Addressing Modes

Ways to get operands i.e. how to address operands?

Do there exist multiple ways?

how machine language instructions identify the operand (or operands) of each instruction.

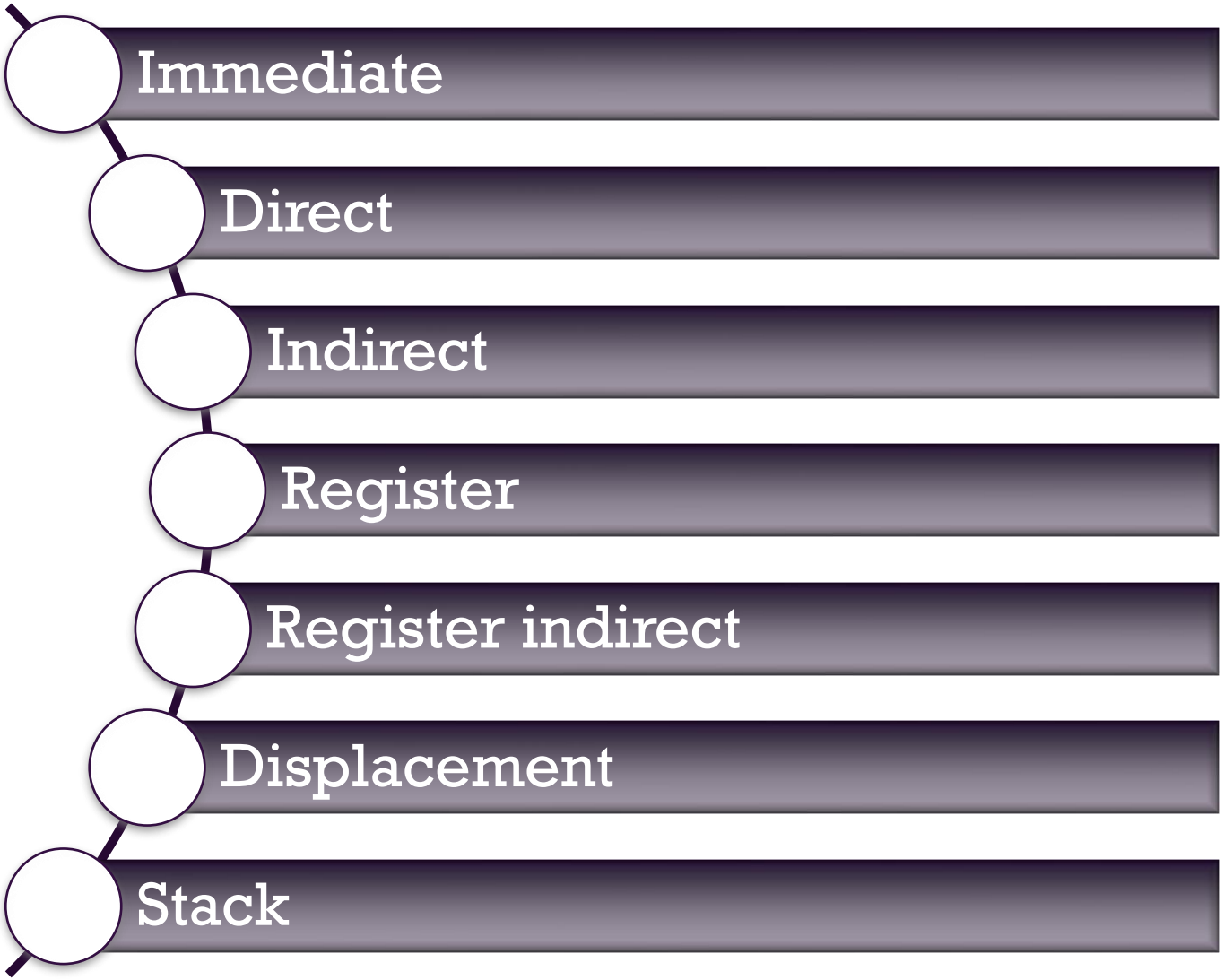
Instruction Set Design

- When we want to make CPU, We design instruction set. Goal is
- Instruction length
  - Must be short
- how operands are addressed?
  - Multiple ways….programmer is at ease
- Instruction format
  - What bits represent what?
    - Opcode is how many bits?
    - Operands are how many?
      - And how many bits?

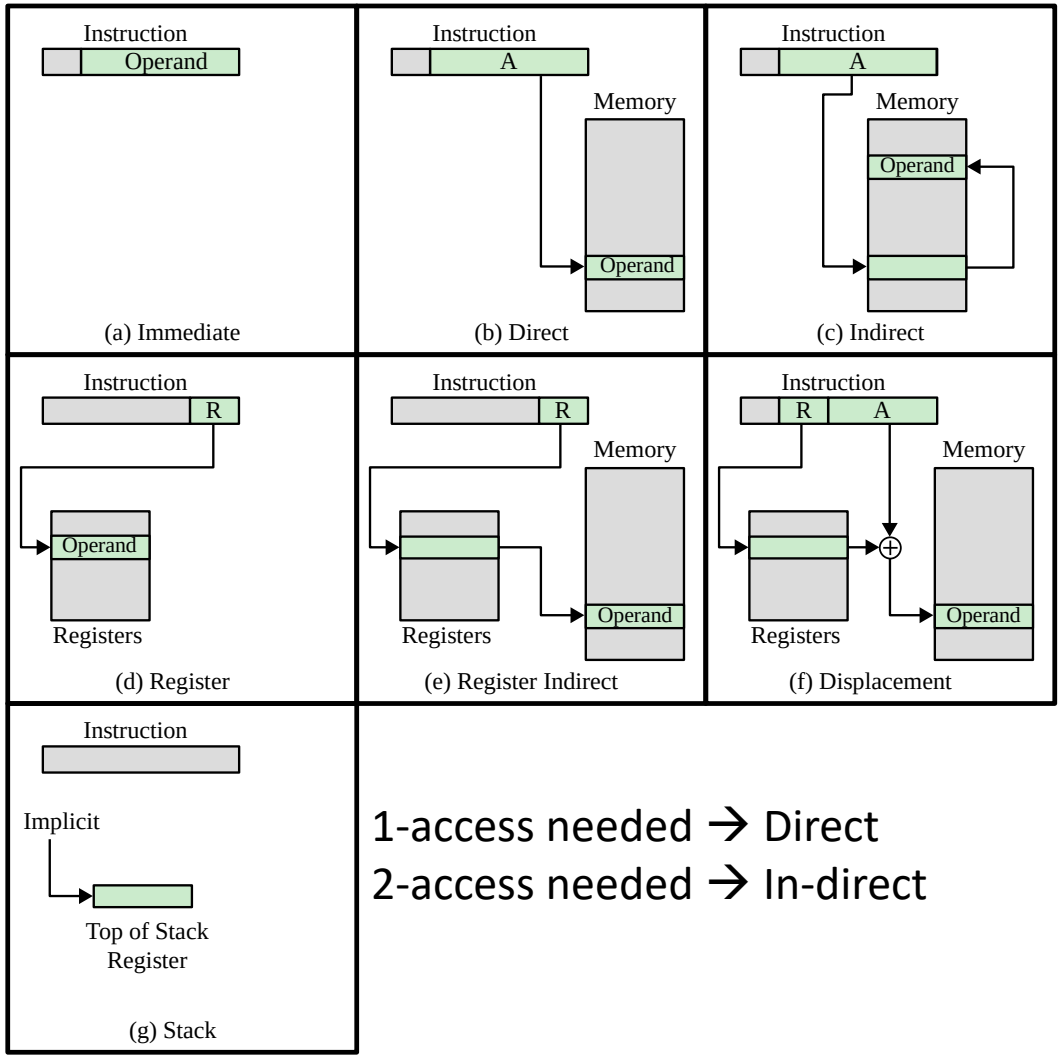
Instruction format trade-offs

- Large instruction set => small programs
- Small instruction set => large programs
- Large memory => longer instructions (address encoding)
- Fixed length instructions same size or multiple of bus width => fast fetch
- Variable length instructions may need extra bus cycles
- If processor Executes faster than Fetch then what to do?
  - Cache memory…..Full fledge Topic we will discuss later on
  - Shorter instructions

Common Addressing Mode



Addressing Modes



1-access needed → Direct

2-access needed → In-direct

Basic Addressing Modes

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) CS+IP	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

Immediate Addressing → ADD 5

- Operand = A
  - At address field, there is operand.
  - MOV CL,12H MOV AX, 1234H
  - We can identify operand easily.
- This mode can be used to define and use constants or set initial values of variables
  - Typically the number will be stored in twos complement form
- Advantage:
  - Fast
    - No memory reference other than the instruction fetch is required to obtain the operand, thus saving one memory or cache cycle in the instruction cycle
- Disadvantage:
  - The size of the number is restricted to the size of the address field, which, in most instruction sets, is small compared with the word length
  - not flexible, since the value is fixed at compile-time

Direct Addressing → ADD X

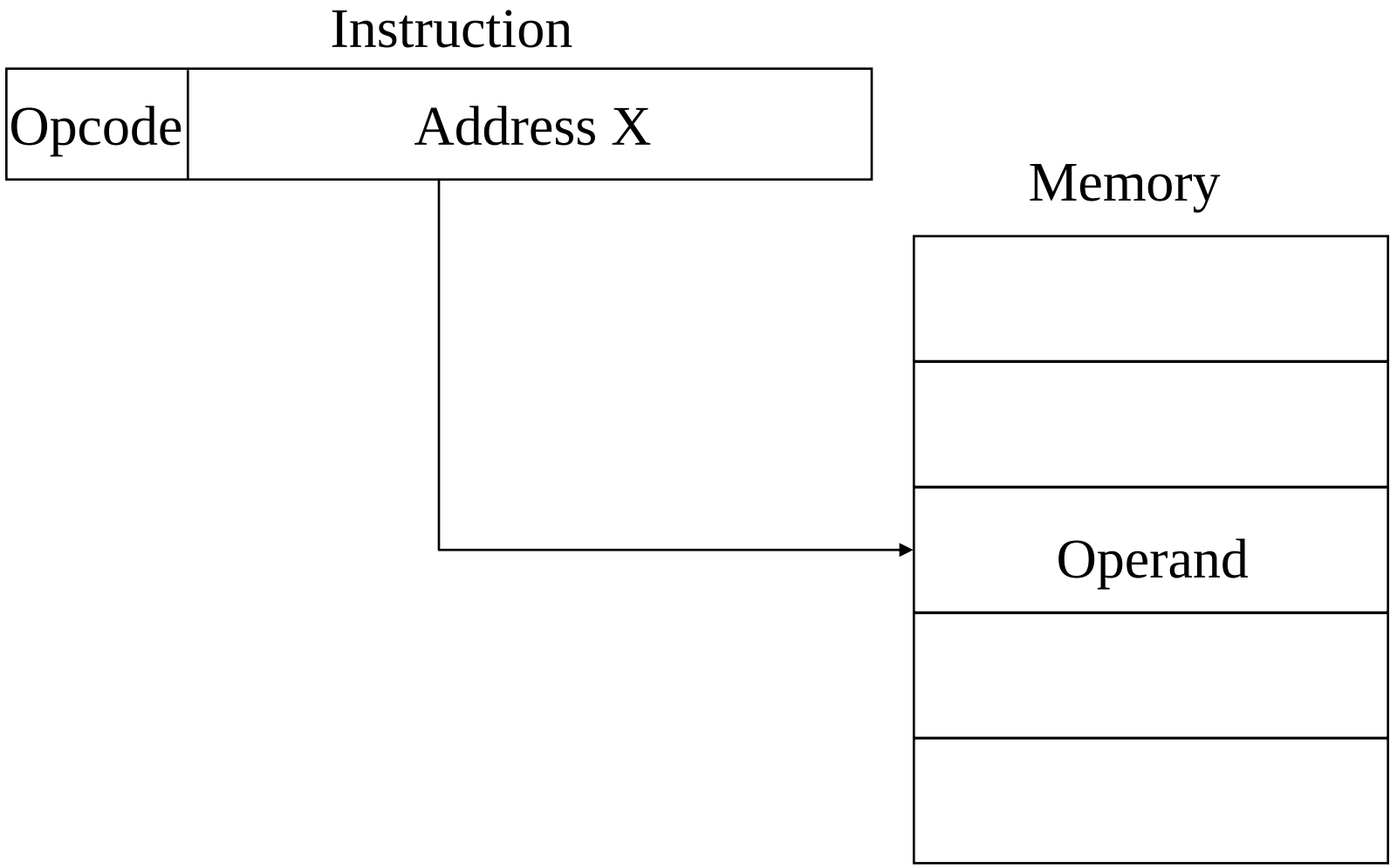
- The instruction tells where the value can be found, but the value itself is out in memory.

- The address field contains the address of the operand
- Effective address (EA) = address field (A)
- e.g. Add X same as Add [1000H]
  - Same as MOV CL, [4321H]
  - Just Offset Address is specified…segment is identified by the instruction and physical address is calculated.
    - Segmentation ---- will discuss later
    - Variables -> in Data Segment, Statements → in Code segment
- In a high-level language, direct addressing is frequently used for things like global variables.

Direct Addressing Pros and Cons

- Advantage
  - Single memory reference to access data
  - No additional calculations to determine effective address
  - More flexible than immediate
- Disadvantage
  - Limited address space

Direct Addressing

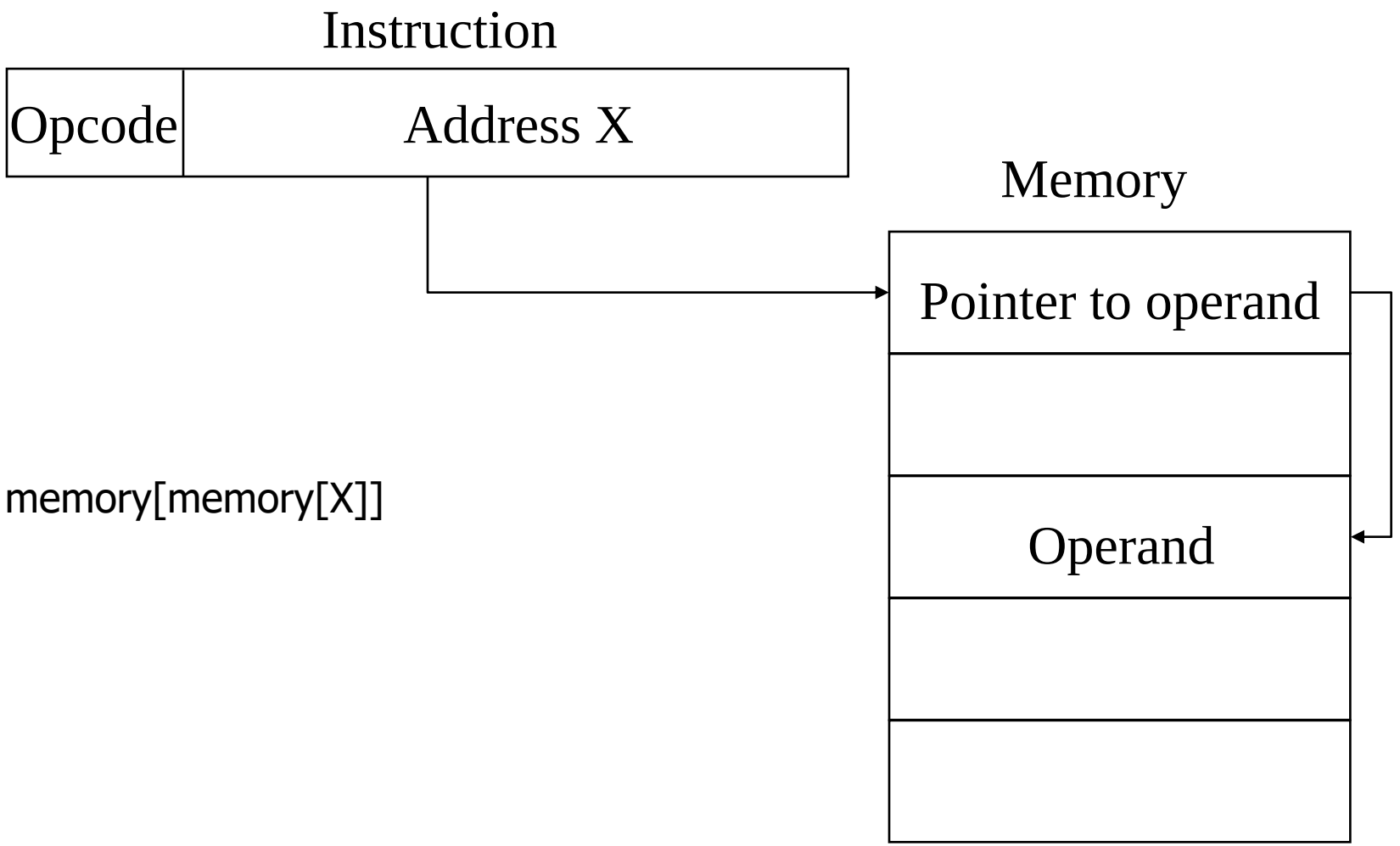


Memory Indirect Addressing

→ADD [X] or ADDI X

- Reference to the address of a word in memory which contains a full-length address of the operand
- EA = [X]
  - Value at X is also an Address
- Advantage:
  - For a word length of *N* an address space of 2^N is now available
- Disadvantage:
  - Instruction execution requires two memory references to fetch the operand
    - One to get its address and a second to get its value

Indirect Addressing



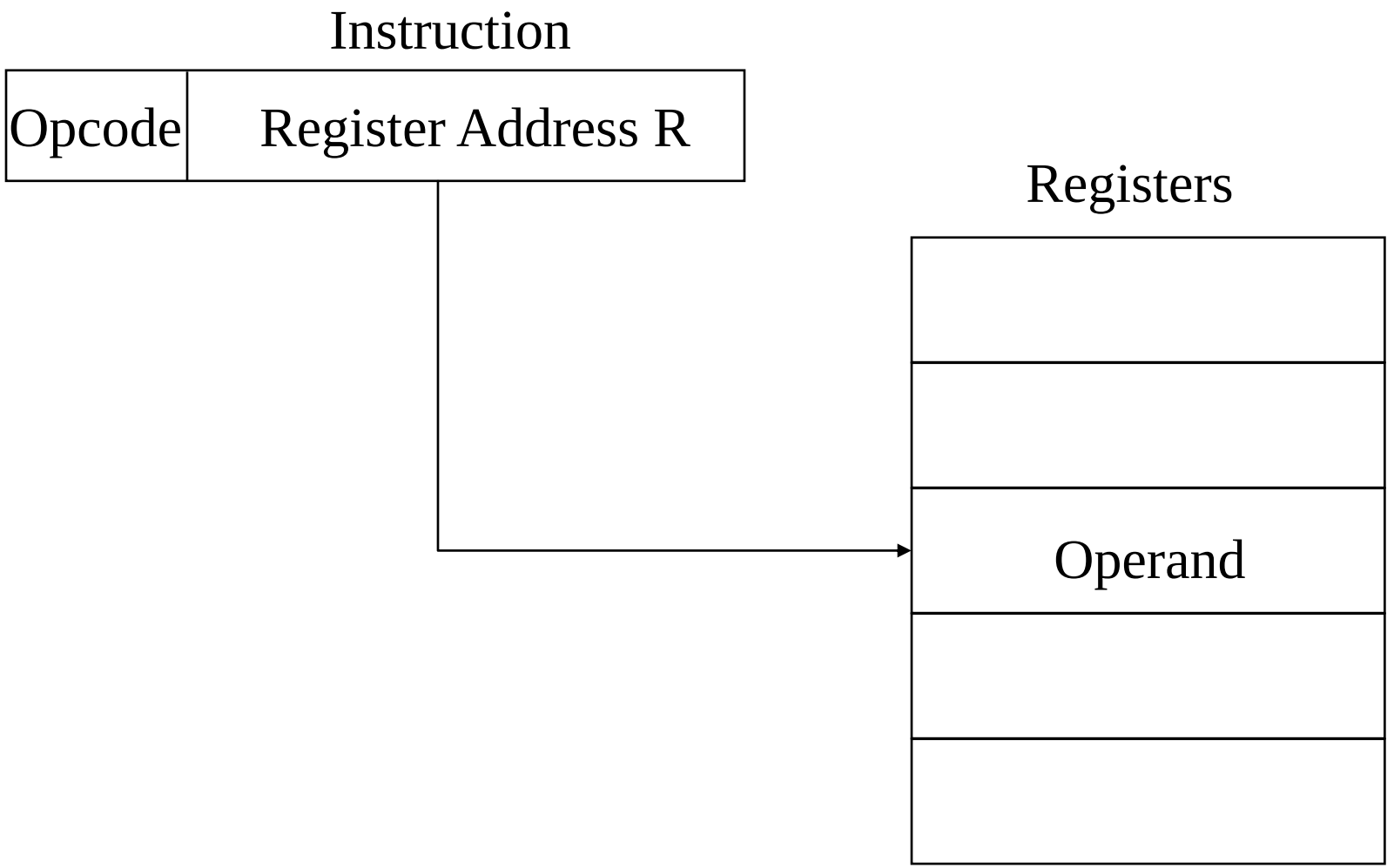
Register Addressing

- Operand is held in register named in address field
- EA = R
- MOV CL, DL MOV AX, DX

Advantages

- No memory accesses
- Very fast execution
- Very small address field needed
  - Shorter instructions (vs. memory operand)
  - Faster instruction fetch

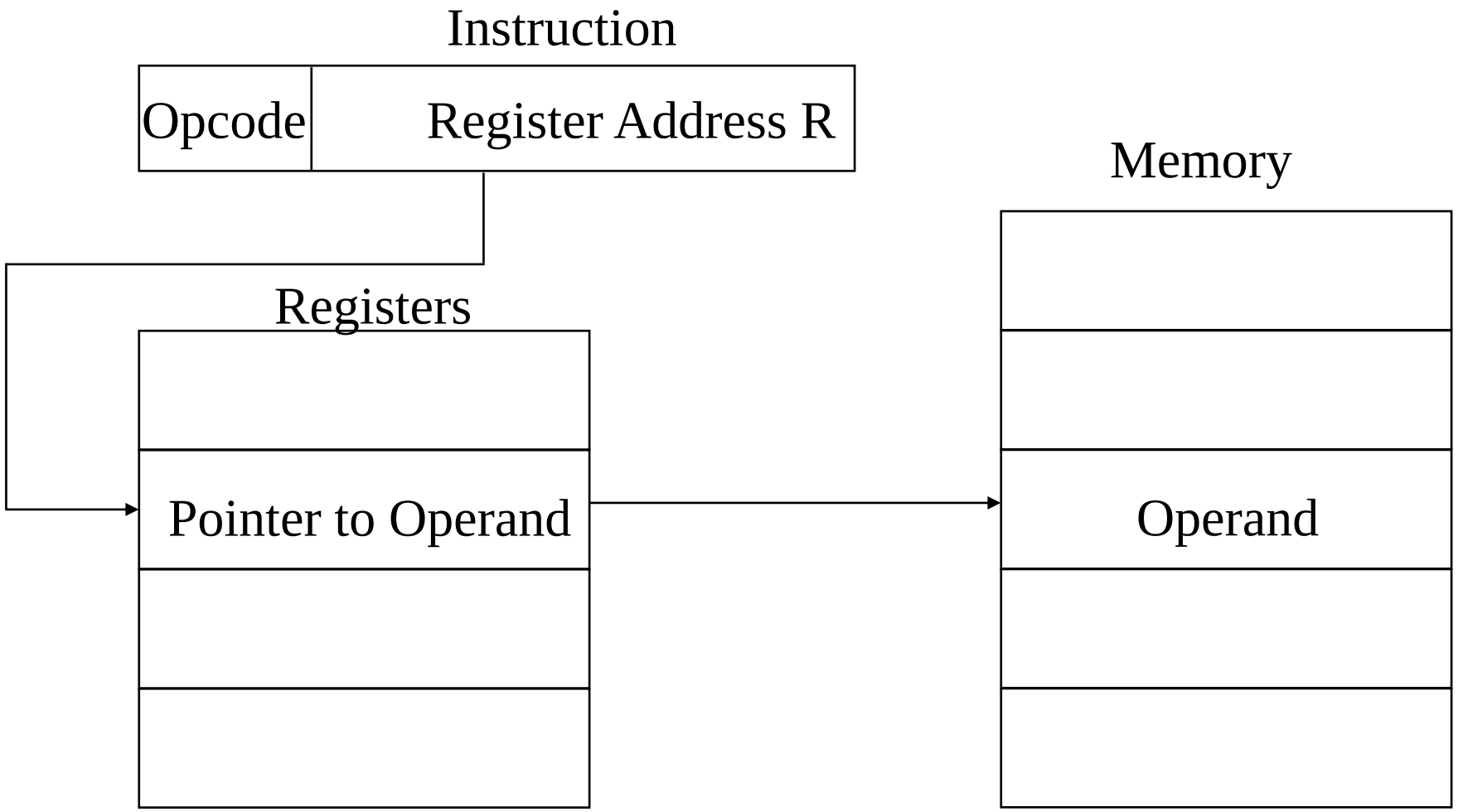
Register Addressing



Register Indirect Addressing

- Similar to memory indirect addressing
- EA = [R]
- Operand is in memory cell pointed to by contents of register R
- Address space limitation of the address field is overcome by having that field refer to a word-length location containing an address
- Uses one less memory reference than memory indirect addressing
  - Large address space (2^n)
- One fewer memory access than indirect addressing
- Base registers BP,BX or index register SI, DI holds offset address of the data byte.
- Default segments are
- BX, SI -→ Data Segment, DI → Extra segment BP → Stack Segment
- MOV CL, [BX]

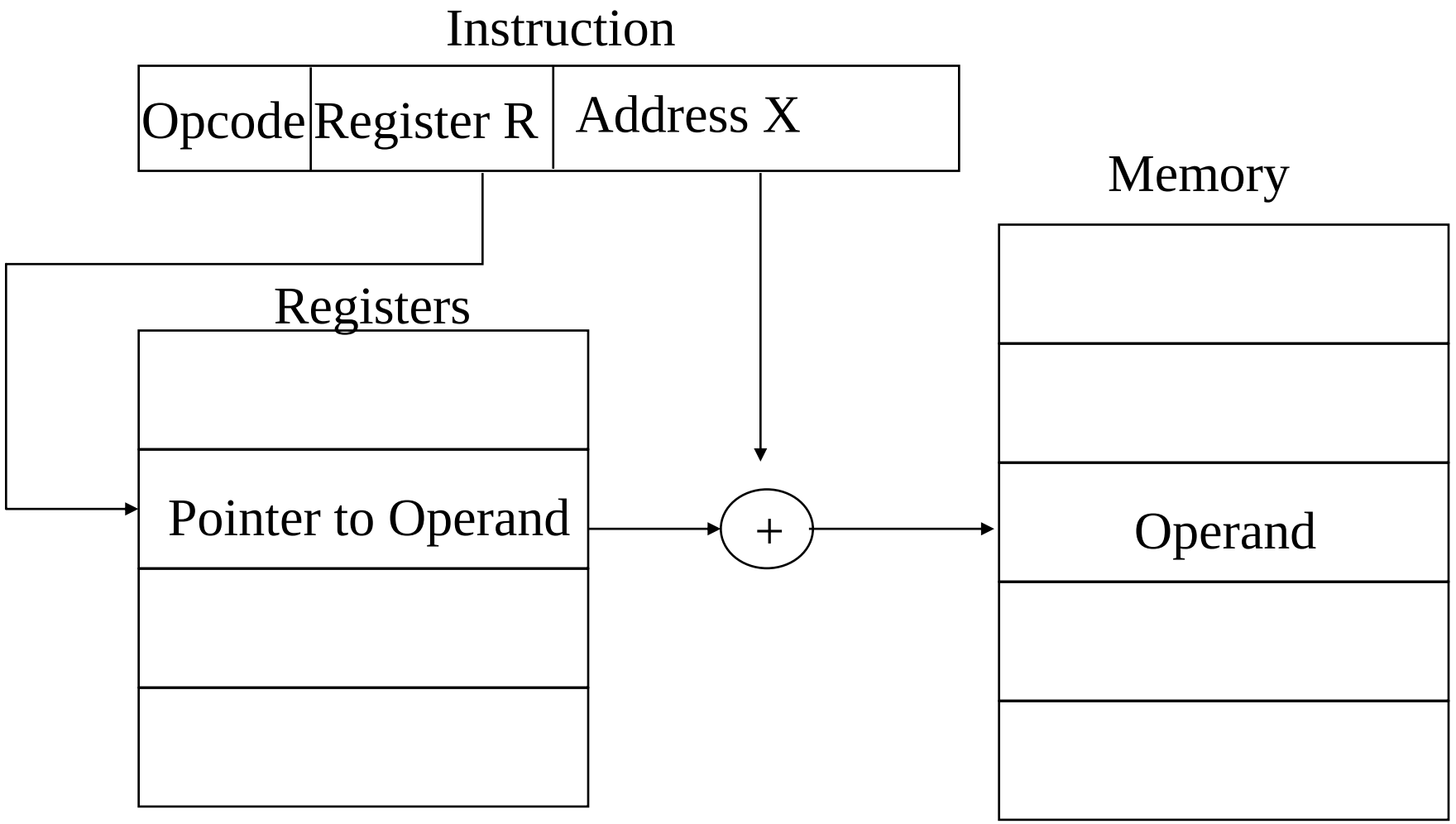
Register Indirect Addressing



Displacement Addressing

- 1. Register relative addressing
  - 8/16 Bits displacement value is added to the register (BP, BX, DI, or SI)
  - MOV AL, [BX+4] OR MOV AL, 4[BX] OR MOV AL, BX[4]
  - Displacement can be a 32-bit number and the register can be any 32-bit register except the ESP
  - Array name can be displacement (constant value→ memory address) Array[DI] → Index addressing mode ---Index changing
    - Address filed contains main memory address and referenced register contains positive displacement from that address
  - If Base register is used instead of index ---Displacement Addressing/Based Addressing –to access structure elements (here constant not memory address)
    - Referenced register contains main memory address and address contains displacement from that address
  - Third form is PC-Relative Addressing
    - Referenced register is PC

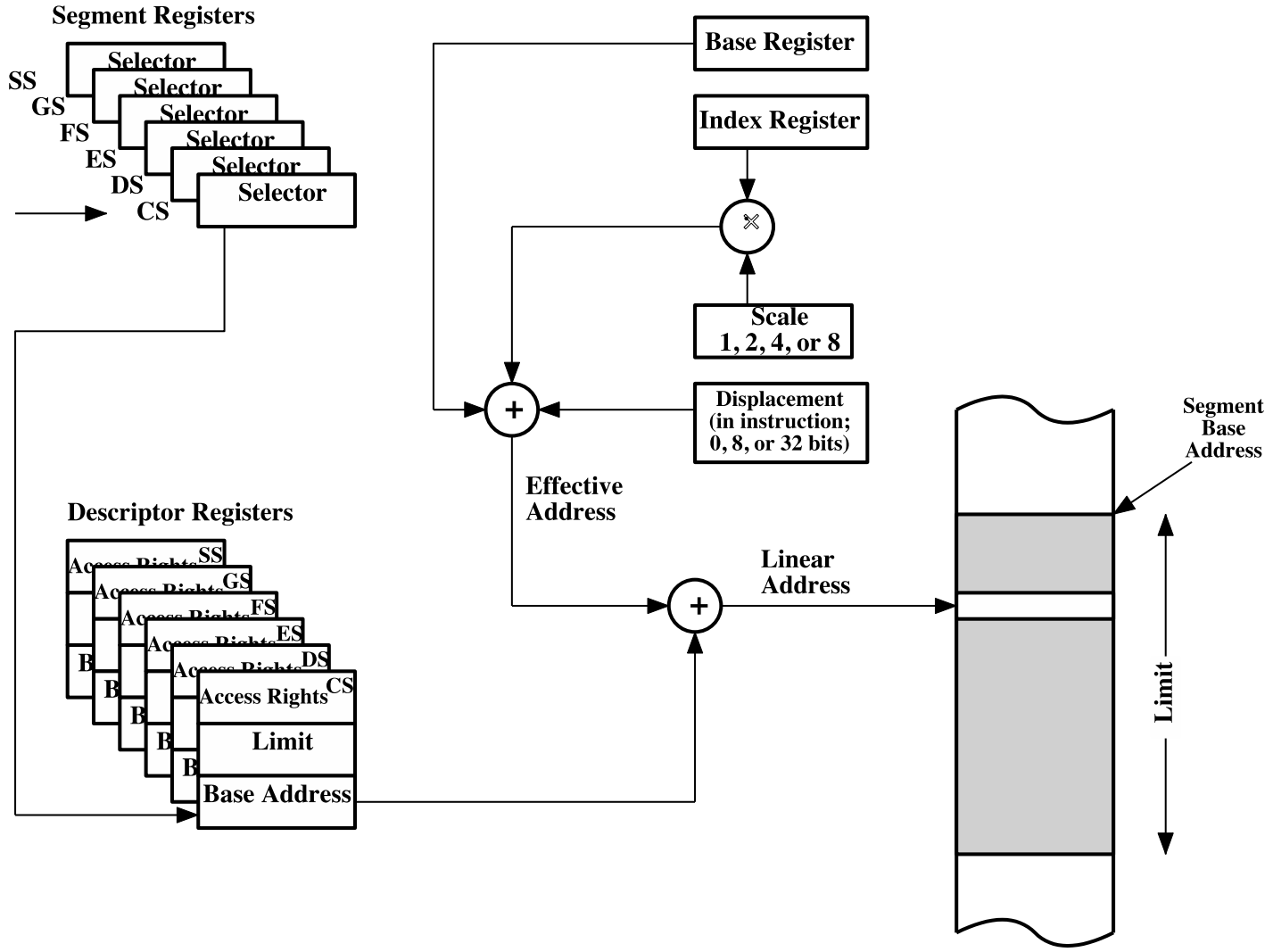
Displacement Addressing



Stack Addressing →PUSH X

- A stack is a linear array of locations
  - Sometimes referred to as a *pushdown list* or *last-in-first-out queue*
- A stack is a reserved block of locations
  - Items are appended to the top of the stack so that the block is partially filled
- Associated with the stack is a pointer whose value is the address of the top of the stack
  - The stack pointer is maintained in a register
  - Thus references to stack locations in memory are in fact register indirect addresses
- Is a form of implied addressing
- The machine instructions need not include a memory reference but implicitly operate on the top of the stack

x86 Addressing Mode Calculation



x86 Addressing Modes

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

LA = linear address
(X) = contents of X
SR = segment register
PC = program counter
A = contents of an address field in the instruction

R = register
B = base register
I = index register
S = scaling factor