Computer Architecture

Ch2: Instruction set architecture and design

2.1. MEMORY LOCATIONS AND OPERATIONS

The (main) memory can be modelled as an array of millions of adjacent cells, each capable of storing a binary digit (bit), having value of 1 or 0. These cells are organized in the form of groups of fixed number, say n, of cells that can be dealt with as an atomic entity. An entity consisting of 8 bits is called a byte. In many systems, the entity consisting of n bits that can be stored and retrieved in and out of the memory using one basic memory operation is called a word (the smallest addressable entity). Typical size of a word ranges from 16 to 64 bits.

The number of bits, L, needed to distinctly address M words in a memory is given by $L = \log_2 M$. For example, if the size of the memory is 64 M (read as 64 mega words), then the number of bits in the address is $\log_2 (64\times 2^{20}) = \log_2 (2^{26}) = 26$ bits. Alternatively, if the number of bits in the address is l, then the maximum memory size (in terms of the number of words that can be addressed using these l bits) is $M = 2^L$.

As mentioned above, there are two basic memory operations. These are the memory write and memory read operations. Typically, memory read and memory write operations are performed by the central processing unit (CPU).

Three basic steps are needed in order for the CPU to perform a write operation into a specified memory location:

1. The word to be stored into the memory location is first loaded by the CPU into a specified register, called the memory data register (MDR).
2. The address of the location into which the word is to be stored is loaded by the CPU into a specified register, called the memory address register (MAR).
3. A signal, called write, is issued by the CPU indicating that the word stored in the MDR is to be stored in the memory location whose address in loaded in the MAR.

It is worth mentioning that the MDR and the MAR are registers used exclusively by the CPU and are not accessible to the programmer.
Similar to the write operation, three basic steps are needed in order to perform a memory read operation:

1. The address of the location from which the word is to be read is loaded into the MAR.
2. A signal, called read, is issued by the CPU indicating that the word whose address is in the MAR is to be read into the MDR.
3. After some time, corresponding to the memory delay in reading the specified word, the required word will be loaded by the memory into the MDR ready for use by the CPU.

Instruction Formats:

1- An Operation code field that specifies the operation to be performed.
2- An address field that designates a memory address or a processor register.

3- A mode field that specifies the way the operand or the effective address is determined.

Most computer fall into one of three types of CPU organizations:

1- Single accumulator organization.
2- General register organization.
3- Stack organization.

2.2 ADDRESSING MODES

Information involved in any operation performed by the CPU needs to be addressed. In computer terminology, such information is called the operand. Therefore, 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.
Instructions can be classified based on the number of operands as: three-address, two-address, one-and-half-address, one-address, and zero-address.

Example: (Three address)
ADD R1,R2,R3 R3 = R1+R2
ADD A,B,C [C] = [A] + [B]
ADD R1,Racc,Racc

Examples: (Two address)
ADD R1,R2 R2 = R1+R2
ADD A,B [B] = [A] + [B]

MOVE B,C [C] = [B]
ADD A,C [C] = [A]+[C]

ADD R1,Racc

Example: (one-and-half- address)
ADD B,R1 R1=R1+[B]

Example: (one- address)
ADD R1 Racc = R1 + Racc
ADD B Racc = [B] + Racc

Example: (zero-address)
ADD (SP)+, (SP)

The different ways in which operands can be addressed are called the addressing modes. Addressing modes differ in the way the address information of operands is specified. The simplest addressing mode is to include the operand itself in the instruction, that is, no address information is needed. This is **called immediate addressing**. A more involved addressing mode is to compute the address of the operand by adding a constant value to the content of a register. This is **called indexed addressing**. Between these two addressing modes there exist a number of other

addressing modes including **absolute addressing, direct addressing, and indirect addressing**. A number of different addressing modes are explained below.

2.2.1 Immediate Mode

According to this addressing mode, the value of the operand is (immediately) available in the instruction itself.
EX: LOAD #1000,Ri

2.2.2 Direct (Absolute) Mode

According to this addressing mode, the address of the memory location that holds the operand is included in the instruction. EX: LOAD 1000,Ri

2.2.3 Indirect Mode

In the indirect mode, what is included in the instruction is not the address of the operand, but rather a name of a register or a memory location that holds the (effective) address of the operand.
EX: LOAD (1000),Ri
LOAD (R1),R2

2.2.4 Indexed Mode

In this addressing mode, the address of the operand is obtained by adding a constant to the content of a register, called the index register. Consider, for example, the instruction LOAD X(Rind), Ri. This instruction loads register Ri with the contents of the memory location whose address is the sum of the contents of register Rind and the value X.

2.2.5 Other Modes

Relative Mode Recall that in indexed addressing, an index register, Rind , is used.
Relative addressing is the same as indexed addressing except that the program counter (PC) replaces the index register. For example, the instruction LOAD X(PC), Ri loads register Ri with the contents of the memory location whose address is the sum of the contents of the program counter (PC) and the value X.

Autoincrement Mode This addressing mode is similar to the register indirect addressing mode in the sense that the effective address of the operand is the content of a register, call it the autoincrement register that is included in the instruction.
However, with autoincrement, the content of the autoincrement register is automatically incremented after accessing the operand.
Ex: LOAD (Rauto)+, Ri

Autodecrement Mode Similar to the autoincrement, the autodecrement mode uses a register to hold the address of the operand. However, in this case the content of the autodecrement register is first decremented and the new content is used as the effective address of the operand. In order to reflect the fact that the content of the autodecrement register is decremented before accessing the operand, a (-) is included before the indirection parentheses.
Ex: LOAD -(Rauto)+, Ri

2.3 INSTRUCTION TYPES

2.3.1 Data Movement Instructions

Data movement instructions are used to move data among the different units of the machine.

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

2.3.2 Arithmetic and Logical Instructions

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

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

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

2.3.3 Sequencing Instructions

Control (sequencing) instructions are used to change the sequence in which instructions are executed. They take the form of CONDITIONAL BRANCHING (CONDITIONAL JUMP), UNCONDITIONAL BRANCHING (JUMP), or CALL instructions.

A common characteristic among these instructions is that their execution changes the program counter (PC) value.

Flag name	Meaning
Negative (N)	Set to 1 if the result of the most recent operation is negative, it is 0 otherwise
Zero (Z)	Set to 1 if the result of the most recent operation is 0, it is 0 otherwise
Overflow (V)	Set to 1 if the result of the most recent operation causes an overflow, it is 0 otherwise
Carry (C)	Set to 1 if the most recent operation results in a carry, it is 0 otherwise

The change made in the PC value can be unconditional or the change made in the PC by the branching instruction can be conditional based on the value of a specific flag. Examples of these flags include the Negative (N), Zero (Z), Overflow (V), and Carry (C). These flags represent the individual bits of a specific register, called the CONDITION CODE (CC) REGISTER.

The CALL instructions are used to cause execution of the program to transfer to a subroutine. A CALL instruction has the same effect as that of the JUMP in terms of loading the PC with a new value from which the next instruction is to be executed.

However, with the CALL instruction the incremented value of the PC (to point to the next instruction in sequence) is pushed onto the stack.

2.3.4 Input/Output Instructions

Input and output instructions (I/O instructions) are used to transfer data between the computer and peripheral devices. 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. Examples of input devices include a keyboard or a mouse. Input devices are interfaced with a computer through dedicated input ports. Computers can use dedicated addresses to address these ports. Suppose that the input port through which a keyboard is connected to a computer carries the unique address 1000. Therefore, execution of the instruction **INPUT 1000** will cause the data stored in a specific register in the interface between the keyboard and the computer, call it the input data register, to be moved into a specific register (called the accumulator) in the computer. Similarly, the execution of the instruction **OUTPUT 2000** causes the data stored in the accumulator to be moved to the data output register in the output device whose address is 2000.

Alternatively, the computer can address these ports in the usual way of addressing memory locations. In this case, the computer can input data from an input device by executing an instruction such as MOVE Rin, R0. This instruction moves the content of the register Rin into the register R0. Similarly, the instruction MOVE R0, Rin

moves the contents of register R0 into the register Rin that is, performs an output operation. This latter scheme is called memory-mapped Input/Output.

2.4 PROGRAMMING EXAMPLES

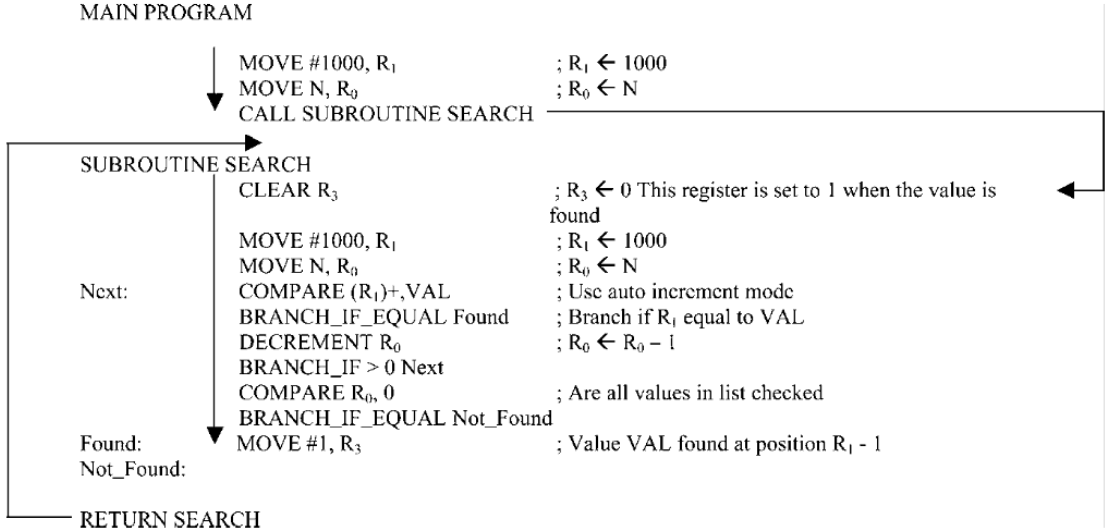
Example 1: In this example, we would like to show a program segment that can be used to perform the task of adding 100 numbers stored at consecutive memory locations starting at location 1000. The results should be stored in memory location 2000.

	CLEAR R_0;	$R_0 \leftarrow 0$
	MOVE # *100*, R_1;	$R_1 \leftarrow 100$
	CLEAR R_2;	$R_2 \leftarrow 0$
LOOP:	*ADD 1000*(R_2), R_0;	$R_0 \leftarrow R_0 + M\ (1000 + R_2)$
	INCREMENT R_2;	$R_2 \leftarrow R_2 + 1$
	DECREMENT R_1;	$R_1 \leftarrow R_1 - 1$
	BRANCH-IF > 0 *LOOP*;	*GO TO LOOP if contents of* $R_1 > 0$
	STORE R_0, *2000*;	$M(2000) \leftarrow R_0$

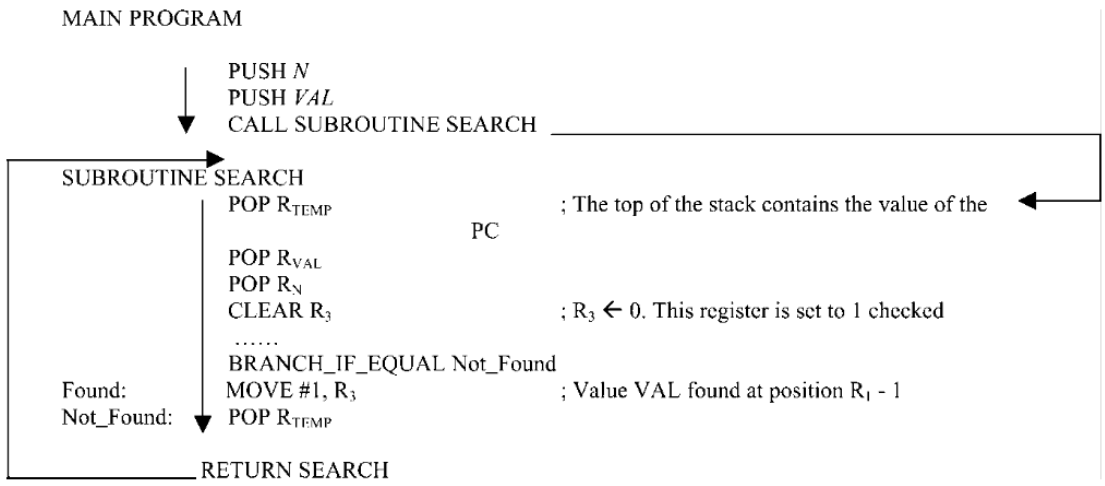
Example 2: This example illustrates the use of a subroutine, SORT, to sort N values in ascending order. The numbers are originally stored in a list starting at location 1000. The sorted values are also stored in the same list and again starting at location 1000. The subroutine sorts the data using the well-known "Bubble Sort" technique. The content of register R3 is checked at the end of every loop to find out whether the list is sorted or not.

```
MAIN PROGRAM

                MOVE #999, R1           ; R1 ← 999
                MOVE N, R0              ; R0 ← N

                CALL SUBROUTINE SORT

SUBROUTINE SORT

Again:          CLEAR R3                ; R3 ← 0

Next:           COMPARE (R1), (R1+1)    ; Compare current and next values
                BRANCH_IF_LOWER Skip    ; Branch if R1 lower than R1 + 1
                MOVE (R1), R2           ; If not, Swap the contents of
                MOVE (R1 + 1), (R1)     ; the current location with the
                MOVE R2 , (R1 + 1)      ; next one.
                MOVE #1, R3             ; Indicate the swap
Skip:           INCEMENT R1             ; R1 ← R1 + 1
                DECREMENT R0            ; R0 ← R0 – 1
                BRANCH_IF_GREATER Next  ; Go to next value

                COMPARE R3, 1           ; Was there any swap?
                BRANCH_IF_EQUAL Again   ; Repeat process
                RETURN SORT
```

Example 3: This example illustrates the use of a subroutine, SEARCH, to search for a value VAL in a list of N values. We assume that the list is not originally sorted and therefore a brute force search is used. In this search, the value VAL is compared with every element in the list from top to bottom. The content of register R3 is used to indicate whether VAL was found. The first element of the list is located at address 1000.



Example 4: This example illustrates the use of a subroutine, SEARCH, to search for a value VAL in a list of N values (as in Example 3). Here, we make use of the stack to send the parameters VAL and N.



2.5 Program Interrupt

Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request. Control returns to the original program after the service program is executed.
The interrupt procedure is, in principle, quite similar to a subroutine call except for three variations:

1- The interrupt is usually initiated by an internal or external signal rather than from the execution of an instruction (except for software interrupt).
2- The address of the interrupt service program is determined by the hardware rather than from the address field of an instruction.
3- An interrupt procedure usually stores all the information necessary to define the state of the CPU rather than storing only the program counter.

Types of Interrupts

1- External interrupts
2- Internal interrupts
3- Software interrupts

External interrupts: come from I/O devices, from a timing device, from circuit monitoring the power supply, or from any other external source. Examples that cause external interrupts are I/O device requesting transfer of data, I/O device finished transfer of data, elapsed time of an event, or power failure.
Internal interrupts: arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called *traps*. Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation.

The difference between internal and external interrupts is that:
the internal interrupts is initiated by some exceptional condition caused by the program itself rather than by an external event.
Internal interrupts are synchronous with the program while external interrupts are asynchronous.
If the program is rerun, the internal interrupts will occur in the same place each time. External interrupts depend on external conditions that independent of the program being executed at the time.

Software interrupts: is initiated by executing an instruction. Software interrupts is a special call instruction that behaves like an interrupt rather than a subroutine call.