



Addressing Modes



Addressing Modes

- The way the operands are chosen during program execution is dependent on the addressing mode of the instructions.
- The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.



Addressing Modes

- Computers use addressing mode techniques for the purpose of accommodating one or both of the following provisions:
 1. to give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data and program relocation.
 2. To reduce the number of bits in the addressing field of the instruction.



Addressing Modes

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

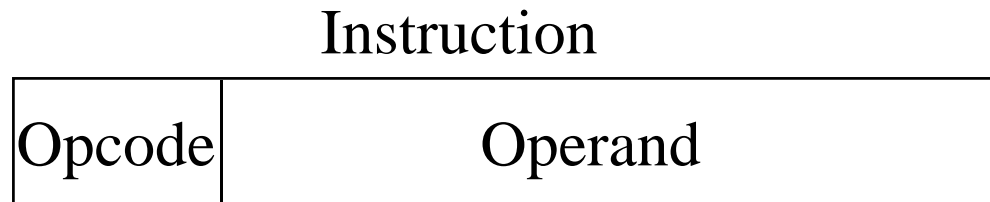


Immediate Addressing

- 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



Immediate Addressing Diagram

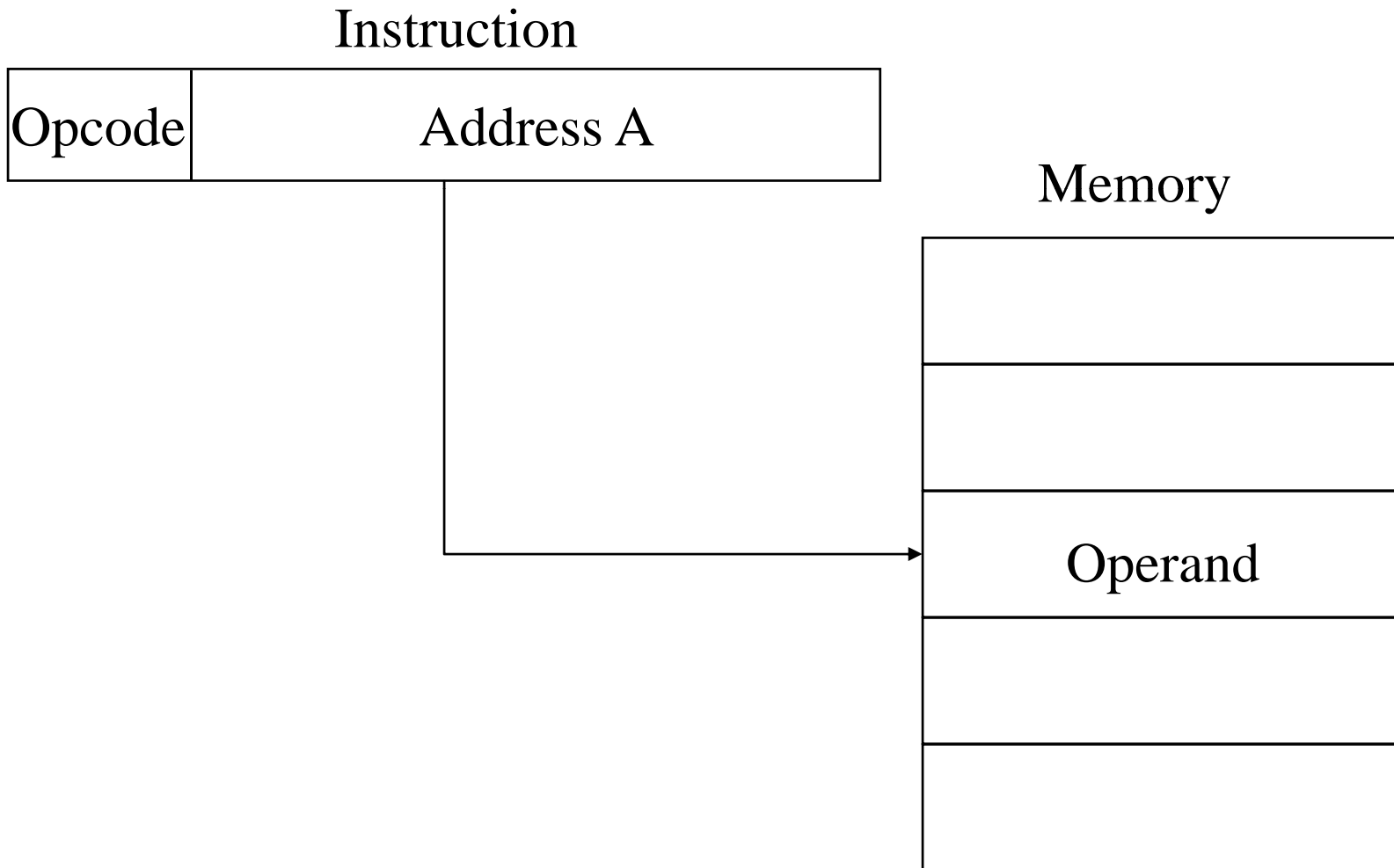




Direct Addressing

- 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)

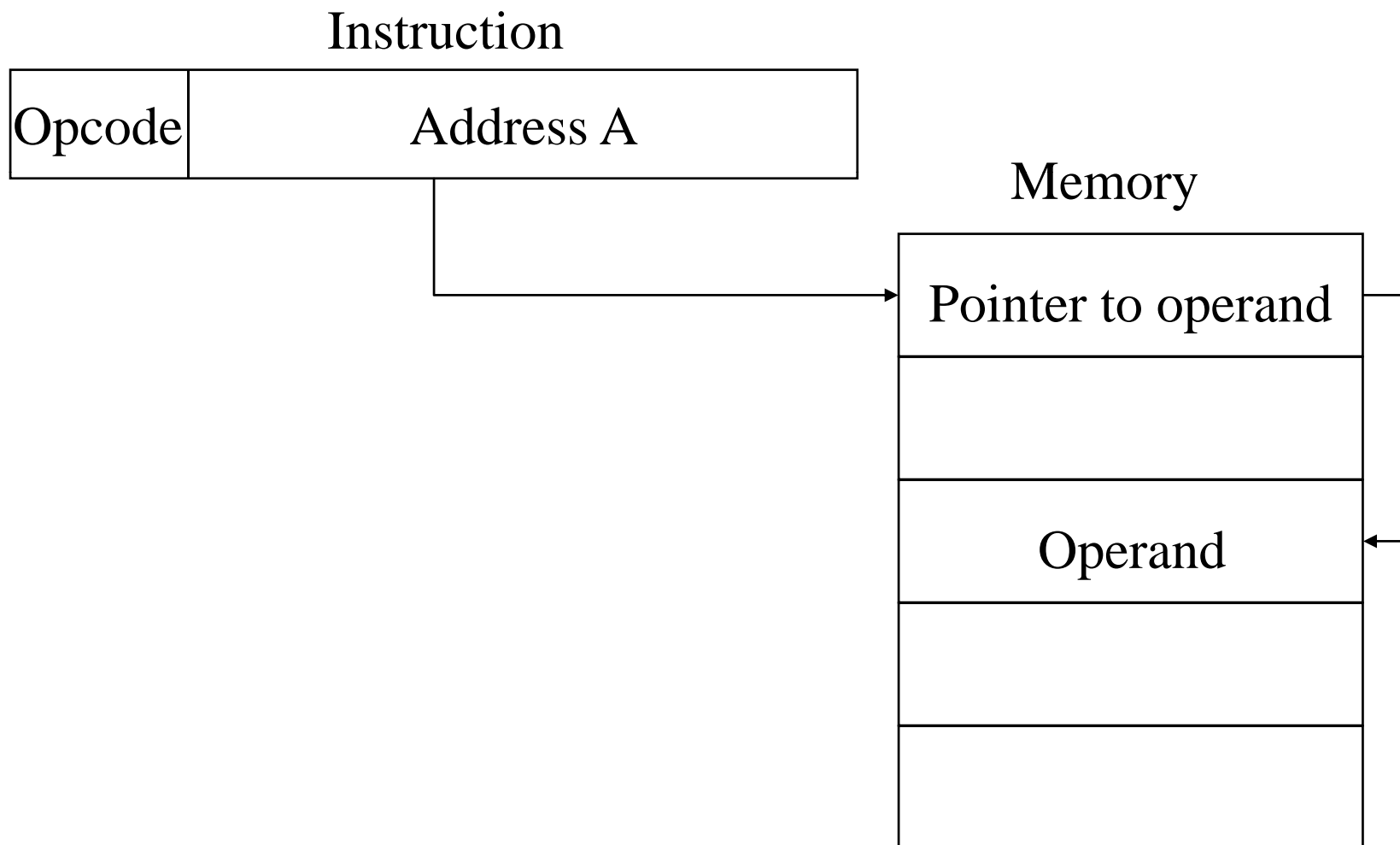
- Memory cell pointed to by address field contains the address of (pointer to) the operand
- $EA = (A)$
 - Look in A, find address (A) and look there for operand
- e.g. ADD (A)
 - Add contents of cell pointed to by contents of A to accumulator



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)

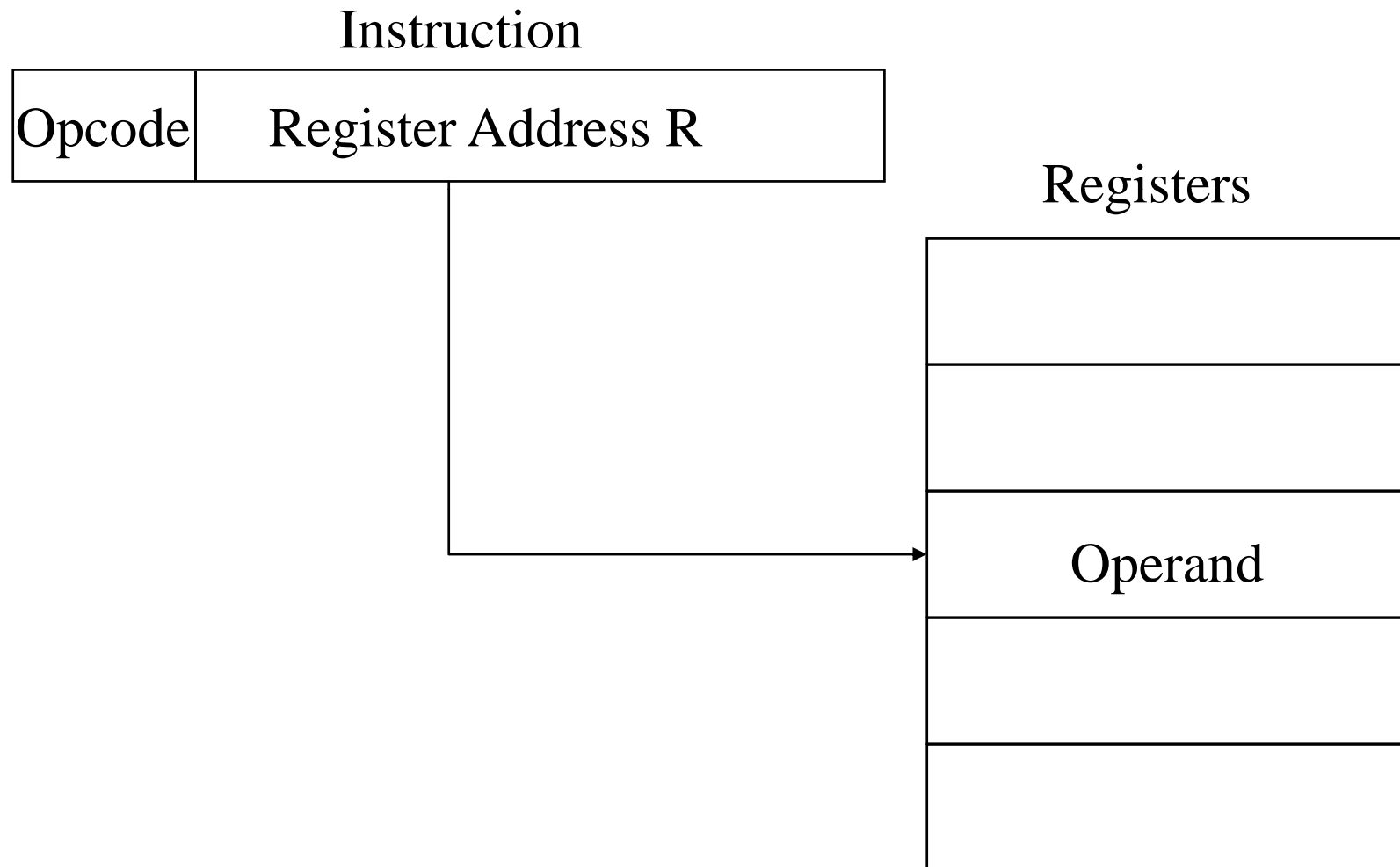
- Operand is hold in register named in address field
- $EA = R$
- Limited number of registers
- Very small address field needed
 - Shorter instructions
 - Faster instruction fetch



Register Addressing (2)

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

Register Addressing Diagram

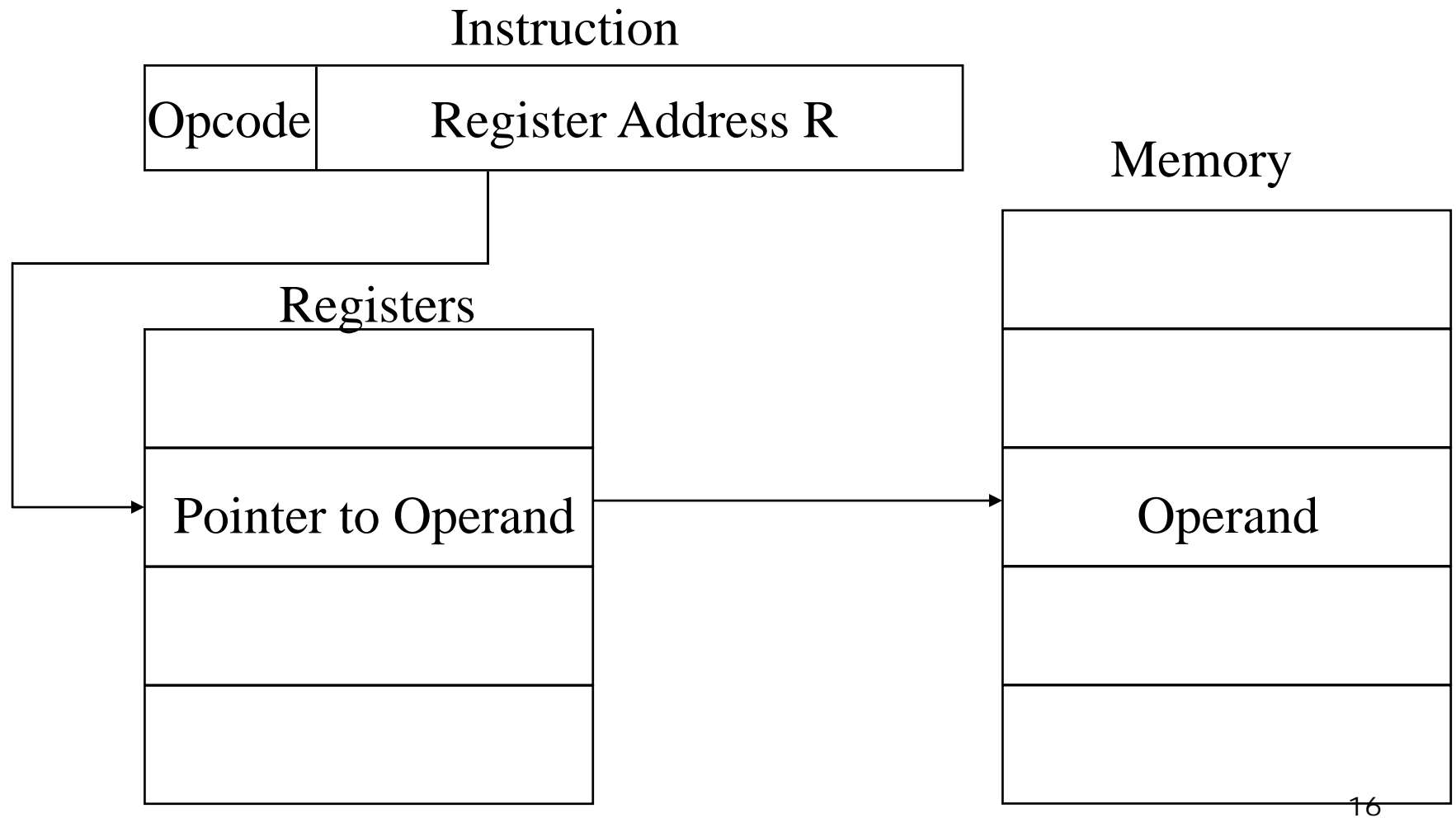




Register Indirect Addressing

- Like 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

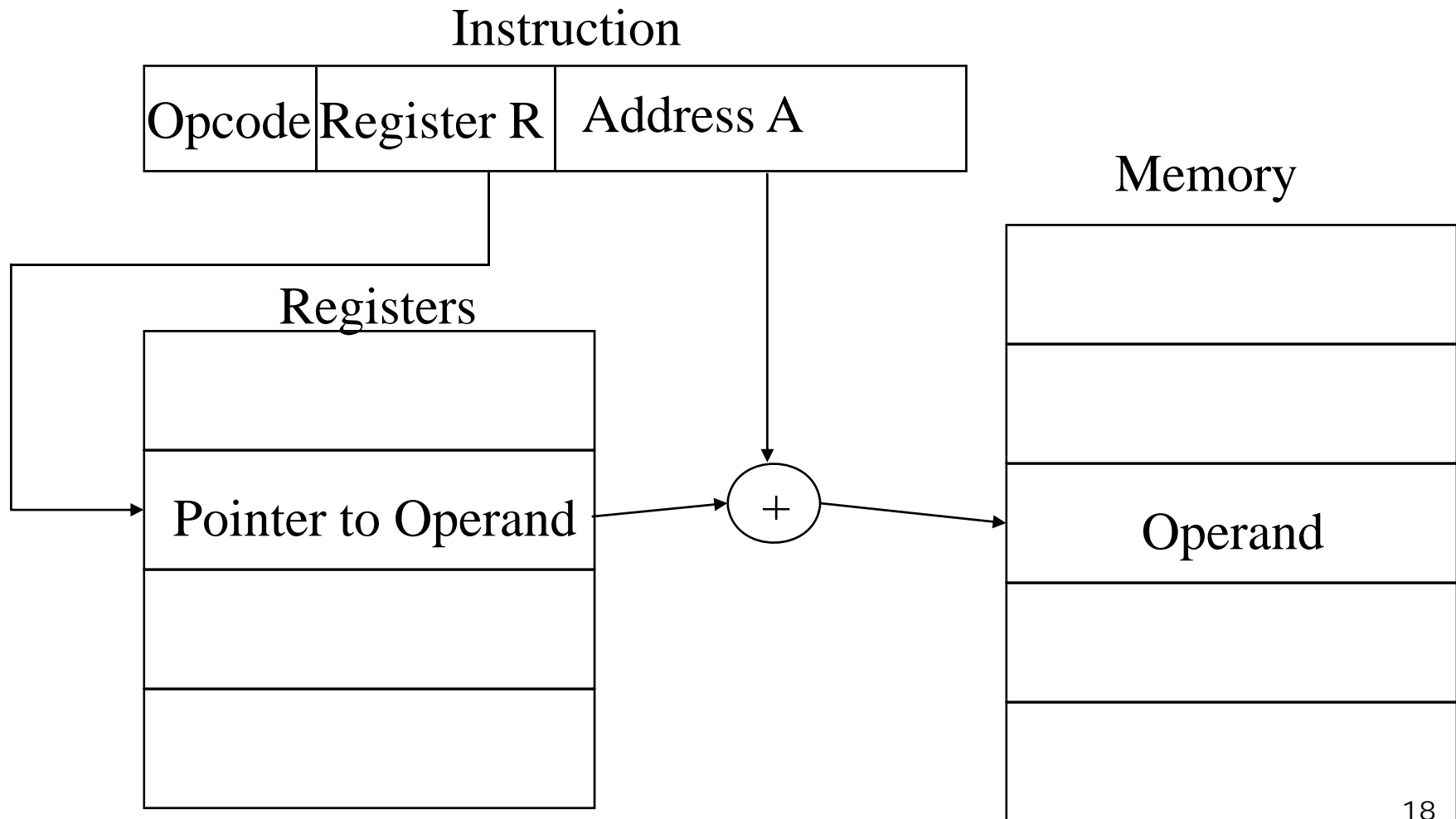




Displacement Addressing

- $EA = A + (R)$
- 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
- $R = \text{Program counter, PC}$
- $EA = A + (PC)$
- 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

- $A = \text{base}$
- $R = \text{displacement}$
- $EA = A + R$
- Good for accessing arrays
 - $EA = A + R$
 - $R++$



Combinations

- Postindex
- $EA = (A) + (R)$
- Preindex
- $EA = (A+(R))$
- (Draw the diagrams)



Stack Addressing

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