ADDRESSING MODE

Aarti Singh, Ananya Anikesh

***Abstract*- Addressing mode is the way of addressing a memory location in instruction. Microcontroller needs data or operands on which the operation is to be performed. The method of specifying source of operand and output of result in an instruction is known as addressing In this paper we explain types of addressing mode getting familiar with 8085 addressing mode, explain 8086 addressing mode**

***Index Terms*- addressing mode, 8086 addressing mode,8085 addressing mode etc**

I. INTRODUCTION

Addressing mode is the way of addressing a memory location in instruction. Microcontroller needs data or operands on which the operation is to be performed. The method of specifying source of operand and output of result in an instruction is known as addressing mode. One of a set of methods for specifying the operand(s) for a machinecode instruction.Differentprocessors vary g reatly in the number of addressingmodes they provi de. The more complex modes described belowcan usually be replaced with a short sequence of instructions using only simpler modes. The most common modes are "register" the operand is stored in a specified register; "absolute" the operand is sto red at a specified memory address; and "immediate" - the operand is contained within the instruction. Most processors also have indirect addressing modes, e.g. "register indirect", "memory indirect" where the specified register or memory location does not contain the operand but contains its address, known as the "effective address". For an absolute addressing mode, the effective address is contained within the instruction. Indirect addressing modes often have options for pr e- or post- increment or decrement, meaning that the register ormemory location containing the effective address is incremented or decremented by some amount (either fixed or alsospecified in the instruction), either before or after the instruction is executed. These are very useful for stacks and for accessing blocks of data. Other variations form the effective address by adding together one or more registers an d oneor more constants which may themselves be direct or indirect. Such complex addressing modes are designed tosupport access to multidimensional arrays and arrays of data struct ures. The addressing mode may be "implicit" - the location of the operand is obvious from the particular instruction. This would be the case for an instruction that modified a particular control register in the CPU or, in a stack based processor where operands are always on the top of the stack.

II. NUMBER OF ADDRESSING MODE

Different computer architectures vary greatly as to the number of addressing modes they provide in hardware. There are some benefits to eliminating complex addressing modes and using only one or a few simpler addressing modes, even though it requires a few extra instructions, and perhaps an extra register.[1] It has proven[2] [3] [4] much easier to designpipelined CPUs if the only addressing modes available are simple ones.

Most RISC machines have only about five simple addressing modes, while CISC machines such as the DEC VAX supermini have over a dozen addressing modes, some of which are quite complicated. The IBM System/360 mainframe had only three addressing modes; a few more have been added for the System/390.When there are only a few addressing modes, the particular addressing mode required is usually encoded within the instruction code (e.g. IBM System/360 and successors, most RISC). But when there are lots of addressing modes, a specific field is often set aside in the instruction to specify the addressing mode. The DEC VAX allowed multiple memory operands for almost all instructions, and so reserved the first few bits of each operand specifier to indicate the addressing mode for that particular operand. Keeping the addressing mode specifier bits separate from the opcode operation bits produces an orthogonal instruction set.

Even on a computer with many addressing modes, measurements of actual programs[5] indicate that the simple addressing modes listed below account for some 90% or more of all addressing modes used. Since most such measurements are based on code generated from high-level languages by compilers, this reflects to some extent the limitations of the compilers being used.[6]

III. TYPES OF ADDRESSING MODE

The common addressing modes are:

- **Immediate**
- **Direct**
- **Indirect**
- **Register**
- **Register indirect**
- **Displacement (Indexed)**
- **Stack**
-

Immediate Addressing:

Instruction

- operand is a part of instruction
- operand = address field
- e.g. ADD 5
  —Add 5 to contents of accumulator
  —5 is operand
- No memory reference to fetch data
- Fast
- Limited range

Direct Addressing*:*

- Address field contains address of operand
- Effective address (EA) = address field (A)
- e.g. ADD A
  —Add contents of cell A to accumulator
  —Look in memory at address A for operand
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space

Indirect Addresssing*:*

- Memory cell pointed to by address field contains the address of (pointer to) the operand
- EA = (A)
  —Look in A, find address (A) and look there for operand
- e.g. ADD (A)
  —Add contents of cell pointed to by contents of A to accumulator
- Large address space
- 2n where n = word length
- May be nested, multilevel, cascaded
  —e.g. EA = (((A)))
- Multiple memory accesses to find operand
- Hence slower

Register Addressing:

- Operand is held in register named in address filed
- EA = R
- Limited number of registers
- Very small address field needed
  —Shorter instructions
  —Faster instruction fetch
- No memory access
- Very fast execution
- Very limited address space
- Multiple registers helps performance
  —Requires good assembly programming or compiler writing
  —N.B. C programming
  –register int a;

Register Indirect Addressing*:*

- C.f. indirect addressing
- EA = (R)
- Operand is in memory cell pointed to by contents of register R
- Large address space (2n)
- One fewer memory access than indirect addressing

Displacement Addressing:

- EA = A + (R)
- Address field hold two values
  —A = base value
  —R = register that holds displacement
  —or vice versa

Stack Addressing*:*

- Operand is (implicitly) on top of stack
- e.g.
  —ADD Pop top two items from stack and add
- The stack mode of addressing is a form of implied addressing
- the machine instructions need not include a memory reference but implicitly operate on top of stack.

(30930)

The most common names for addressing mode

Addressing modes	**Example Instruction**	**Meaning**	**When used**
Register	Add R4,R3	R4 <- R4 + R3	When a value is in a register
Immediate	Add R4, #3	R4 <- R4 + 3	For constants
Displacement	Add R4, 100(R1)	R4 <- R4 + M[100+R1]	Accessing local variables
Register deffered	Add R4,(R1)	R4 <- R4 + M[R1]	Accessing using a pointer or a computed address
Indexed	Add R3, (R1 + R2)	R3 <- R3 + M[R1+R2]	Useful in array addressing: R1 - base of array R2 - index amount
Direct	Add R1, (1001)	R1 <- R1 + M[1001]	Useful in accessing static data
Memory deferred	Add R1, @(R3)	R1 <- R1 + M[M[R3]]	If R3 is the address of a pointer *p*, then mode yields **p*
Auto-increment	Add R1, (R2)+	R1 <- R1 +M[R2] R2 <- R2 + *d*	Useful for stepping through arrays in a loop. R2 - start of array *d* - size of an element
Auto-decrement	Add R1,-(R2)	R2 <-R2-*d* R1 <- R1 + M[R2]	Same as autoincrement. Both can also be used to implement a stack as push and pop
Scaled	Add R1, 100(R2)[R3]	R1<- R1+M[100+R2+R3**d*]	Used to index arrays. May be applied to any base addressing mode in some machines.

Getting Familiar with the 8051's Addressing Modes

You don't need to know a lot about the 8051's addressing; we use most modes, of its modest set, but not all. When you are learning assembly language, it's good news that the 8051 isn't very versatile (in contrast, say, to Motorola's 68000, which we used some years ago; it offered fourteen addressing modes). There's less to learn than for a more complex machine. On the other hand, when you're trying to write code to get something done, the 8051's restrictions are less pleasing. Many addressing modes that make perfect sense—such as "MOVX @DPTR, #012h" or "CLR R5" or "MOV R3, R4"—just aren't available

ADDRESSING MODE ON 8086

The x86 instructions use five different operand types: registers, constants, and three memory addressing schemes. Each form is called an *addressing mod*e. The x86 processors support

the *register* addressing mode ,
the *immediate* addressing mode,
the *direct* addressing mode,
the *indirect* addressing mode,
the *base plus index* addressing mode,
the *register relative* addressing mode,

and the *base relative plus index* addressing mode.

Register operands are the easiest to understand. Consider the following forms of the mov instruction:

mov ax, ax
mov ax, bx
mov ax, cx
mov ax, dx

The first instruction accomplishes absolutely nothing. It copies the value from the ax register back into the ax register. The remaining three instructions copy the value of bx, cx and dx into ax. Note that the original values of bx, cx, and dx remain the same. The first operand (the *destinatio*n) is not limited to ax; you can move values to any of these registers.

Constants are also pretty easy to deal with. Consider the following instructions:

mov ax, 25
mov bx, 195
mov cx, 2056
mov dx, 1000

These instructions are all pretty straightforward; they load their respective registers with the specified hexadecimal constant.

There are three addressing modes which deal with accessing data in memory. These addressing modes take the following forms:

mov ax, [1000]
mov ax, [bx]
mov ax, [1000+bx]

The first instruction above uses the *direct* addressing mode to load ax with the 16 bit value stored in memory starting at location 1000 hex.

The mov ax, [bx] instruction loads ax from the memory location specified by the contents of the bx register. This is an *indirect* addressing mode. Rather than using the value in bx, this instruction accesses to the memory location whose address appears in bx.

Note that the following two instructions:

mov bx, 1000
mov ax, [bx]

are equivalent to the single instruction:

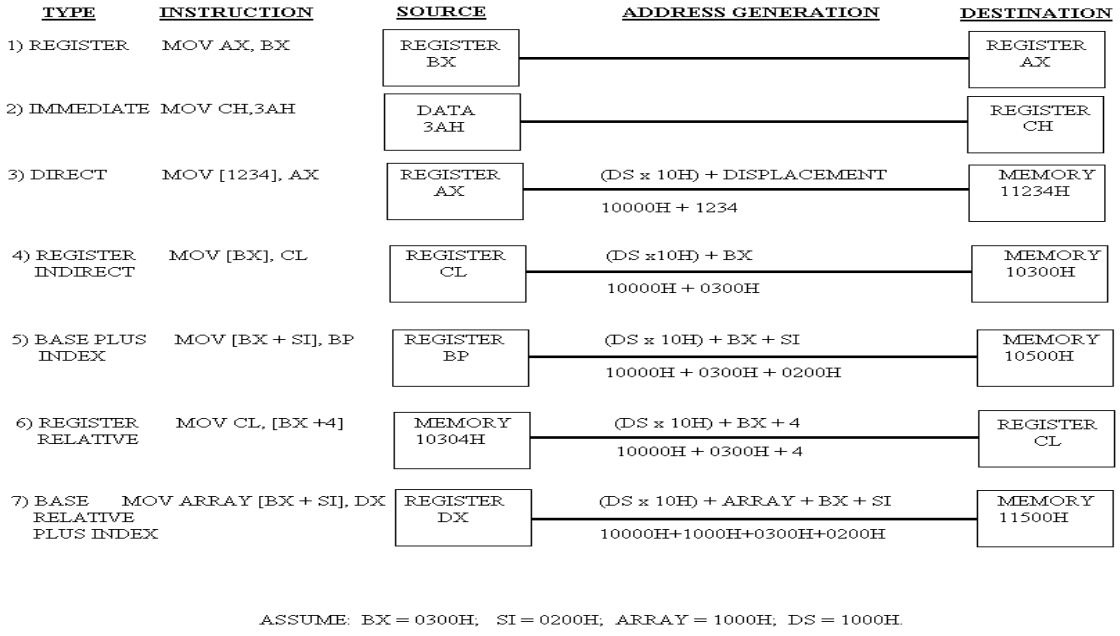
mov ax, [1000]

Of course, the second sequence is preferable. However, there are many cases where the use of indirection is faster, shorter, and better.

Another addressing mode is the *base plus index* addressing mode. An example of this memory addressing mode is

mov ax, [1000+bx]

8086 ADDRESS MODES



ASSUME: BX = 0300H; SI = 0200H; ARRAY = 1000H; DS = 1000H.

IV. CONCLUSION

The term ***addressing modes*** refers to the way in which the operand of an instruction is specified. Information contained in the instruction code is the value of the operand or the address of the result/operand. The type of addressing mode we have explain above.

REFERNCES

[1] "How many addressing modes are enough?" by F. Chow, S. Correll, M. Himelstein, E. Killian, L. Weber, all from MIPS Computer Systems, Inc. 1987 "An Overview of the MIPS-X-MP Project" by John L. Hennessy and Mark A. Horowitz 1986: "MIPS-X uses a single addressing mode: base register plus offset. This simple addressing mode allows the computation of the effective address to begin very early"

[2] http://www.csee.umbc.edu/~squire/cs411_l19.html

[3] http://hpc.serc.iisc.ernet.in/~govind/hpc/L10-Pipeline.txt

[4] John Paul Shen, Mikko H. Lipasti (2004). *Modern Processor Design*. McGraw-Hill Professional.

[5] *Reference Manual IBM 7090 Data Processing System*. IBM. 1962. pp. 9–10.

[6] Jones, Douglas, *Reference Instructions on the PDP-8*, retrieved 1 July 2013

[7] Friend, Carl, *Data General NOVA Instruction Set Summary*, retrieved 1 July 2013

[8] "C Reference: function malloc()"

[9] Dave Brooks. "Some Old Computers".

[10] Bill Purvis. "Some details of the Elliott 803B hardware"