Computer Organization
module-1

Prepared by:
Soumya Das
Asst. Prof in Computer science engg.
GCE, Kalahandi

Computer Organization

- **CPU - central processing unit**

Where decisions are made, computations are performed, and input/output requests are delegated

- **Memory**

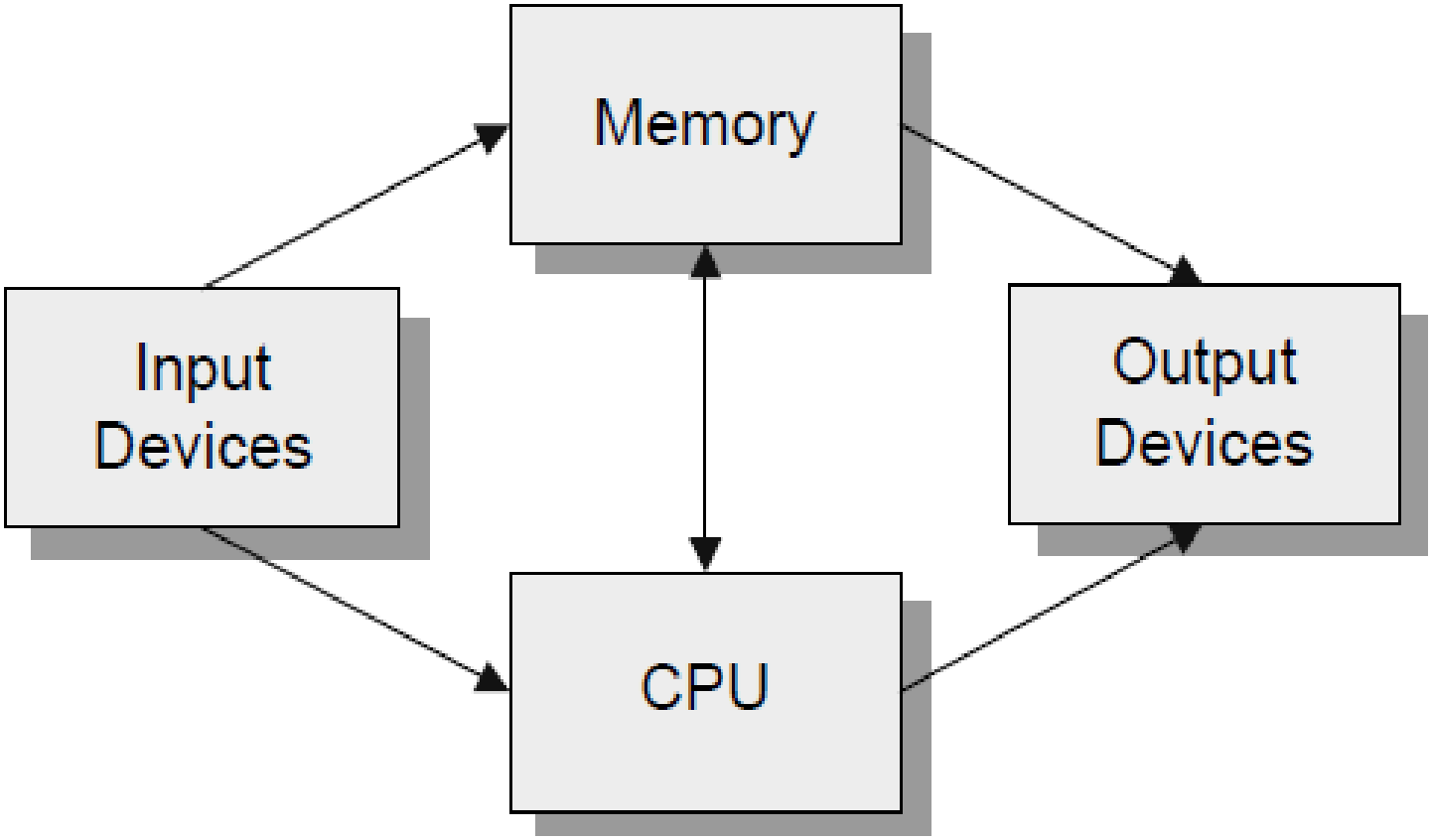
Stores information being processed by the CPU

- **Input devices**

Allows people to supply information to computers

- **Output devices**

Allows people to receive information from computers


Memory
Input Devices
Output Devices
CPU

Computer organization vs Architecture

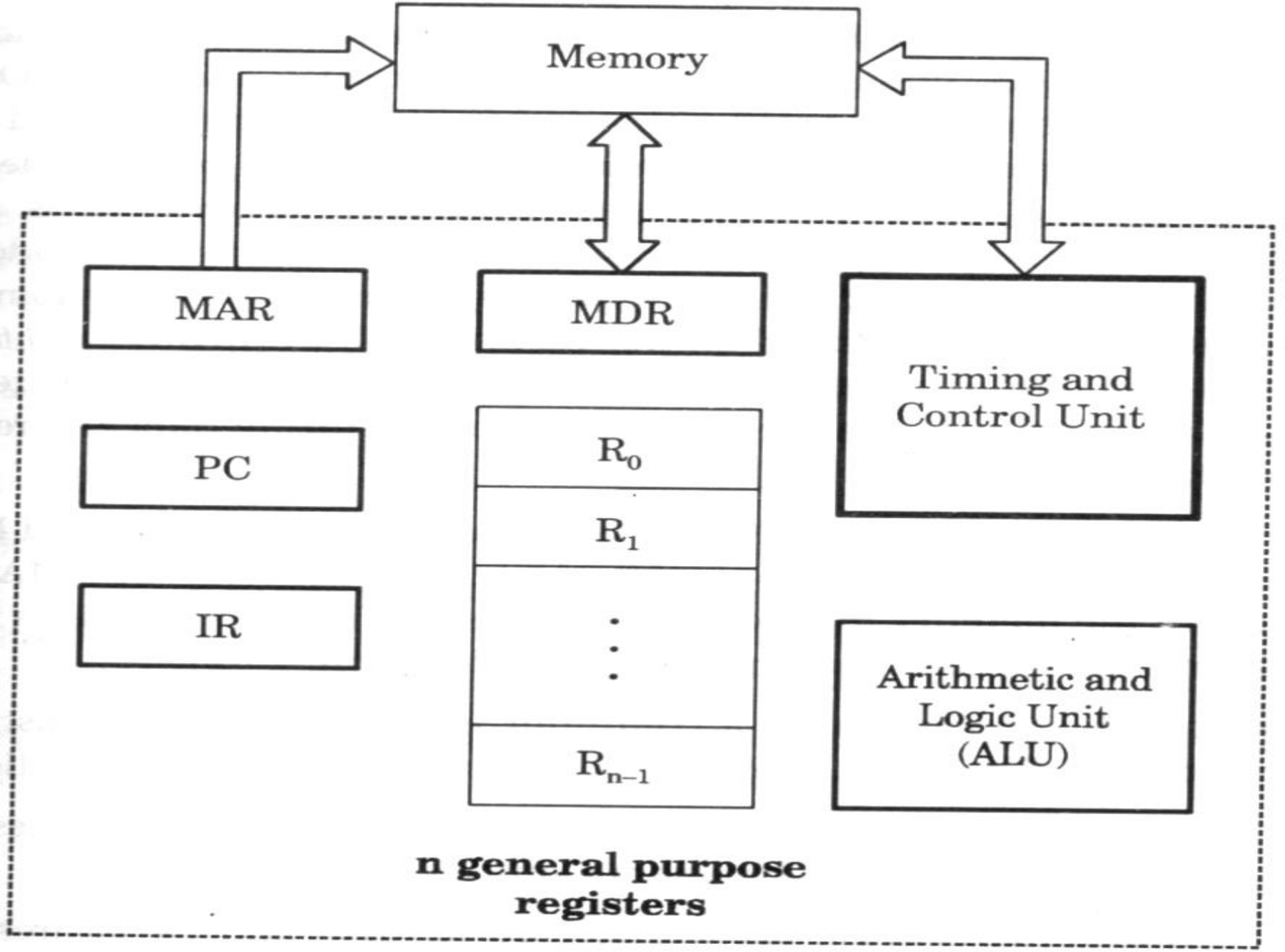
The components from which computers are built, i.e., computer organization.

- In contrast, computer architecture is the science of integrating those components to achieve a level of functionality and performance.
- It is as if computer organization examines the lumber, bricks, nails, and other building material
- While computer architecture looks at the design of the house.

Basic Operational Concepts

Basic Function of Computer

- To Execute a given task as per the appropriate program
- Program consists of list of instructions stored in memory


Memory
MAR
MDR
Timing and Control Unit
PC
R_0
R_1
IR
R_{n-1}
Arithmetic and Logic Unit (ALU)
n general purpose registers

Registers

Registers are fastest and alone storage locations that hold data temporarily. Multiple registers are needed to facilitate the operation of the CPU. Some of these registers are

MAR(Memory Address Register)and MDR(Memory Data Register):To handle the data transfer between main memory and processor. MAR-Holds addresses, MDR-Holds data

Instruction register(IR):Hold the Instructions that is currently being executed

Program counter: Points to the next instructions that is to be fetched from memory

Operation

- (PC) ⟶ (MAR)(the contents of PC transferred to MAR)

- (MAR) ⟶ (Address bus) Select a particular memory location
- Issues RD control signals

- Reads instruction present in memory and loaded into MDR
- Will be placed in IR (Contents transferred from MDR to IR)

Operation contd.

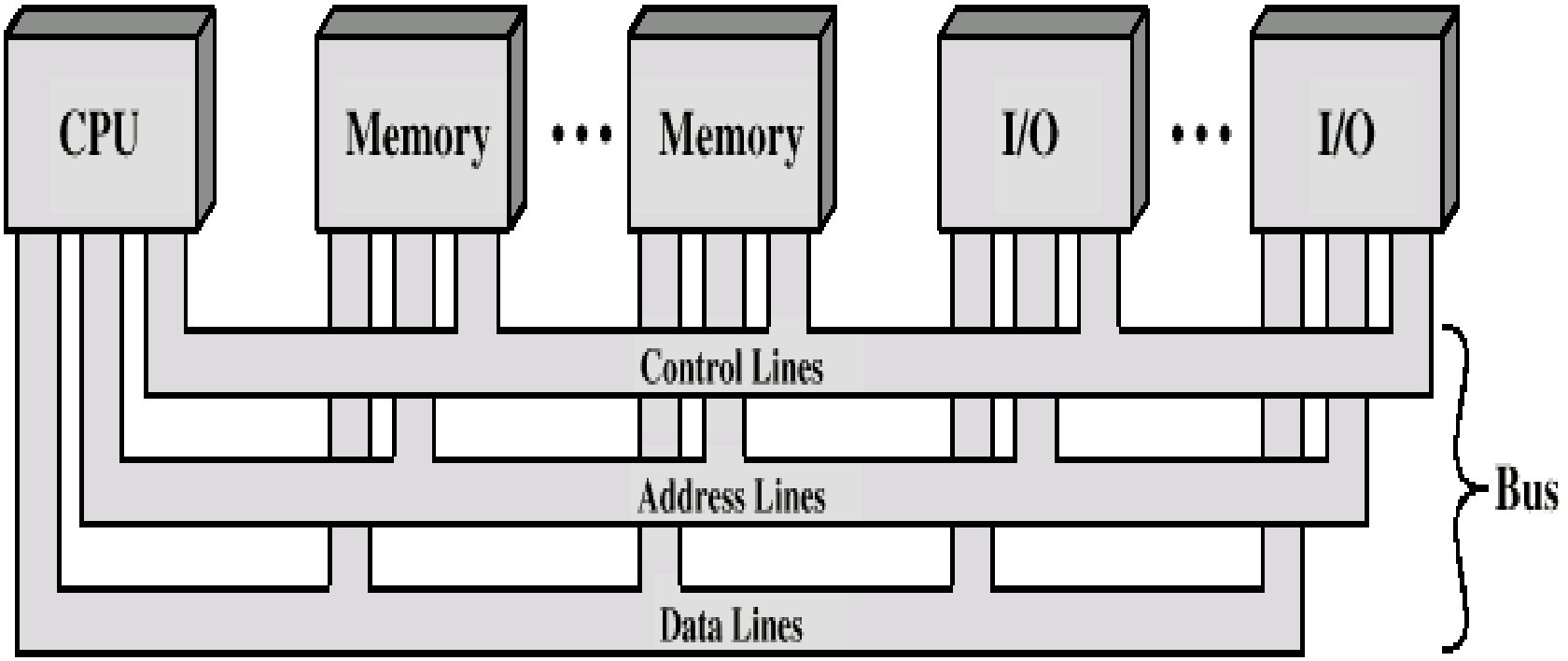
- Instruction present in IR will be decoded by which processor understand what operation it has to perform
- Increments the contents of PC by 1, so that it points to the next instruction address
- If data required for operation is available in register, it performs the operation
- If data is present in memory following sequence is performed

Address of the data ⟶ MAR

- MAR ⟶ Address bus ⟶ select memory location where is issued RD signal

- Reads data via data bus ⟶ MDR
- From MDR data can be directly routed to ALU or it can be placed in register and then operation can be performed
- Results of the operation can be directed towards output device, memory or register

Bus Interconnection Scheme



Data Bus

- Carries data
  - Remember that there is no difference between “data” and “instruction” at this level
- Width (number of lines) is a key determinant of performance, since this determines how many bits can be transferred in one go (cycle)
  - 32 to hundreds of bits

Address bus

- Identify the source or destination of data
- e.g. CPU needs to read an instruction (data) from a given location in memory
- Bus width determines maximum memory capacity of system
  - e.g. 8080 has 16 bit address bus giving 64k address space

Control Bus

- Memory or I/O read/write signals
- Interrupt request/acknowledgment
- Clock signals
- Bus request/grant signals

Software

Application software:

- Programs designed to perform specific tasks that are transparent to the user
- Application software is the software that has made using computers indispensable and popular Common application software

Example :

- Word processors
- Desktop publishing programs
- Spreadsheets
- Presentation managers

System software:

- Programs that support the execution and development of

Application Software

- Learning how to develop application software is our focus
- Controls and manages the computing resources

Examples

- Windows®, UNIX®, Mac OS X®

Performance

Time taken by the system to execute a program

Parameters which influence the performance are

- Clock speed
- Type and number of instructions available
- Average time required to execute an instruction
- Memory access time
- Power dissipation in the system
- Number of I/O devices and types of I/O devices connected
- The data transfer capacity of the bus

MEMORY LOCATIONS AND ADDRESSES

- Main memory is the second major subsystem in a computer. It consists of a collection of storage locations, each with a unique identifier, called an address.

- Data is transferred to and from memory in groups of bits called words. A word can be a group of 8 bits, 16 bits, 32 bits or 64 bits (and growing).

- If the word is 8 bits, it is referred to as a byte. The term “byte” is so common in computer science that sometimes a 16-bit word is referred to as a 2-byte word, or a 32-bit word is referred to as a 4-byte word.

Address space

- To access a word in memory requires an identifier. Although programmers use a name to identify a word (or a collection of words), at the hardware level each word is identified by an address.
- The total number of uniquely identifiable locations in memory is called the address space.
- For example, a memory with 64 kilobytes (16 address line required) and a word size of 1 byte has an address space that ranges from 0 to 65,535.

Assignment of byte addresses

- Little Endian (e.g., in DEC, Intel)

  low order byte stored at lowest address

  byte0 byte1 byte2 byte3
- Big Endian (e.g., in IBM, Motorolla, Sun, HP)

  high order byte stored at lowest address

  byte3 byte2 byte1 byte0

INSTRUCTION SET ARCHITECTURE

BASIC 4 TYPES OF OPERATION:-

- Data transfer between memory and processor register
- Arithmetic and logic operation
- Program sequencing and control
- I/O transfer

Register transfer notation (RTN)

- Transfer between processor registers & memory, between processor register & I/O devices, Memory locations, registers and I/O register names are identified by a symbolic name in uppercase alphabets
- LOC,PLACE,MEM are the address of memory location. R1 , R2,… are processor registers , DATA_IN, DATA_OUT are I/O registers,
- Contents of location is indicated by using square brackets [ ]
- RHS of RTN always denotes a values, and is called Source.LHS of RTN always denotes a symbolic name where value is to be stored and is called destination

- Source contents are not modified
- Destination contents are overwritten

Examples of RTN statements

R2 $\longrightarrow$ [LOCN

R4 $\longrightarrow$ [R3] +[R2]

ASSEMBLY LANGUAGE NOTATION (ALN)

RTN is easy to understand and but cannot be used to represent machine instructions

Mnemonics can be converted to machine language, which processor understands using assembler

Eg:

1. MOVE LOCN, R2
2. ADD R3, R2, R4

TYPE OF INSTRUCTION

Three address instruction

Syntax: Operation source 1, source 2, destination

- Eg: ADD D,E,F where D,E,F are memory location
- Advantage: Single instruction can perform the complete operation
- Disadvantage : Instruction code will be too large to fit in one word location in memory

TWO ADDRESS INSTRUCTION

Syntax : Operation source, destination

Eg: MOVE E,F MOVE D,F ADD D,F OR ADD E,F

Perform ADD A,B,C using 2 instructions

- Disadvantage: Single instruction is not sufficient to perform the entire operation.

ONE ADDRESS INSTRUCTION

Syntax- Operation source/destination

In this type either a source or destination operand is mentioned in the instruction

Other operand is implied to be a processor register called Accumulator Eg: ADD B (general)

Zero address instruction

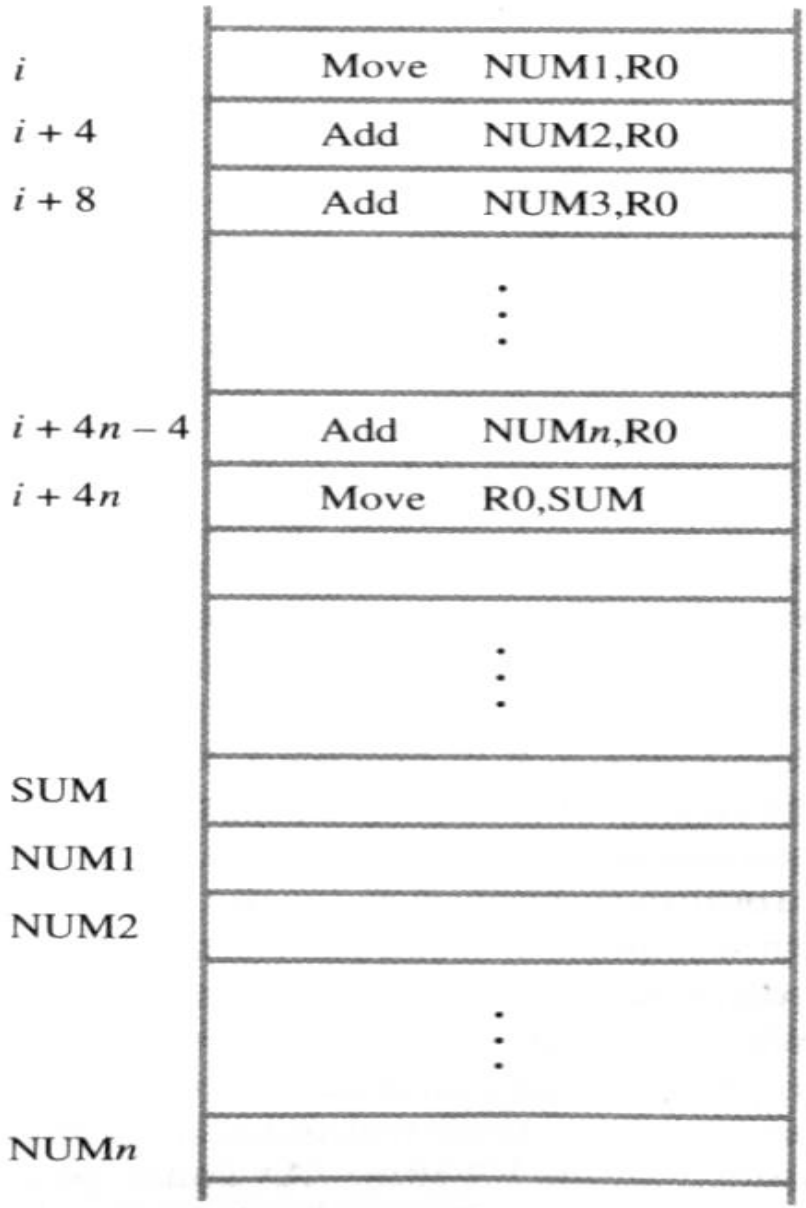
Location of all operands are defined implicitly

Operands are stored in a structure called pushdown stack

Instruction execution

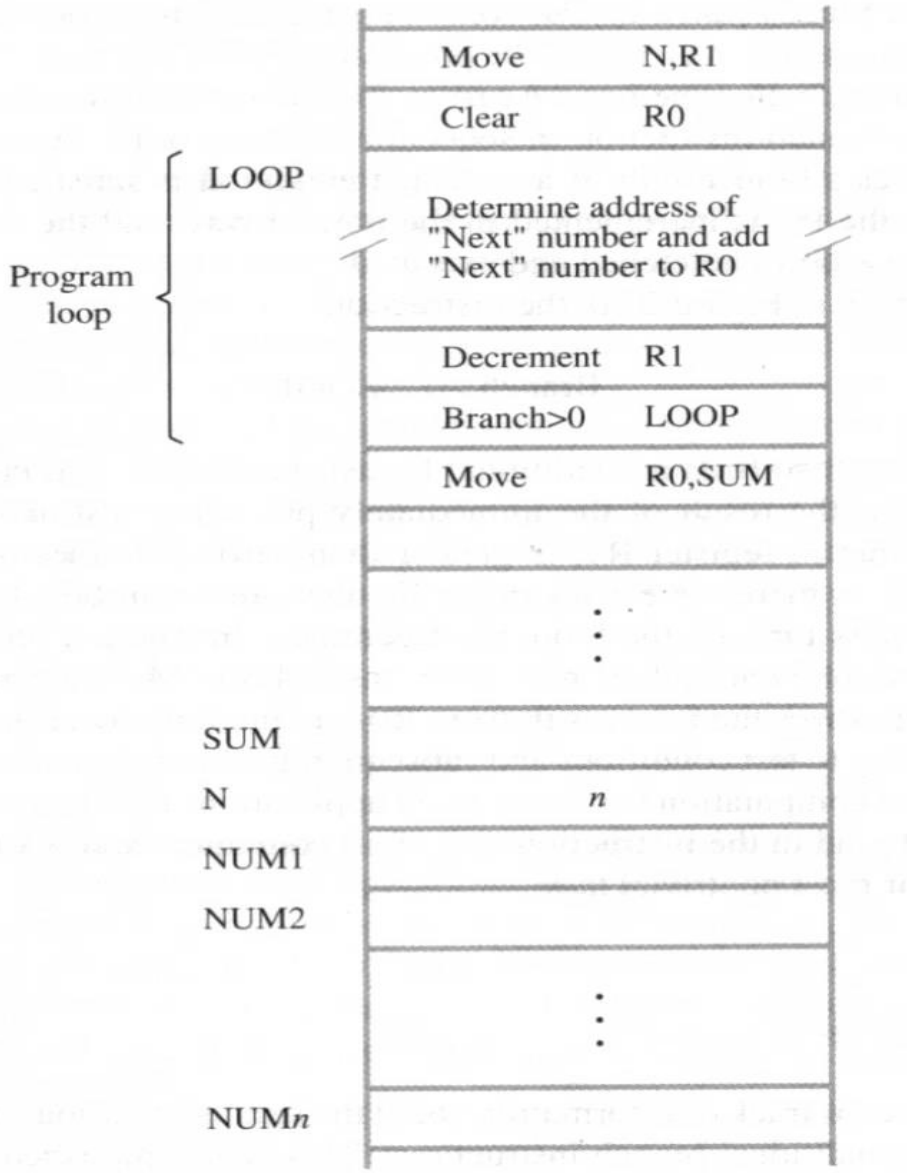
Straight line sequencing: If fetching and executing of instructions is carried out one by one from successive addresses of memory, it is called straight line sequencing.

An example is given at the right hand side



Branching

- Branch instruction are those which changes the normal sequence of execution.
  - Sequence can be changed either conditionally or unconditionally.
  - Accordingly we have conditional branch instructions and unconditional branch instruction.
  - Conditional branch instruction changes the sequence only when certain conditions are met.
  - Unconditional branch instruction changes the sequence of execution irrespective of condition of the results.
- An example is given at the right hand side



CONDITION CODES

- CONDITIONAL CODE FLAGS: The processor keeps track of information about the results of various operations for use by subsequent conditional branch instructions
- N – Negative 1 if results are Negative
  0 if results are Positive
- Z – Zero 1 if results are Zero
  0 if results are Non zero
- V – Overflow 1 if arithmetic overflow occurs
  0 non overflow occurs
- C – Carry 1 if carry and from MSB bit
  0 if there is no carry from MSB bit

Basic Instruction Cycle

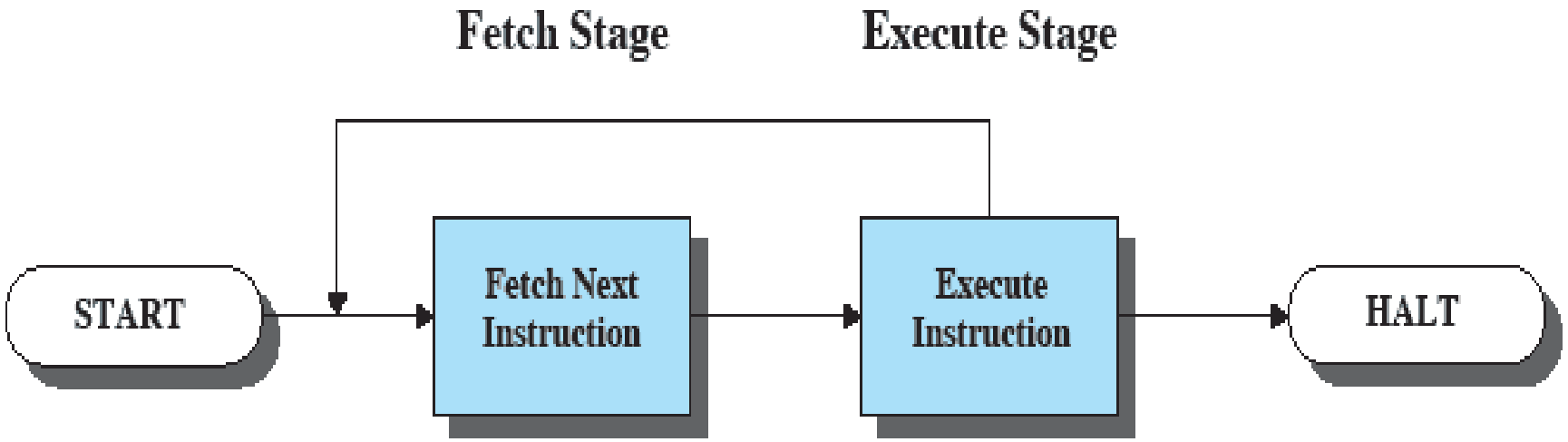


Figure 1.2 Basic Instruction Cycle

Fetch Cycle

- Program Counter (PC) holds address of next instruction to be fetched
- Processor fetches instruction from memory location pointed to by PC
- Increment PC
  - Unless told otherwise
- Instruction loaded into Instruction Register (IR)
- Processor interprets instruction and performs required actions

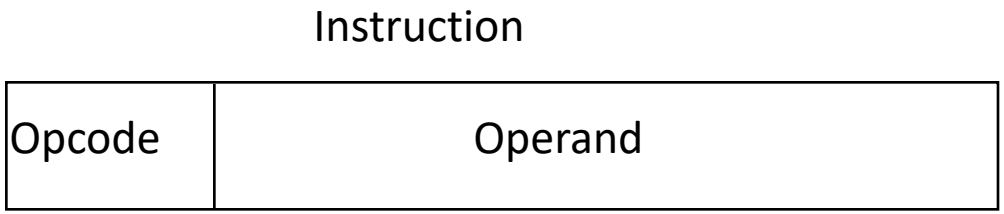
Addressing Modes

- 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

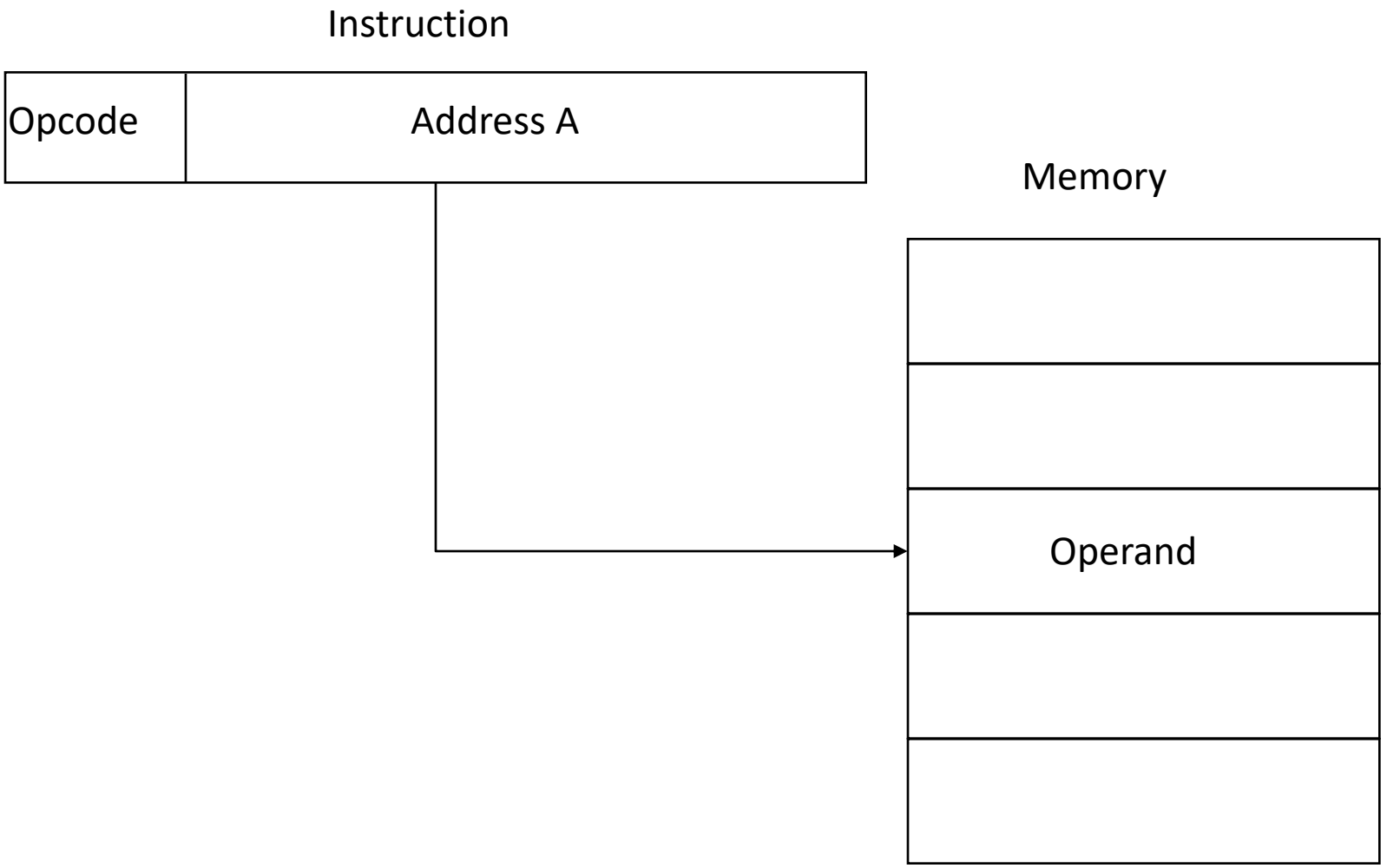
Immediate Addressing Diagram



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



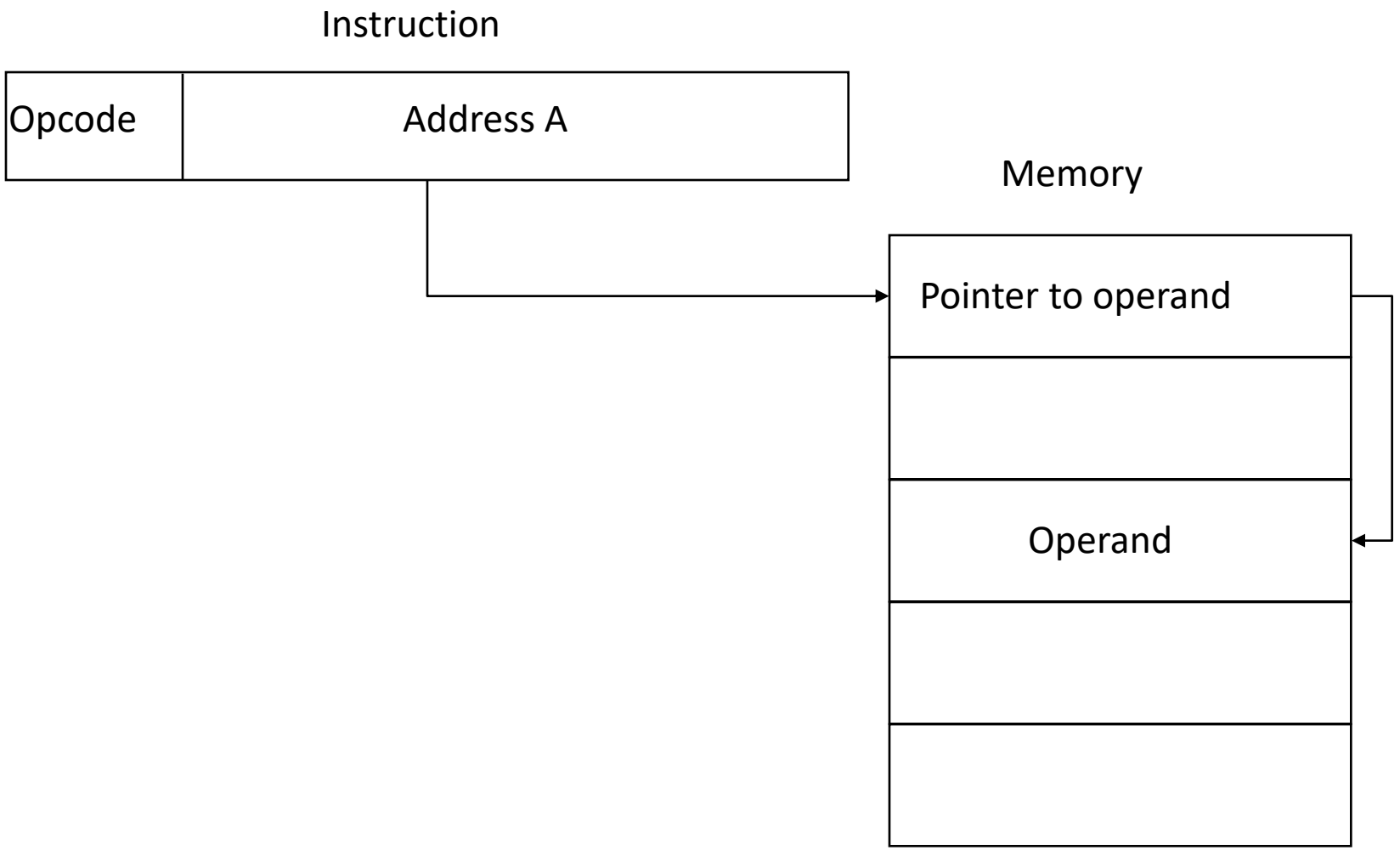
Indirect Addressing (1)

- 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



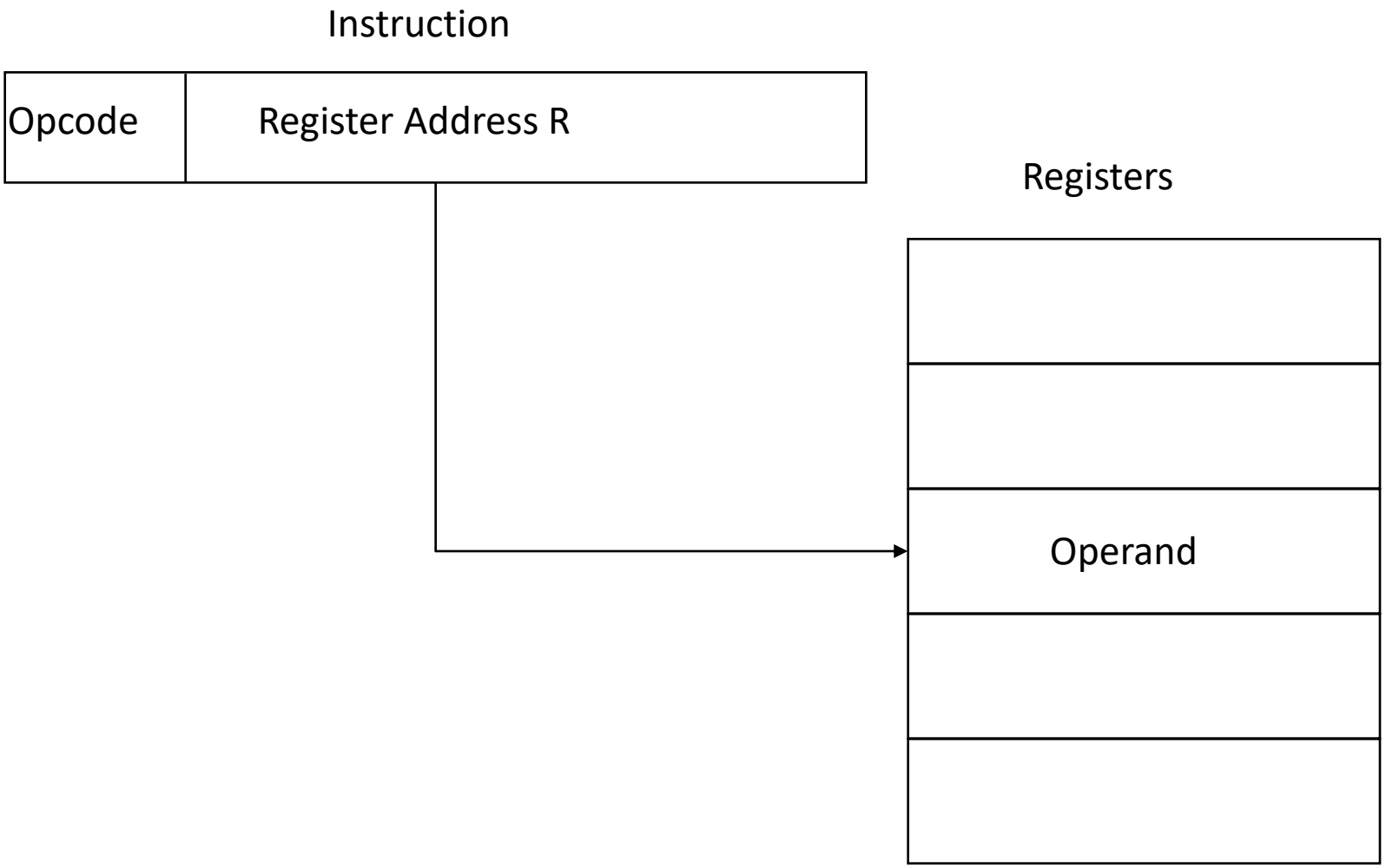
Register Addressing (1)

- Operand is 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
  - N.B. C programming
    - register int a;
- c.f. Direct addressing

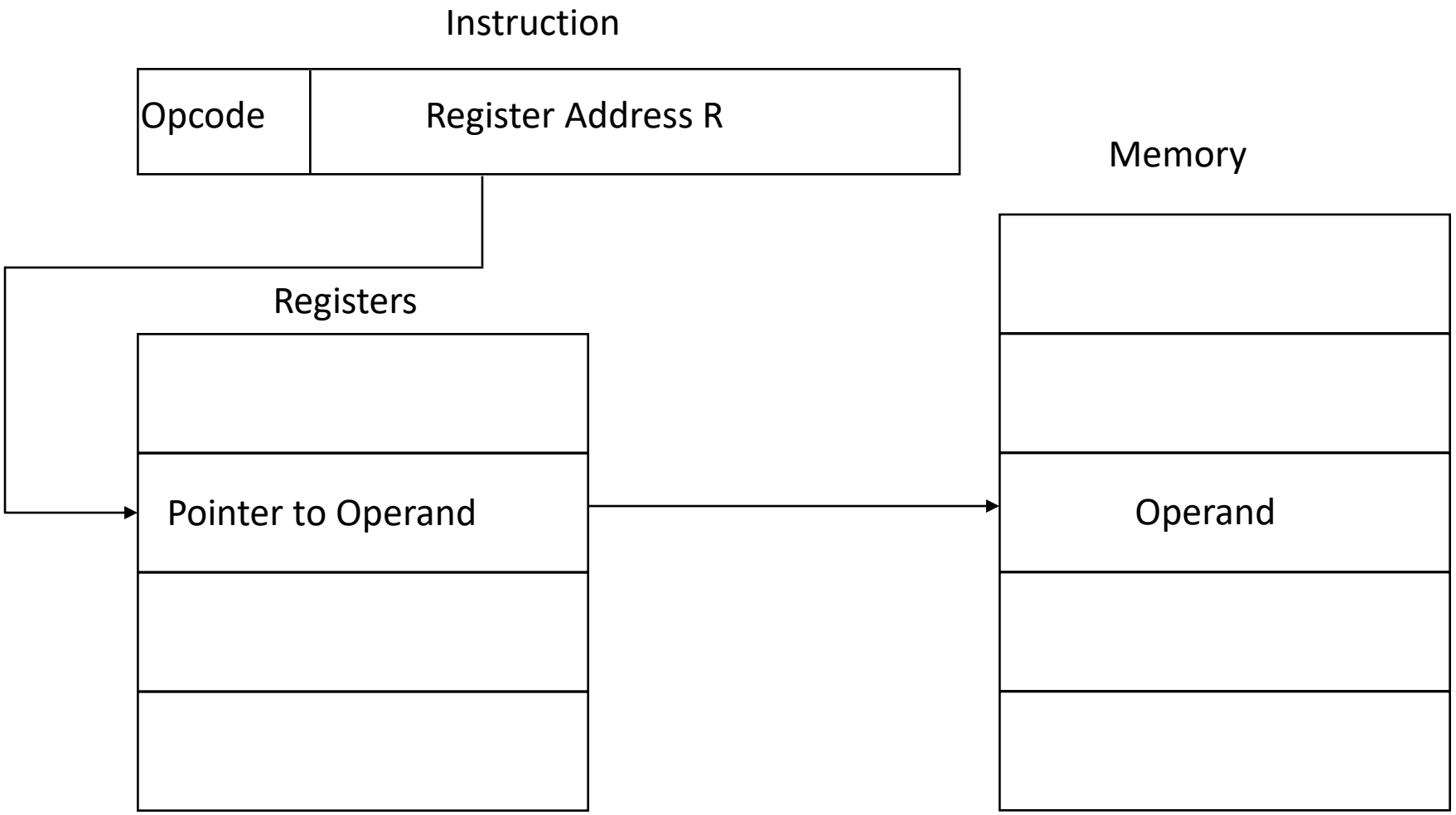
Register Addressing Diagram



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

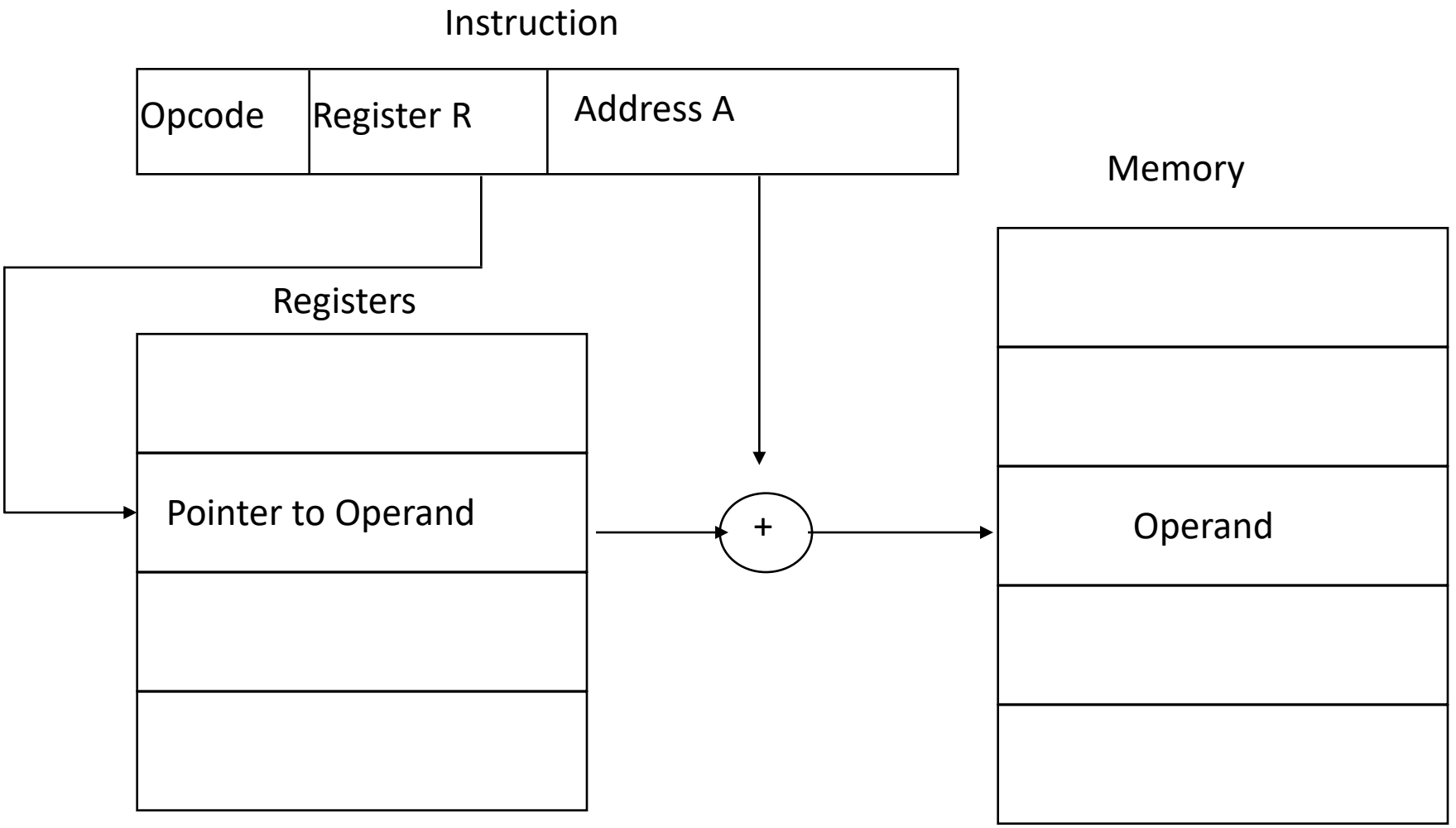
Register Indirect Addressing Diagram



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
- c.f 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))

- (Draw the diagrams)

Stack Addressing

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