Computer Architecture
3. Instruction Sets

Lecturer: A.Prof.Dr. Hoàng Xuân Dậu

Email: dauhx@ptit.edu.vn

Faculty of Information Security

Posts & Telecommunications
Institute of Technology

Main topics

- Introduction to instruction set
- Formats and elements of instructions
- Instruction addresses/operands
- Addressing modes
- Instruction types

Instructions and their components

- Instruction is a binary word which implements a single pre-defined operation of a processor.
  - Instructions are stored in memory
  - Instructions are fetched from memory to CPU for execution
  - One instruction has it own function
- Instructions are divided into groups: data movement, computational, conditional and branching, etc.

Instructions and their components

- Instruction execution is divided into phases or stages. One instruction can be executed in 5 stages:
  - Instruction fetch (IF): instruction is read from memory to CPU;
  - Instruction decode (ID): instruction is decoded by CPU;
  - Instruction execution: instruction is executed by CPU;
  - Memory Access (MEM): CPU access memory for data for instruction's operands (if any);
  - Write back (WB): instruction results (if any) are saved into registers or memory.

Instruction execution cycle

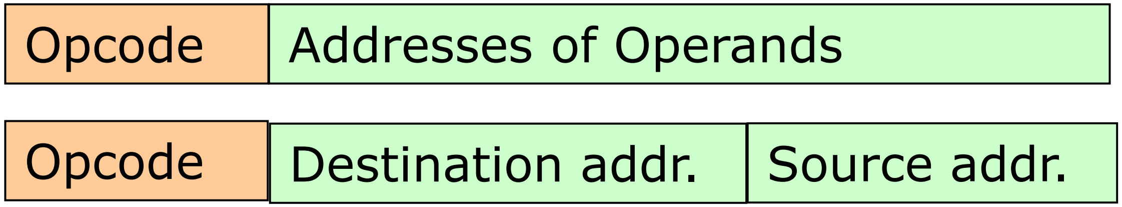
- Instruction execution cycle is the period of time that CPU complete the execution of the instruction.
  - An instruction execution cycle consist of some instruction execution stages.
  - An execution stage may consist of some machine cycle.
  - One machine cycle consist of some clock cycle.

Instruction execution cycle (cont.)

- An instruction execution cycle may have up to seven elements:
  - Instruction fetch cycle
  - Memory read cycle (for data)
  - Memory write cycle (for data)
  - Peripheral read cycle
  - Peripheral write cycle
  - Interrupt acceptance cycle
  - Bus free cycle

Instruction formats

- General format of instruction has two parts:
  - Opcode (operation code): each instruction has its own opcode.
  - Addresses of Operands: depends on instructions. There may be 3, 2, 1, 1.5 and 0 addresses.



Opcode | Addresses of Operands

Opcode | Destination addr. | Source addr.

Operands – 3 addresses

- Format: opcode addr1, addr2, addr3
  - Each of addr1, addr2, addr3 refers to a register or a memory location.
- Example:

  ADD R_3, R_1, R_2; $R_3 \leftarrow R_1 + R_2$

  Adds R_1 and R_2 then assigns the result to R_3.

  R_i is CPU register.

  ADD C, A, B; $M[C] \leftarrow M[A]+M[B]$

  A, B, C are memory locations.

Operands – 2 addresses

- Format: opcode addr1, addr2
  - Each of addr1, addr2 refers to a register or a memory location.
- Example:

  ADD R_2, R_1; $R_2 \leftarrow R_1 + R_2$

  Adds R_1 and R_2 then assigns the result to R_2.

  R_i is CPU register.

  ADD B, A; $M[B] \leftarrow M[A]+M[B]$

  A and B are memory locations.

Operands – 1 address

- Format: opcode addr1
  - addr1 refers to a register or a memory location.
  - This format uses R_{acc} (Accumulator) as the default register as the 2nd address.
- Example:

  ADD R_1; $R_{acc} \leftarrow R_1 + R_{acc}$

  Adds R_1 and R_{acc} then assigns the result to R_{acc}.

  R_i is CPU register.

  ADD B; $R_{acc} \leftarrow M[B]+ R_{acc}$

  B is a memory location.

Operands – 1.5 address

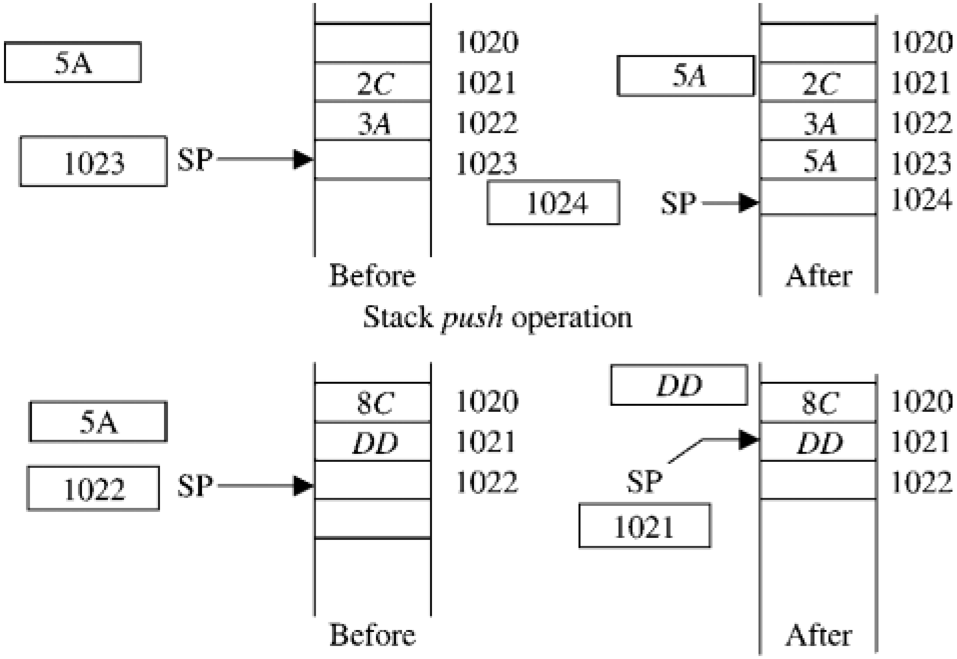
- Format: opcode addr1, addr2
  - addr1 refers to a register and addr2 refers a memory location or vice versa.
  - 1.5 or one and a half address is the mixed operands between register and memory location.
- Example:

  ADD R_1, B; $R_1 \leftarrow M[B] + R_1$

  R_1 is CPU register and B is a memory location.

Operands – 0 address

- 0 address is used in instructions that perform stack operations: push & pop



Stack *push* operation

Stack *pop* operation

Addressing modes

- Addressing modes are methods that instruction operands are organized.
- Some typical addressing modes:
  - Immediate
  - Direct
  - Register indirect
  - Memory indirect
  - Indexed
  - Relative

Addressing modes - Immediate

- The value of the source operand is immediately available in the instruction.
- The destination operand may be a register or a memory location.
- Example:

  LOAD R_1, #1000; $R_1 \leftarrow 1000$

  Load the value of 1000 into register R_1.

  LOAD B, #500; $M[B] \leftarrow 500$

  Load the value of 500 into memory location B.

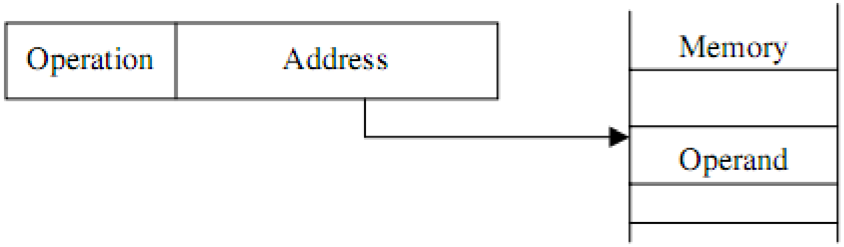
Addressing modes – Direct/Absolute

- The address of the memory location that holds the operand is included in the instruction.
- The other operand may be a register or a memory location.
- Example:

  LOAD R_1, 1000; $R_1 \leftarrow M[1000]$

  Load the value stored in memory location 1000 into register R_1.

Addressing modes – Direct/Absolute

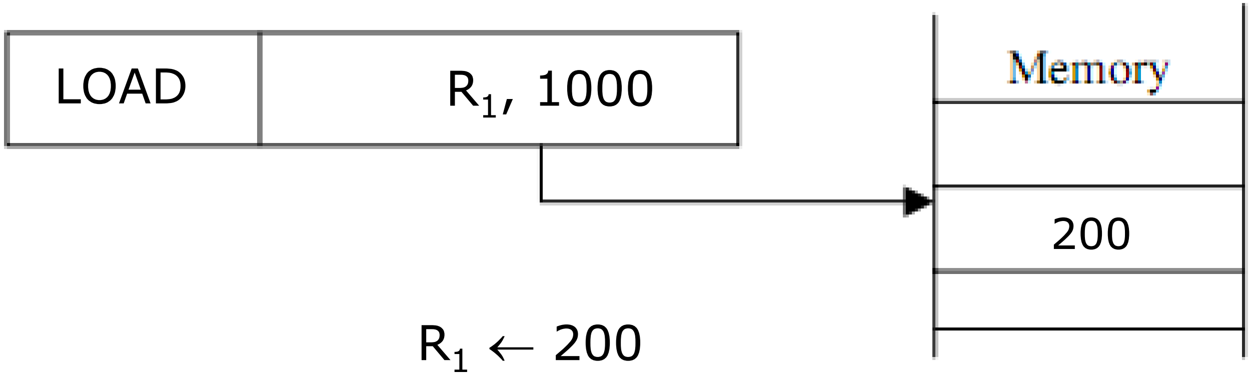


Operation

Address

Memory

Operand



LOAD

R_1, 1000

Memory

200

$R_1 \leftarrow 200$

Addressing modes – Indirect

- In indirect addressing modes, a register or a memory location is used to store the address of the operand.
  - Register indirect:

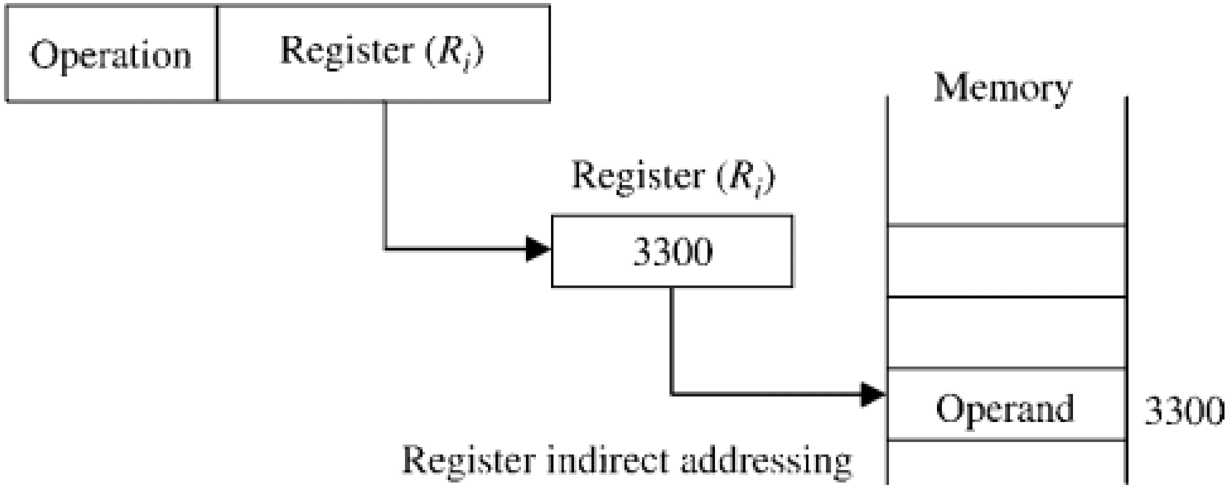
    LOAD R_j, (R_i); $R_j \leftarrow M[R_i]$

    Load the value in memory location that has address stored in R_i into R_j.
  - Memory indirect:

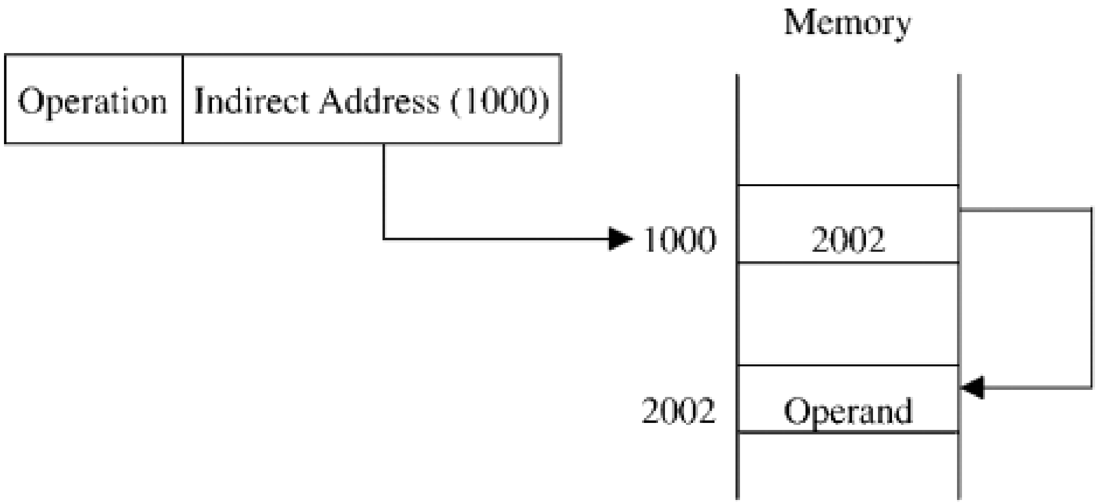
    LOAD R_i, (1000); $R_i \leftarrow M[M[1000]]$

    Load the value in memory location that has address stored in memory location 1000 into R_i.

Addressing modes – Indirect



Register indirect addressing

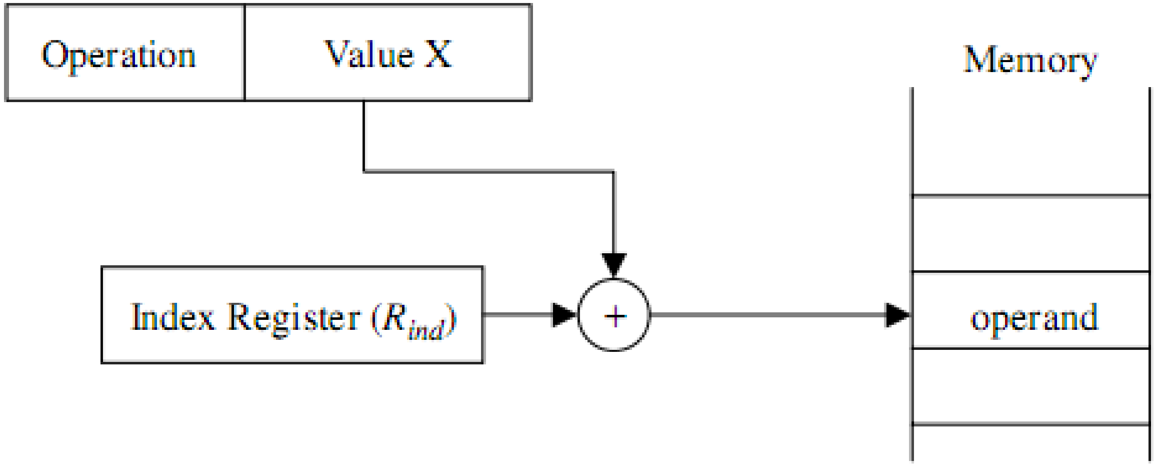


Memory indirect addressing

Indexed addressing mode

- The address of the operand is obtained by adding a constant to the content of a register, called the indexed register.
- Example:

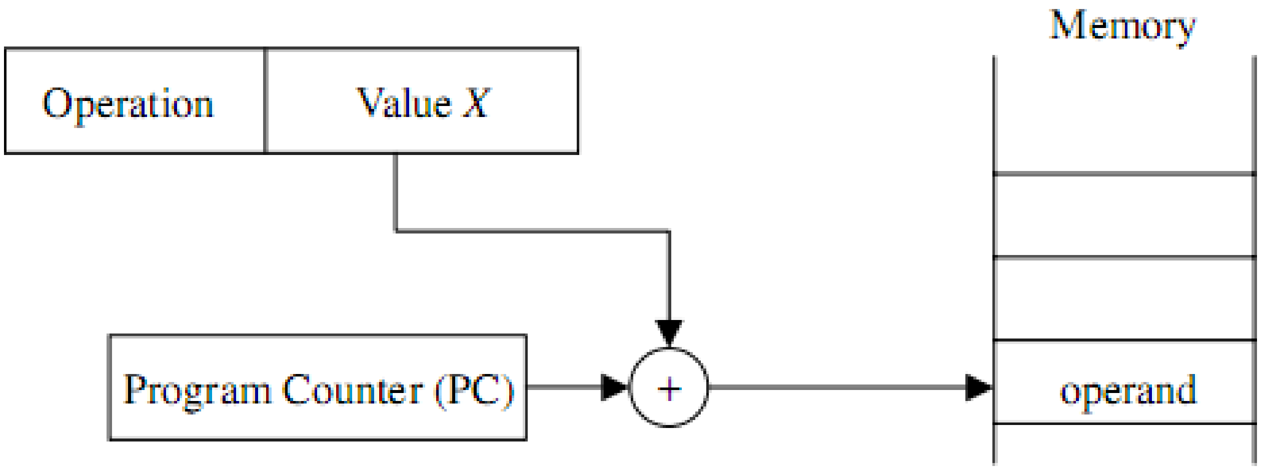
  LOAD R_i, X(R_{ind}); $R_i \leftarrow M[X+R_{ind}]$



Relative addressing mode

- The address of the operand is obtained by adding a constant to the content of a register, called the PC (program counter) register.
- Example:

LOAD R_i, X(PC); $R_i \leftarrow M[X+PC]$



Addressing mode summary

Addressing mode	Definition	Example	Operation
Immediate	Value of operand is included in the instruction	load R_i, #1000	$R_i \leftarrow 1000$
Direct/ Absolute	Address of operand is included in the instruction	load R_i, 1000	$R_i \leftarrow M[1000]$
Register Indirect	Operand is a memory location whose is in the register specified in the instruction	load R_i, (R_j)	$R_i \leftarrow M[R_j]$
Memory Indirect	Operand is a memory location whose is in the memory location specified in the instruction	load R_i, (1000)	$R_i \leftarrow M[M[1000]]$
Indexed	Address of operand is the sum of an index value and the content of the index register	load R_i, $X(R_{ind})$	$R_i \leftarrow M[R_{ind}+X]$
Relative	Address of operand is the sum of an index value and the content of the program counter	load R_i, X(PC)	$R_i \leftarrow M[PC+X]$

Instruction Types

- Typical instruction types include:
  - Data Movement Instructions
  - Arithmetic and Logical Instructions
  - Control/Sequencing Instructions
  - Input/Output Instructions

Data Movement Instructions

- Move data among units of the computer:
  - Between CPU registers:

    MOVE R_j, R_i; $R_j \leftarrow R_i$

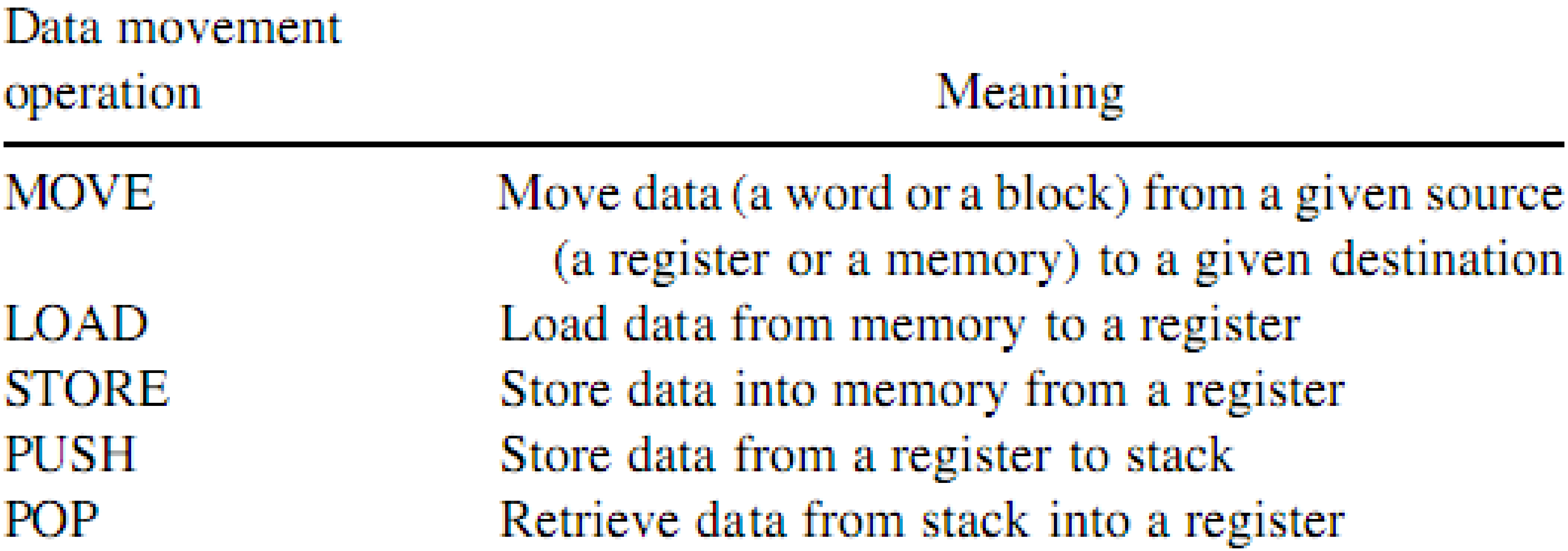
  - Between a CPU register and a memory location:

    MOVE R_j, 1000; $R_j \leftarrow M[1000]$

  - Between memory locations:

    MOVE (R_j), 1000; $M[R_j] \leftarrow M[1000]$

Common Data Movement Instructions



Data movement operation	Meaning
MOVE	Move data (a word or a block) from a given source (a register or a memory) to a given destination
LOAD	Load data from memory to a register
STORE	Store data into memory from a register
PUSH	Store data from a register to stack
POP	Retrieve data from stack into a register

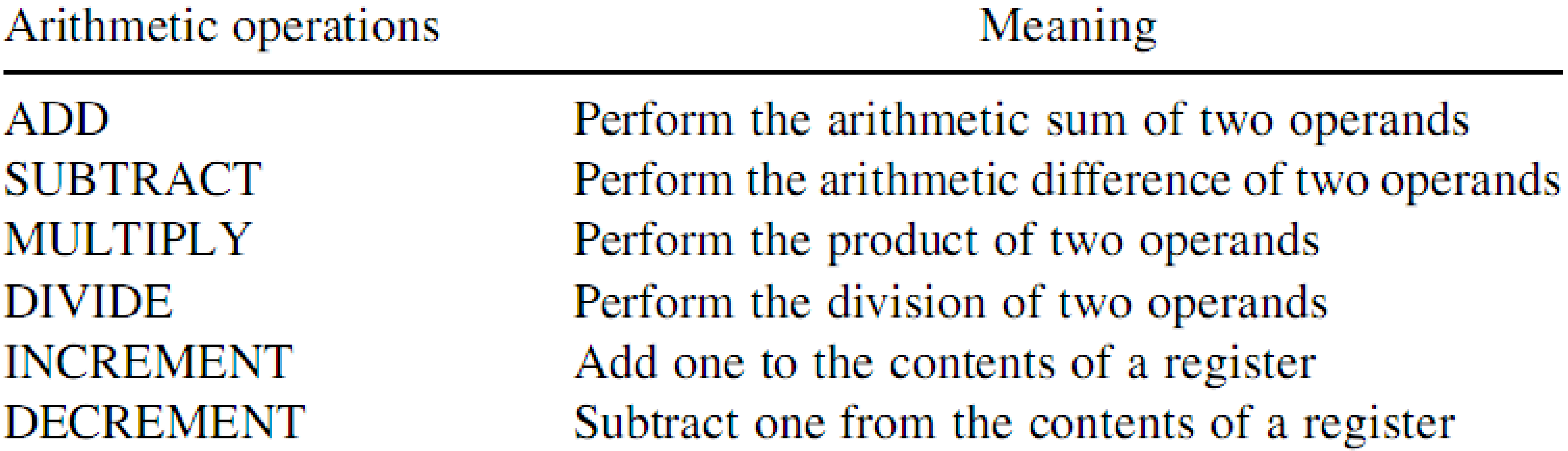
Arithmetic and Logical Instructions

- Arithmetic and logical instructions are those used to perform arithmetic and logical manipulation of registers and memory contents.
- Examples:

ADD R1, R2, R3; $R_1 \leftarrow R_2 + R_3$

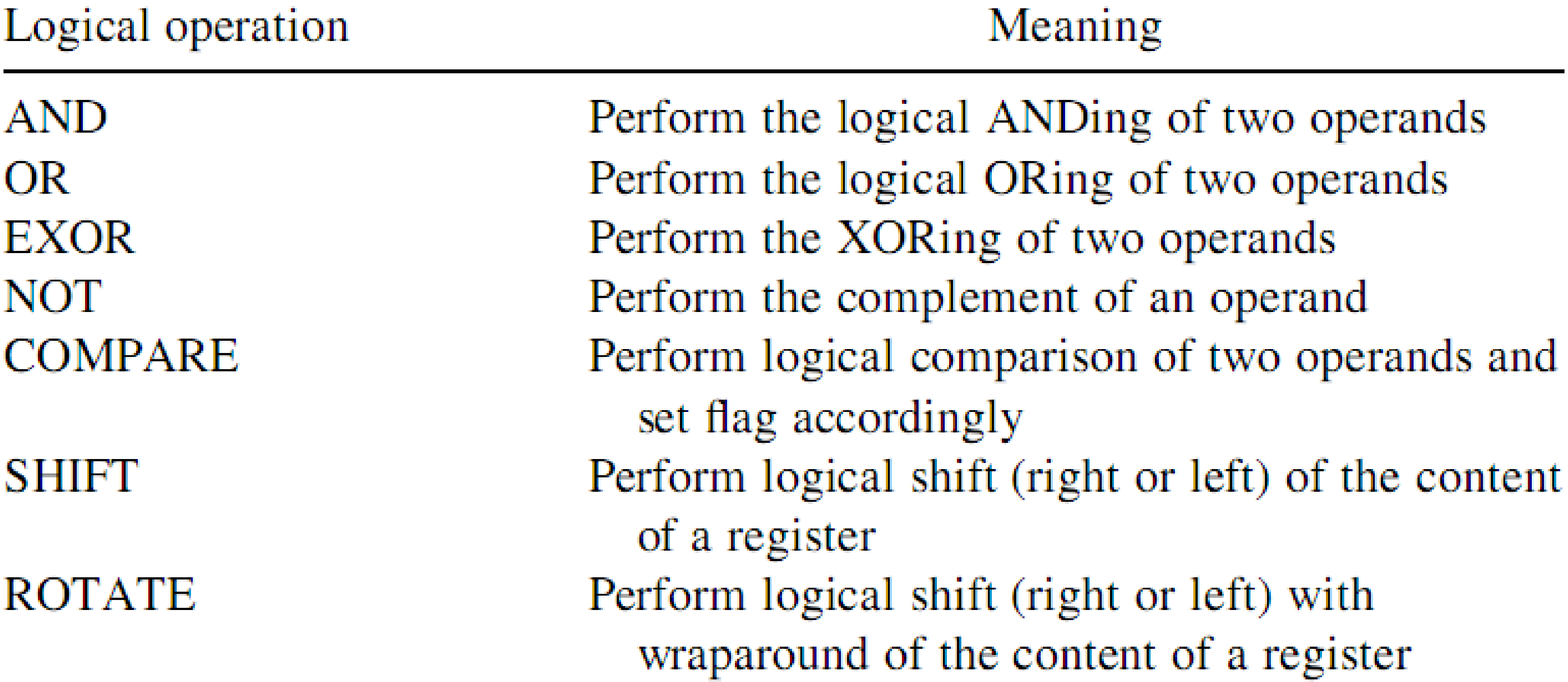
SUBSTRACT R1, R2, R3; $R_1 \leftarrow R_2 - R_3$

Common Arithmetic Instructions



Arithmetic operations	Meaning
ADD	Perform the arithmetic sum of two operands
SUBTRACT	Perform the arithmetic difference of two operands
MULTIPLY	Perform the product of two operands
DIVIDE	Perform the division of two operands
INCREMENT	Add one to the contents of a register
DECREMENT	Subtract one from the contents of a register

Common Logic Instructions

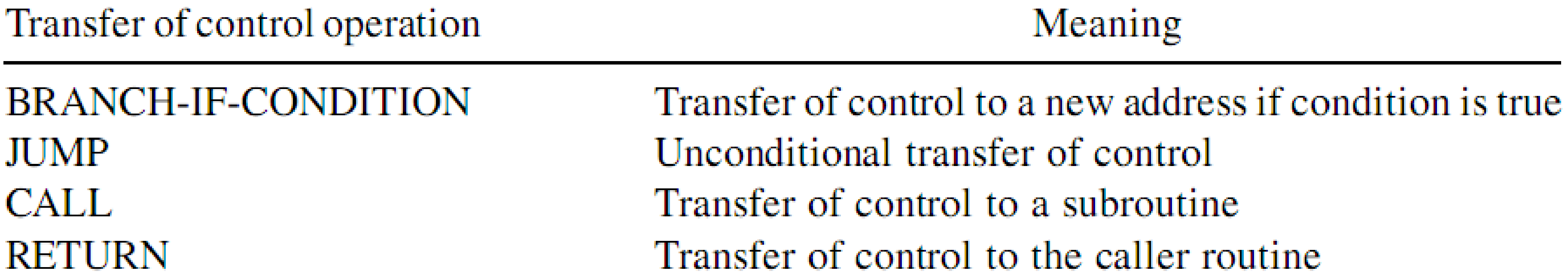


Logical operation	Meaning
AND	Perform the logical ANDing of two operands
OR	Perform the logical ORing of two operands
EXOR	Perform the XORing of two operands
NOT	Perform the complement of an operand
COMPARE	Perform logical comparison of two operands and set flag accordingly
SHIFT	Perform logical shift (right or left) of the content of a register
ROTATE	Perform logical shift (right or left) with wraparound of the content of a register

Control/Sequencing Instructions

- Control instructions are used to change the sequence in which instructions are executed:
  - CONDITIONAL BRANCHING (CONDITIONAL JUMP)
  - UNCONDITIONAL BRANCHING (JUMP)
  - CALL and RETURN
- A common characteristic among these instructions is that their execution changes the program counter (PC) value.
- Use ALU flags to determine the conditions.

Control/Sequencing Instructions



Transfer of control operation	Meaning
BRANCH-IF-CONDITION	Transfer of control to a new address if condition is true
JUMP	Unconditional transfer of control
CALL	Transfer of control to a subroutine
RETURN	Transfer of control to the caller routine

Control/Sequencing Instructions

```
       LOAD $R_1$, #100
       LOAD $R_2$, #1000
       LOAD $R_0$, #0
*Loop*: ADD R0, ($R_2$)
       INCREMENT $R_2$
       DECREMENT $R_1$
       BRANCH-IF-GREATER-THAN *Loop*
```

Calculate the sum of the contents of 100 memory locations started from address of 1000. The loop ends when R_1 is down to 0.

Input/Output Instructions

- Input and output (I/O) instructions are used to transfer data between the computer and peripheral devices.
- Peripheral devices are interfaced with a computer through dedicated ports. Each port has a dedicated address.
- The two basic I/O instructions used are the INPUT and OUTPUT instructions.
  - The INPUT instruction is used to transfer data from an input device to the processor.
  - The OUTPUT instruction is used to transfer data from the processor to an output device.

Programming Example

```
        LOAD $R_1$, #100;         $R_1 \leftarrow 100$
        LOAD $R_2$, #1000;        $R_2 \leftarrow 1000$
        LOAD $R_0$, #0;           $R_0 \leftarrow 0$
Loop:   ADD $R_0$, ($R_2$);         $R_0 \leftarrow R_0 + M[R_2]$
        INCREMENT $R_2$;          $R_2 \leftarrow R_2 + 1$
        DECREMENT $R_1$;          $R_1 \leftarrow R_1 - 1$
        BRANCH-IF-GREATER-THAN  Loop;
        ; Go back to execute the instruction after Loop label
        ; if $R_1$ is greater than 0.
        STORE 2000, $R_0$; $M[2000] \leftarrow R_0$
```

A code segment that performs the task of adding 100 numbers stored at consecutive memory locations starting at location 1000. The results should be stored in memory location 2000.