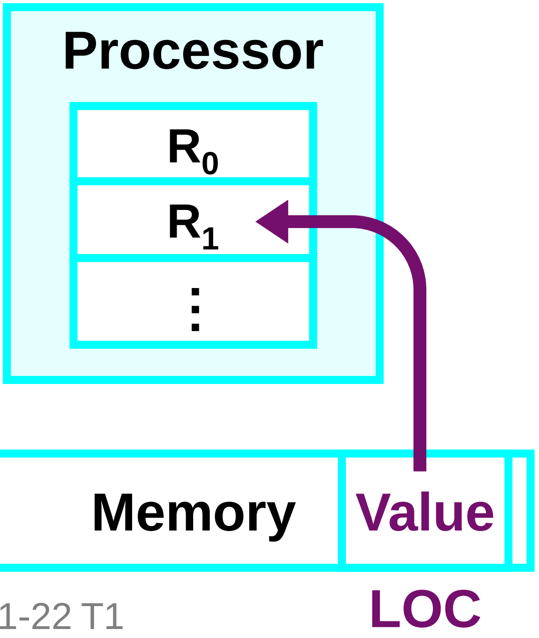


## Addressing Mode 3) Absolute

- **Absolute Mode**: the operand is the content of a memory location.
  - That is, the address of this location is explicitly specified as the **effective address** of the operand.

  **Ex.** `Load R1, LOC`

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



# Addressing Mode 4) Register Indirect (1/2)

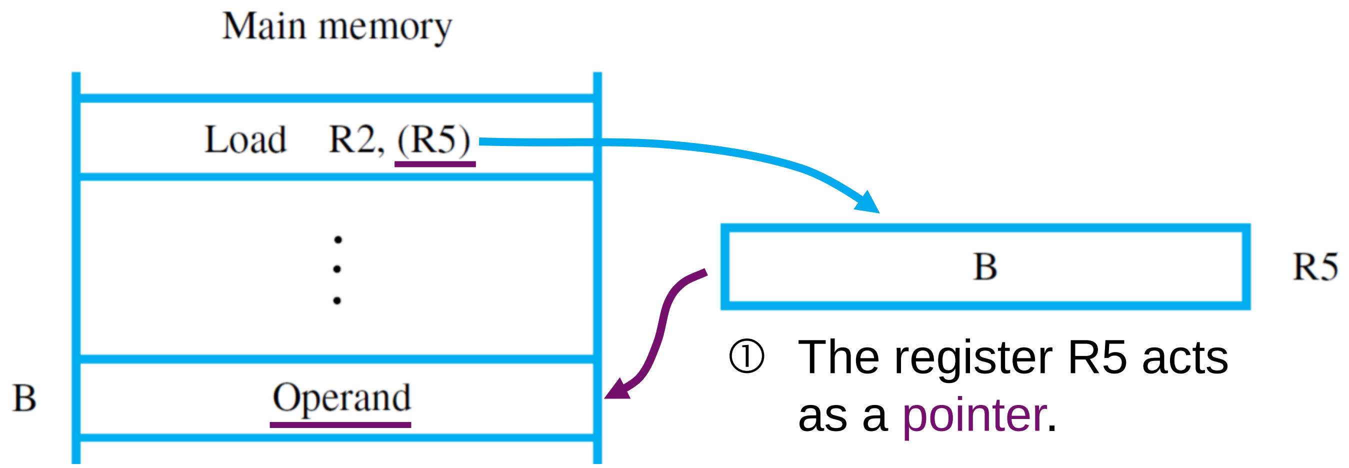
- **Register Indirect Mode**: the **effective address** of the operand is the content of a register.

  **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.

## Addressing Mode 4) Register Indirect (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              R4, addr NUM1  Get address of the first number.
LOOP:  Load              R5, (R4)       Get the next number.
       Add               R3, R3, R5     Add this number to sum.
       Add               R4, R4, #4     Increment the pointer to the list.
       Subtract          R2, R2, #1     Decrement the counter.
       Branch_if_[R2]>0  LOOP           Branch back if not finished.
       Store             R3, SUM        Store the final sum.
```

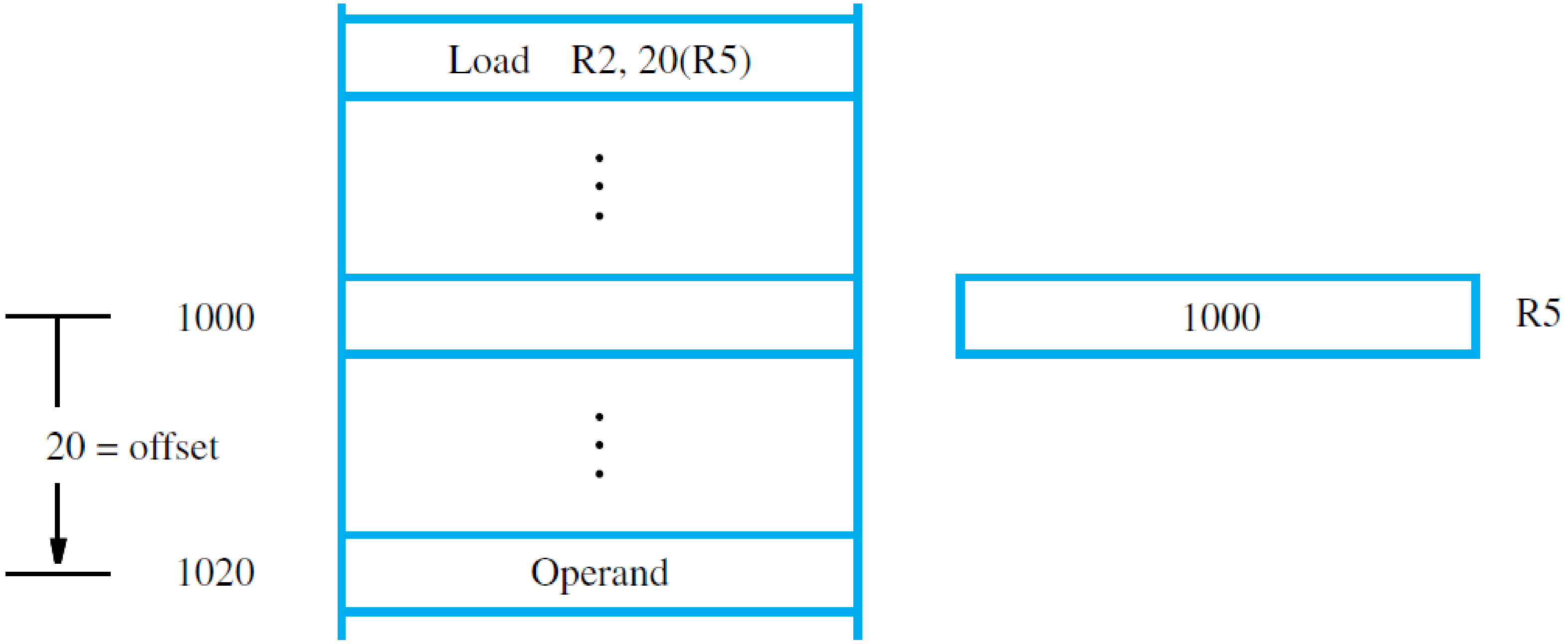
  - Register **R4** is used as a pointer to the numbers in the list, and the operands are accessed indirectly through R4.
  - We will illustrate this code segment in Lecture 05.

# Addressing Mode 5) Index

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



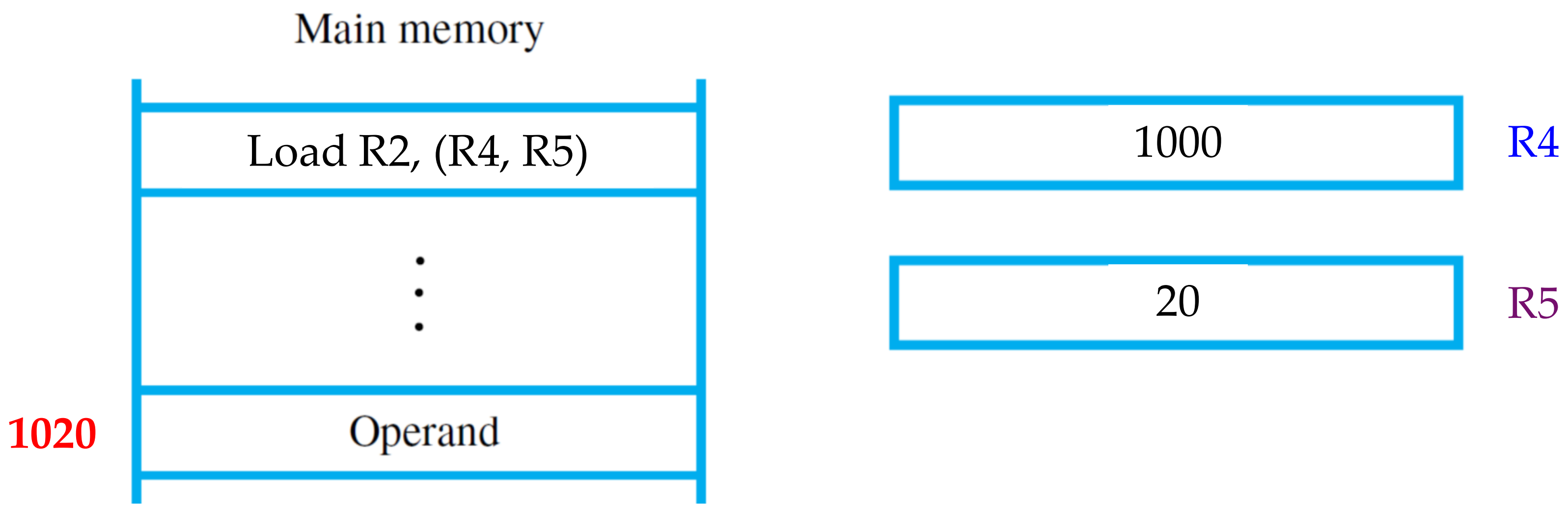
# Addressing Mode 6) Base with Index

- **Base with Index Mode**: the **effective 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: ____________________

## Outline

- 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

## RISC and CISC Styles

- 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 <u>one word</u> in memory.
  - Arithmetic/logic operations can be performed <u>only on</u> operands in the processor registers.

**2) Complex Instruction Set Computer (CISC)**

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

## Outline

- 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

## Introduction to RISC Instruction Sets

- 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 <u>fewer registers</u> using RISC instructions?
- Answer:

## Outline

- 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

## Introduction to CISC Instruction Sets

- 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 ← [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:

## Outline

- 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

- **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.

# Relative Mode

- 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 Ri.

  **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 | CISC |
|---|---|
| Simple addressing modes | More complex addressing modes |
| All instructions fitting in a single word | More complex instructions, where an instruction may span multiple words |
| Fewer instructions in the instruction set, and simpler addressing modes | Many instructions that implement complex tasks, and complicated addressing modes |
| Arithmetic and logic operations that can be performed only on operands in processor registers | Arithmetic and logic operations that can be performed on operands in both memory and processor registers |
| Don’t allow direct transfers from one memory location to another<br>Note: Such transfers must take place via a processor register. | Possible to transfer from one memory location to another by using a single Move instruction |
| Programs that tend to be larger in size, because more but simpler instructions are needed to perform complex tasks | Programs that tend to be smaller in size, because fewer but more complex 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) | |

## Class Exercise 4.5

- Given the following two programs that compute the dot product of two vectors of length n. Can you tell which one is RISC-style and which one is CISC-style?

**Program 1**

```
        Move             R2, addr AVEC
        Move             R3, addr BVEC
        Load             R4, N
        Clear            R5
LOOP:   Load             R6, (R2)
        Load             R7, (R3)
        Multiply         R8, R6, R7
        Add              R5, R5, R8
        Add              R2, R2, #4
        Add              R3, R3, #4
        Subtract         R4, R4, #1
        Branch_if_[R4]>0 LOOP
        Store            R5, DOTPROD
```

**Program 2**

```
        Move       R2, addr AVEC
        Move       R3, addr BVEC
        Move       R4, N
        Clear      R5
LOOP:   Move       R6, (R2)+
        Multiply   R6, (R3)+
        Add        R5, R6
        Subtract   R4, #1
        Branch>0   LOOP
        Move       DOTPROD, R5
```

## Addressing Modes in MASM

- **Addressing Modes**: the ways for specifying the **effective address (EA)** of an instruction operand.

| Address Mode | Assembler Syntax | MASM Syntax | Addressing Function |
|---|---|---|---|
| 1) Immediate | $\#Value$ | $Value$ (e.g., 25) | $Operand = Value$ |
| 2) Register | $Ri$ | $Ri$ (e.g., EAX) | $EA = Ri$ |
| 3) Absolute | $LOC$ | $LOC$ (e.g., offset data) | $EA = LOC$ |
| 4) Register indirect | $(Ri)$ | $[Ri]$ (e.g., [EAX]) | $EA = [Ri]$ |
| 5) Index | $X(Ri)$ | $X[Ri]$ (e.g., 4[EAX]) | $EA = [Ri] + X$ |
| 6) Base with index | $(Ri, Rj)$ | $[Ri][Rj]$ or $[Ri + Rj]$ | $EA = [Ri] + [Rj]$ |

*$Value$: a signed number*

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

*$X$: an index value*

## Summary

- 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