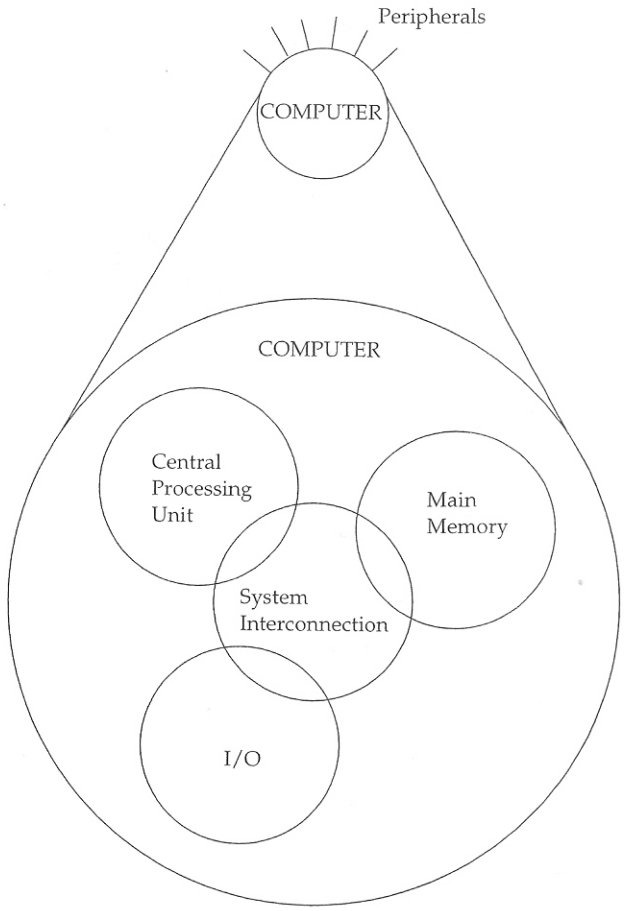
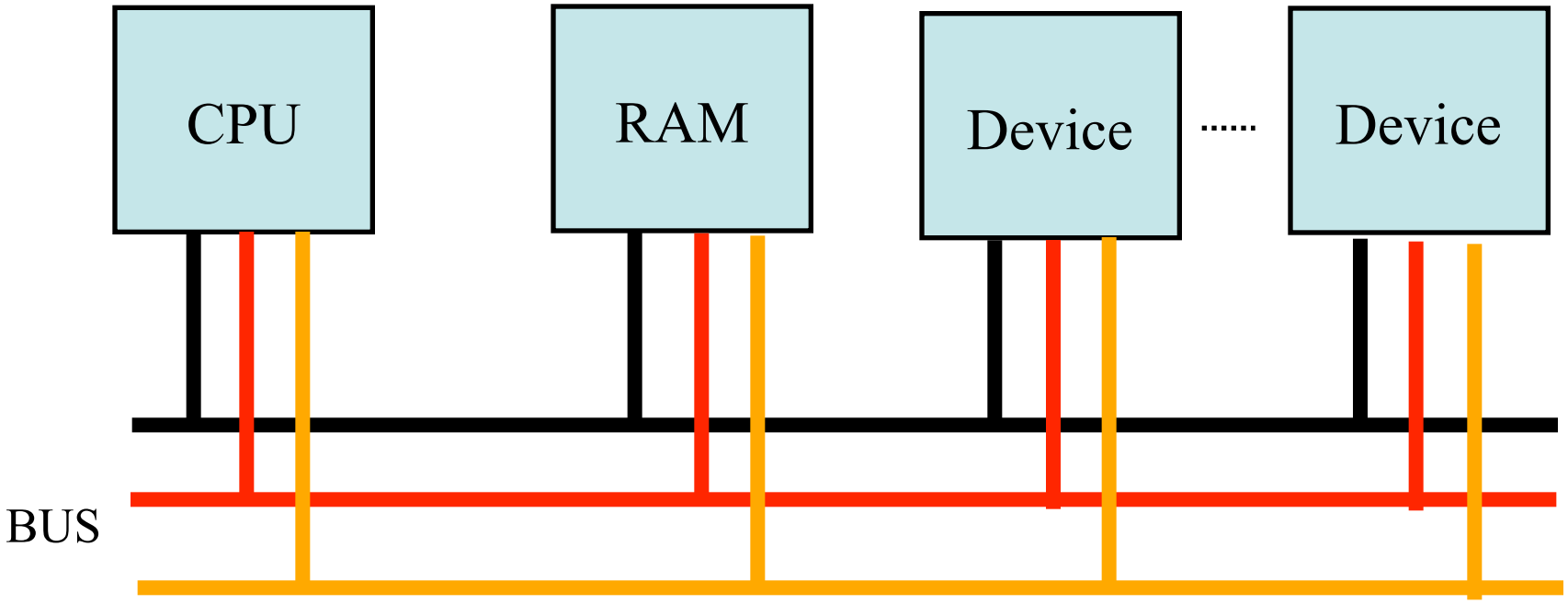
ASSEMBLY LANGUAGE PROGRAMMING

COMPUTER TOP LEVEL STRUCTURE

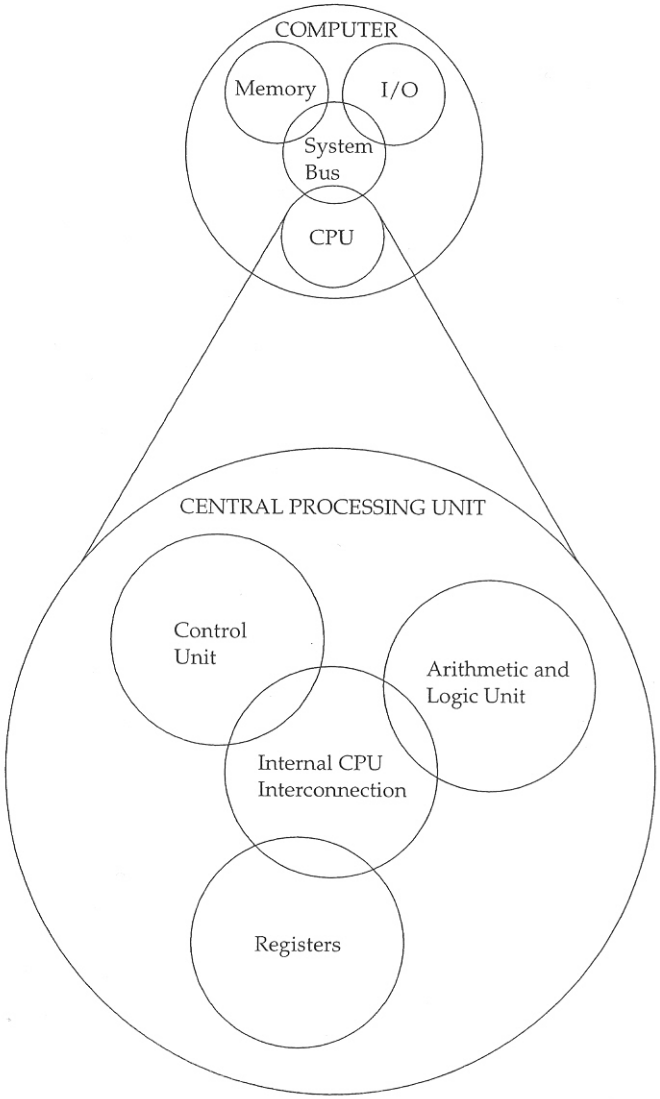


Von Neumann Architecture

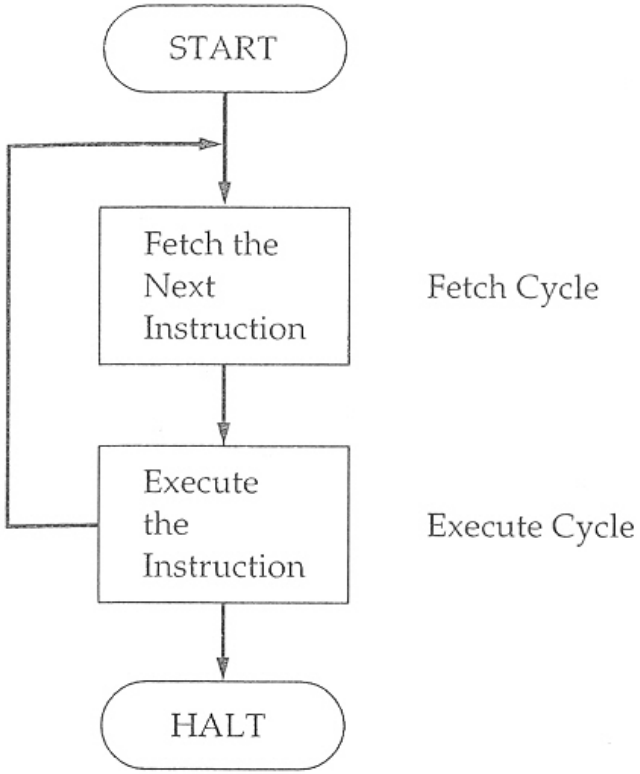


- sequential computer

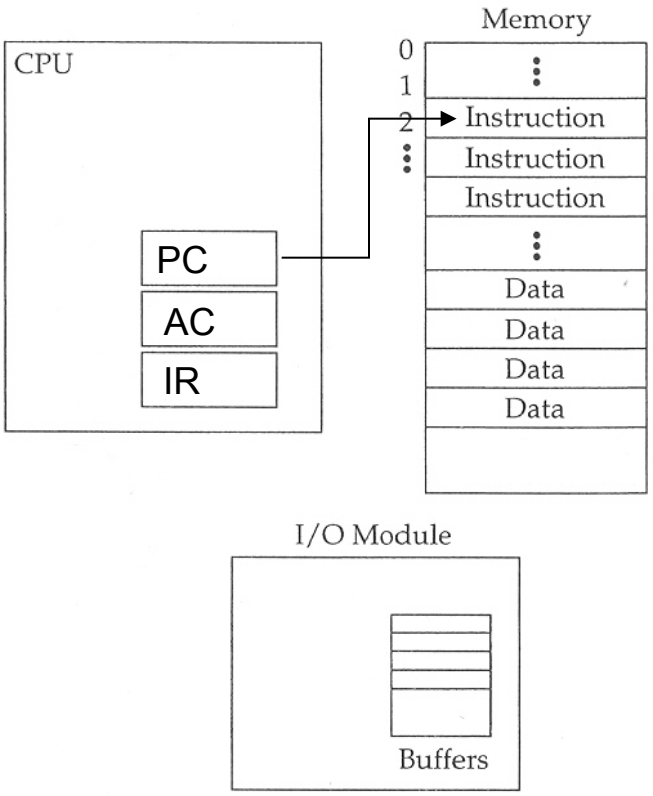
THE CPU


COMPUTER
Memory
I/O
System Bus
CPU
CENTRAL PROCESSING UNIT
Control Unit
Arithmetic and Logic Unit
Internal CPU Interconnection
Registers

BASIC INSTRUCTION CYCLE

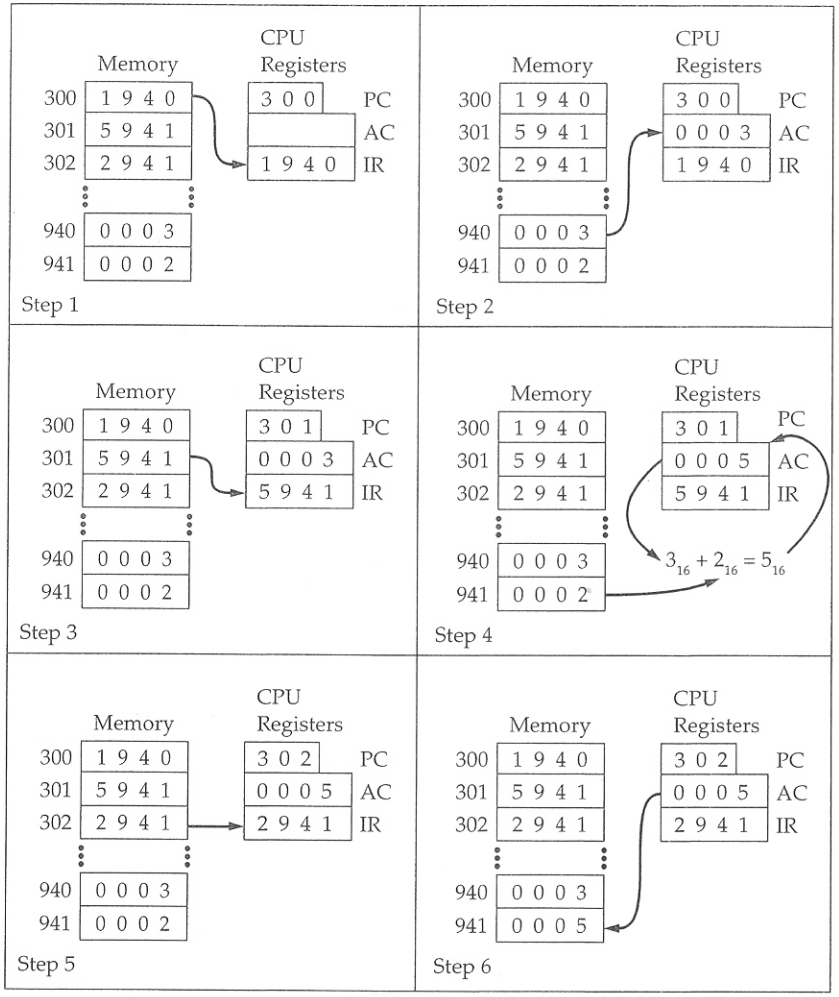


COMPUTER COMPONENTS



EXAMPLE OF PROGRAM EXECUTION

Instructions
1 Load AC from memory
2 Store AC to memory
5 Add to AC from memory



PROGRAM TO COMPUTE Y=(A-B) / (C+D*E)

Instruction		Comment
SUB	Y, A, B	$Y \leftarrow A - B$
MPY	T, D, E	$T \leftarrow D \times E$
ADD	T, T, C	$T \leftarrow T + C$
DIV	Y, Y, T	$Y \leftarrow Y \div T$

(a) Three-Address Instructions

Instruction		Comment
MOVE	Y, A	$Y \leftarrow A$
SUB	Y, B	$Y \leftarrow Y - B$
MOVE	T, D	$T \leftarrow D$
MPY	T, E	$T \leftarrow T * E$
ADD	T, C	$T \leftarrow T + C$
DIV	Y, T	$Y \leftarrow Y \div T$

(b) Two-Address Instructions

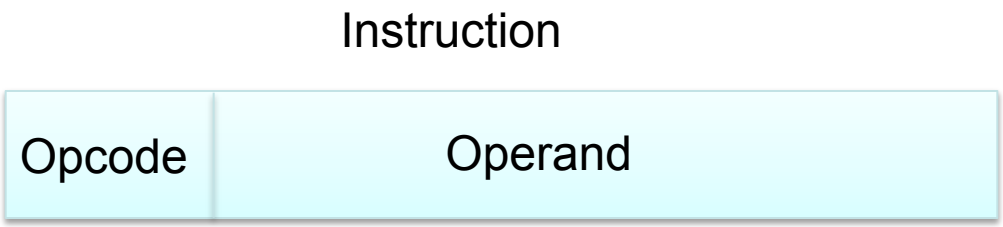
Instruction		Comment
LOAD	D	$AC \leftarrow D$
MPY	E	$AC \leftarrow AC * E$
ADD	C	$AC \leftarrow AC + C$
STOR	Y	$Y \leftarrow AC$
LOAD	A	$AC \leftarrow A$
SUB	B	$AC \leftarrow AC - B$
DIV	Y	$AC \leftarrow AC \div Y$
STOR	Y	$Y \leftarrow AC$

(c) One-Address Instructions

Addressing Modes

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

Immediate Addressing Diagram

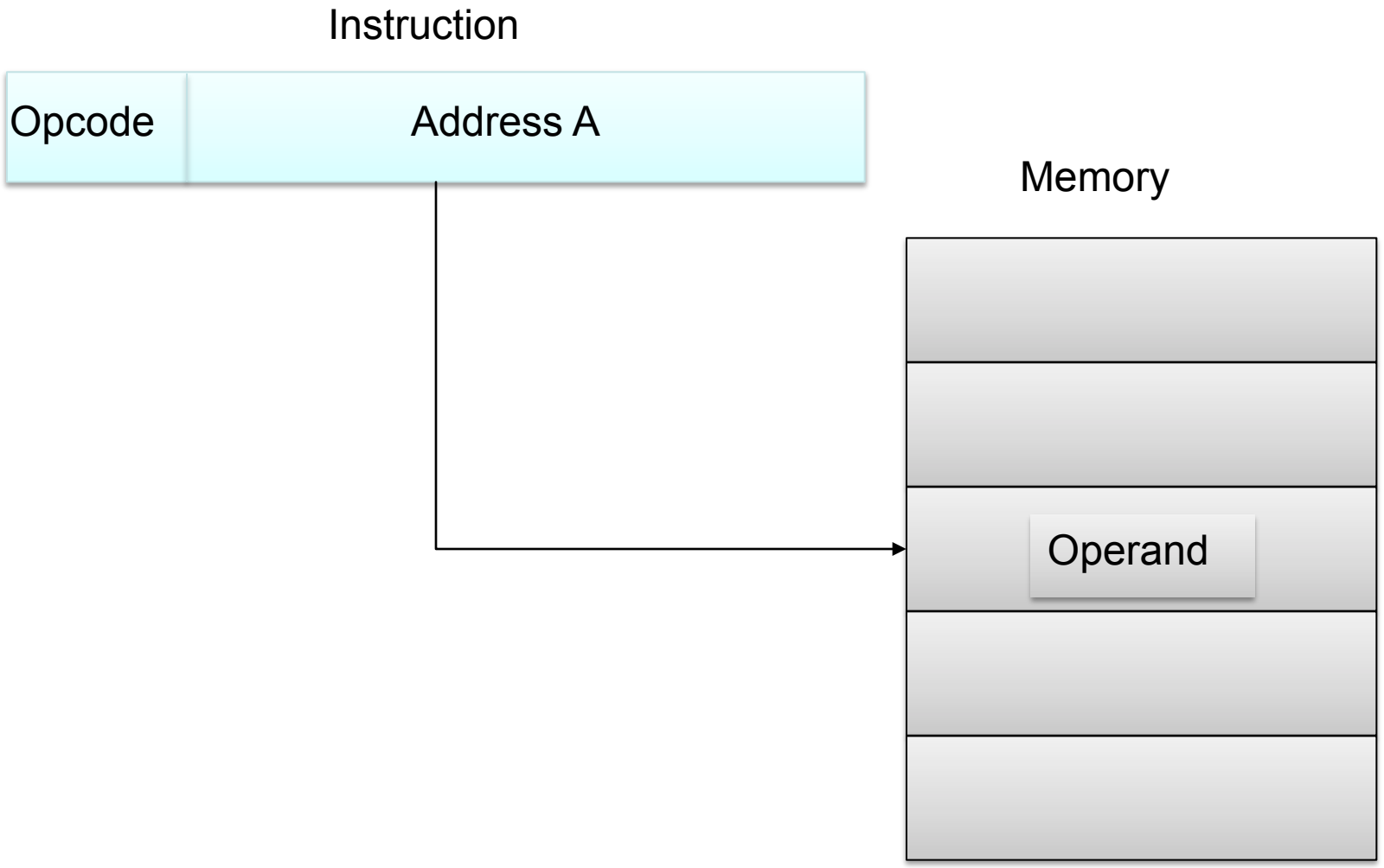


- Operand is part of instruction
- Operand = address field
- e.g. ADD 5
- No memory reference to fetch data
- Fast
- Limited range

Direct Addressing

- Address field contains address of operand
- Effective address (EA) = address field (A)
- e.g. ADD A
  - 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



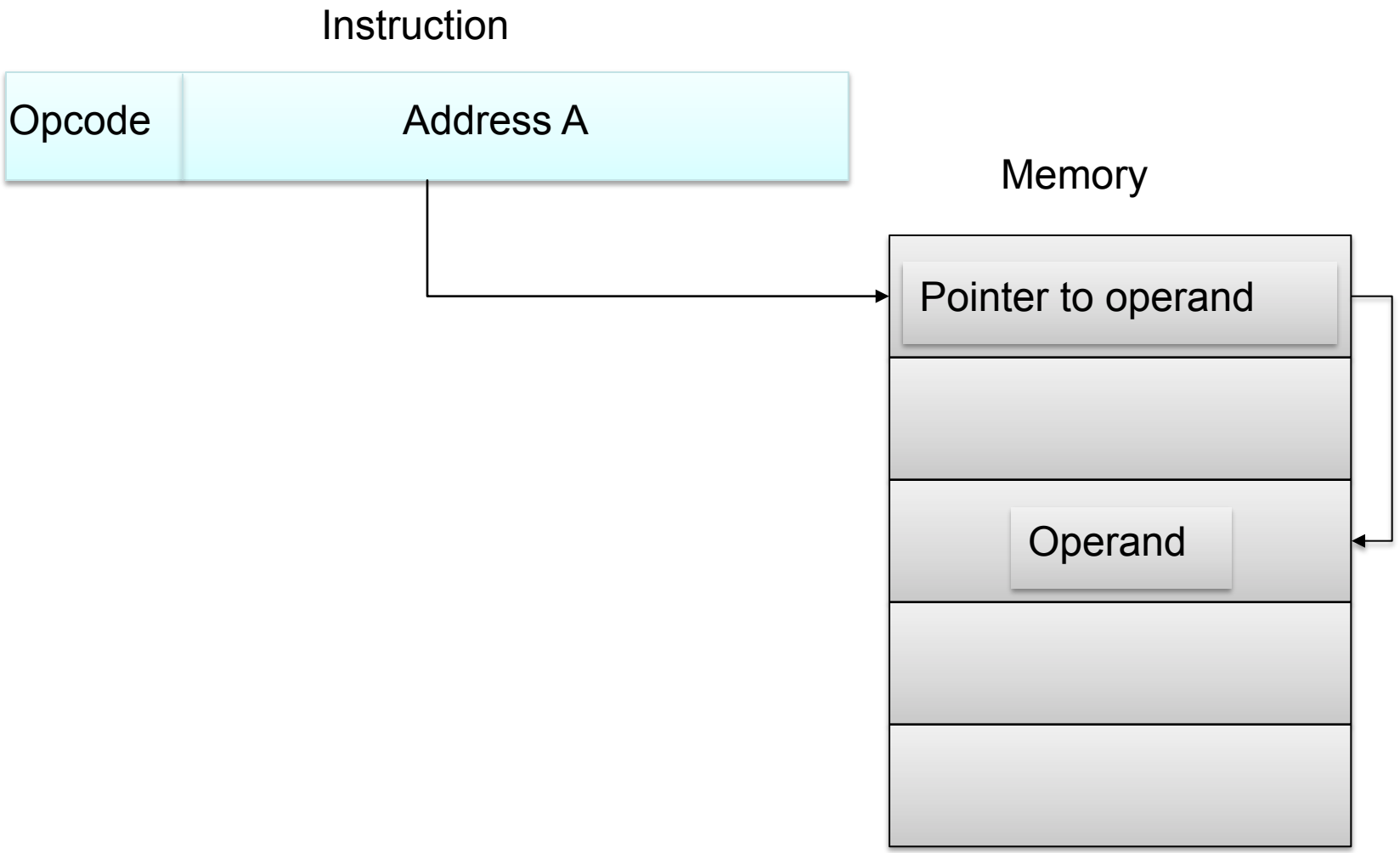
Indirect Addressing (1)

- 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

Indirect Addressing (2)

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

Indirect Addressing Diagram



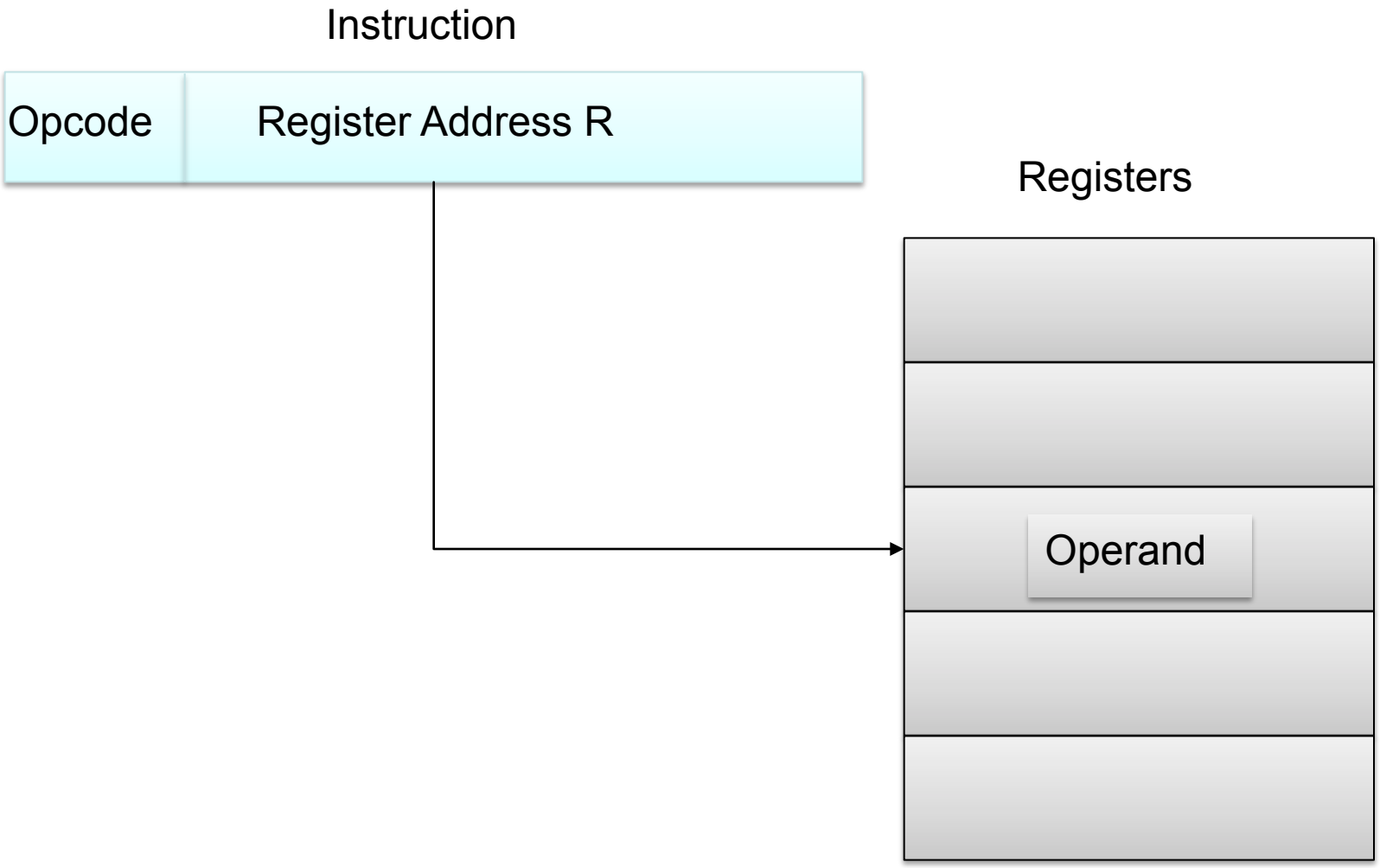
Register Addressing (1)

- 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 language (hint to the compiler)
    - register int a;
- Compare with Direct addressing

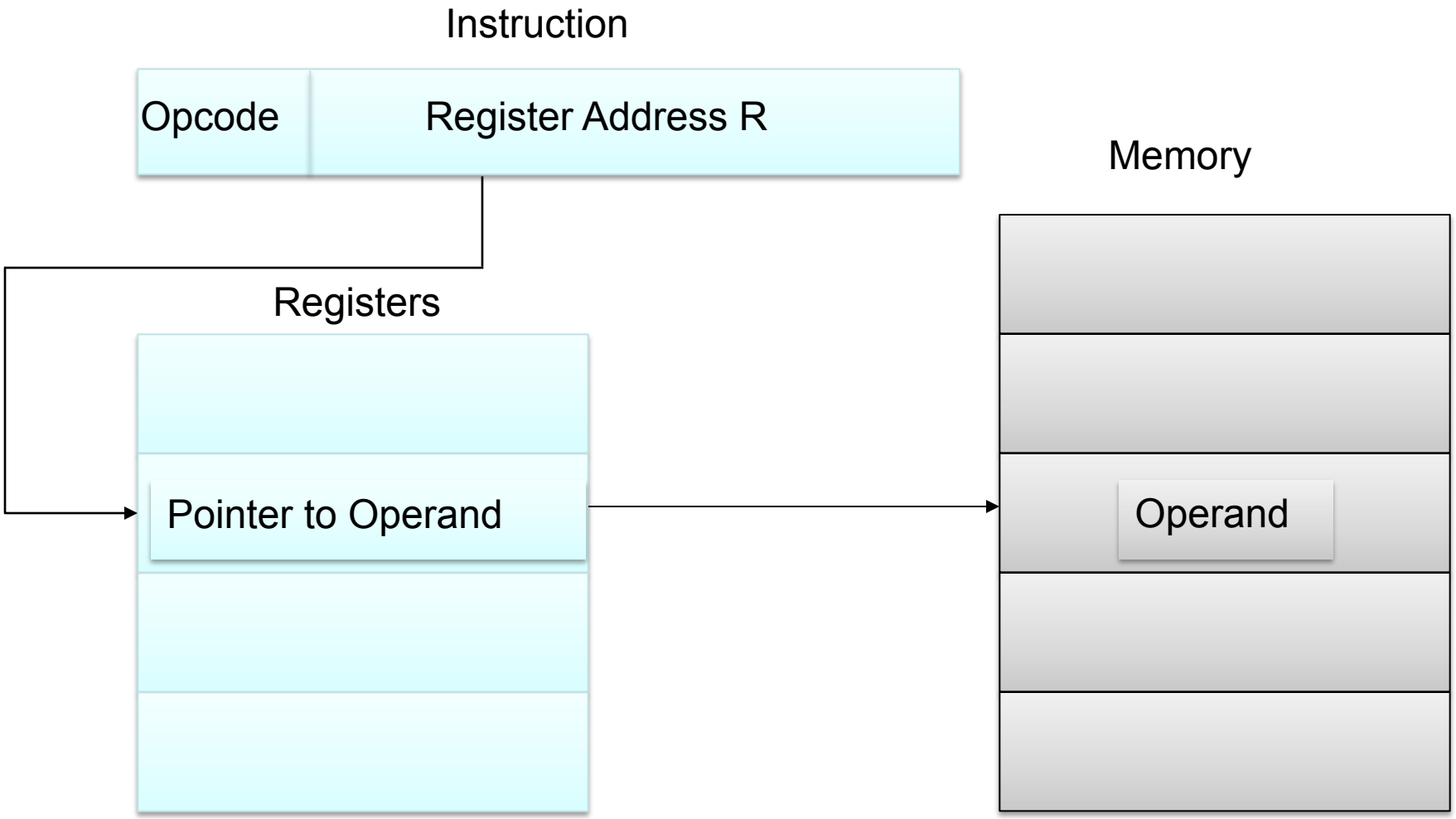
Register Addressing Diagram



Register Indirect Addressing

- Compare with 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

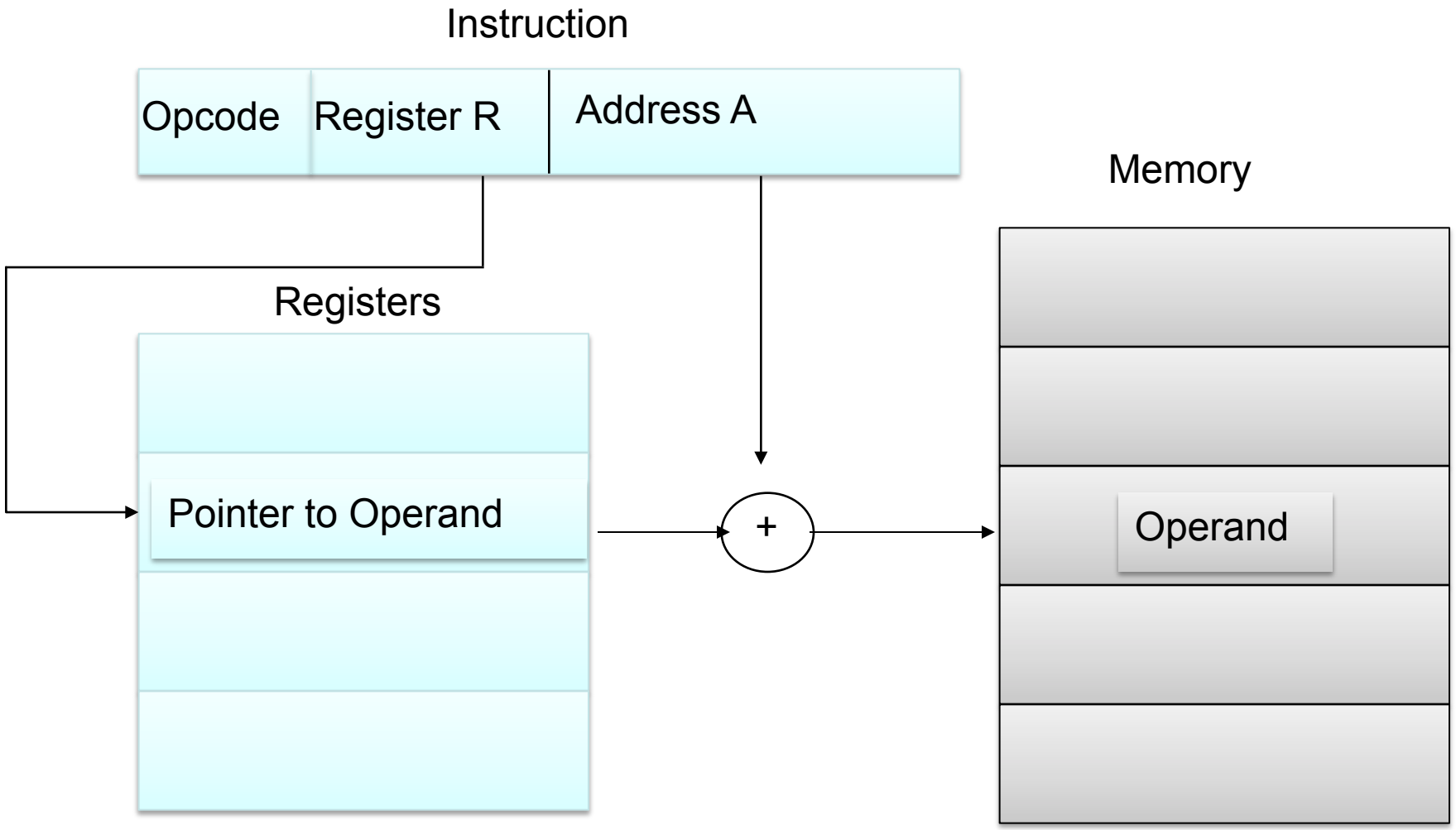
Register Indirect Addressing Diagram



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
- Compare with: locality of reference & cache usage

Base-Register Addressing

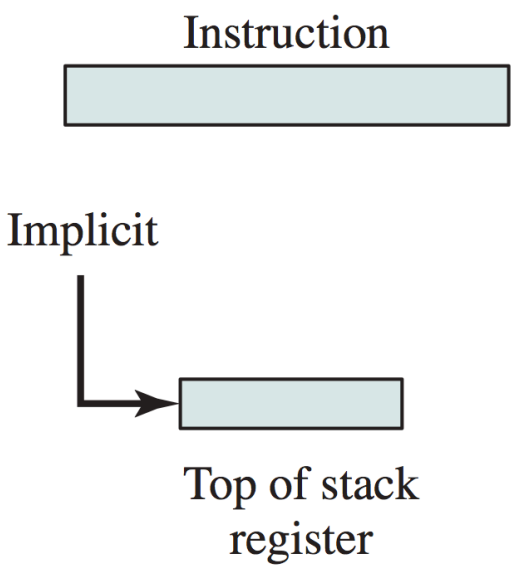
- A holds displacement
- R holds pointer to base address
- R may be explicit or implicit
- e.g. segment registers in 80x86

Indexed Addressing

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

Stack Addressing

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



Basic Addressing Modes

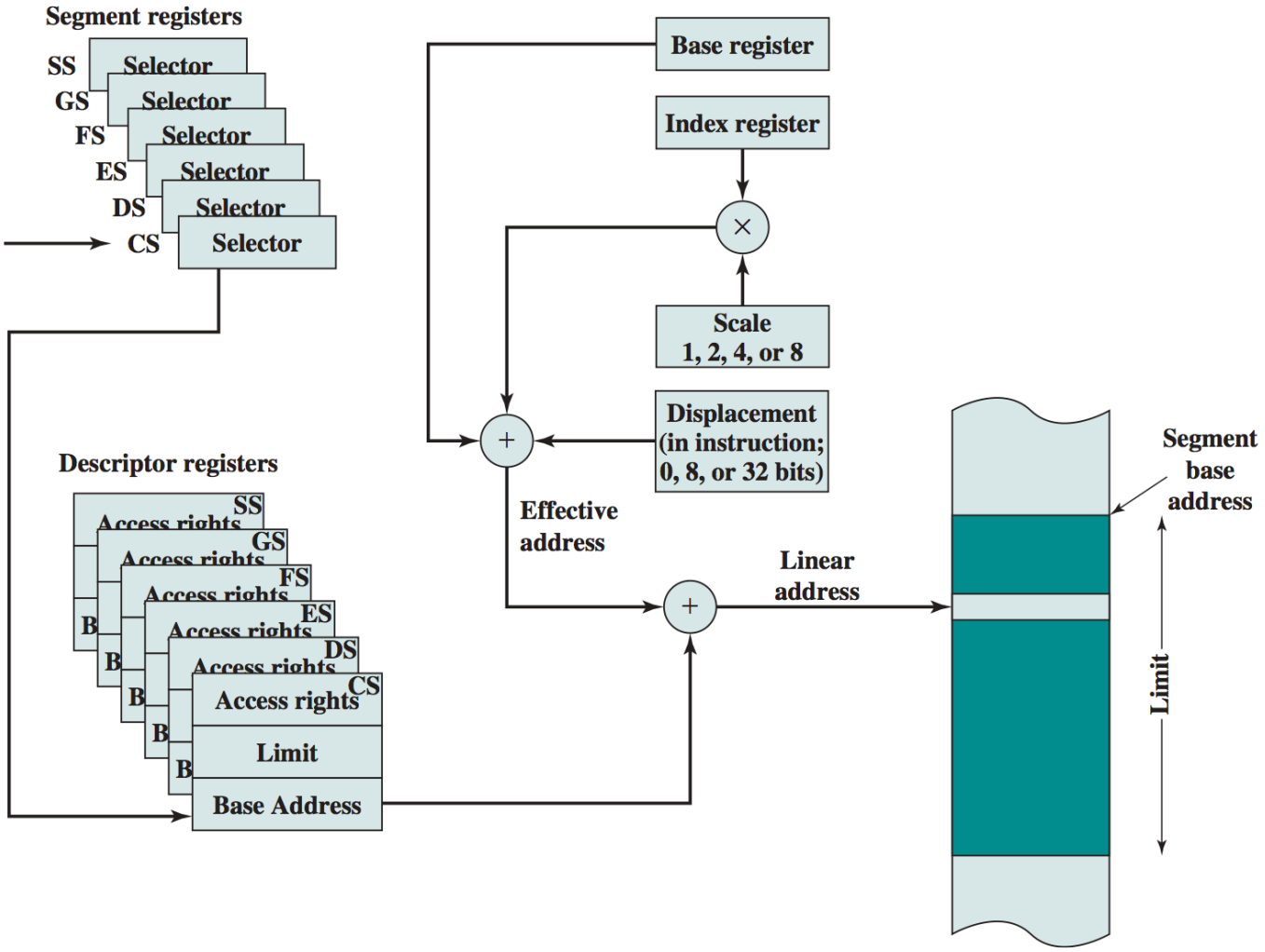
Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

x86 Addressing Modes

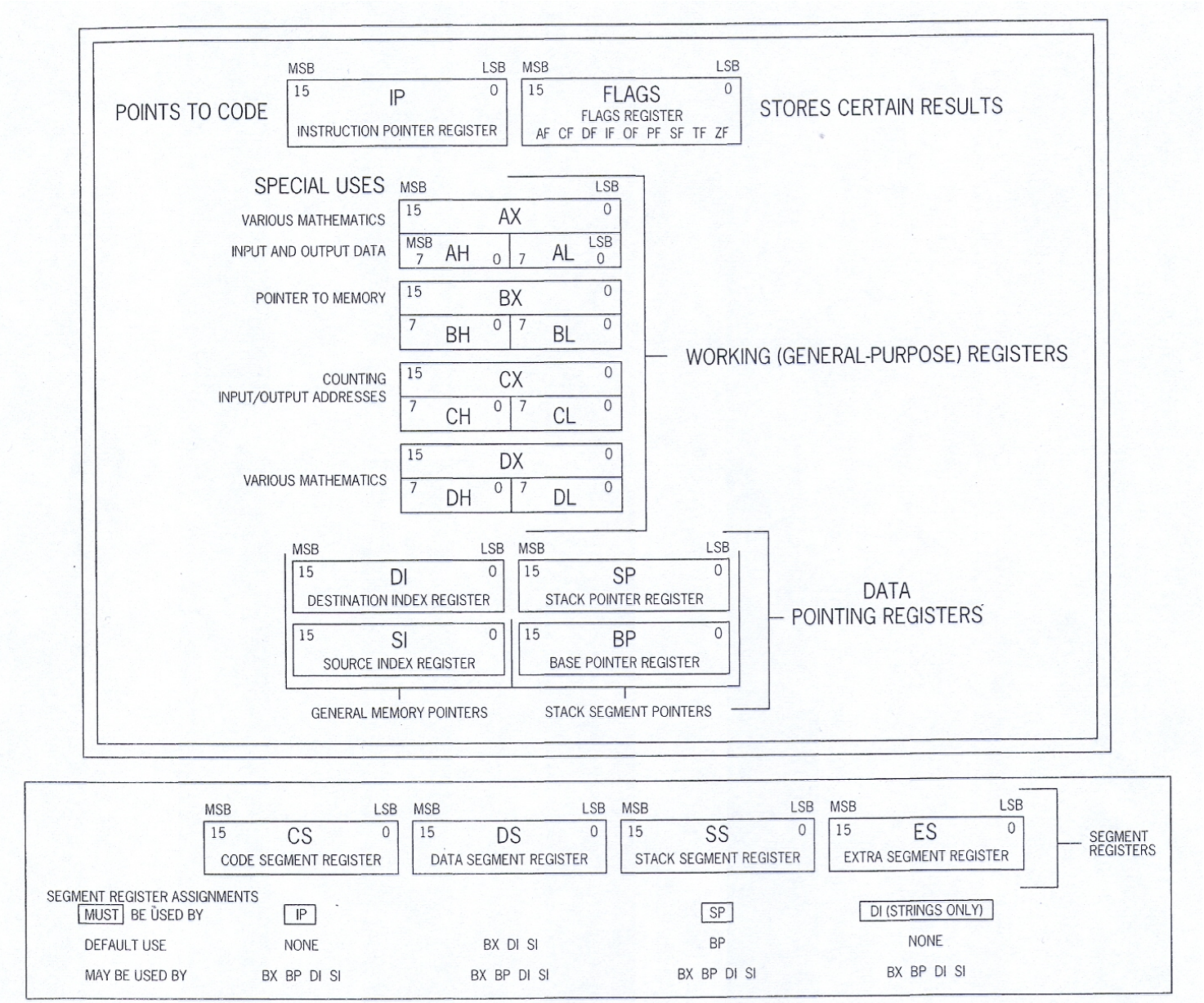
Mode	Algorithm
Immediate	Operand = A
Register Operand	LA = R
Displacement	LA = (SR) + A
Base	LA = (SR) + (B)
Base with Displacement	LA = (SR) + (B) + A
Scaled Index with Displacement	LA = (SR) + (I) × S + A
Base with Index and Displacement	LA = (SR) + (B) + (I) + A
Base with Scaled Index and Displacement	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

x86 Addressing Mode Calculation



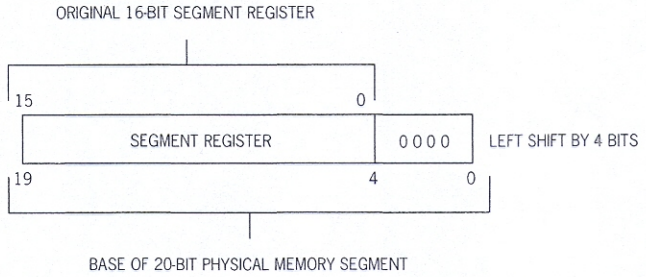
8086 Registers



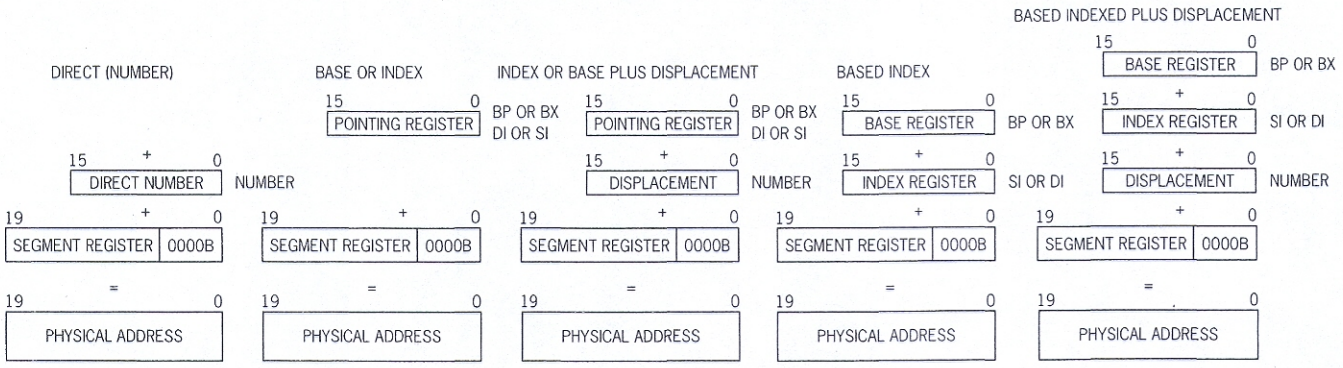
Flags Register

Flag	Meaning of the Flag if Set to 1
Auxiliary Carry flag (AF) [S]	A carry out of the low nibble of a result or a borrow into the low nibble of a result.
Carry flag (CF) [S] [P]	A carry out of the most significant bit of a result or a borrow into the result.
Direction flag (DF) [C] [P]	Forces certain opcodes to process data in memory from high to low addresses.
Interrupt flag (IF) [C] [P]	Allows the CPU to be interrupted by an externally generated interrupt.
Overflow flag (OF) [S]	Applies only to signed numbers used in a program. Indicates that the result is too large for the signed number range chosen.
Parity flag (PF) [S]	The operation generated a result with even number of 1's in the least significant byte.
Sign flag (SF) [S]	The result of an operation contains a 1 bit in the most significant bit position.
Trap flag (TF) [S]	Performs a software interrupt at the end of the next instruction.
Zero flag (ZF) [S]	The last CPU operation resulted in a quantity that contains all binary 0s.

Addressing Modes



FORMING A 20-BIT ADDRESS FROM A 16-BIT SEGMENT REGISTER



MOV Instruction

MOV Move Source to Destination

Mnemonic	Operands	Size	Operation	Flags
MOV	DESTINATION, SOURCE	B/W	(DST) ← (SRC)	None

Addressing Modes
(Destination, Source)

register, immediate
register, register
register, memory
memory, immediate
memory, register

Examples:

```
mov ax,1234h        ;MOV the number 1234h to register AX
mov bx,ax           ;MOV the contents of register AX to register BX
mov [1234h],ax      ;MOV the contents of AX to memory offset 1234h
mov [1234h],5678h   ;MOV the number 5678h to memory offset 1234h
mov cx,[0abcdh]     ;MOV the contents of memory offset ABCDh to CX
mov cx,[bx]         ;MOV the contents of memory offset pointed to
                    ;by BX to CX
```

Operation:

MOV copies data from the source address to the destination address. Byte- or word-length data may be moved.