Computer Organization

Instruction Set Characteristics, Instruction Formats, Addressing Modes, RTL & Micro-Operations, CISC, RISC.

Chapters (10 + 11 + Mano Ch.4 + 13)

Typical Instructions

Data Movement	Load (from memory) Store (to memory) memory-to-memory move register-to-register move input (from I/O device) output (to I/O device) push, pop (to/from stack)
Arithmetic	Data Types: (signed & unsigned) Integer (binary + decimal) (signed & unsigned) Floating Point Numbers Operations: Add, Subtract, Multiply, Divide
Logical	Not, and, or, set, clear
Shift	Arithmetic (& Logical) shift (left/right), rotate (left/right)
Control (Jump/Branch)	unconditional, conditional
Subroutine Linkage	call, return
Interrupt	trap, return
Synchronisation	test & set (atomic r-m-w)
String	search, compare, translate

Interpret the Assembly code:

```
        mov a1, 10
        mov t0, 0
loop:
        BREQ a1, 0, finish
        add t0, a0
        sub a1, 1
        BR loop
finish:
        add t0, t0, 100
        add v0, t0, 0
```

Convert the C code to Assembly

```
a = 8
b = 2
c = 4
r0 = 0
for ( i = 0; i < a; i++){
   r0 += i*b + c
}
```

Addressing Modes

Chapter 11

Types of Operand

- Addresses
- Numbers
  - Integer/floating point
- Characters
  - ASCII etc.
- Logical Data
  - Bits or flags

Memory Locations and Operations

- The (main) memory can be modeled as an array of millions of adjacent cells, each capable of storing a binary digit (bit), having value of 1 or 0.
- These cells are organized in the form of groups of fixed number, say n, of cells that can be dealt with as an atomic entity. An entity consisting of 8-bit is called a byte.
- The entity consisting of n-bit that can be stored and retrieved in and out of the memory using one basic memory operation is called a word.

Memory Locations and Operations

- In order to be able to move a word in and out of the memory, a distinct address has to be assigned to each word.
- This address will be used to determine the location in the memory in which a given word is to be stored. This is called a memory *write operation.*
- Similarly, the address will be used to determine the memory location from which a word is to be retrieved from the memory. This is called a memory *read* operation.

Registers and Operations

- CPU must have some working space (temporary storage) Called registers
- Number and function vary between processor designs
- Top level of memory hierarchy
- User Visible Registers
  - Data register
  - Address register
- In order to be able to move a word in and out of the Register, a distinct address or register number has to be assigned.
- This address will be used to determine the specific register in which a given word is to be stored or read.

Addressing Modes

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

Immediate Addressing

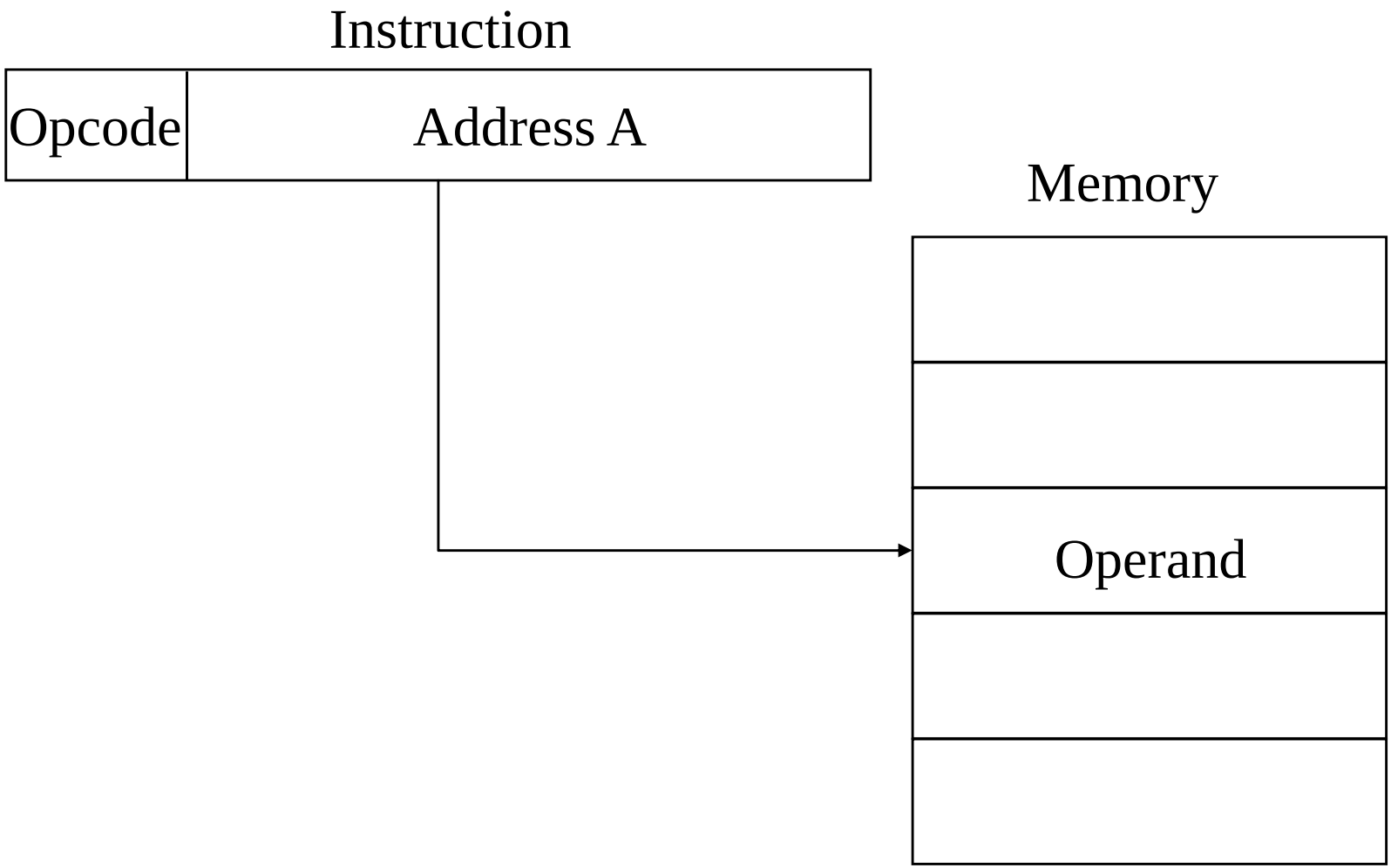
- Operand is part of instruction
- Operand = address field
- e.g. ADD 5
  - Add 5 to contents of accumulator
  - 5 is operand

- No memory reference to fetch data
- Fast
- Limited range
- The use of immediate addressing leads to poor programming practice. This is because a change in the value of an operand requires a change in every instruction that uses the immediate value of such operand.

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

Direct Addressing Diagram



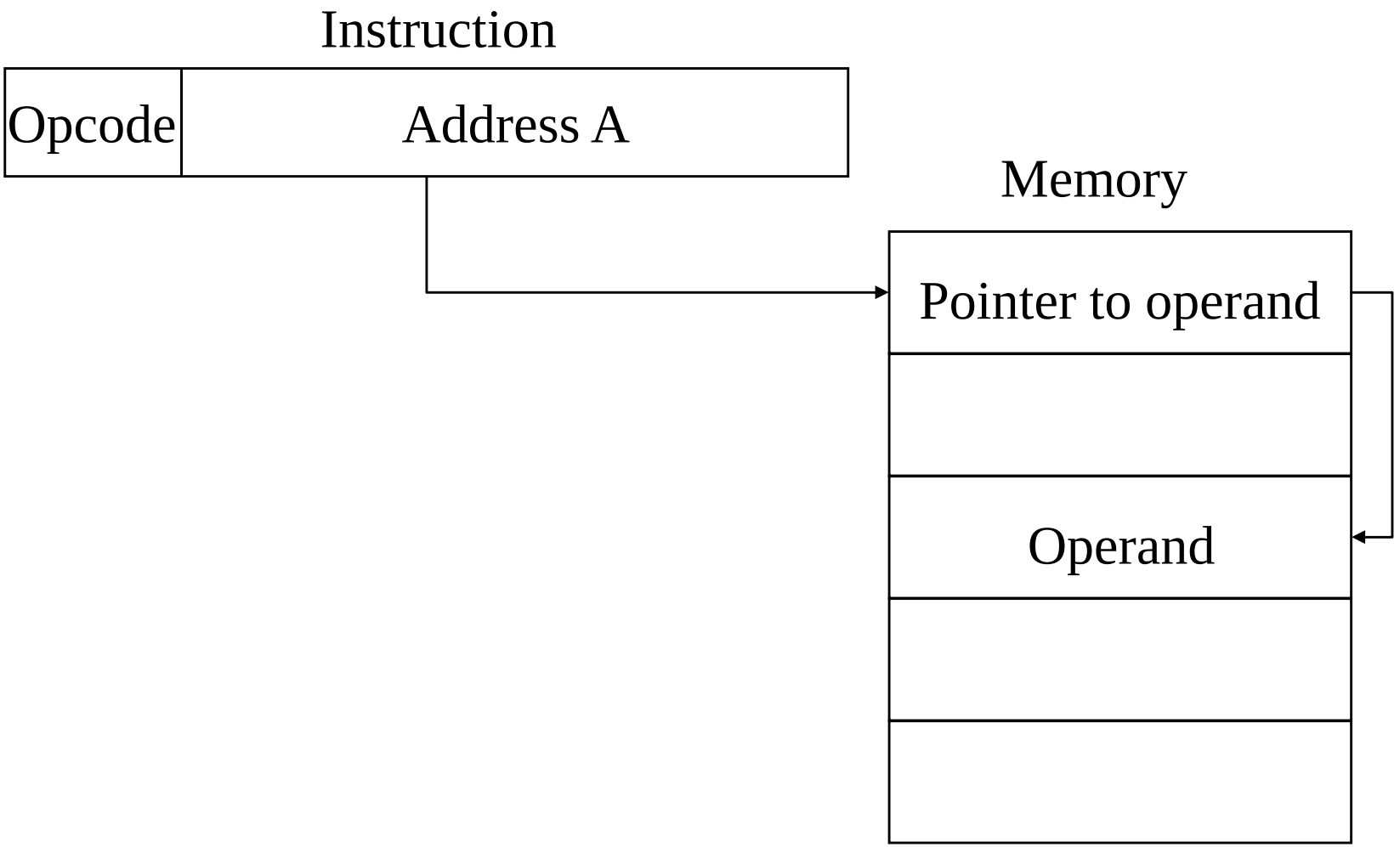
Indirect Addressing (1)

- 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

Indirect Addressing (2)

- Large address space
- 2^n where n = word length
- May be nested, multilevel, cascaded
  - —e.g. EA = (((A)))
    - – Draw the diagram yourself
- Multiple memory accesses to find operand
- Hence slower

Indirect Addressing Diagram



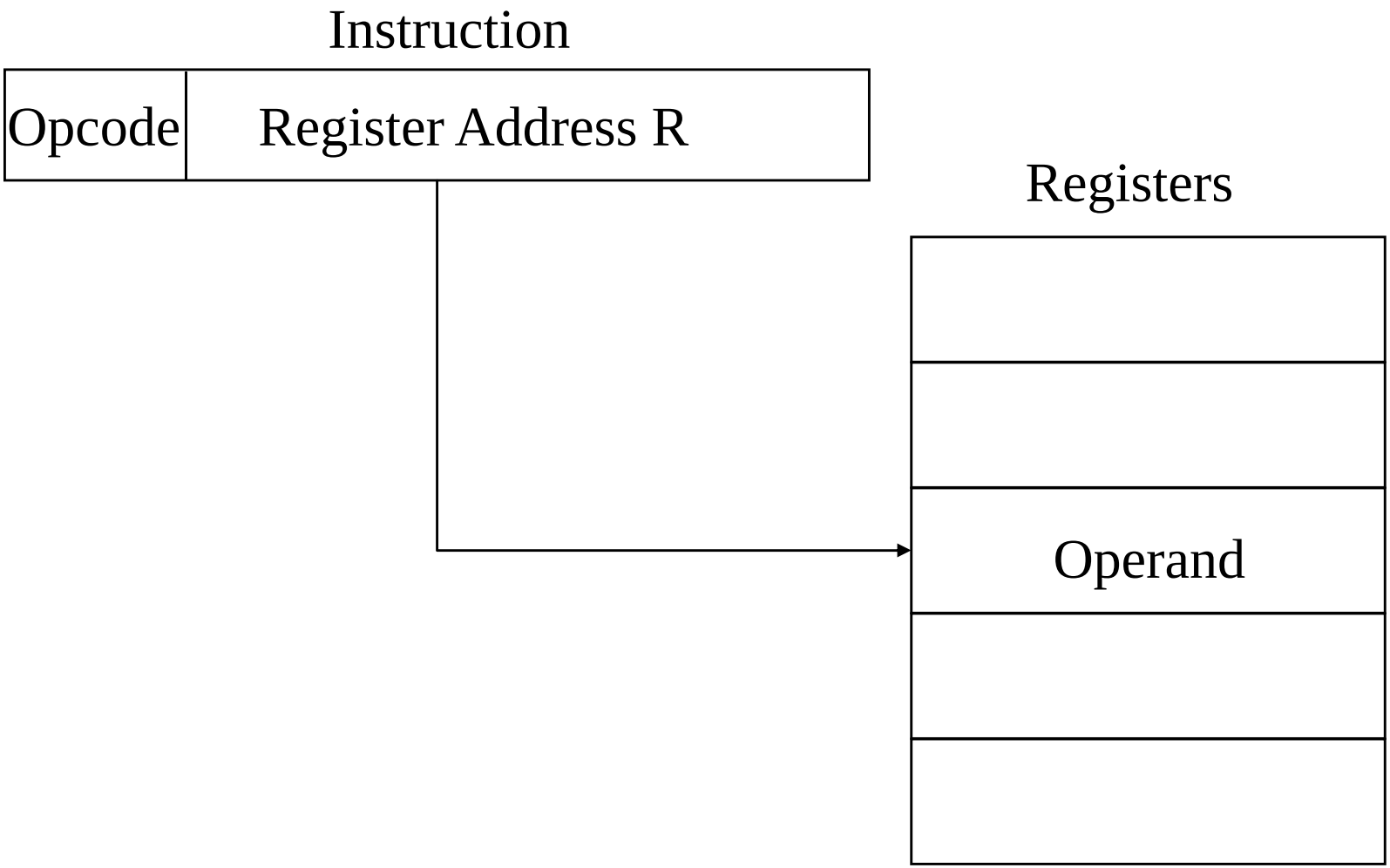
Register Addressing (1)

- 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

Register Addressing (2)

- No memory access
- Very fast execution
- Very limited address space
- Multiple registers helps performance
  - Requires good assembly programming or compiler writing
  - C programming
    - register int a;
- Direct addressing

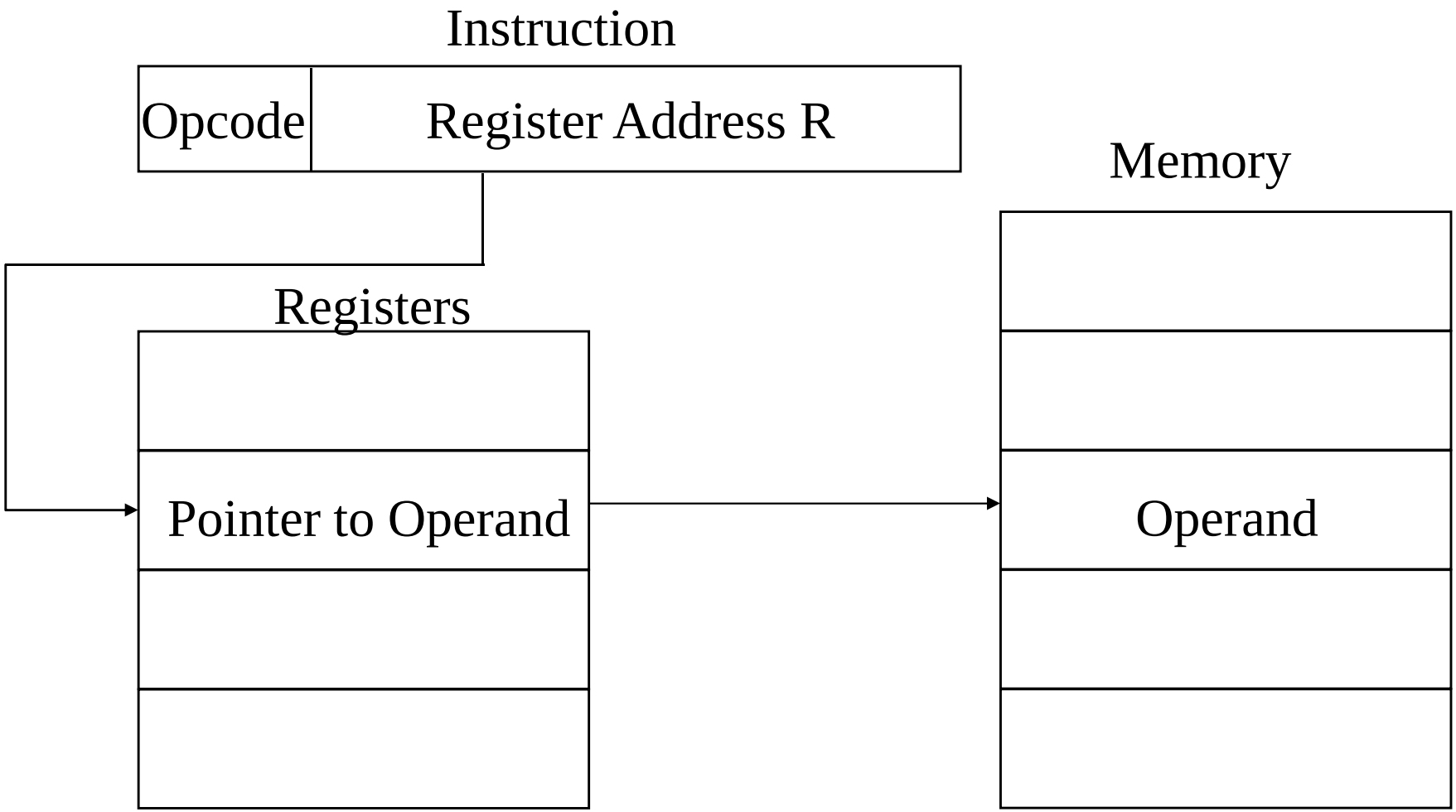
Register Addressing Diagram



Register Indirect Addressing

- C.f. 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

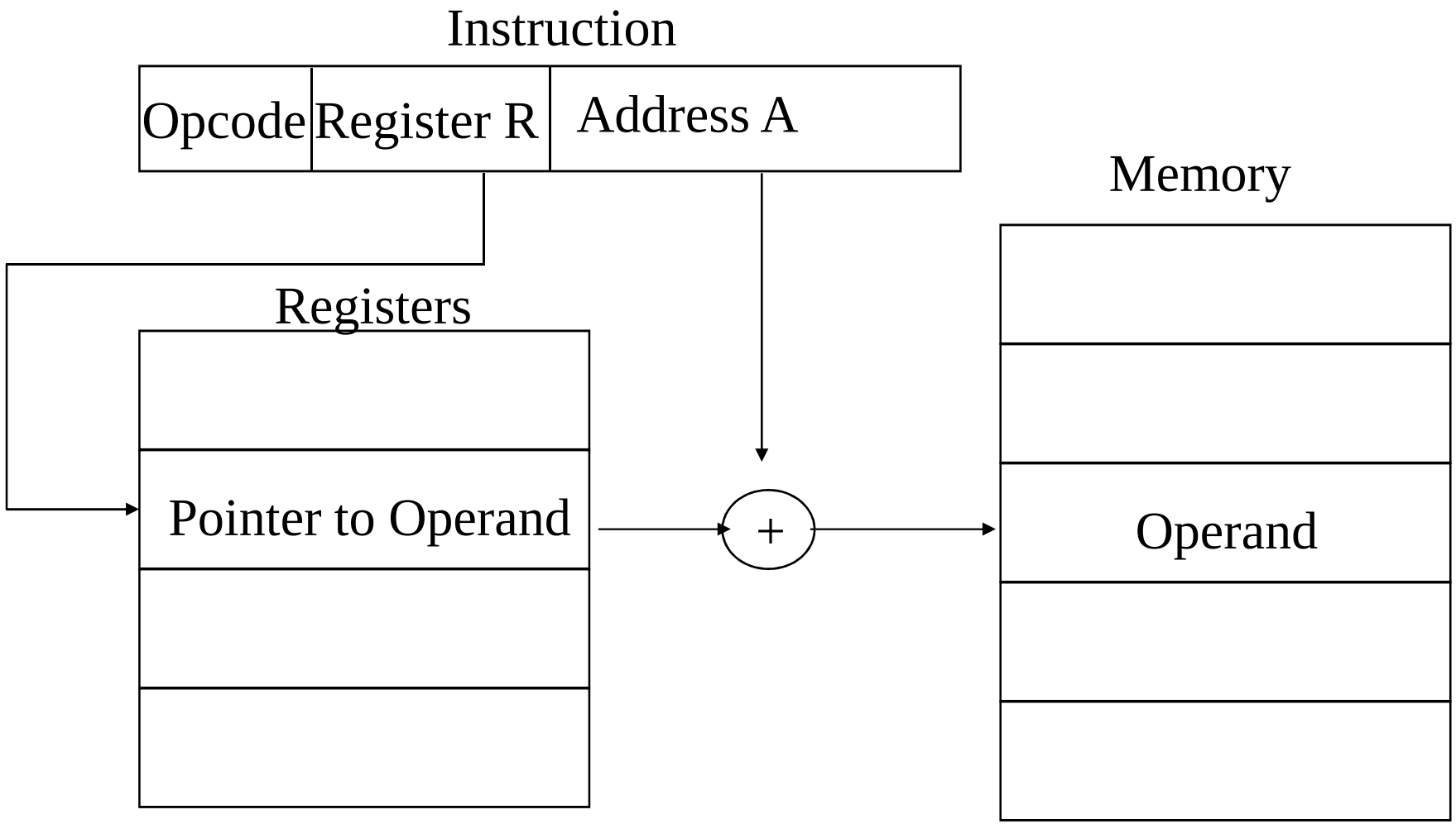
Register Indirect Addressing Diagram



Displacement Addressing

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

Displacement Addressing Diagram



Relative Addressing

- 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
- locality of reference & cache usage

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++

Combinations

- Postindex
- EA = (A) + (R)

- Preindex
- EA = (A+(R))

Stack Addressing

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