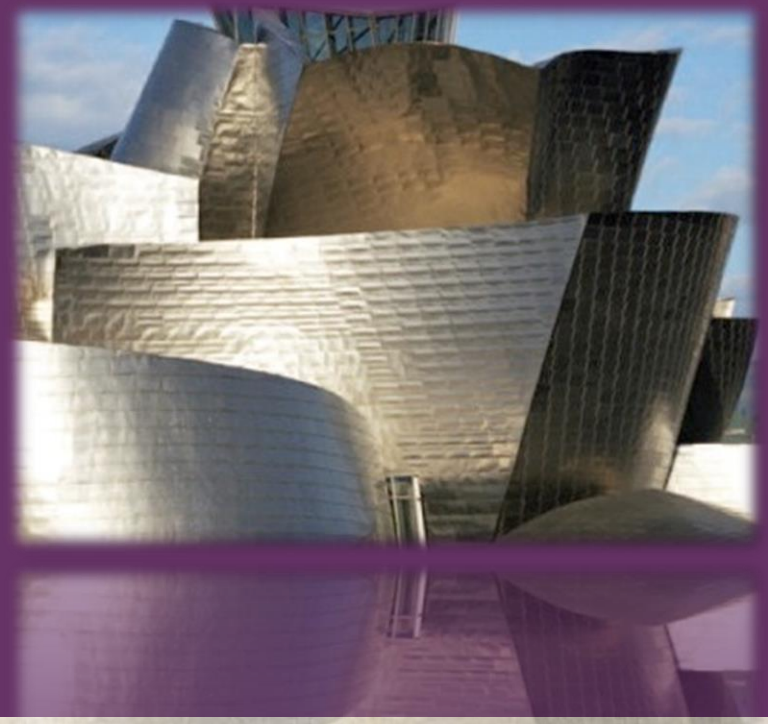


Chapter 13

Instruction Sets:

Addressing Modes and Formats

Objectives

After studying this chapter, you should be able to:

- Describe the various types of addressing modes common in instruction sets.
- Present an overview of x86 and ARM addressing modes (ARM: Advanced RISC Machine, RISC: Reduced Instruction Set Computer).
- Summarize the issues and trade-offs involved in designing an instruction format.
- Present an overview of x86 and ARM instruction formats.
- Understand the distinction between machine language and assembly language.

Contents

13.1- Addressing Modes

Ways to specify an operand in an instruction:

- Immediate
- Direct
- Indirect
- Register
- Register indirect
- Displacement (replace)
- Stack

Addressing Modes

Immediate : Operand is a specific value

Indirect : Operand address is stored in another variable

Register Indirect : Operand address is stored in a register

Implicit : Operand address is stored in stack register

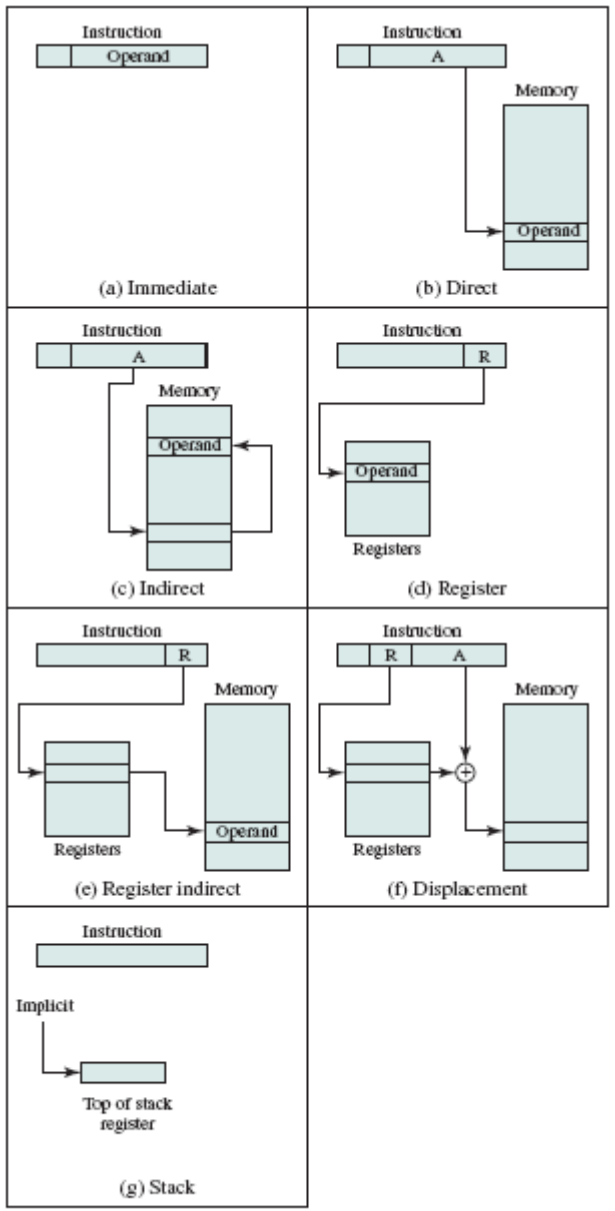


Figure 13.1 Addressing Modes

Direct : Operand is the value of a variable

Register : Operand is a specific register

Displacement : Replace the value of a variable with an expression

Basic Addressing Modes

A = contents of an address field in the instruction
R = contents of an address field in the instruction that refers to a register
EA = actual (effective) address of the location containing the referenced operand
(X) = contents of memory location X or register X

Table 13.1 Basic Addressing Modes

Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

Immediate Addressing

- Simplest form of addressing
- Operand = A
- This mode can be used to define and use constants or set initial values of variables
  - Typically the number will be stored in twos complement form
  - The leftmost bit of the operand field is used as a sign bit
- Advantage:
  - no memory reference other than the instruction fetch is required to obtain the operand, thus saving one memory or cache cycle in the instruction cycle
- Disadvantage:
  - The size of the number is restricted to the size of the address field, which, in most instruction sets, is small compared with the word length

Direct Addressing

- Address field contains the effective address of the operand
- Effective address (EA) = address field (A)
- Was common in earlier generations of computers
- Requires only one memory reference and no special calculation
- Limitation is that it provides only a limited address space

Indirect Addressing

- Reference to the address of a word in memory which contains a full-length address of the operand
- EA = (A)
  - Parentheses are to be interpreted as meaning *contents of*
- Advantage:
  - For a word length of *N* an address space of 2^N is now available
- Disadvantage:
  - Instruction execution requires two memory references to fetch the operand
    - One to get its address and a second to get its value
- A rarely used variant of indirect addressing is multilevel or cascaded indirect addressing
  - EA = (. . . (A) . . .)
  - Disadvantage is that three or more memory references could be required to fetch an operand

Register Addressing

- Address field refers to a register rather than a main memory address
- EA = R
- Advantages:
  - Only a small address field is needed in the instruction
  - No time-consuming memory references are required
- Disadvantage:
  - The address space is very limited

Register Indirect Addressing

- Analogous to indirect addressing
  - The only difference is whether the address field refers to a memory location or a register
- EA = (R)
- Address space limitation of the address field is overcome by having that field refer to a word-length location containing an address
- Uses one less memory reference than indirect addressing

Displacement Addressing

- Combines the capabilities of direct addressing and register indirect addressing
- EA = A + (R)
- Requires that the instruction have two address fields, at least one of which is explicit
  - The value contained in one address field (value = A) is used directly
  - The other address field refers to a register whose contents are added to A to produce the effective address
- Most common uses:
  - Relative addressing
  - Base-register addressing
  - Indexing

Relative Addressing

- The implicitly referenced register is the program counter (PC)
  - The next instruction address is added to the address field to produce the EA
  - Typically the address field is treated as a twos complement number for this operation
  - Thus the effective address is a displacement relative to the address of the instruction
- Exploits the concept of locality
- Saves address bits in the instruction if most memory references are relatively near to the instruction being executed

Base-Register Addressing

- The referenced register contains a main memory address and the address field contains a displacement from that address
- The register reference may be explicit or implicit
- Exploits the locality of memory references
- Convenient means of implementing segmentation
- In some implementations a single segment base register is employed and is used implicitly
- In others the programmer may choose a register to hold the base address of a segment and the instruction must reference it explicitly

Indexed Addressing

- **The address field references a main memory address and the referenced register contains a positive displacement from that address**
- The method of calculating the EA is the same as **for base-register addressing**
- An important use is to provide an efficient mechanism for **performing iterative operations**
- **Autoindexing**
  - Automatically increment or decrement the index register after each reference to it
  - EA = A + (R)
  - (R) ← (R) + 1
- **Postindexing**
  - Indexing is performed after the indirection
  - EA = (A) + (R)
- **Preindexing**
  - Indexing is performed before the indirection
  - EA = (A + (R))

Stack Addressing

- A stack is a linear array of locations
  - Sometimes referred to as a *pushdown list* or *last-in-first-out queue*
- A stack is a reserved block of locations
  - Items are appended to the top of the stack so that the block is partially filled
- Associated with the stack is a pointer whose value is the address of the top of the stack
  - The stack pointer is maintained in a register
  - Thus references to stack locations in memory are in fact register indirect addresses
- Is a form of implied addressing
- The machine instructions need not include a memory reference but implicitly operate on the top of the stack

13.3- Instruction Formats

Define the layout of the bits of an instruction, in terms of its constituent fields

Must include an opcode and, implicitly or explicitly, indicate the addressing mode for each operand

For most instruction sets more than one instruction format is used

Instruction Length

- Most basic design issue
- Affects, and is affected by:
  - Memory size
  - Memory organization
  - Bus structure
  - Processor complexity
  - Processor speed
- Should be equal to the memory-transfer length or one should be a multiple of the other
- Should be a multiple of the character length, which is usually 8 bits, and of the length of fixed-point numbers

Allocation of Bits

- Number of addressing modes
- Number of operands
- Register versus memory
- Number of register sets
- Address range
- Address granularity (cốt lõi)

PDP-8 Instruction Format

Memory reference instructions

Opcode			D/I	Z/C	Displacement						
0		2	3	4	5						11

Input/output instructions

1	1	0	Device						Opcode		
0		2	3					8	9		11

Register reference instructions

Group 1 microinstructions

1	1	1	0	CLA	CLL	CMA	CML	RAR	RAL	BSW	IAC
0	1	2	3	4	5	6	7	8	9	10	11

Group 2 microinstructions

1	1	1	0	CLA	SMA	SZA	SNL	RSS	OSR	HLT	0
0	1	2	3	4	5	6	7	8	9	10	11

Group 3 microinstructions

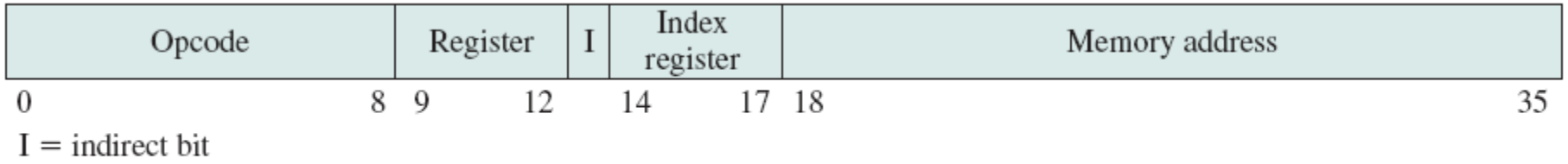
1	1	1	0	CLA	MQA	0	MQL	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11

D/I = Direct/Indirect address
Z/C = Page 0 or Current page
CLA = Clear Accumulator
CLL = Clear Link
CMA = CoMplement Accumulator
CML = CoMplement Link
RAR = Rotate Accumulator Right
RAL = Rotate Accumulator Left
BSW = Byte SWap

IAC = Increment ACcumulator
SMA = Skip on Minus Accumulator
SZA = Skip on Zero Accumulator
SNL = Skip on Nonzero Link
RSS = Reverse Skip Sense
OSR = Or with Switch Register
HLT = HaLT
MQA = Multiplier Quotient into Accumulator
MQL = Multiplier Quotient Load

Figure 13.5 PDP-8 Instruction Formats

PDP-10 Instruction Format



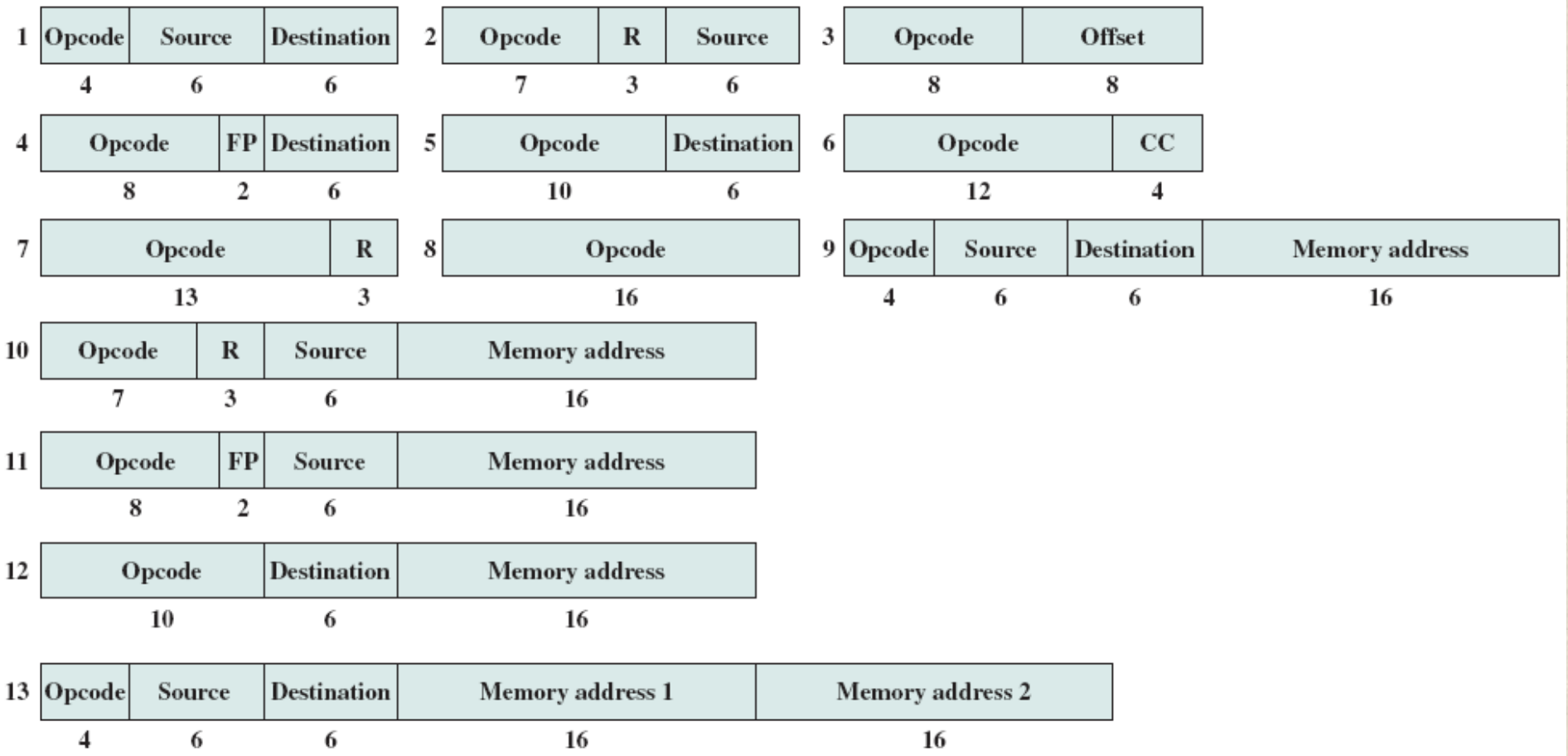
I = indirect bit

Figure 13.6 PDP-10 Instruction Format

Variable-Length Instructions

- Variations can be provided efficiently and compactly
- Increases the complexity of the processor
- Does not remove the desirability of making all of the instruction lengths integrally related to word length
  - Because the processor does not know the length of the next instruction to be fetched a typical strategy is to fetch a number of bytes or words equal to at least the longest possible instruction
  - Sometimes multiple instructions are fetched

PDP-11 Instruction Format



Numbers below fields indicate bit length
Source and destination each contain a 3-bit addressing mode field and a 3-bit register number
FP indicates one of four floating-point registers
R indicates one of the general-purpose registers
CC is the condition code field

Figure 13.7 Instruction Formats for the PDP-11

VAX Instruction Examples

Hexadecimal Format	Explanation	Assembler Notation and Description
8 bits 0 5	Opcode for RSB	RSB Return from subroutine
D 4 5 9	Opcode for CLRL Register R9	CLRL R9 Clear register R9
B 0 C 4 6 4 0 1 A B 1 9	Opcode for MOVW Word displacement mode, Register R4 356 in hexadecimal Byte displacement mode, Register R11 25 in hexadecimal	MOVW 356(R4), 25(R11) Move a word from address that is 356 plus contents of R4 to address that is 25 plus contents of R11
C 1 0 5 5 0 4 2 D F [ ]	Opcode for ADDL3 Short literal 5 Register mode R0 Index prefix R2 Indirect word relative (displacement from PC) Amount of displacement from PC relative to location A	ADDL3 #5, R0, @A[R2] Add 5 to a 32-bit integer in R0 and store the result in location whose address is sum of A and 4 times the contents of R2

Figure 13.8 Example of VAX Instructions

x86 Instruction Format

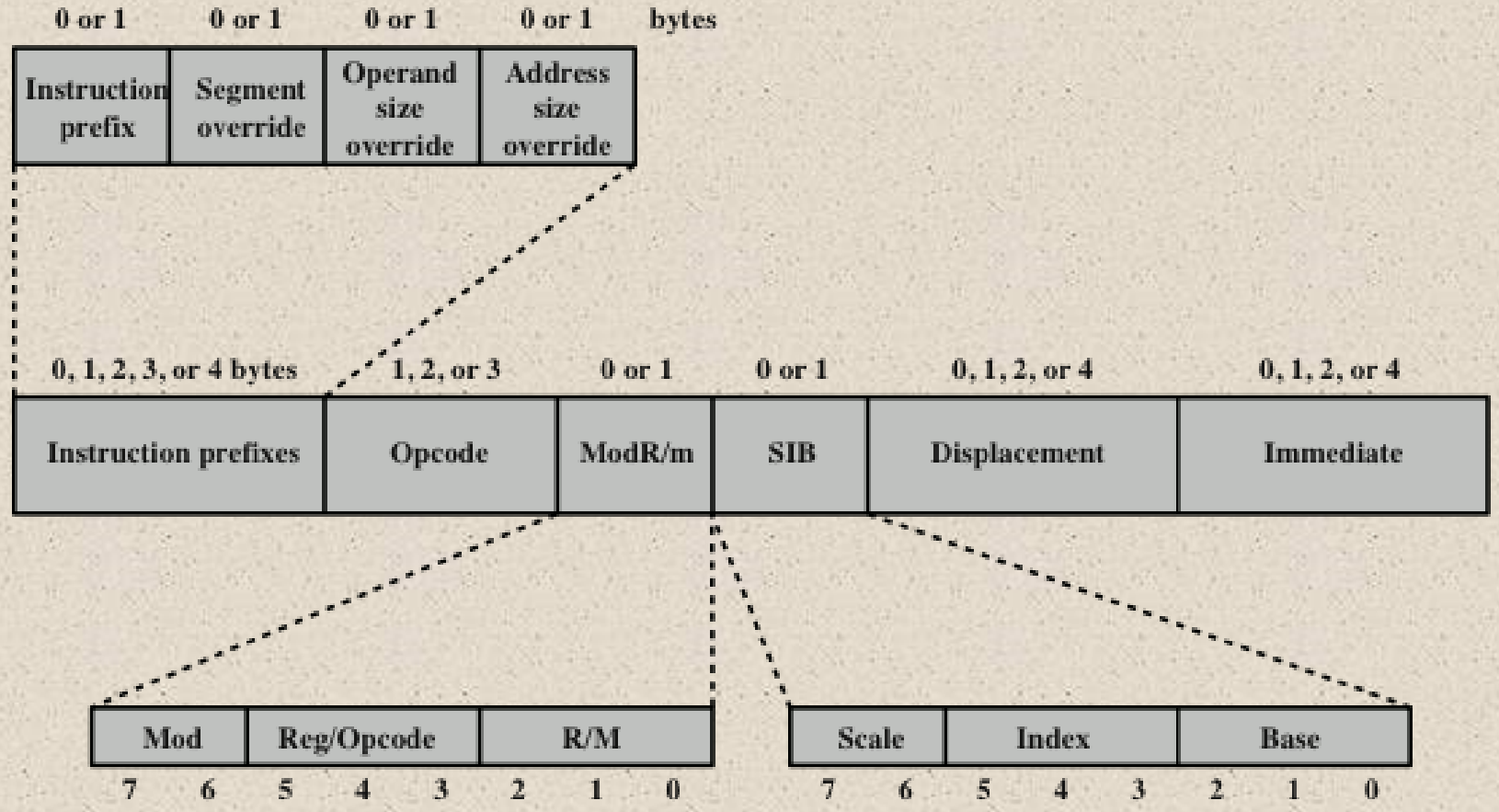


Figure 13.9 x86 Instruction Format

13.5- Assembly Language

Consider the simple BASIC statement: **N = I + J + K**

Suppose we wished to program this statement in machine language and to initialize I, J, and K to 2, 3, and 4, respectively.
This is shown in Figure 13.13a. (next slide)
The program starts in location 101 (hexadecimal). Memory is reserved for the four variables starting at location 201.

The program consists of four instructions:

1. Load the contents of location 201 into the AC (variable I)
2. Add the contents of location 202 to the AC (J)
3. Add the contents of location 203 to the AC.(K)
4. Store the contents of the AC in location 204(N)

This is clearly a tedious (buồn tẻ) and very error-prone (dễ mắc lỗi) process.
➔ Assembly Language

Assembler – Assembly Compiler

Address	Contents			
101	0010	0010	101	2201
102	0001	0010	102	1202
103	0001	0010	103	1203
104	0011	0010	104	3204
201	0000	0000	201	0002
202	0000	0000	202	0003
203	0000	0000	203	0004
204	0000	0000	204	0000

(a) Binary program

Address	Contents
101	2201
102	1202
103	1203
104	3204
201	0002
202	0003
203	0004
204	0000

(b) Hexadecimal program

How can we understand it?

Address	Instruction	
101	LDA	201
102	ADD	202
103	ADD	203
104	STA	204
201	DAT	2
202	DAT	3
203	DAT	4
204	DAT	0

(c) Symbolic program

Label	Operation	Operand
FORMUL	LDA	I
	ADD	J
	ADD	K
	STA	N
I	DATA	2
J	DATA	3
K	DATA	4
N	DATA	0

(d) Assembly program

More understandable
More readable
(Assembly program looks like this)

Figure 13.13 Computation of the Formula N = I + J + K

Assembly Language (Wiki)

- **Assembly language** (or assembler language) is a low-level programming language for a computer, or other programmable device
- There is a very strong (generally one-to-one) **correspondence** between the language and the architecture's machine code instructions.
- Each assembly language is specific to a particular computer architecture, in contrast to most high-level programming languages, which are generally portable across multiple architectures, but require interpreting or compiling.
- Assembly language is **converted** into executable machine code by a utility program referred to as an ***assembler***; the conversion process is referred to as *assembly*, or *assembling* the code.

Assembly Language (Wiki)…

•**Assembly language uses a mnemonic** to represent each low-level machine instruction or operation. Typical operations require one or more operands in order to form a complete instruction, and most assemblers can therefore take labels, symbols and expressions as operands to represent addresses and other constants, freeing the programmer from tedious manual calculations.

• **Macro assemblers** include a macro instruction facility so that (parameterized) assembly language text can be represented by a name, and that name can be used to insert the expanded text into other code.

• Many assemblers offer additional mechanisms to facilitate program development, to control the assembly process, and to aid debugging.

Exercises

13.1 Briefly define immediate addressing.
13.2 Briefly define direct addressing.
13.3 Briefly define indirect addressing.
13.4 Briefly define register addressing.
13.5 Briefly define register indirect addressing.
13.6 Briefly define displacement addressing.
13.7 Briefly define relative addressing.
13.8 What is the advantage of autoindexing?
13.9 What is the difference between postindexing and preindexing?
13.10 What facts go into determining the use of the addressing bits of an instruction?
13.11 What are the advantages and disadvantages of using a variable-length instruction format?

Summary

Chapter 13

Instruction Sets: Addressing Modes and Formats

- Addressing modes
  - Immediate addressing
  - Direct addressing
  - Indirect addressing
  - Register addressing
  - Register indirect addressing
  - Displacement addressing
  - Stack addressing
- Instruction formats
  - Instruction length
  - Allocation of bits
  - Variable-length instructions
- Instruction to Assembly language