Addressing Modes

Tanmay Bhowmik

Operand, Operator and Opcode

- Operands are the objects that are manipulated and operators are the symbols that represent specific actions.
  - For example, in the expression x + 5, x and 5 are operands and + is an operator.
- Opcode is an instruction that tells processor what to do with the variable or data written besides it.
  - For Example: ADD R1,R2
  - ADD is the Opcode and R1 and R2 are the Operand.

Addressing Modes – What is…

- The term addressing modes refers to the way in which the operand of an instruction is specified.
- The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually executed.
- An addressing mode specifies how to calculate the effective memory address of an operand by using information held in registers and/or constants contained within a machine instruction

Format

- The control unit is designed to go through an instruction cycle that is divided into three major phases:
  - Fetch the instruction
  - Decode the instruction
  - Execute the instruction
- There is one register called Program Counter (PC) that keeps track of the instructions in the program stored in memory.
- An example of an instruction format with a distinct addressing mode field is shown in following figure:

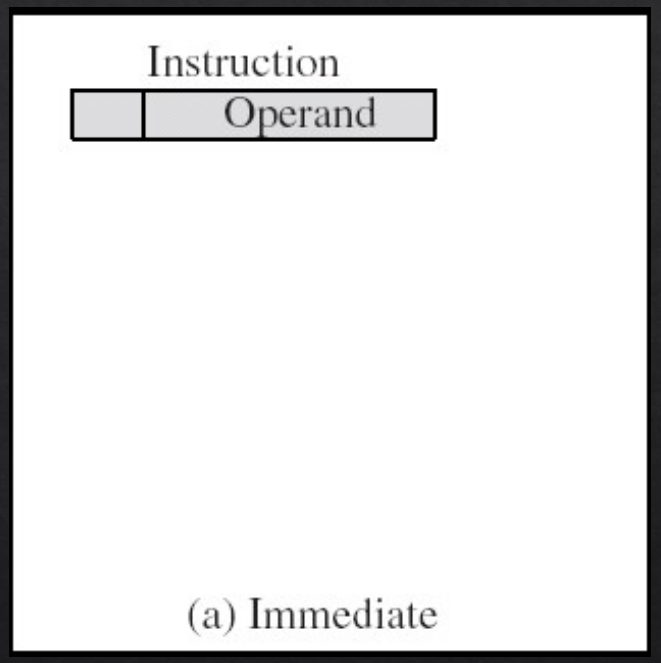
Figure 8-6 Instruction format with mode field.

Addressing Modes – Types

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

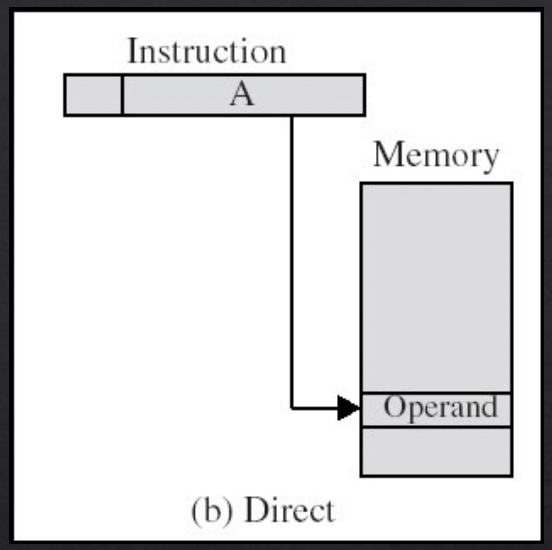
Immediate Addressing

- Operand is part of instruction
- Operand = address field
- e.g. ADD AX, 5h
- LDA #5
  - Add 5 to contents of accumulator
  - 5 is operand
- No memory reference to fetch data
- Fast
- Limited range



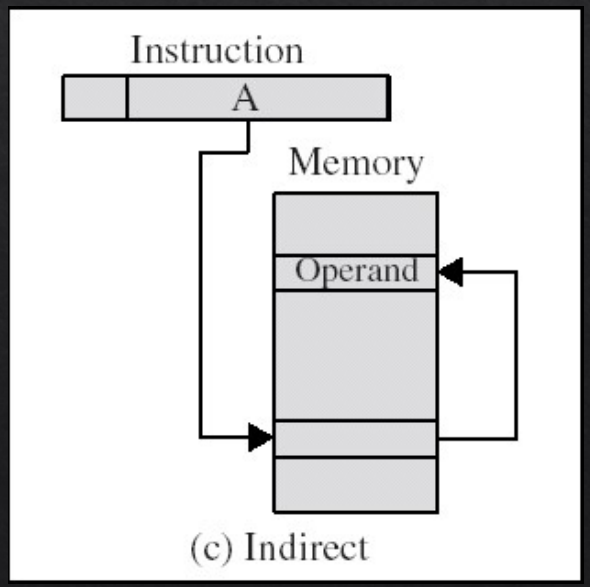
Direct Addressing

- Address field contains address of operand
- Effective address EA = address field (A)
- ADD AX, value
- Value DB 05h
  - Add contents of cell value to accumulator AX
  - Look in memory at address value for operand
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space



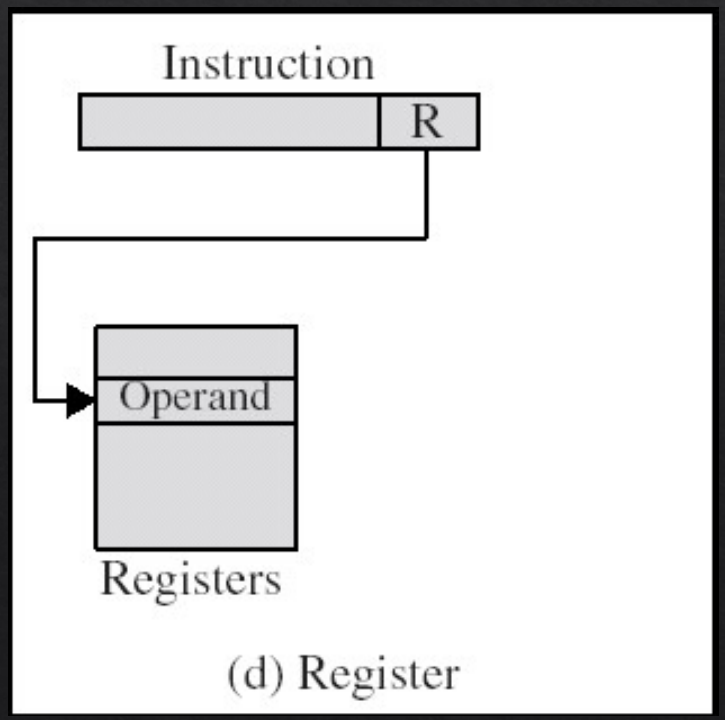
Indirect Addressing

- 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 AX, (A)
  - Add contents of cell pointed to by contents of A to accumulator



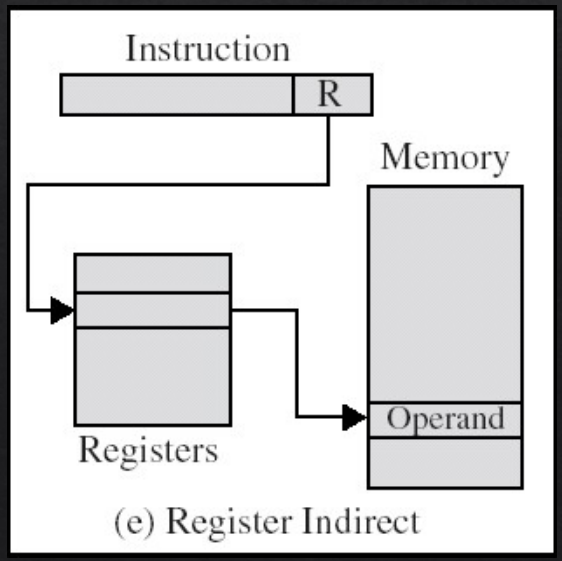
Register Addressing

- 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
  - MOV AX, BX
  - ADD AX, BX



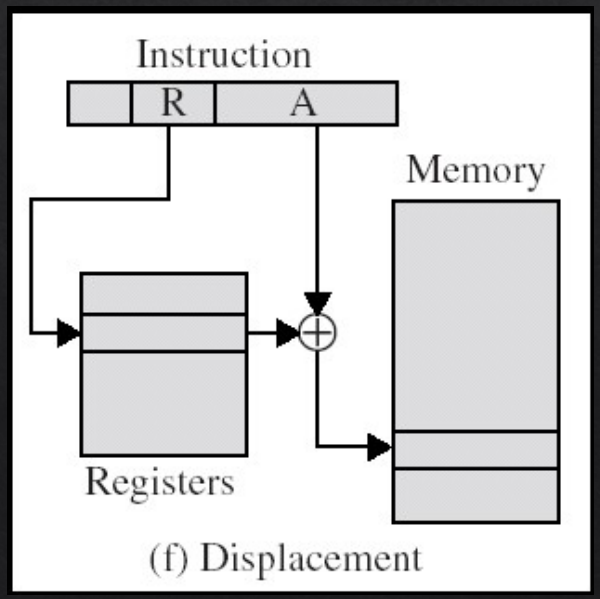
Register Indirect Addressing

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



Displacement Addressing

- EA = A + (R)
- Effective address=start address + displacement
- Effective address=Offset + (Segment Register)
- Use direct and register indirect
- Address field hold two values
  - A = base value
  - R = register that holds displacement
  - or vice versa



Relative Addressing (PC-Relative)

- A version of displacement addressing
- R = Program counter, PC
- EA = A + (PC)
- i.e. get operand from A cells from current location pointed to by PC
- c.f locality of reference & cache usage

Base-Register Addressing

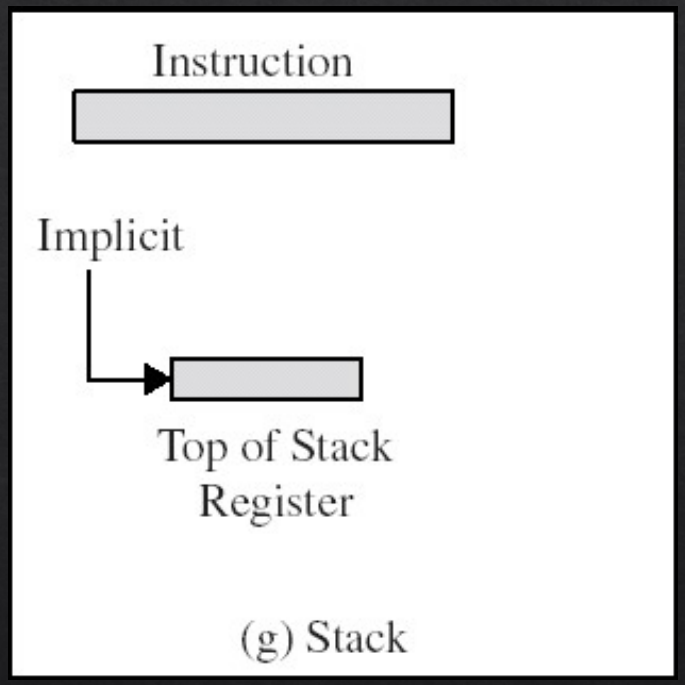
- A holds displacement
  - EA = (CS) + A
- R holds pointer to base address
- R may be explicit or implicit

Indexed Addressing

- A = base
- R = displacement
  - EA = A + (R)
- Good for accessing arrays
  - EA = A + (R)
  - R++

Stack Addressing

- Operand is (implicitly) on top of stack
- e.g.
  - ADD Pop top two items from stack and add and push



Finally…

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