Chapter 1. Basic Structure of Computers

Functional Units

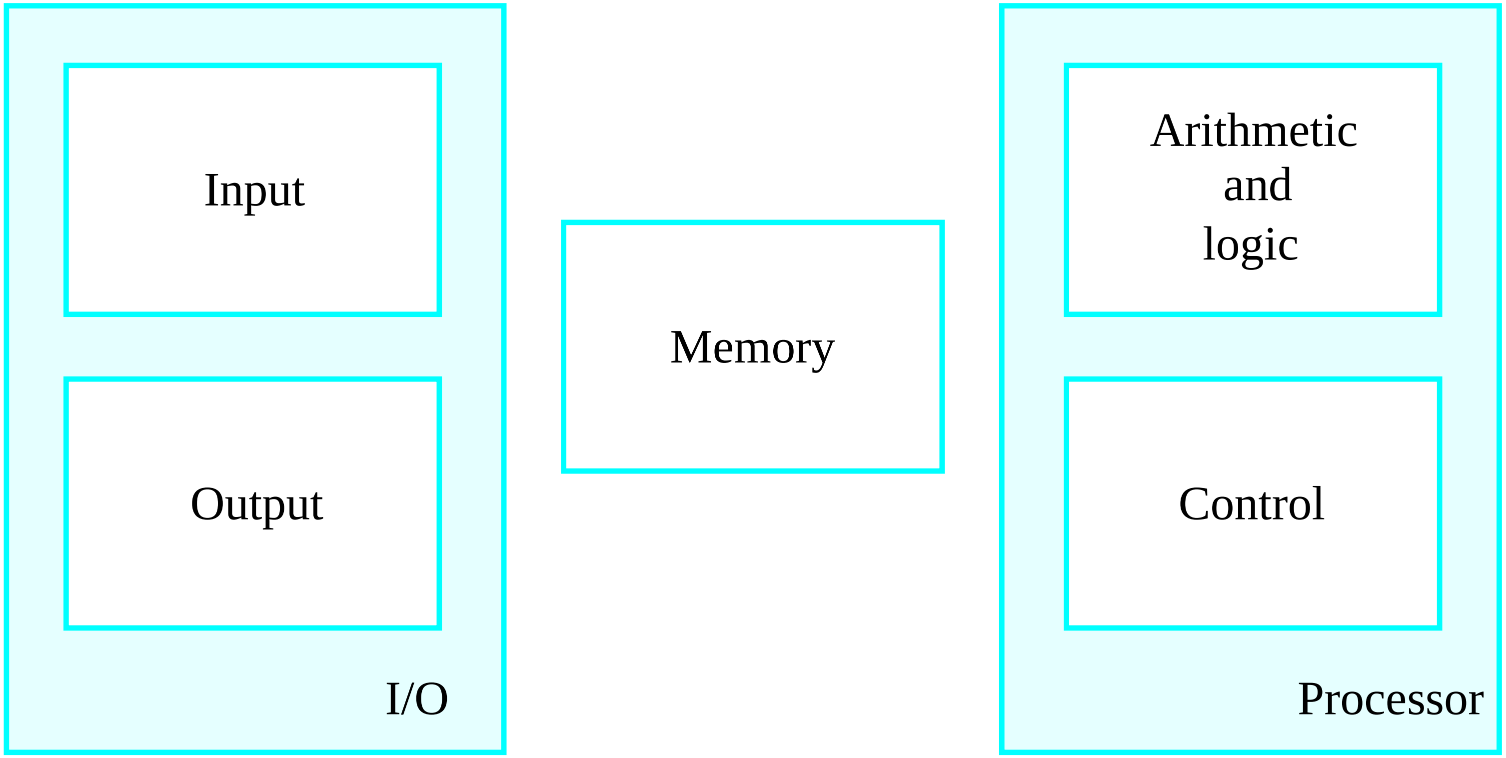


Figure 1.1. Basic functional units of a computer.

Information Handled by a Computer

- Instructions/machine instructions
  - Govern the transfer of information within a computer as well as between the computer and its I/O devices
  - Specify the arithmetic and logic operations to be performed
  - Program
- Data
  - Used as operands by the instructions
  - Source program
- Encoded in binary code – 0 and 1

Memory Unit

- Store programs and data
- Two classes of storage
  - ➢ Primary storage
    - ❖ Fast
    - ❖ Programs must be stored in memory while they are being executed
    - ❖ Large number of semiconductor storage cells
    - ❖ Processed in words
    - ❖ Address
    - ❖ RAM and memory access time
    - ❖ Memory hierarchy – cache, main memory
  - ➢ Secondary storage – larger and cheaper

Arithmetic and Logic Unit (ALU)

- Most computer operations are executed in ALU of the processor.
- Load the operands into memory – bring them to the processor – perform operation in ALU – store the result back to memory or retain in the processor.
- Registers
- Fast control of ALU

Control Unit

- All computer operations are controlled by the control unit.
- The timing signals that govern the I/O transfers are also generated by the control unit.
- Control unit is usually distributed throughout the machine instead of standing alone.
- Operations of a computer:
  - Accept information in the form of programs and data through an input unit and store it in the memory
  - Fetch the information stored in the memory, under program control, into an ALU, where the information is processed
  - Output the processed information through an output unit
  - Control all activities inside the machine through a control unit

Basic Operational Concepts

Review

- Activity in a computer is governed by instructions.
- To perform a task, an appropriate program consisting of a list of instructions is stored in the memory.
- Individual instructions are brought from the memory into the processor, which executes the specified operations.
- Data to be used as operands are also stored in the memory.

A Typical Instruction

- Add LOCA, R0
- Add the operand at memory location LOCA to the operand in a register R0 in the processor.
- Place the sum into register R0.
- The original contents of LOCA are preserved.
- The original contents of R0 is overwritten.
- Instruction is fetched from the memory into the processor – the operand at LOCA is fetched and added to the contents of R0 – the resulting sum is stored in register R0.

Separate Memory Access and ALU Operation

- Load LOCA, R1
- Add R1, R0
- Whose contents will be overwritten?

Connection Between the Processor and the Memory

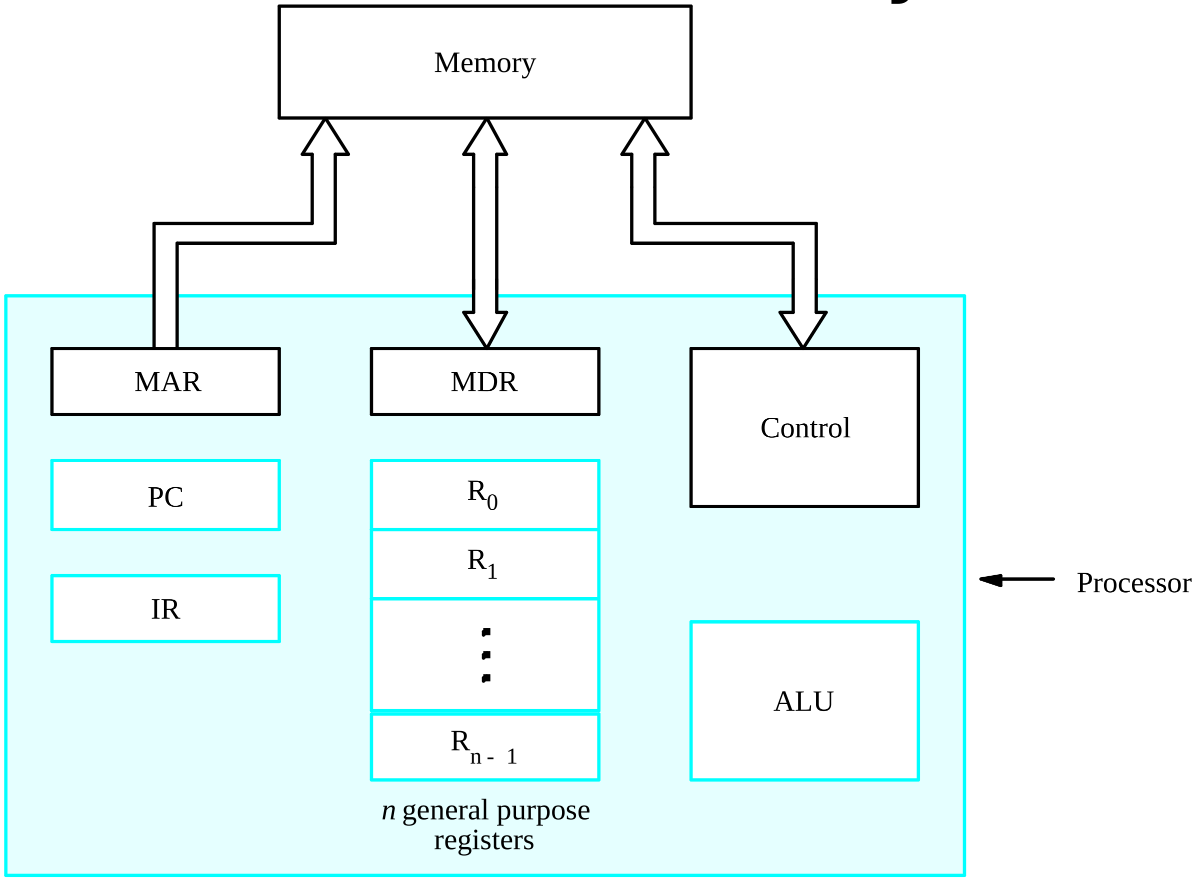


Figure 1.2. Connections between the processor and the memory.

Computer Components: Top-Level View

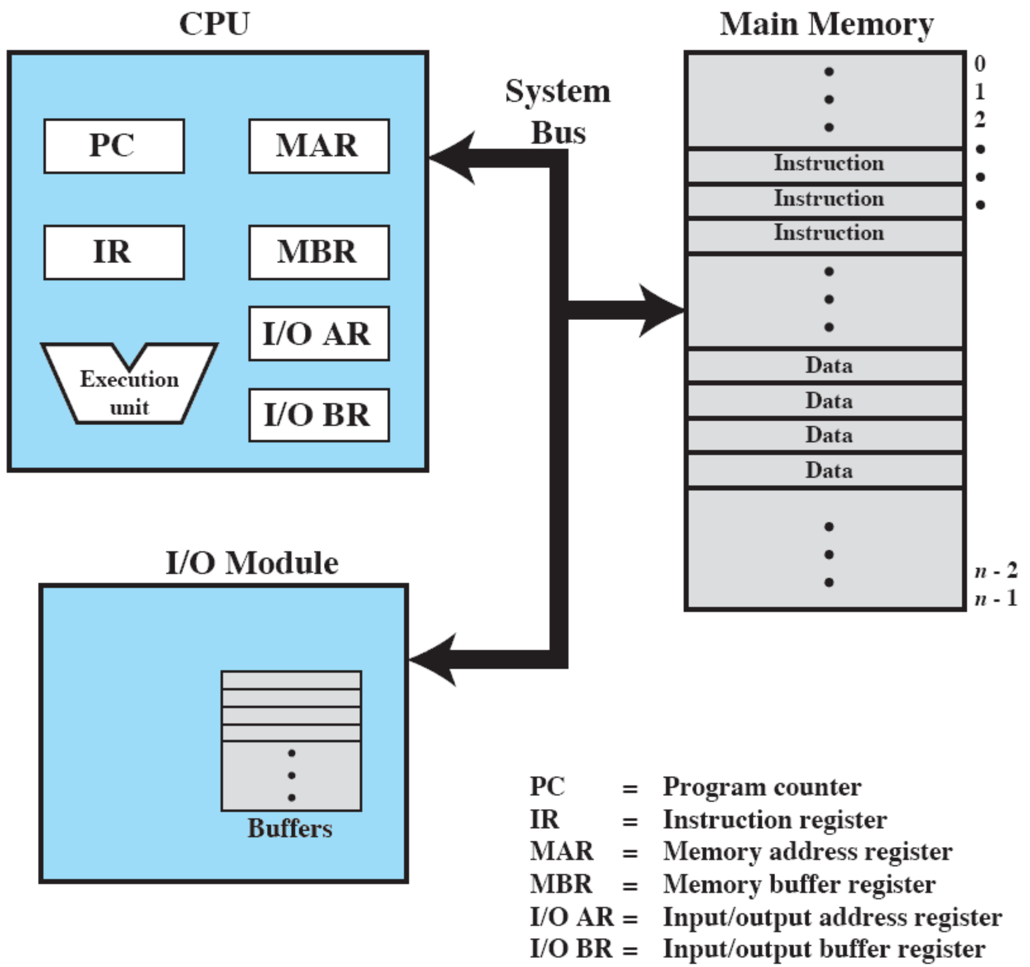


Figure 1.1 Computer Components: Top-Level View

Registers

- Instruction register (IR)
- Program counter (PC)
- General-purpose register (R_0 – R_{n-1})
- Memory address register (MAR)
- Memory data register (MDR)

Instruction Execution

- Two steps
  - Processor reads (fetches) instructions from memory
  - Processor executes each instruction

Basic Instruction Cycle

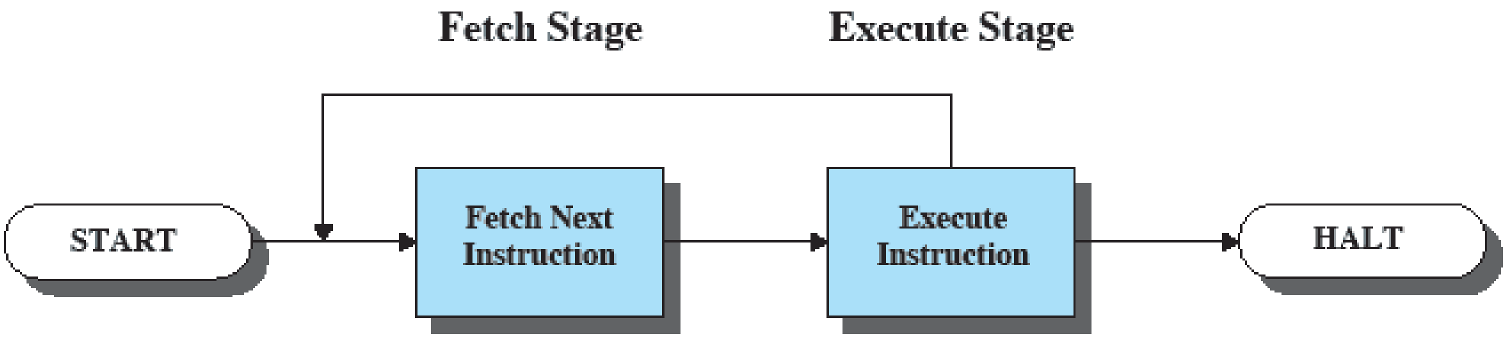


Figure 1.2 Basic Instruction Cycle

Instruction Fetch and Execute

- The processor fetches the instruction from memory
- Program counter (PC) holds address of the instruction to be fetched next
- PC is incremented after each fetch

Instruction Register

- Fetched instruction loaded into instruction register
- Categories
  - Processor-memory, processor-I/O, data processing, control

Typical Operating Steps

- Programs reside in the memory through input devices
- PC is set to point to the first instruction
- The contents of PC are transferred to MAR
- A Read signal is sent to the memory
- The first instruction is read out and loaded into MDR
- The contents of MDR are transferred to IR
- Decode and execute the instruction

Typical Operating Steps (Cont’)

- Get operands for ALU
  - ➢ General-purpose register
  - ➢ Memory (address to MAR – Read – MDR to ALU)
- Perform operation in ALU
- Store the result back
  - ➢ To general-purpose register
  - ➢ To memory (address to MAR, result to MDR – Write)
- During the execution, PC is incremented to the next instruction

Bus Structures

- There are many ways to connect different parts inside a computer together.
- A group of lines that serves as a connecting path for several devices is called a *bus*.
- Address/data/control

Bus Structure

- Single-bus

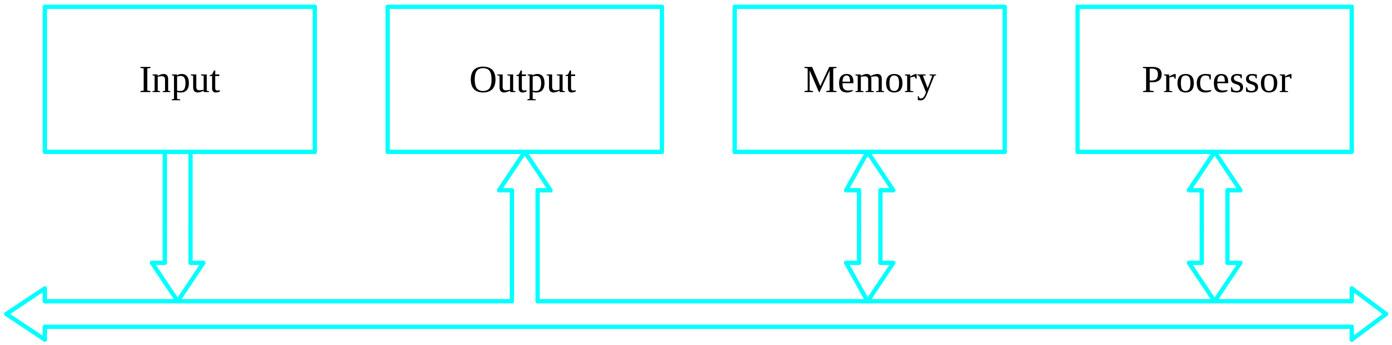


Figure 1.3. Single-bus structure.

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

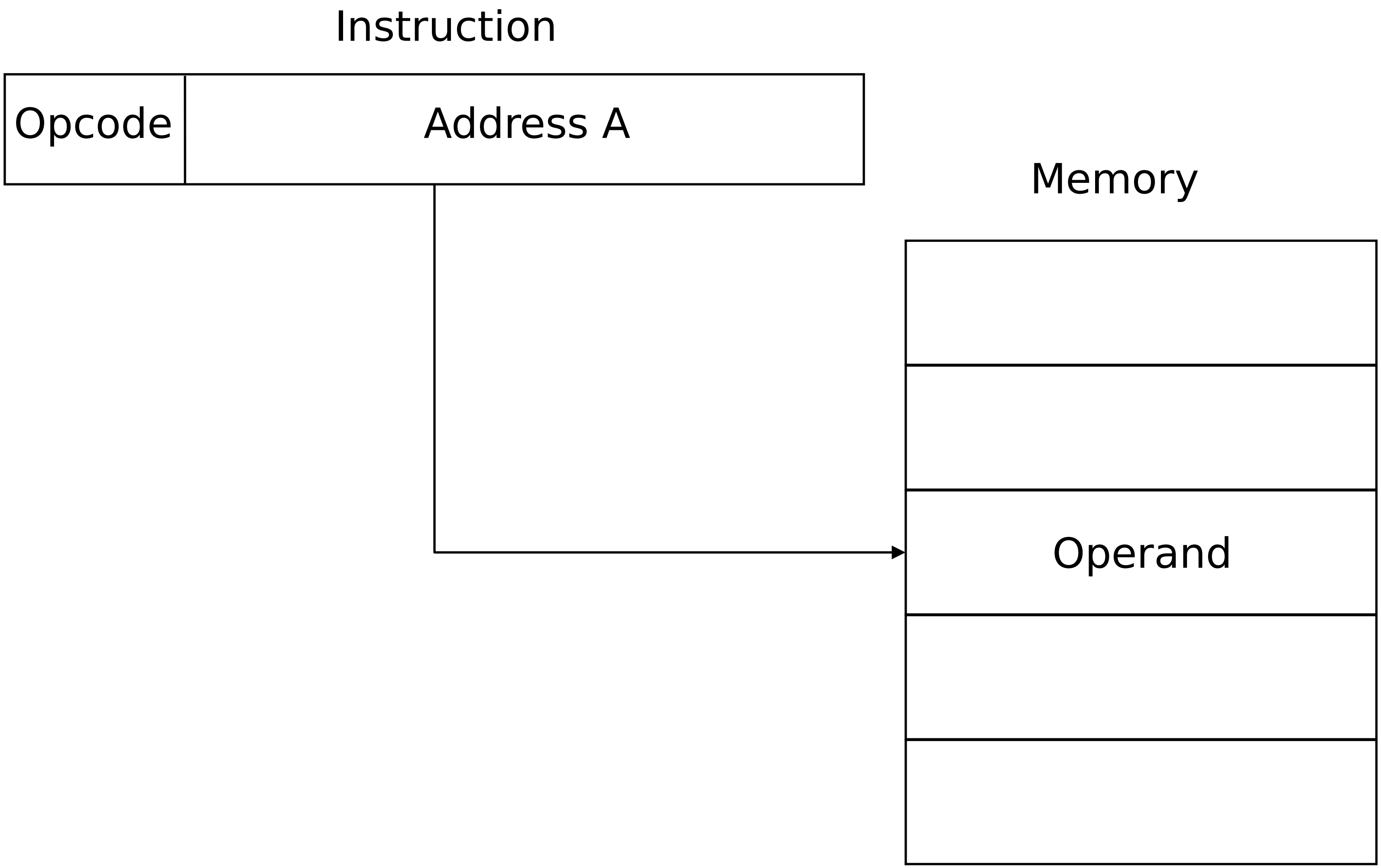
Instruction

Opcode	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



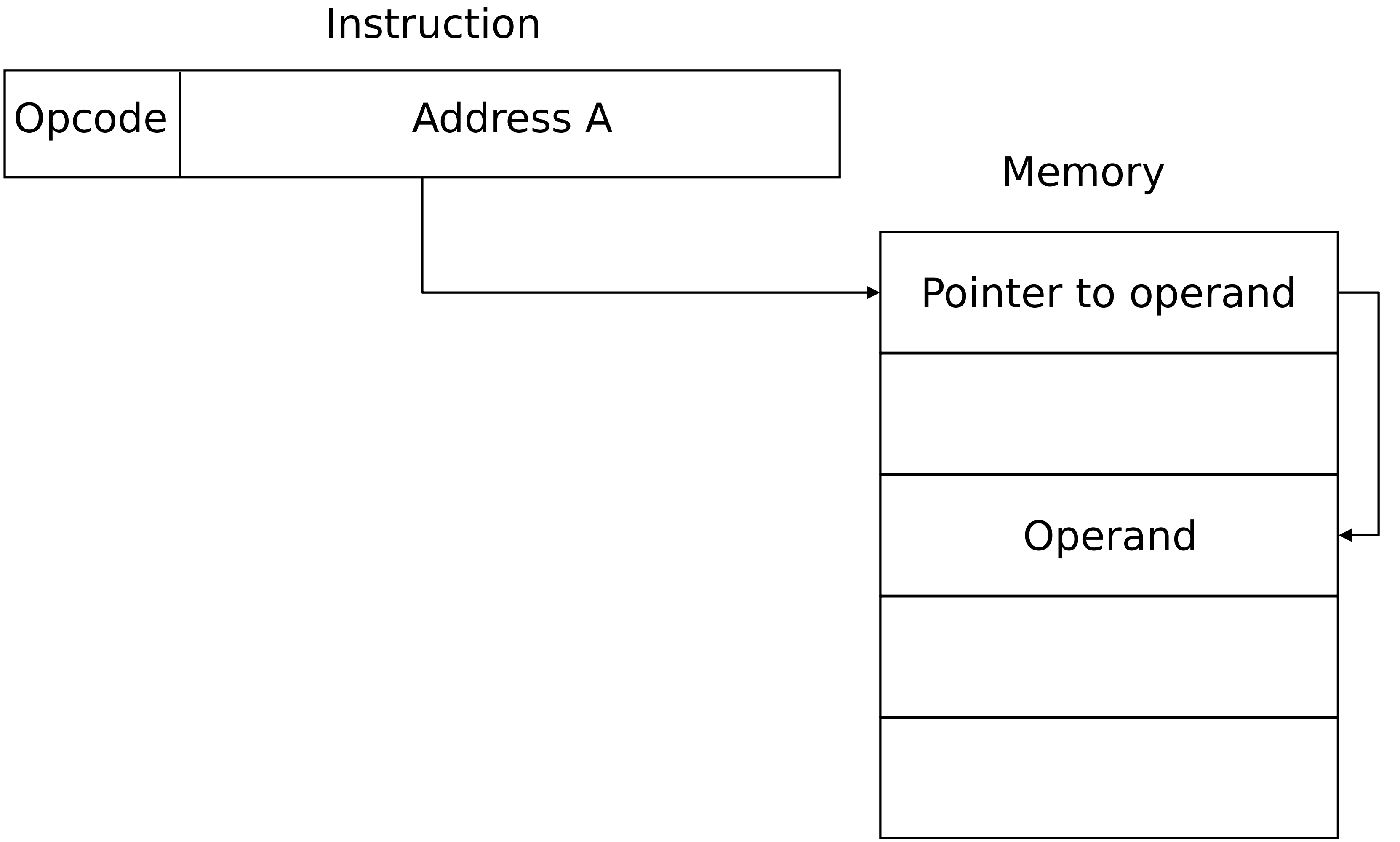
Indirect Addressing (1)

- 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



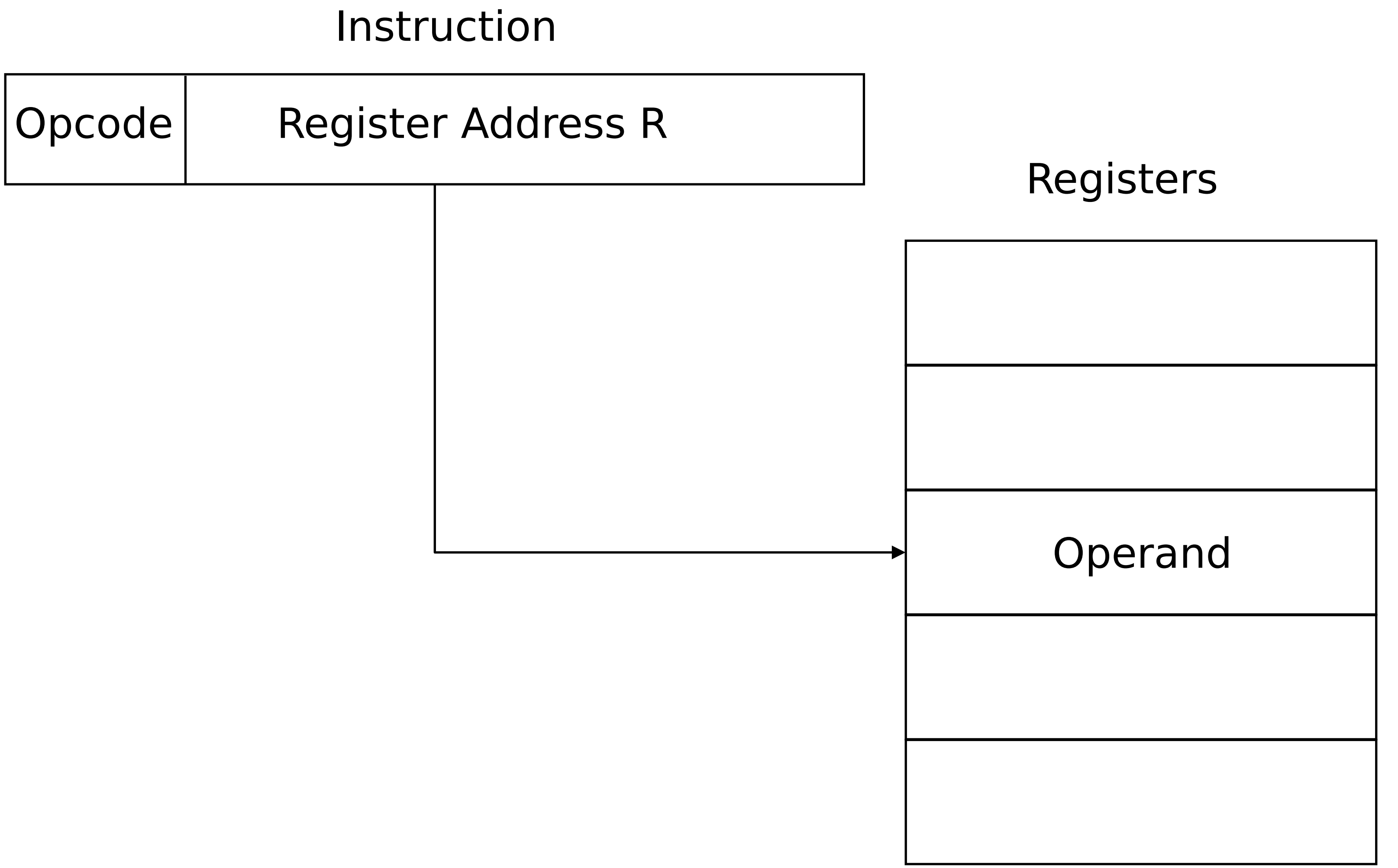
Register Addressing (1)

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

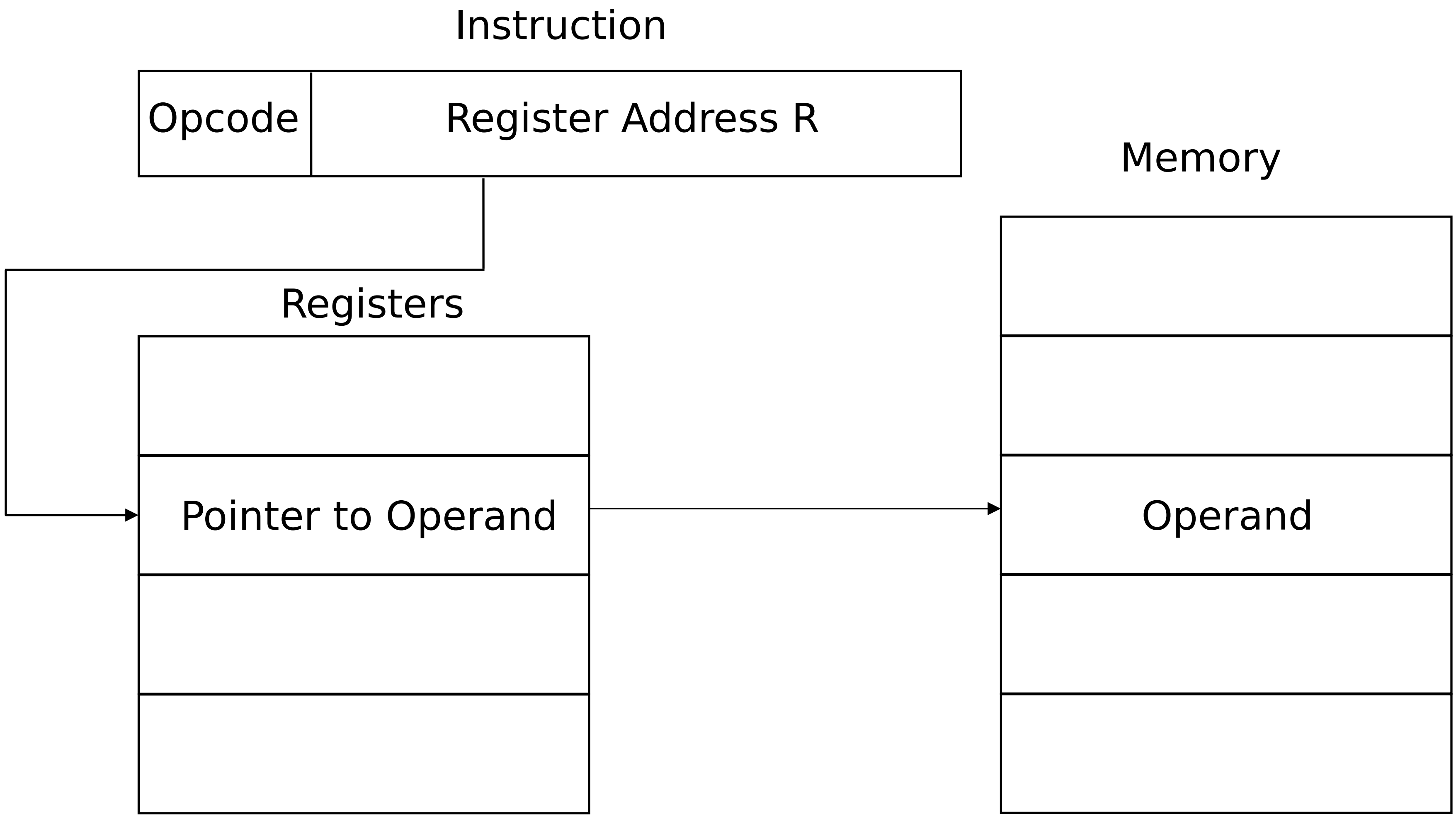
Register Addressing Diagram



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

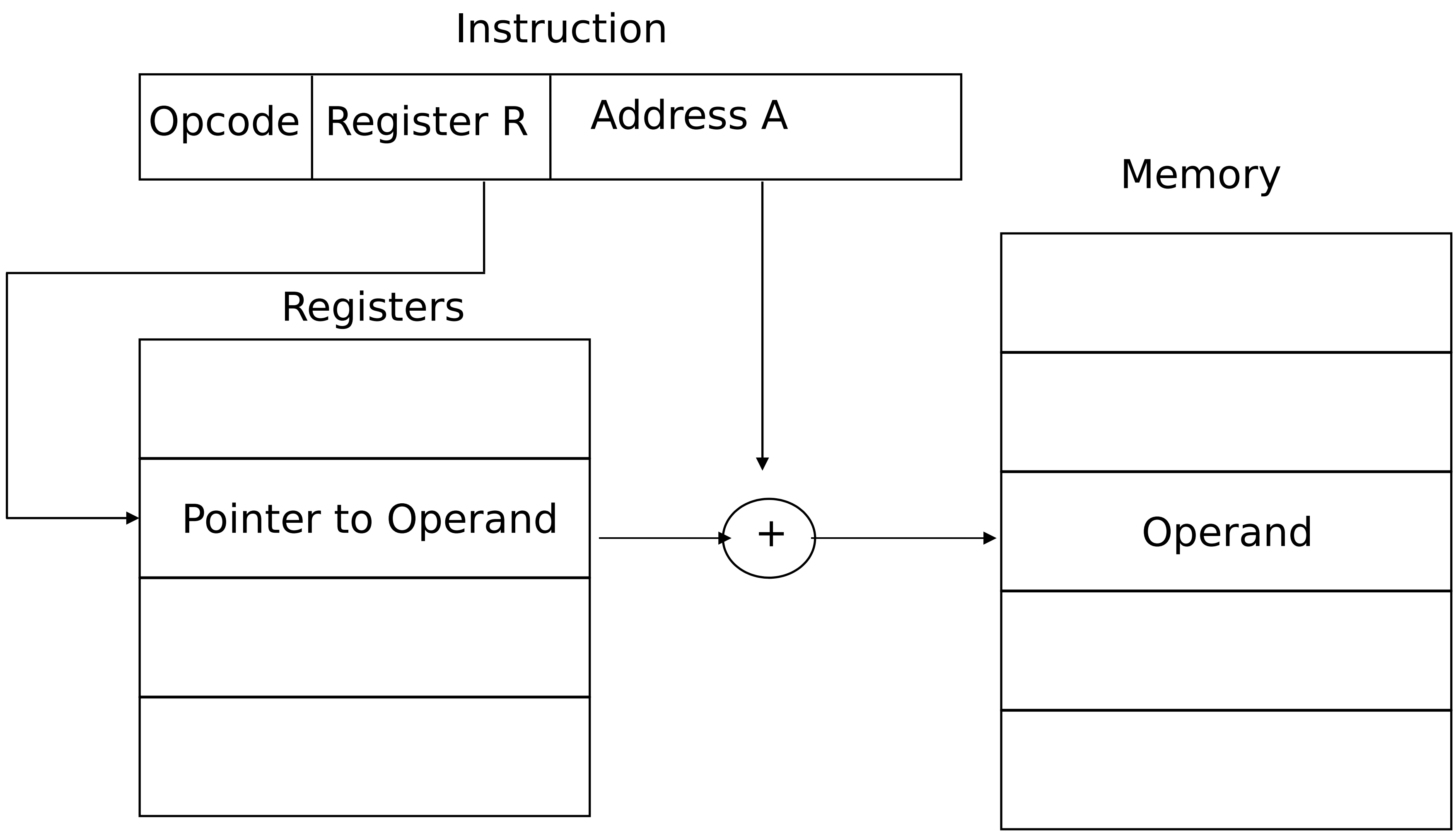
Register Indirect Addressing Diagram



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++