BCN1043

COMPUTER ARCHITECTURE & ORGANIZATION

By
Dr. Mritha Ramalingam

Faculty of Computer Systems & Software Engineering
mritha@ump.edu.my

http://ocw.ump.edu.my/

AUTHORS

- **Dr. Mohd Nizam Mohmad Kahar** (mnizam@ump.edu.my)
- **Jamaludin Sallim** (jamal@ump.edu.my)
- **Dr. Syafiq Fauzi Kamarulzaman** (syafiq29@ump.edu.my)
- **Dr. Mritha Ramalingam** (mritha@ump.edu.my)

Faculty of Computer Systems & Software Engineering

Universiti Malaysia PAHANG
Engineering • Technology • Creativity

BCN1043

COMPUTER ARCHITECTURE & ORGANIZATION

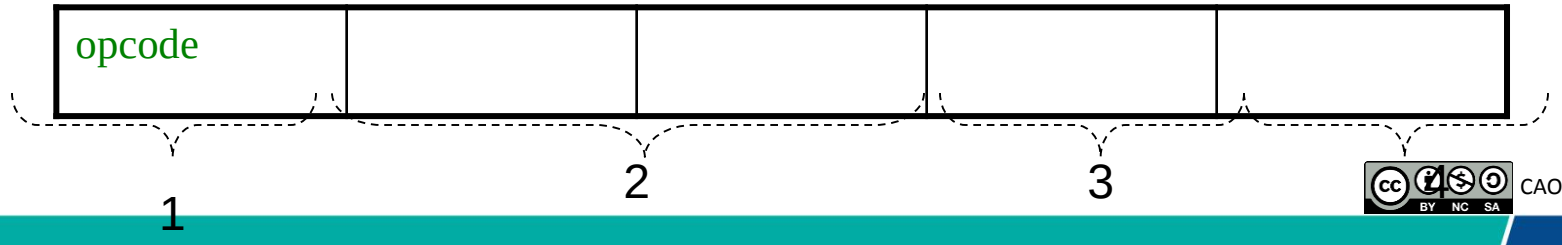
Chapter 4 continues…

CHAPTER 4
ASSEMBLY LANGUAGE

- Introduction
- The Computer Organization
- Instruction Format
- Addressing Mode
- DEBUG program

Instruction Format

- The operation of CPU is determined by the instructions it executes (machine or computer instructions)
- CPU's **instruction set** – the collection of different instructions that CPU can execute
- Each instruction must contain the information required by CPU for execution :-
  1. Operation code (opcode) -- specifies the operation to be performed (eg: ADD, I/O) →*do this*
  2. Source operand reference -- the operation may involve one or more source operands (input for the operation) → *to this*
  3. Result operand reference -- the operation may produce a result → *put the answer here*
  4. Next instruction reference -- to tell the CPU where to fetch the next instruction after the execution of this instruction is complete → *do this when you have done that*



- Operands (source & result) can be in one of the 3 areas:-
  - Main or Virtual Memory
  - CPU register
  - I/O device
- It is not efficient to put all the information required by CPU in a machine instruction
- Each instruction is represented by sequence of bits & is divided into 2 fields; opcode & address

opcode	address

- Processing become faster if all information required by CPU in one instruction or one instruction format

opcode	address for Operand 1	address for Operand 2	address for Result	address for Next instruction

- Problems → instruction become long (takes a few words in main memory to store 1 instruction)
- Solution → provide a few instruction formats (format instruction); 1, 2, 3 and addressing mode.
- Instruction format with 2 address is always used; INTEL processors

- Opcodes are represented by abbreviations, called mnemonics, that indicate the operation. Common examples:

  ADD Add

  SUB Subtract

  DIV Divide

  LOAD Load data from memory

  STOR Store data to memory

Instruction-3-address

opcode	address for Result	address for Operand 1	address for Operand 2

- Example : SUB Y A B Y = A - B
  - Result → Y
  - Operand 1 → A
  - Operand 2 → B
  - Operation = subtracts
  - Address for Next instruction? → program counter (PC)

Instruction-2-address

opcode	address for Operand 1 & Result	address for Operand 2

- Example : SUB Y B Y = Y - B
  - Operand 1 → Y
  - Operand 2 → B
  - Result → replace to operand 1
  - Address for Next instruction? → program counter (PC)

Instruction-1-address

opcode	**address for Operand 2**

- Example :

  LOAD A

  ADD B AC = AC + B

  or

  SUB B AC = AC - B

  – Operand 1 & Result → in AC (accumulator), a register
  – **SUB B** → B subtracts from AC, the result stored in AC
  – Address for Next instruction? → program counter (PC)

- Short instruction (requires less bit) but need more instructions for arithmetic problem

Y = (A-B) / (C+D x E)

Instruction	Comment
• SUB Y, A, B	• Y ⟵ A - B
• MPY T, D, E	• T ⟵ D x E
• ADD T, T, C	• T ⟵ T + C
• DIV Y, Y, T	• Y ⟵ Y / T

Three-address instructions

Y = (A-B) / (C+D x E)

- MOVE Y, A
- SUB Y, B
- MOVE T, D
- MPY T, E
- ADD T, C
- DIV Y, T

Two-address instructions

Y = (A-B) / (C+D x E)

INSTRUCTIONS

- LOAD D
- MPY E
- ADD C
- STOR Y
- LOAD A
- SUB B
- DIV Y
- STOR Y

Comment

- AC ← D
- AC ← AC x E
- AC ← AC + C
- Y ← AC
- AC ← A
- AC ← AC – B
- AC ← AC / Y
- Y ← AC

One-address Instructions

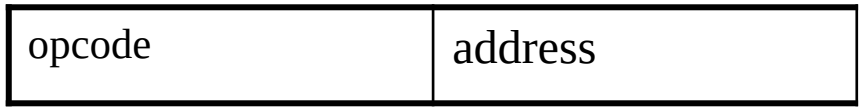
Utilization of Instruction Addresses

Number of Addresses	Symbolic Representation	Interpretation
3	OP A, B, C	A ← B OP C
2	OP A, B	A ← A OP B
1	OP A	AC ← AC OP A
0	OP	T ← (T – 1) OP T

AC = accumulator
T = top of stack
A, B, C = memory or register locations

** Zero-address instructions are applicable to a special memory organisation, called a stack.

Addressing Mode

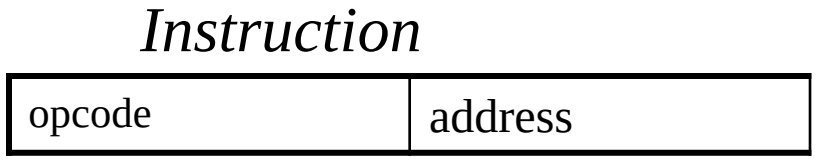


- Address field → address for operand & result
- Number of bit required to store data (operand & result)
  - Eg: field size for an operand = 4 bit, 2^4=16 space for address can be used to store an operand
- How is the address of an operand specified?
- Addressing mode - A technique to identify address to access operands

- The most common addressing modes are:
  - Immediate
  - Direct
  - Indirect
  - Register
  - Register Indirect
  - Displacement
  - Stack

Immediate Addressing Mode

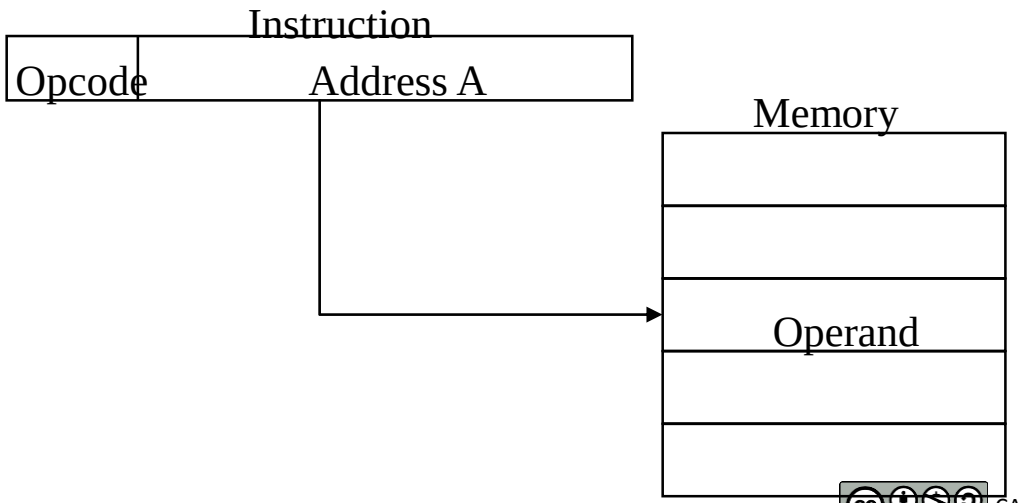
- The data or operand is contained in the instruction → address field



- Eg: **ADD 20**
  - Operand = 20
  - add 20 to contents of accumulator (AC)
  - If value in AC = 10, the result → 30
- No memory reference to fetch data
- Fast
- Limited address space

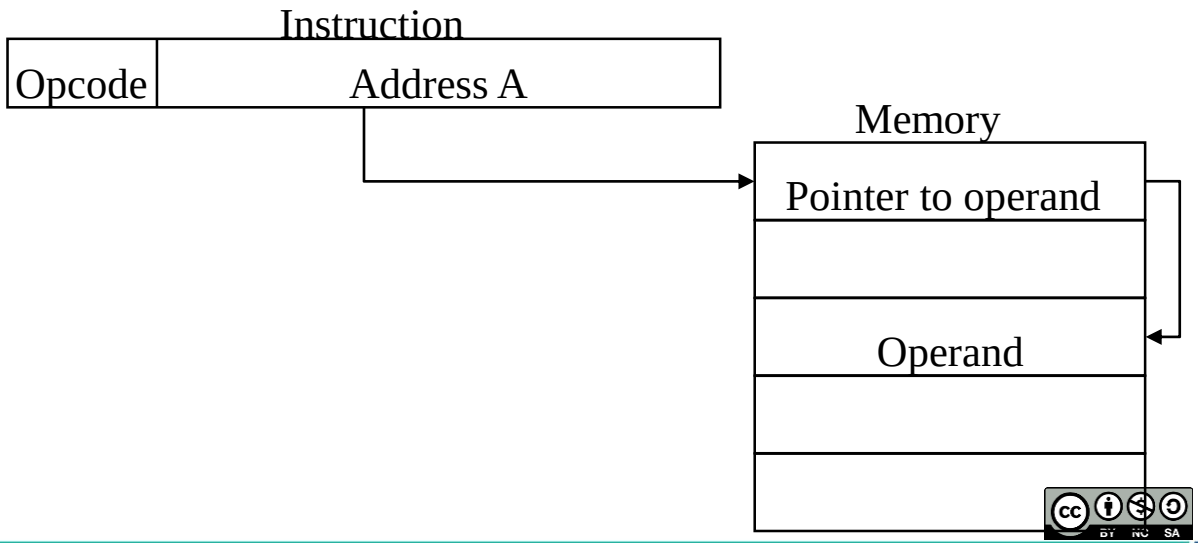
Direct Addressing Mode

- Address field → address of operand
- E.g. ADD A
  - Add contents of cell A to accumulator (AC)
  - Look in memory at address A for operand
- One memory reference to access data
- No additional calculations to work out **A** (effective address)
- Limited address space



Indirect Addressing Mode

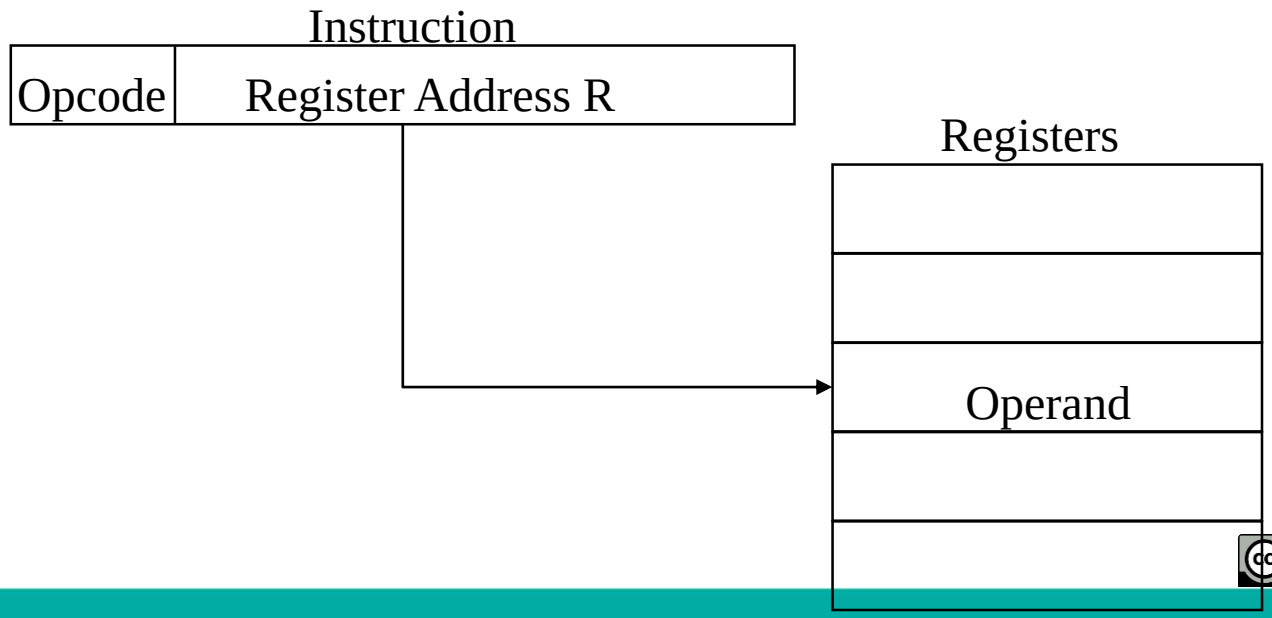
- Memory cell pointed to by address field contains the address of the operand
- E.g. ADD A
  - Look in A, find address (A) and look there for operand
  - Add contents of cell pointed to by contents of A to accumulator (AC)
- Multiple memory accesses to find operand
- Slower



Register Direct Addressing Mode

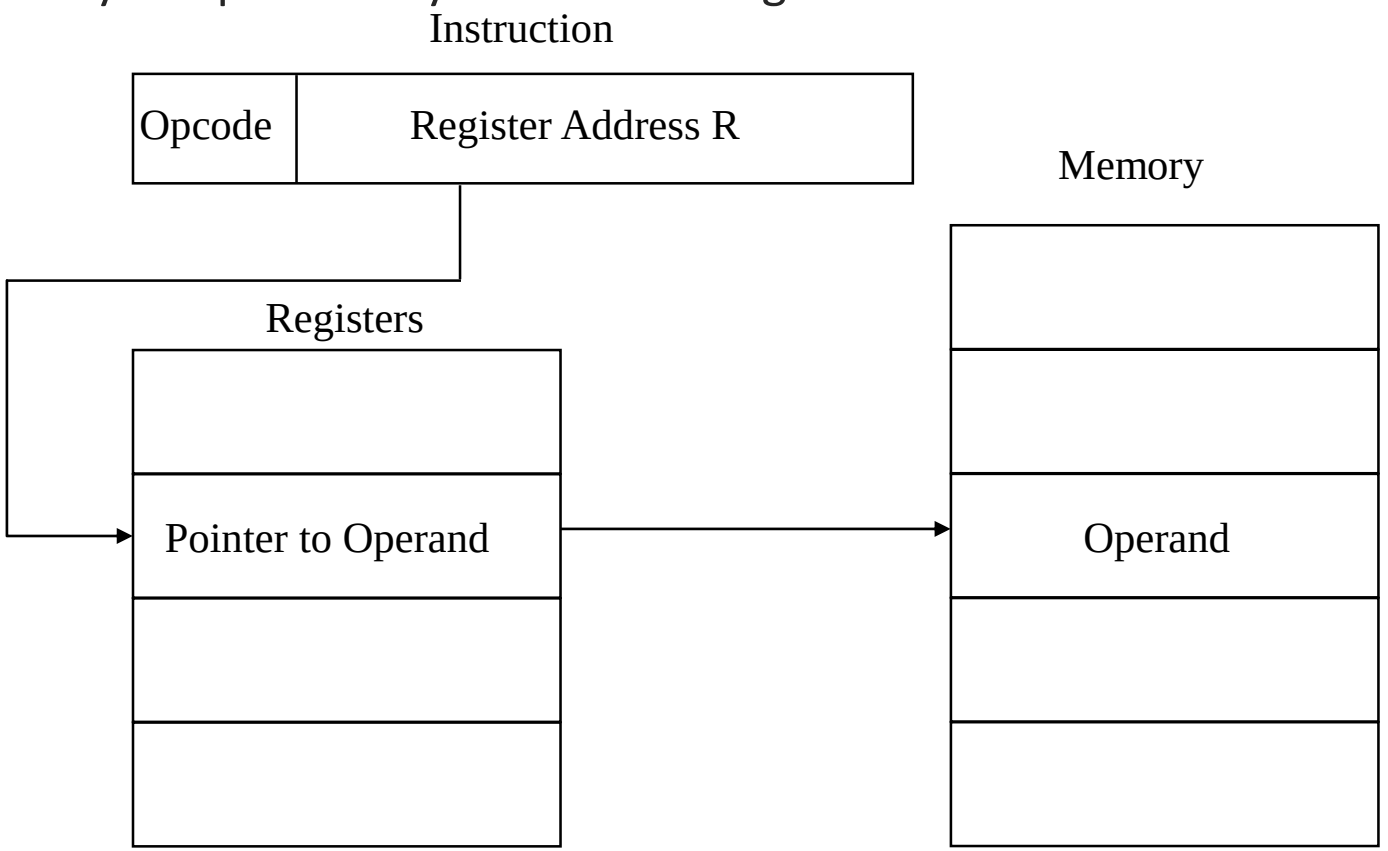
- the operand is held in register named in address field

- Small address field is needed
- No memory references are required
- Fast execution
- But, address space is limited



Register Indirect Addressing Mode

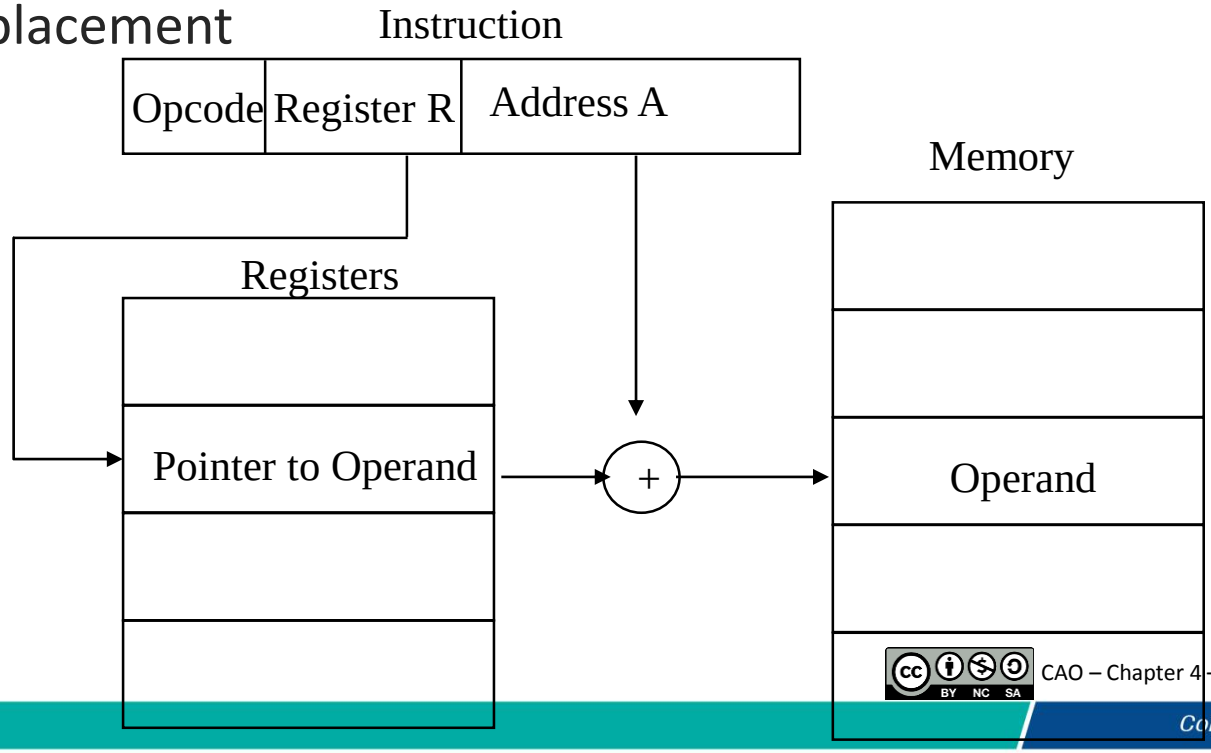
- using a register as a pointer to memory
- Operand is in memory cell pointed by contents of register



- Disadvantages the same as indirect addressing

Displacement Addressing Mode

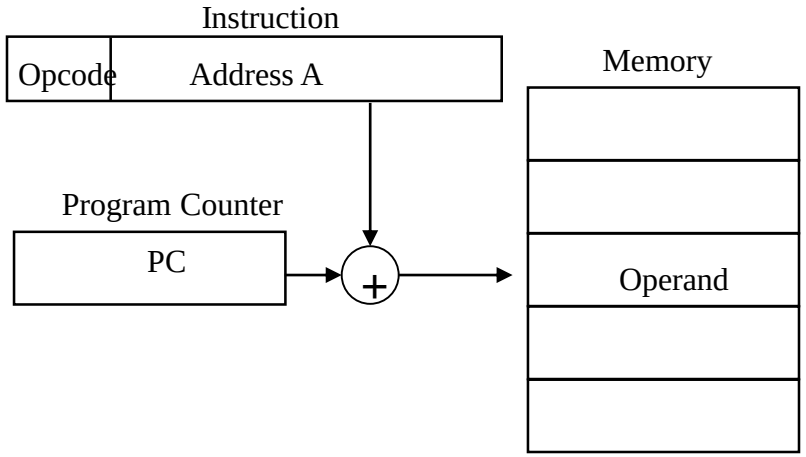
- Combines the capabilities of direct addressing & register indirect addressing.
- Address field = A + (R)
- Address field hold two values
  - A = base value
  - R = register that holds displacement
  - or vice versa



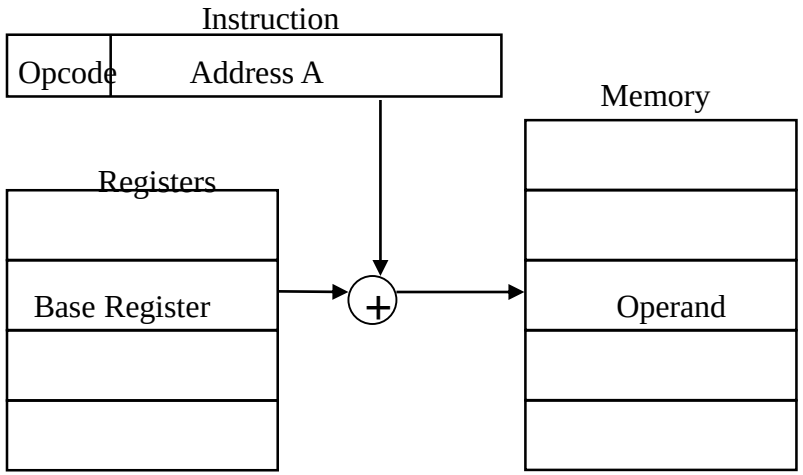
• Common uses of displacement addressing:

- **Relative Addressing**
  - R = Program counter, PC
  - EA = A + (PC)
  - i.e. get operand from A cells from current location pointed to by PC
- **Base-Register Addressing**
  - A holds displacement
  - R holds pointer to base address
  - R may be explicit or implicit
  - e.g. segment registers in 80x86
- **Indexed Addressing**
  - A = base
  - R = displacement
  - EA = A + R
  - Good for accessing arrays
    - EA = A + R
    - R++

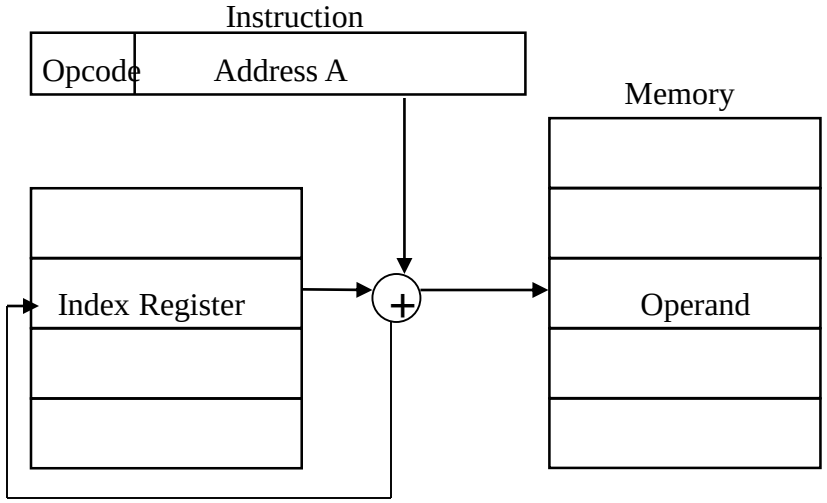
Displacement Addressing



Relative Addressing



Base-Register Addressing

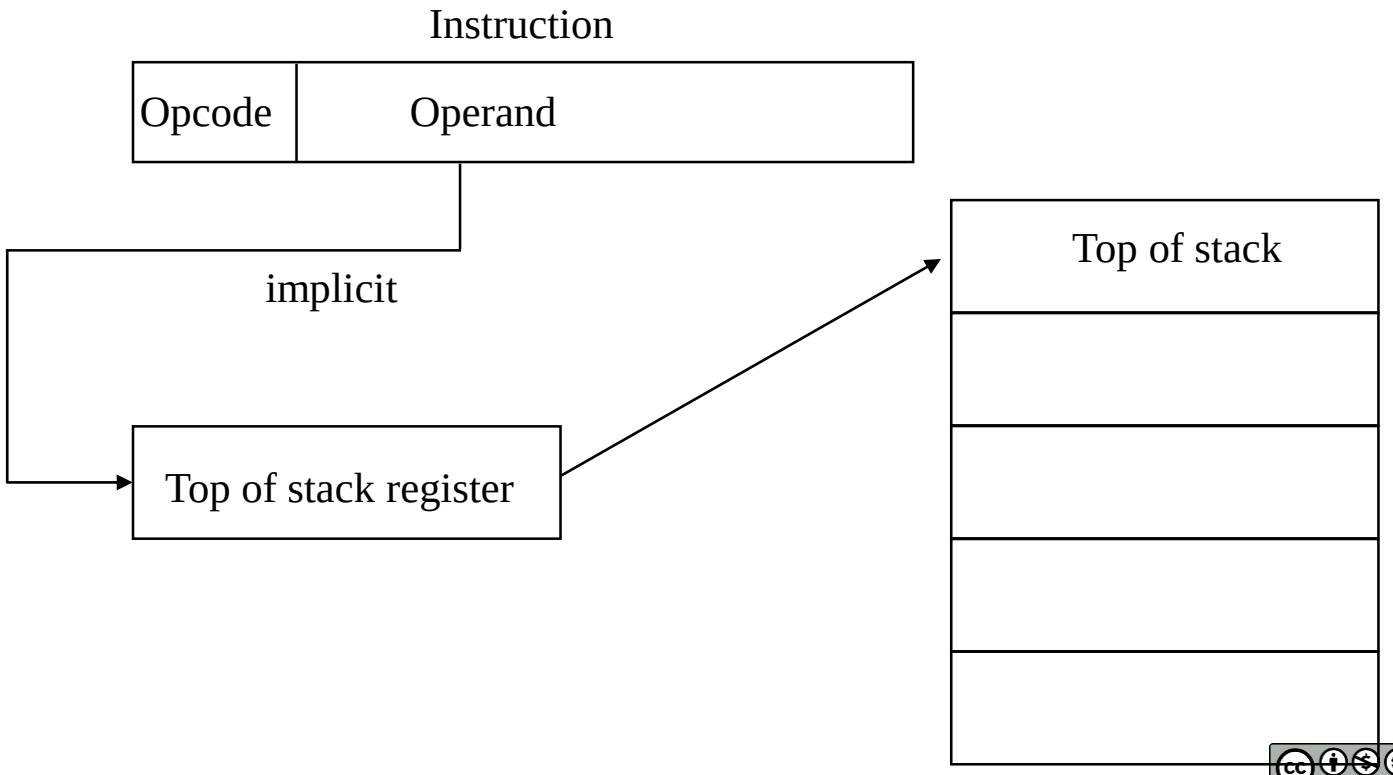


Indexed Addressing

- Better distinction of the base and indexing might be who/what does the reference.
- Examples:
  - Indexing is used within programs for accessing data structures
  - Base addressing is used as a control measure by the OS to implement segmentation

Stack Addressing Mode

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



- There are 3 types of addressing for INTEL processor:
  - Immediate addressing
  - Register addressing
  - Memory addressing
- Memory addressing → 6 types :
  - Direct memory addressing
  - Direct-offset addressing – same as direct memory addressing but use arithmetic operation to edit address.
  - Indirect memory addressing
  - Base displacement addressing – content in base registers (BX & BP) & index registers (DI & SI), plus with displacement value in instruction format to get the correct address for the data.
  - Base index addressing – content in base registers, plus with the content in index registers to get the correct address.
  - Base index with displacement addressing – content in base registers, plus with the content in index registers & displacement value to get the correct address.

General Purpose Registers (continue)

- Are available for addition and subtraction of 8-, 16- or 32-bits values:

```
MOV   EAX, 225 ;Move 225 to EAX
ADD   AX, CX          ; Add CX to AX (words)
SUB  BL, AL   ; Subtract AL from BL (bytes)
```

Will continue…