Basic Accumulator Based CPU

- Early computers had accumulator (a register used to store the intermediate result of CPU's ALU temporarily, overwrites the previous value) based CPUs (one address machine).
  - The instruction format is one address field:

    opcode address
- In this type of CPUs, the first operand is always stored in the accumulator.
- One address instruction is used here. For example, ADD A, LOAD A, STORE A.
- Requires less memory space and instruction cycle takes less time (advantage).
- For complex expressions/computations, program and memory size increases due to a lot of short instructions (Disadvantage).
- In modern CPUs, accumulators are replaced by general-purpose registers (generally from 8-32) , because they offer more flexibility. Data access is much faster.

Basic Instruction Types

- One address instruction:
  - Instruction format: Operation Address

    ADD A
- Two address instruction:
  - Instruction format: Operation Source,Destination

    ADD A,B (C←A+B)

    MOVE B,C
- Three address instruction:
  - Instruction format: Operation Source1,Source2,Destination

    ADD A,B,C (C←A+B)

In modern computers, the order of source and destination depends on the manufacture companies, no single rule has been adopted.

Straight Line Sequencing of Instruction Execution

- Execution of C←A+B
  - Suppose the computer has word length of 32 bits
  - The memory is byte addressable
  - Memory location starts at i
  - The three instructions are at successive memory locations

- The instructions are executed sequentially. The processor's control circuits use the information in PC to execute instruction one at a time in a sequential manner. This is known as **straight line sequencing**.

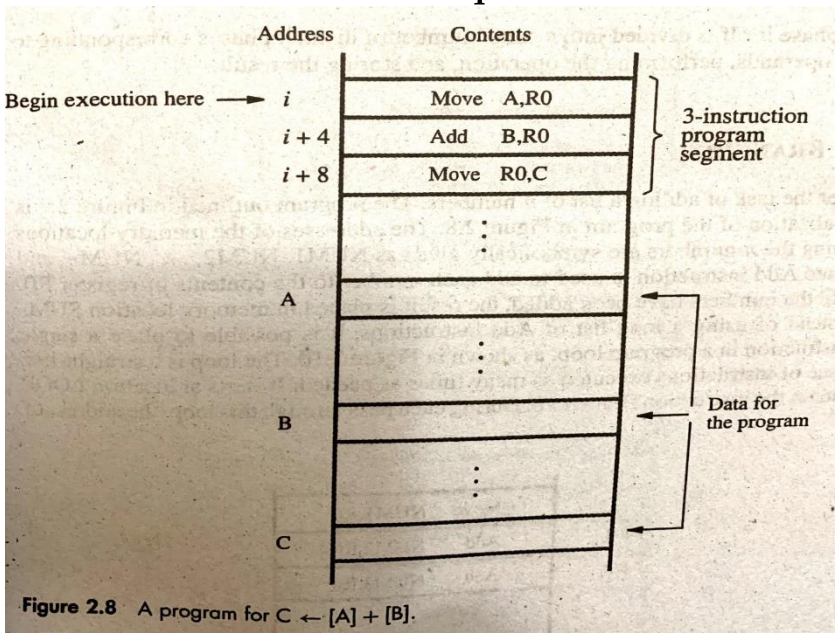

Figure 2.8 A program for C ← [A] + [B].

Branching

- Execution of adding a list of n numbers.

```
MOVE NUM1,R0
ADD NUM2,R0
        .
        .
        .
ADD NUMn,R0
MOVE R0,SUM
```

See Figure 2.9

Branching(contd.)

- Branch is an instruction switching execution to a different instruction sequence only if the condition is satisfied. Otherwise, PC is incremented in normal sequential manner.
- Instead of using a list of ADD operation, loop can be used.

  Here, n is the number of entries in the list

  R1 is used as a counter

  Branch is a conditional branch instruction

  LOOP is a branching location

  LOOP is repeated as long as R1 is greater than zero

Branching(contd.)

- See Figure 2.10

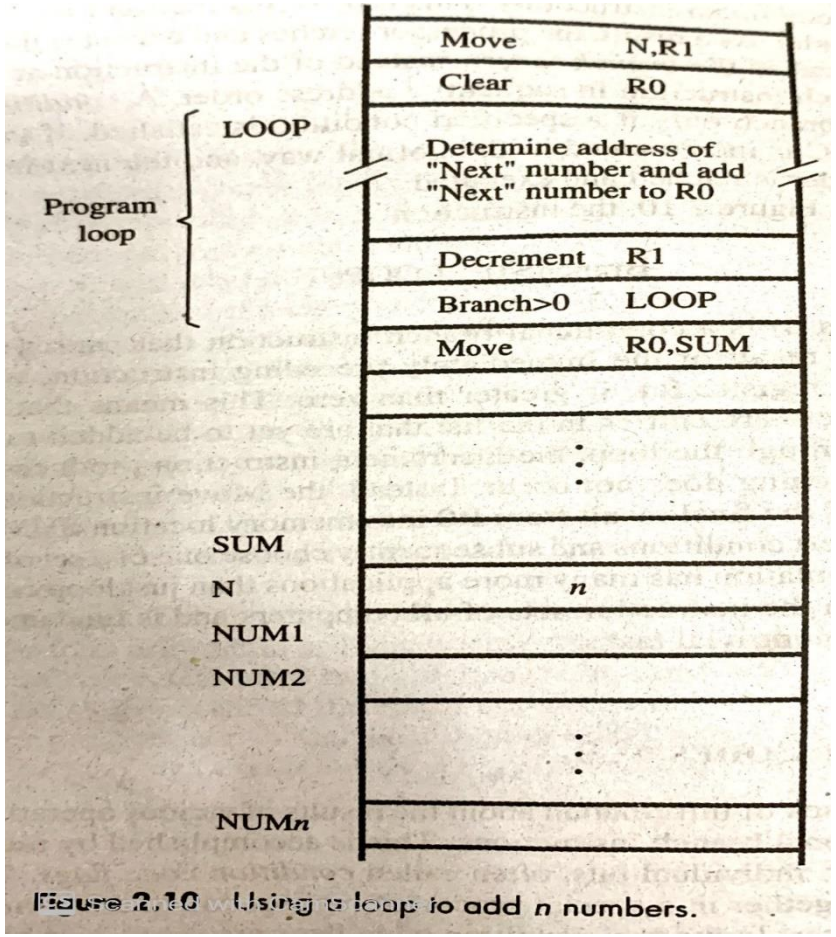

Figure 2.10 Using a loop to add *n* numbers.

Branching(contd.)

Status Register/Condition code register/Flag register: Indicates the status of the processor. These are collective flag bits inside a processor.

- The processor keeps track of information about the results of various operations for use by subsequent conditional branch instruction. This is accomplished by individual flag bits.
- Commonly used flags are:
  - N (negative): set to 1 if the result of an arithmetic or logic operation is negative; otherwise cleared to 0
  - Z (zero): set to 1 if the result of an arithmetic or logic operation is 0; otherwise cleared to 0
  - V(overflow): set to 1 if arithmetic overflow occurs; otherwise cleared to 0
  - C (carry): set to 1 if a carry-out results from the operation; otherwise cleared to 0

Addressing Modes

The different ways in which the location of an operand is specified in an instruction are referred to as **addressing modes**.

- In general, a program operates on data that are stored in the memory.
- Programs use different data structures to represent these data used in computation.
- Programs use constant, local and global variables, arrays, and pointers.
- Translating programs from high level language to assembly language, the compiler must be able to implement these constructs by the instruction set.

See Table 2.1

Addressing Modes(contd.)

Variables: A variable can be accessible by specifying the name of the register or the address of the memory where it is located. These two addressing modes are:

- **Register mode:** The name of the register (operand is the content of the register) is given in the instruction.
- **Absolute/Direct mode:** The address of the memory location (where the operand is located) is given in the instruction.

- MOVE LOC, R1 which addressing mode is used here?
- High level language program instruction: int a,b; absolute mode

Addressing Modes(contd.)

Constants: Address and data constants are represented using immediate mode.

- **Immediate mode:** The operand is given explicitly in the instruction.

- MOVE #100,R1 places the value 100 in the register R1, # indicates the value is an immediate operand
- The immediate mode specifies values of source operand only.
- High level language program instruction: A=B+6; would be compiled as MOVE B,R1

  ADD #6,R1

  MOVE R1,A

Addressing Modes(contd.)

Pointers: Store memory addresses of other operands. In this case, the addressing mode provides information from which the memory address of the operand is specified. This is referred as the effective address (EA).

- **Indirect mode:** The effective address of the operand is the content of a register or memory location whose address appears in the instruction.

- ADD (R1),R2 parenthesis denotes effective address
- ADD (A),R2

See Figure 2.11

Home Work: Figure 2.12