College of Computer and Information Sciences

Department of Software Engineering

Computer Organization
CIS 343

ADDRESSING MODES

PROJECT SUPERVISOR
Dr.Hathal Alwageed

GROUP MEMBERS
Kadi Almasoud 421205513
Fatimh Alsrhani 421204562
Retaj Alrasheed 421204537

2022/2023

Table of Contents

Figures and Tables

List of Figures

List of Tables

1.1

Abstract

The addressing mode is one of the fields in an instruction which is determines how and where an operand is fetched from the memory.
Some of the various types of addressing modes are:[5]

- Direct addressing
- Indirect addressing
- Register direct addressing
- Register indirect addressing
- Stack addressing

1.2 Introduction

The 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 but here we explain addressing modes.

The addressing modes refer to the many methods in which operands can be addressed. The method address information for operands is given differs between addressing modes. The address field or fields are rather small in a common instruction style. We'd like to be able to refer to a large number of places in main memory or virtual memory on some systems.

An **addressing mode** is how we tell the processor where to find the data that it needs to use with each instruction[1].

First, virtually all computer architectures provide more than one of these addressing modes. The question is how the processor can figure out which address mode is being used in a given instruction. A variety of ways are used. Various addressing modes are frequently used by different opcodes. A mode field can also be one or more bits in the instruction structure. The mode field's value specifies which addressing mode is utilized.

Second, concerns the interpretation of the effective address (EA). In a system without virtual memory, the effective address will be either a main memory address or a register. In a virtual memory system, the effective address is a virtual address or a register. The actual mapping to a physical address is a function of the memory management unit (MMU) and is invisible to the programmer.

1.3 Direct Addressing

According to this addressing mode, the address of the memory location that holds the operand is included in the instruction. Consider, for example, the case of loading the value of the operand stored in memory location 1000 into register Ri. This operation can be performed using an instruction such as LOAD 1000, Ri. In this instruction, the source operand is the value stored in the memory location whose address is 1000, and the destination is the register Ri. Note that the value 1000 is not prefixed with any special characters, indicating that it is the (direct or absolute) address of the source operand.

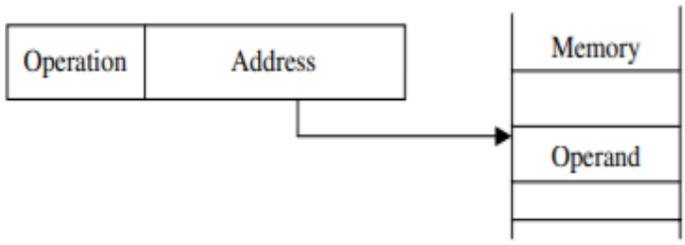

Figure 1.1: Illustration of the direct addressing mode[3]

Figure 1.1 shows an illustration of the direct addressing mode example, if the content of the memory location whose address is 1000 was (2345) at the time when the instruction LOAD 1000, Ri is executed, then the result of executing such instruction is to load the value (2345) into register Ri. Direct (absolute) addressing mode provides more flexibility compared to the immediate mode. However, it requires the explicit inclusion of the operand address in the instruction. A more flexible addressing mechanism is provided through the use of the indirect addressing mode. This is explained below[3].

Direct addressing is a relatively simple kind of addressing in which the address field contains the operand's effective address, The approach was widely used in previous generations of computers, but it is no longer widely used in modern systems. It just needs one memory reference and no additional calculations. The obvious restriction is that it only allows for a limited number of addresses.

1.4 Indirect Addressing

Indirect addressing is a little more complex than direct addressing.

With direct addressing, the length of the address field is usually less than the word length, thus limiting the address range. One solution is to have the address field refer to the address of a word in memory, which in turn contains a full-length address of the operand. This is known as indirect addressing[2].

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. In order to indicate the use of indirection in the instruction, it is customary to include the name of the register or the memory location in parentheses.

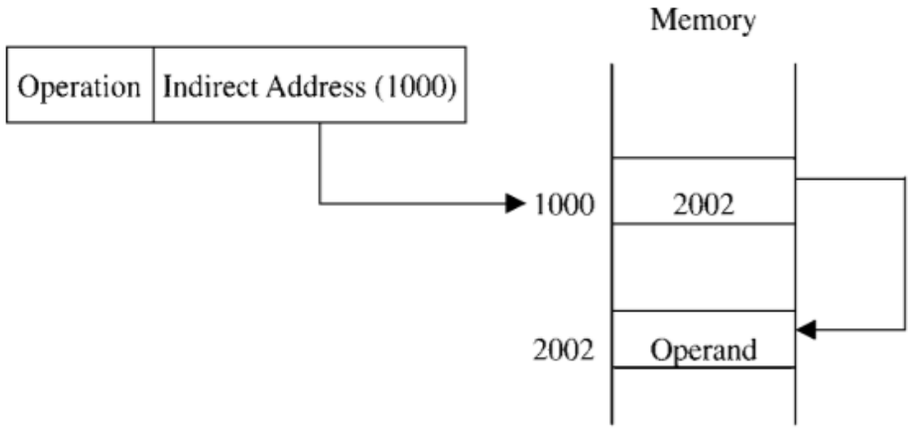

Figure 1.2: Memory indirect addressing[3]

Figure 1.2 shows an example of indirect addressing; Consider, the instruction LOAD (1000), Ri. This instruction has the memory location 1000 enclosed in parentheses, thus indicating indirection. The meaning of this instruction is to load register Ri with the contents of the memory location whose address is stored at memory address 1000[3].

1.5 Register Direct Addressing

Direct addressing and register addressing are comparable. The only change is that instead of the main memory address, the address field relates to a register: R = EA To explain, if a register address field in an instruction contains the number 5, then register R5 is the intended address, and R5 contains the operand value. An address field that refers to registers typically has 3 to 5 bits, allowing for a total of 8 to 32 general-purpose registers to be addressed. The advantages of register addressing are that:

1- The instruction only requires a short address field.
2- No time-consuming memory references are necessary.

A processor's internal register has a substantially shorter memory access time than the main memory address. The downside of register addressing is that it has a very small address space. The processor registers are likely to be substantially used when register addressing is heavily employed in an instruction set. Because there are so few registers (in comparison to main memory locations), using them in this way makes sense only if they are used effectively. A superfluous intermediary step has been added if every operand is brought into a register from the main memory, acted on once, and then returned to the main memory. Instead, if the operand in a register is reused for several operations, actual savings are realized. The programmer or compiler must decide which values should be stored in the main memory and which should remain in registers. Most current processors have several general-purpose registers, putting the assembly-language programmer under a lot of pressure to get things done quickly (e.g., compiler writer)[2].

In this technique of addressing mode:

- The operand is stored in a set of registers.
- The instruction's address field refers to a CPU register that contains the operand.
- The operand can be retrieved without requiring a memory reference.

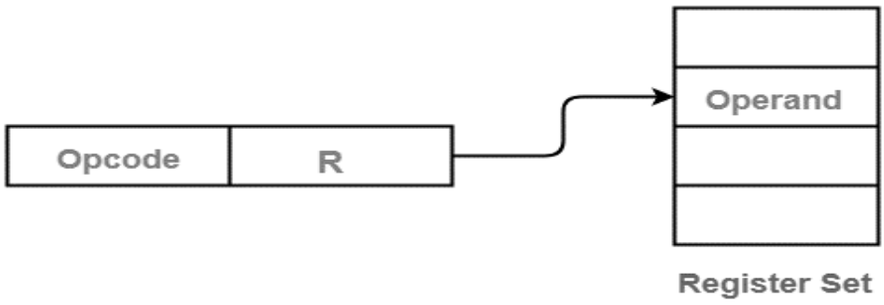

Figure 1.3: Register direct addressing mode[4]

This figure shows an example of register direct addressing mode:

ADD R will increment the value stored in the accumulator by the content of register R.

$AC \leftarrow AC + [R]$[4]

1.6 Register Indirect Addressing

Register indirect addressing is analogous to indirect addressing, just as register addressing corresponds to direct addressing. The only distinction is whether the address field relates to a memory location or a register in both circumstances. As a result, in order to register an indirect address, EA = (R)

The benefits and drawbacks of register indirect addressing are similar to those of indirect addressing. In both situations, the address field's address space limitation (limited range of addresses) is solved by referring to a word length location holding an address. Furthermore, compared to indirect addressing, register indirect addressing utilizes one fewer memory reference[2].

In this technique of addressing mode:

- The instruction's address field refers to a CPU register that holds the operand's effective address.
- To get the operand, only one memory reference is necessary.

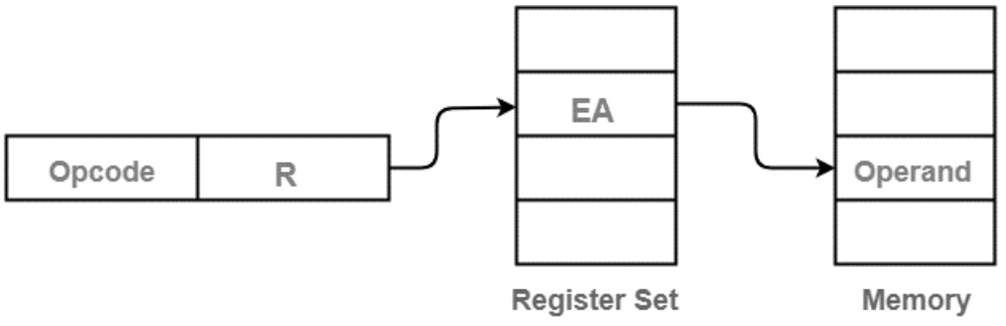

Figure 1.4: Register indirect addressing mode[4]

This figure shows an example of register indirect addressing mode:

-ADD R will increment the value stored in the accumulator by the content of memory location specified in register R.

AC ← AC + [[R]][4]

1.7 Stack Addressing

A stack is a linear array of locations, as defined in Appendix I. It's also known as a pushdown list or a last-in-first-out queue. The stack is a set of sites that has been set aside. Items are added to the top of the stack such that the block is only half-filled at any given time. A pointer is associated with the stack, and its value is the address of the stack's top. The top two elements of the stack could also be in CPU registers, in which case the stack pointer will correspond to the third member of the stack. A register is used to keep track of the stack pointer. As a result, references to memory stack locations are actually registered indirect addresses. Addressing in the stack mode is a type of implied addressing. Machine instructions do not need to specify a memory reference because they act on the top of the stack implicitly[2].

1.8 x86 Addressing Modes

The virtual or effective address generated by the x86 address translation mechanism is offset into a segment. A linear address is created by adding the segment's starting address and the effective address. If paging is employed, this linear address must be translated into a physical address via a page translation mechanism. This last step is skipped in the next sections since it is transparent to the instruction set and the programmer. The x86 is equipped with a variety of addressing modes intended to allow the efficient execution of high- level languages [2].

Mode	Algorithm
Immediate	Operand = A
Register Operand	LA = R
Displacement	LA = (SR) + A
Base	LA = (SR) + (B)
Base with Displacement	LA = (SR) + (B) + A
Scaled Index with Displacement	LA = (SR) + (I) × S + A
Base with Index and Displacement	LA = (SR) + (B) + (I) + A
Base with Scaled Index and Displacement	LA = (SR) + (I) × S + (B) + A
Relative	LA = (PC) + A

Table 1.1: x86 Addressing Modes[2]

R = register
B = base register
I = index register
S = scaling factor
X = contents of X
LA = linear address
SR = segment register
PC = program counter
A = contents of an address field in the instruction

Table 1.1 lists the x86 addressing modes.

Let us consider each of these in turn.

- For the immediate The operand is included in the instruction for the **immediate mode**. The operand can be a single byte, a single word, or a double word[2].

- The operand in **register operand mode** is located in a register. The operand for general instructions like data transmission, arithmetic, and logic can be one of the 32-bit general registers (EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP), one of the 16-bit general registers (AX, BX, CX, DX, SI, DI, SP, BP),

  or one of the 8-bit general registers (AX, BX, CX, DX (AH, BH, CH, DH, AL, BL, CL, DL). There are also several instructions (CS, DS, ES, SS, FS, GS) that relate to the segment selector registers[2].

The remaining addressing modes refer to memory locations. The memory location must be described in terms of the segment that contains the location as well as the offset from the segment's start. A segment is supplied directly in certain circumstances, but in others, it is specified by simple rules that assign a segment by default.

- In the **displacement mode**, the operand's offset (the effective address) is encoded as an 8-, 16-, or 32-bit displacement as part of the instruction. All addresses in instructions correspond to an offset in a segment when segmentation is used. Few machines employ displacement addressing because, as previously stated, it results in large instructions. The displacement value on the x86 can be as long as 32 bits, resulting in a 6-byte instruction. For accessing global variables, displacement addressing might be handy[2].

- The remaining addressing modes are indirect, in that the address component of the command instructs the processor to hunt for the address. The **base mode** indicates that the effective address is stored in one of the 8-, 16-, or 32-bit registers. This is the same as what we've been calling register indirect addressing[2].

- In the **base with displacement mode**, A displacement to be added to a base register, which can be any of the general-purpose registers, is included in the instruction[2].

- The command includes a displacement to be added to a register, in this example an index register, in the **scaled index with displacement mode**. Except for the ESP register, which is used for stack processing, the index register can be any of the general-purpose registers[2].

- To produce the effective address, the **base with index and displacement mode** adds the contents of the base register, the index register, and a displacement. The index register can be any general-purpose register, while the base register can be any general-purpose register. Except for ESP, any general-purpose register can be used[2].

- The contents of the index register multiplied by a scaling factor, the contents of the base register, and the displacement are added in the **based scaled index with displacement mode**. If an array is stored on a stack frame, this is advantageous since the array elements will be 2, 4, or 8 bytes long. When the array members are 2, 4, or 8 bytes in length, this method also allows efficient indexing of a two-dimensional array[2].

- Finally, in transfer-of-control instructions, **relative addressing** can be employed. The value of the program counter, which points to the next instruction, is increased by a displacement. The displacement is handled as a signed byte, word, or doubleword value in this scenario, and that value either raises or reduces the program counter address[2].

1.9 Conclusion

We conclude that addressing modes are The different ways of specifying the location of an operand in an instruction.

Those who write in assembly languages and compiler programmers are most interested in addressing modes in computer programming.

1.10 References

[1] Burrell, Mark, 2004, *Fundamentals of Computer Architecture*, , Macmillan Education UK. (28 Jan 2023)

[2] Stallings, William. *Computer Organization and Architecture: Designing Performance*, Tenth Edition. Ji Xie Gong Ye Chu Ban She, 2019. (29 Jan – 3 Feb 2023)

[3] Mostafa Abd-El-Barr, Hesham El-Rewini - *Fundamentals of Computer Organization and Architecture (Wiley Series on Parallel and Distributed Computing)*-Wiley-Interscie. (29 Jan 2023)

[4] https://www.gatevidyalay.com/addressing-modes/ (1 Feb 2023)

[5] https://unacademy.com/content/nta-ugc/study-material/computer-science/addressing-modes/ (28 Jan 2023)