Software Systems-1

INSTRUCTION FORMATS

Dr. Azam E. Al-Rawachy
COLLEGE OF COMPUTER SCIENCE AND MATHEMATICS

1 Introduction

Any instruction issued by the processor must carry at least two types of information. These are the **operation** to be performed, encoded in what is called the **op-code field**, and the **address information of the operand** on which the operation is to be performed, encoded in what is called the **address field**.

Op-Code	Addresses

Computers may have instructions of several different lengths containing a varying number of addresses. The number of address fields in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three types of CPU organizations:

- Single accumulator organization.
- General register organization.
- Stack organization.

A single accumulator organization is performed with an implied accumulator register. The instruction format in this type of computer uses one address field. For example, the instruction that specifies an arithmetic addition is defined by an assembly language instruction as ADD.

The ADD instruction in this case results in the operation AC ← AC + M[X]. AC is the accumulator register, X is the address of the operand and M[X] symbolizes the memory word located at address X.

General register organization, the instruction format in this type of computer needs **three register address** fields. Thus the instruction for an arithmetic addition may be written in an assembly language as

ADD R1, R2, R3

To denote the operation R1 ← R2 + R3. The number of address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers. Thus the instruction

ADD R1, R2

Computers with **stack organization** would have PUSH and POP instructions that require an address field. Thus the instruction PUSH X will push the word at address X to the top of the stack. The stack pointer is updated automatically. Operation-type instructions do not need an address field in stack-organized computers. This is because the operation is performed on the ***two items*** that are on **top of the stack** (TOS).

The instruction ADD in a stack computer consists of an ***operation code only*** with ***no address field***. This operation has the effect of popping the two top numbers from the stack, adding the numbers, and pushing the sum into the stack. There is no need to specify operands with an address field since all operands are implied to be in the stack.

Most computers fall into one of the three types of organizations that have just been described. Some computers combine features from more than one organizational structure. For example, the Intel 808- microprocessor has seven CPU registers, one of which is an accumulator register. As a consequence; the processor has some of the characteristics of a general register type and some of the characteristics of an accumulator type.

1.1 THREE-ADDRESS INSTRUCTIONS

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates ***X = (A + B) * (C + D)*** is shown below, together with comments that explain the register transfer operation of each instruction.

ADD	R1, A, B	R1 ← M [A] + M [B]
ADD	R2, C, D	R2 ← M [C] + M [D]]
MUL	X, R1, R2	M [X] ← R1 * R2

It is assumed that the computer has two processor registers, R1 and R2. The symbol M[A] denotes the operand at the memory address symbolized by A. The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

1.2 TWO-ADDRESS INSTRUCTIONS

Two address instructions are the most common in commercial computers. Here again, each address field can specify either a processor register or a memory word. The program to evaluate X = (A + B) * (C + D) is as follows:

MOV	R1, A	R1 ← M [A]
ADD	R1, B	R1 ← R1 + M [B]
MOV	R2, C	R2← M [C]
ADD	R2, D	R2 ← R2 + M [D]
MUL	R1, R2	R1 ← R1*R2
MOV	X, R1	M [X] ← R1

The MOV instruction moves or transfers the operands to and from memory and processor registers. The first symbol listed in instruction is assumed to be both a source and the destination where the result of the operation is transferred.

1.3 ONE-ADDRESS INSTRUCTIONS

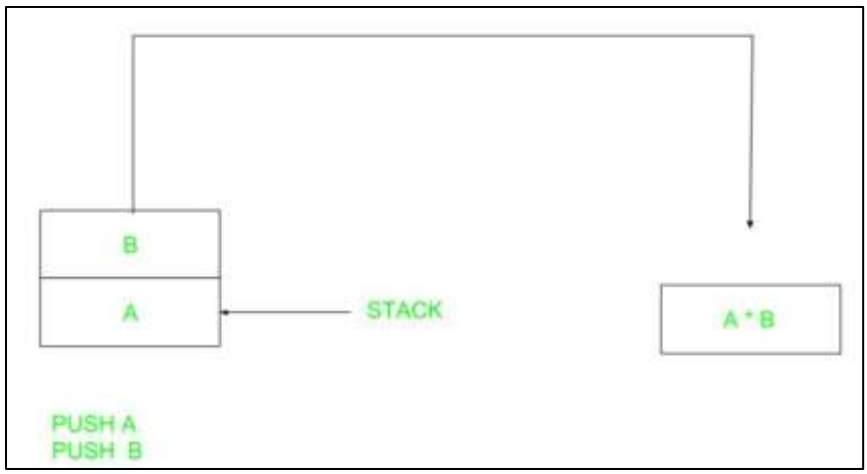
One-address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division, there is a need for a second register. However, here we will neglect the second and assume that the AC contains the result of all operations. The program to evaluate ***X = (A + B) * (C + D)*** is

LOAD	A	AC ← M [A]
ADD	B	AC ← AC + M [B]
STORE	T	M [T] ← AC
LOAD	C	AC ← M [C]
ADD	D	AC ← AC + M [D]
MUL	T	AC ← AC * M [T]
STORE	X	M [X] ← AC

All operations are done between the AC register and a memory operand. T is the address of a temporary memory location required for storing the intermediate result.

1.4 ZERO-ADDRESS INSTRUCTIONS

A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack. The following program shows how the below question will be written for a stack organized computer.



$$X = (A + B) * (C + D)$$

AB+ CD + *

PUSH	A	TOS ← A
PUSH	B	TOS ← B
ADD		TOS ← (A + B)
PUSH	C	TOS ← C
PUSH	D	TOS ← D
ADD		TOS ← (C + D)
MUL		TOS ← (C + D) * (A + B)
POP		

There are three Algebraic Notation possible:

Infix: a + b
Postfix: ab+
Prefix: +ab

To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse Polish notation (Or simply postfix notation, is a mathematical notation in which ***operators*** follow their ***operands***). The name "zero-address" is given to this type of computer because of the absence of an address field in the computational instructions.

EX2: Write a sequence of instructions that will compute the value of $X = \frac{A+B*C}{(D-E*F+G*H)}$ using:

a. Three-Address Instructions

b. Two-Address Instructions

c. One-Address Instructions

d. Zero-Address Instructions

Answer:

a. Three-Address Instructions

MUL	R1, B, C	R1 ← M [B] * M [C]
ADD	R1, R1, A	R1 ← R1+ M [A]
MUL	R2,E,F	R2 ← M [E] * M [F]
SUB	R2, D, R2	R2 ← M [D] – R2
MUL	R3, G, H	R3 ← M [G] * M [H]
ADD	R2, R2, R3	R2 ← R2+ R3
DIV	X, R1, R2	M [X] ← R1/R2

$$X = \frac{A+B*C}{(D-E*F+G*H)}$$

b. Two-Address Instructions

MOV	R1, B	R1 ← M [B]
MUL	R1, C	R1 ← R1* M [C]
ADD	R1, A	R1 ← R1+ M [A]
MOV	R2, D	R2 ← M [D]
MOV	R3, E	R2 ← M [E]
MUL	R3, F	R3 ← R3* M [F]
SUB	R2, R3	R2 ← R2-R3
MOV	R3, G	R2 ← M [G]
MUL	R3, H	R3 ← R3* M [H]
ADD	R2, R3	R2 ← R2+ R3
DIV	R1, R2	R1 ← R1/ R2
MOV	X, R1	M[X] ← R1

$$X = \frac{A+B*C}{(D-E*F+G*H)}$$

c. *One-Address Instructions*

$$X = \frac{A + B * C}{(D - E * F + G * H)}$$

LOAD	E	AC ← M [E]
MUL	F	AC ← AC* M [F]
STORE	T	M [T] ← E*F
LOAD	D	AC ← M [D]
SUB	T	AC ← D-E*F
STORE	T	M [T] ← AC
LOAD	G	AC ← M [G]
MUL	H	AC ← AC* M [H]
ADD	T	AC ← AC+ M [T]
STORE	T	M [T] ← D-E*F+G*H
LOAD	B	AC ← M [B]
MUL	C	AC ← AC* M [C]
ADD	A	AC ← AC+ M [A]
DIV	T	AC ← AC/ M [T]
STORE	X	M [X] ← AC

d. *Zero-Address Instructions*

ABC*+ DEF*- GH*+/

$$X = \frac{A + B * C}{(D - E * F + G * H)}$$

PUSH	A	TOS ← A
PUSH	B	TOS ← B
PUSH	C	TOS ← C
MUL		TOS ← B*C
ADD		TOS ← A+B*C
PUSH	D	TOS ← D
PUSH	E	TOS ← E
PUSH	F	TOS ← F
MUL		TOS ← E*F
SUB		TOS ← D-E*F
PUSH	G	TOS ← G
PUSH	H	TOS ← H
MUL		TOS ← G*H
ADD		TOS ← D-E*F+ G*H
DIV		TOS ← (A+B*C) / (D- E*F+ G*H)
POP	X	