Memory Addressing

- Instruction Formats
- Addressing Modes
- Intel 8086 Architecture

Memory Resources

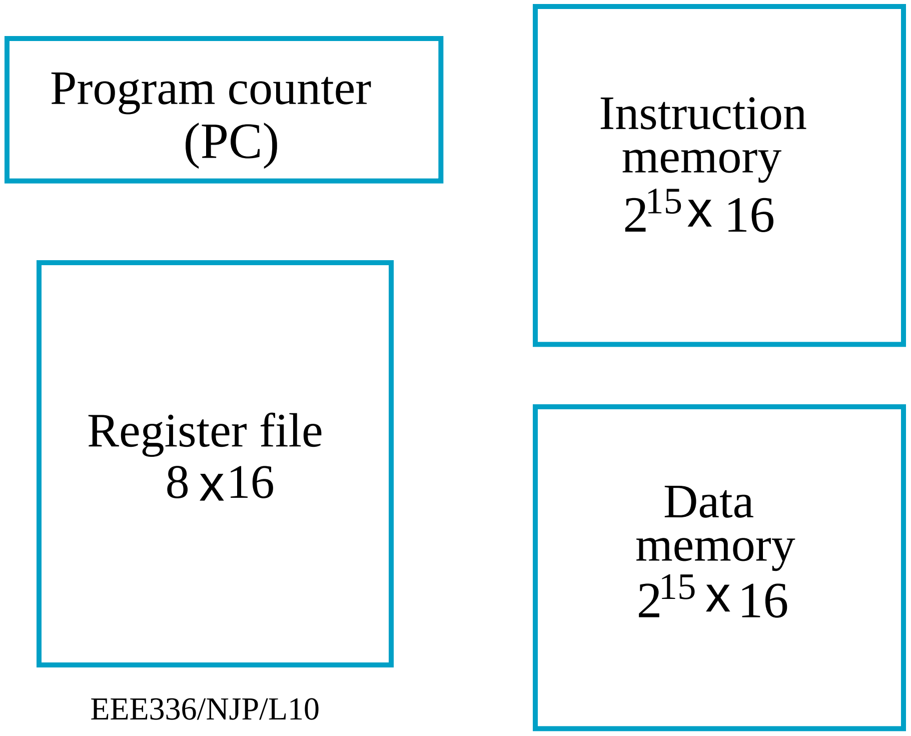
Instructions are stored in RAM or ROM as a *program*.

The program is executed, a line at a time, with addresses provided by a counter known as the *Program Counter*.

The program counter can count up or load a new address based on an instruction and, optionally, information from status flags.

These resources are visible to the programmer at the assembly level.

These sizes are just examples.


Program counter (PC)
Instruction memory $2^{15} \times 16$
Register file 8×16
Data memory $2^{15} \times 16$

Program Instructions

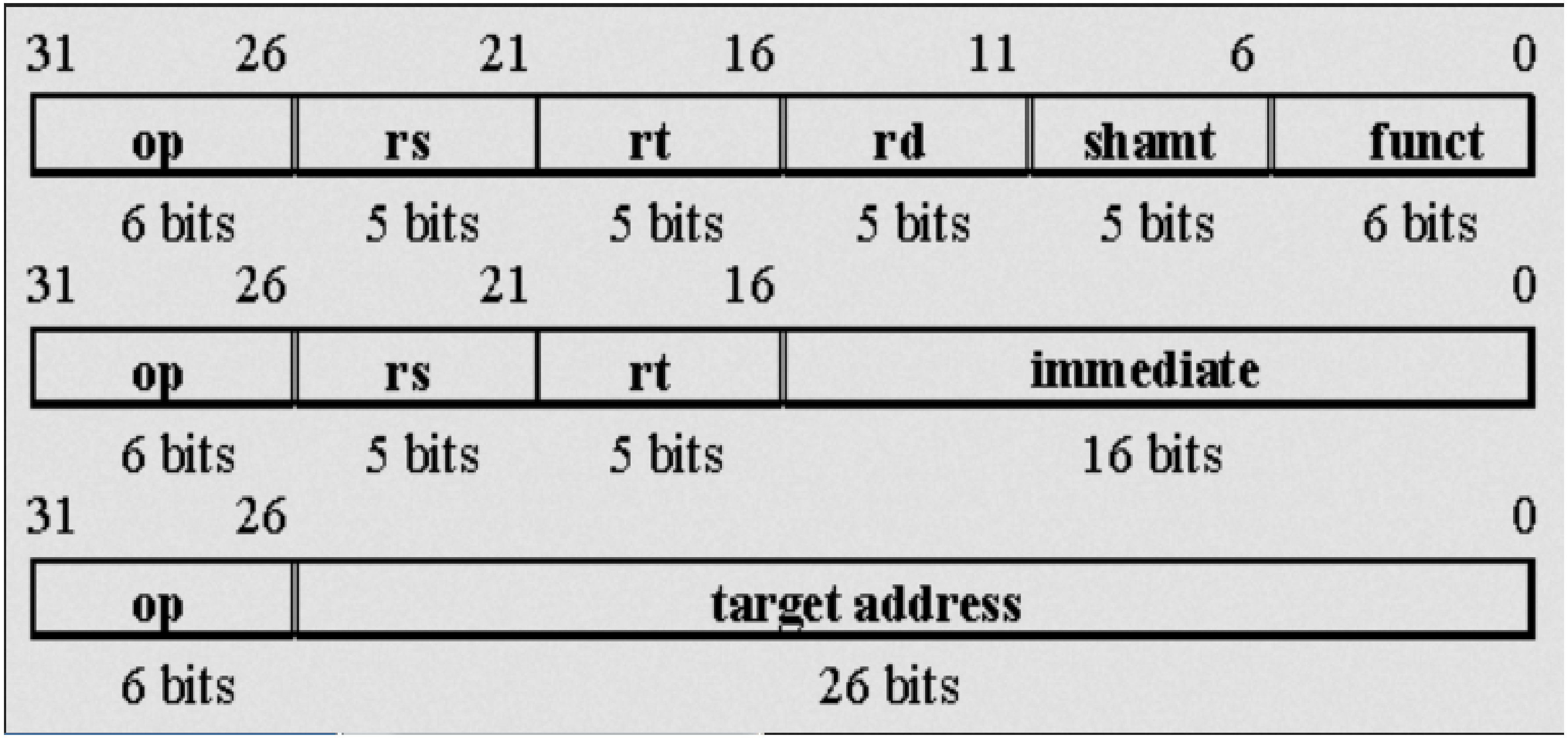
A microprocessor uses a sequence of *instructions* to control its operation.

An typical instruction specifies:

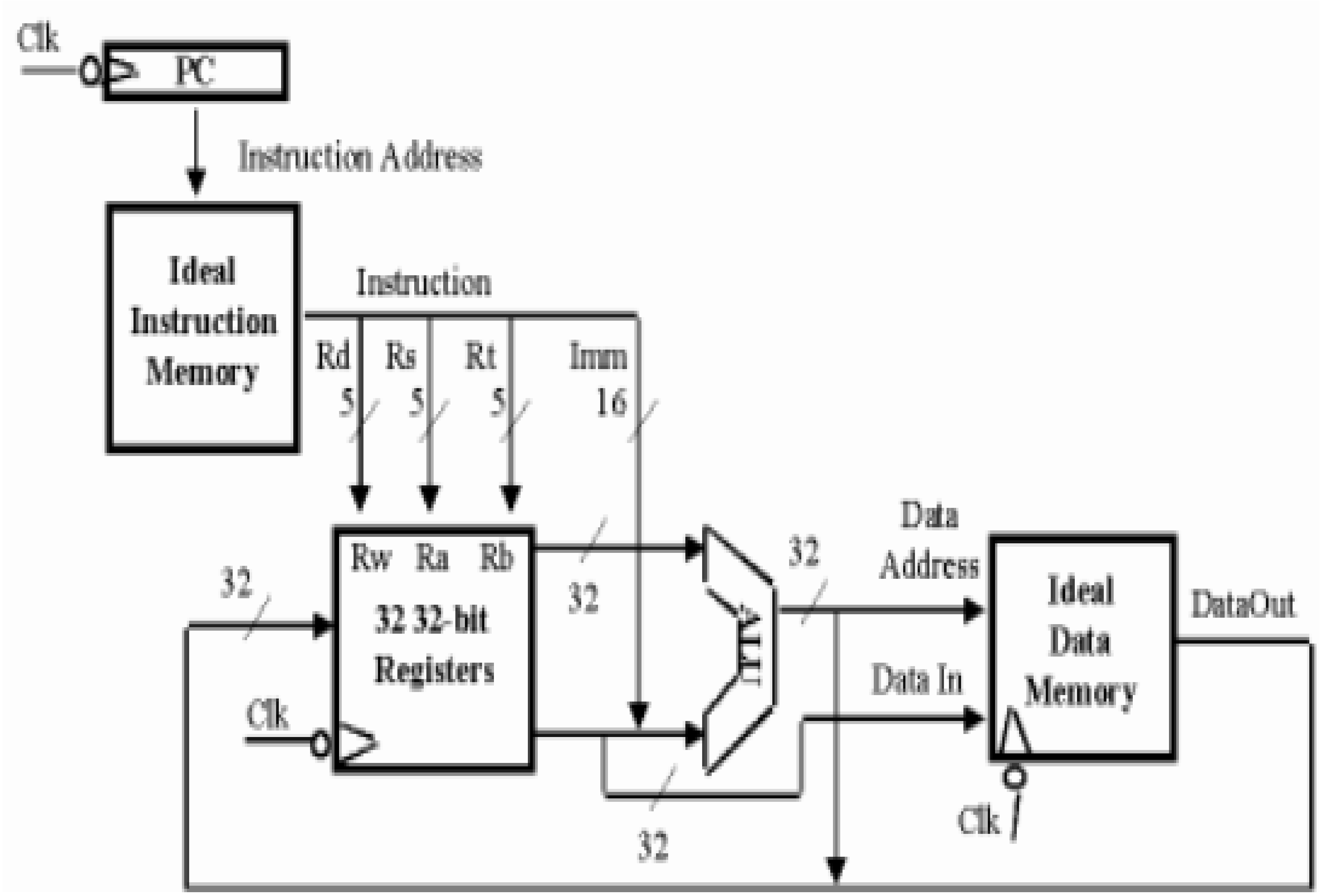
- the operation to be performed (opcode)
- the operands to use for the operation
- where to place the results of the operation
- which instruction to execute next (in some cases)

Example: 32 bit MIPS instruction format

opcode	rs	rt	rd	shift amount	function variant
6	5	5	5	5	6



- Variations of the instruction format are possible and can be interpreted from the opcode.
- In this example, the 5-bit register address enables 32 registers to be addressed in a register file.


Clk
PC
Instruction Address
Ideal Instruction Memory
Instruction
Rd
Rs
Rt
Imm
5
5
5
16
Rw Ra Rb
32 32-bit Registers
32
32
Clk
32
ALU
32
Data Address
Data In
Ideal Data Memory
DataOut
Clk

Internal Register Transfer Instructions

Instructions are separated into fields. The number of bits in the opcode depends upon the number of operations in the instruction set.

$$m \text{ bits gives } 2^m \text{ operations}$$

The designer chooses the bit pattern for each operation.

15 — 9	8 — 6	5 — 3	2 — 0
Opcode	Destination register (DR)	Source register A (SA)	Source register B (SB)

(a) Register

15 — 9	8 — 6	5 — 3	2 — 0
Opcode	Destination register (DR)	Source register A (SA)	Operand (OP)

(b) Immediate

15 — 9	8 — 6	5 — 3	2 — 0
Opcode	Address (AD) (Left)	Source register A (SA)	Address (AD) (Right)

(c) Jump and Branch

(a) Register

15 9	8 6	5 3	2 0
Opcode	Destination register (DR)	Source reg- ister A (SA)	Source reg- ister B (SB)

This format supports instructions represented by:

R1 ← R2 + R3

R1 ← shl R2

There are three 3-bit register fields:

DR - specifies destination register (R1 in the examples)

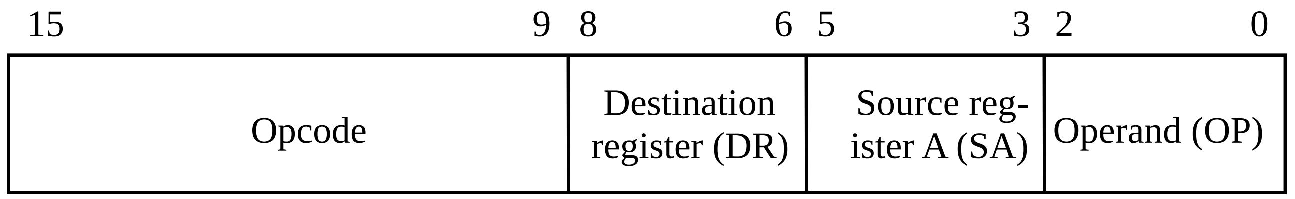
SA - specifies the A source register (R2 in the first example)

SB - specifies the B source register (R3 in the first example and R2 in the second example)

Why is R2 in the second example SB instead of SA?

The source for the shifter in the datapath to be used in this implementation is Bus B rather than Bus A

(b) Immediate



(b) Immediate

This format supports instructions described by:

R1 ← R2 + 3

The B Source Register field is replaced by an Operand field OP which specifies a constant.

The Operand:

3-bit constant

Values from 0 to 7

(c) Jump and Branch

15 — 9	8 — 6	5 — 3	2 — 0
Opcode	Address (AD) (Left)	Source register A (SA)	Address (AD) (Right)

This instruction supports changes in the sequence of instruction execution by adding an extended, 6-bit, signed 2s-complement *address offset* to the Program Counter value

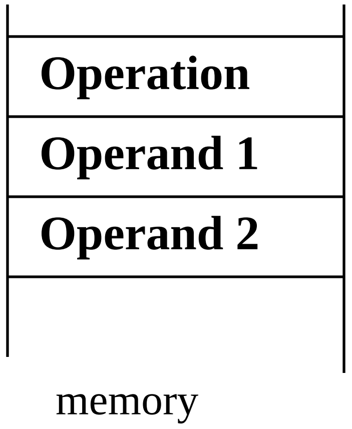
The 6-bit Address (AD) field replaces the DR and SB fields

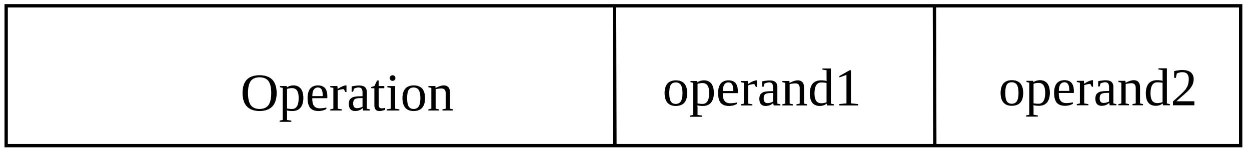
> Example: Suppose that a jump is specified by the Opcode and the PC contains 45 (0…0101101) and Address contains –12 (110100). Then the new PC value will be:
> 0…0101101 + (1…110100) = 0…0100001 (45 + (–12) = 33)

The SA field is retained to permit jumps and branches based on the contents of Source register A (e.g. JZ , jump on zero)

Memory Transfer Instructions

- Most instructions require data
- How is the data derived from the operand of the instruction?
- Any instruction may consist of two parts:
  - operation
  - operand(s)
- usually stored consecutively in main memory
- Operations can take zero, one or two operands





typical instruction format

Within the operation code, there will be bits which describe the addressing mode for operands 1 and 2. The addressing mode relates operand to actual data. There are several ways in which the operands of the instruction relate to the data:

- Implied
- Immediate
- Register
- Memory

*The operand is **implied** by the instruction itself*

*Only a 'source' operand. The operand **is** the data*

Data will be found in a register in the μP. If there are 8 registers then 3 bits are needed to address the registers

Most operands are derived from memory and there are a number of ways in which this can be done

Implied Mode

The operand is specified implicitly in the in the definition of the opcode. An example of this is the Intel 8085 command CMA.

This is an instruction to complement the accumulator. It is an implied mode instruction because the operand in the accumulator register is implied in the definition of the instruction.

Immediate Addressing

The operand is specified in the instruction itself. Useful for initializing registers to a constant value.

Advantages:

- Immediate operands are fetched from memory at the same time as the instruction itself and are thus immediately available for use
- Does not require an extra memory reference to fetch operand
- Fast
- Useful when small integer constants are required

Disadvantages:

- Only a constant can be supplied
- Value of constant is limited by the size of the field

Register Addressing

Operand is held in a processor register R which is named in the address field.

Advantages:

- Only a small address field required
- Shorter instructions
- No memory access, very fast execution
- Useful for variable which are accessed frequently such as loop variables

Disadvantages:

- The number of registers is limited
- Limited address space

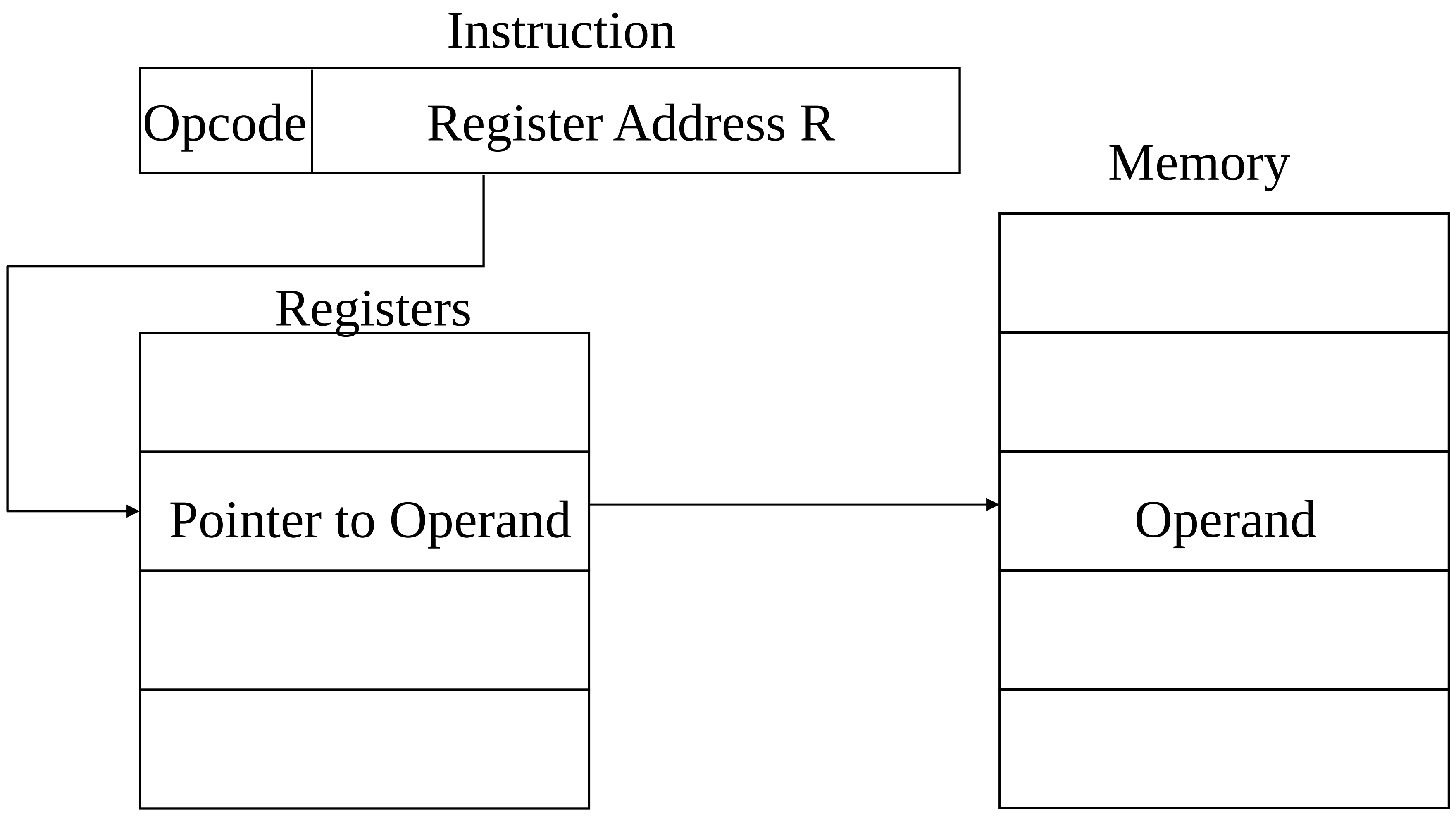
Register-Indirect Mode

Operand is held in a memory cell pointed to by the contents of register R, i.e. the register in the processor contains the address of the operand in memory. This gives a larger address space than Register Direct addressing but requires one more memory access.

Effective Address = [R]

Auto-increment or auto-decrement are similar to register-indirect mode. In this mode, the address in the register is incremented or decremented after it has been used. This is useful when dealing with an array of data in memory.

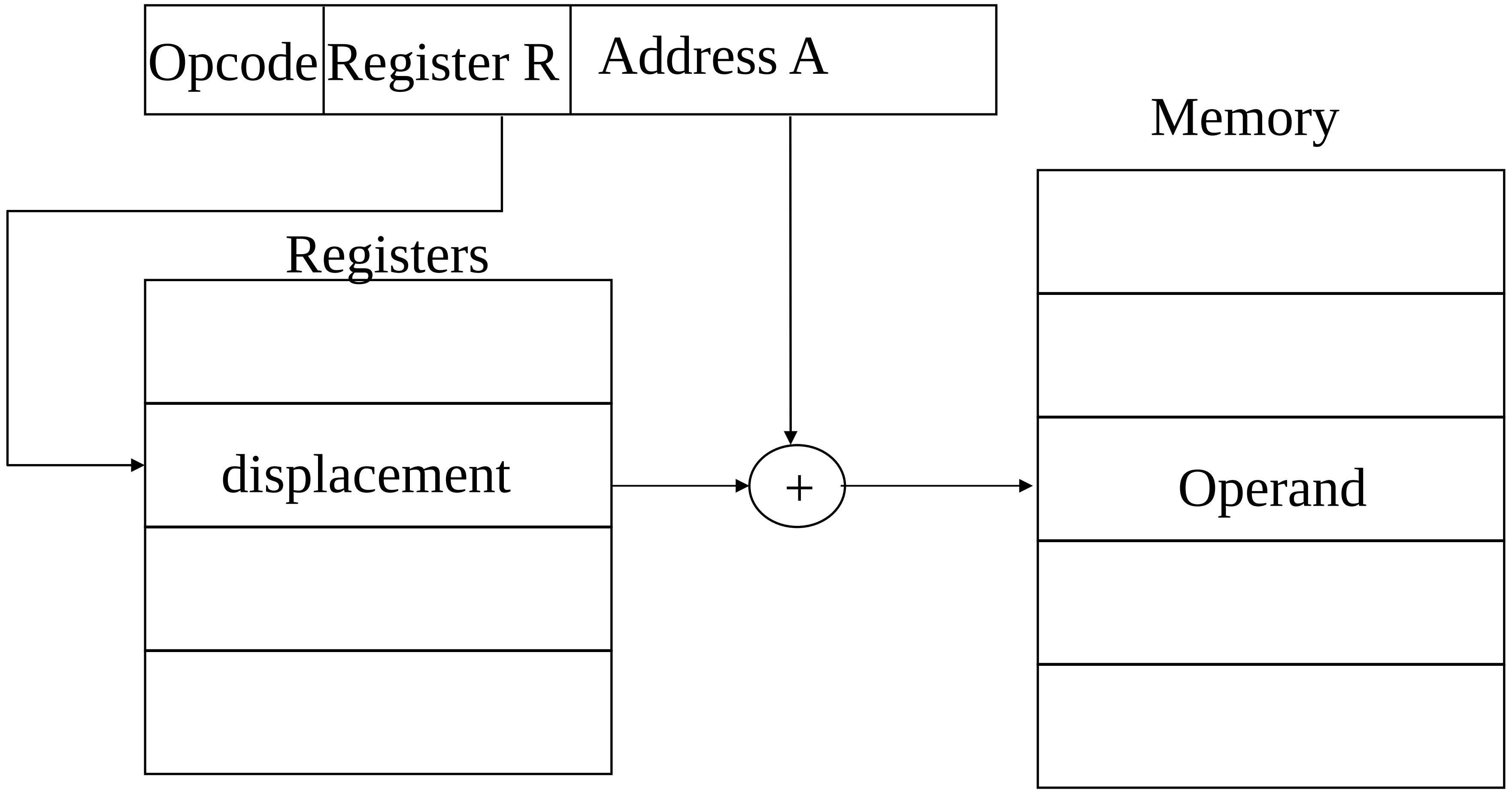
Register Indirect Addressing



Displacement Addressing

A = base value, R = register that holds displacement

Effective address = A + [R]



Versions of Displacement Addressing

Relative Addressing

A version of displacement addressing

R = Program Counter

Effective address = 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

e.g. segment registers in 80x86

Direct Memory Addressing

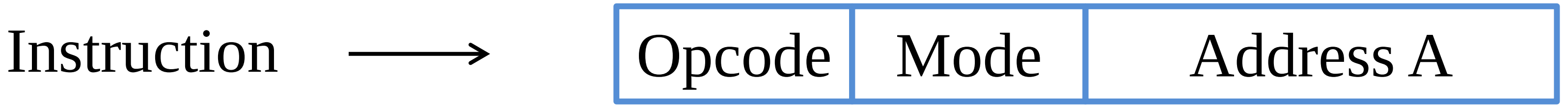
Instruction address field gives the address of the operand in memory.

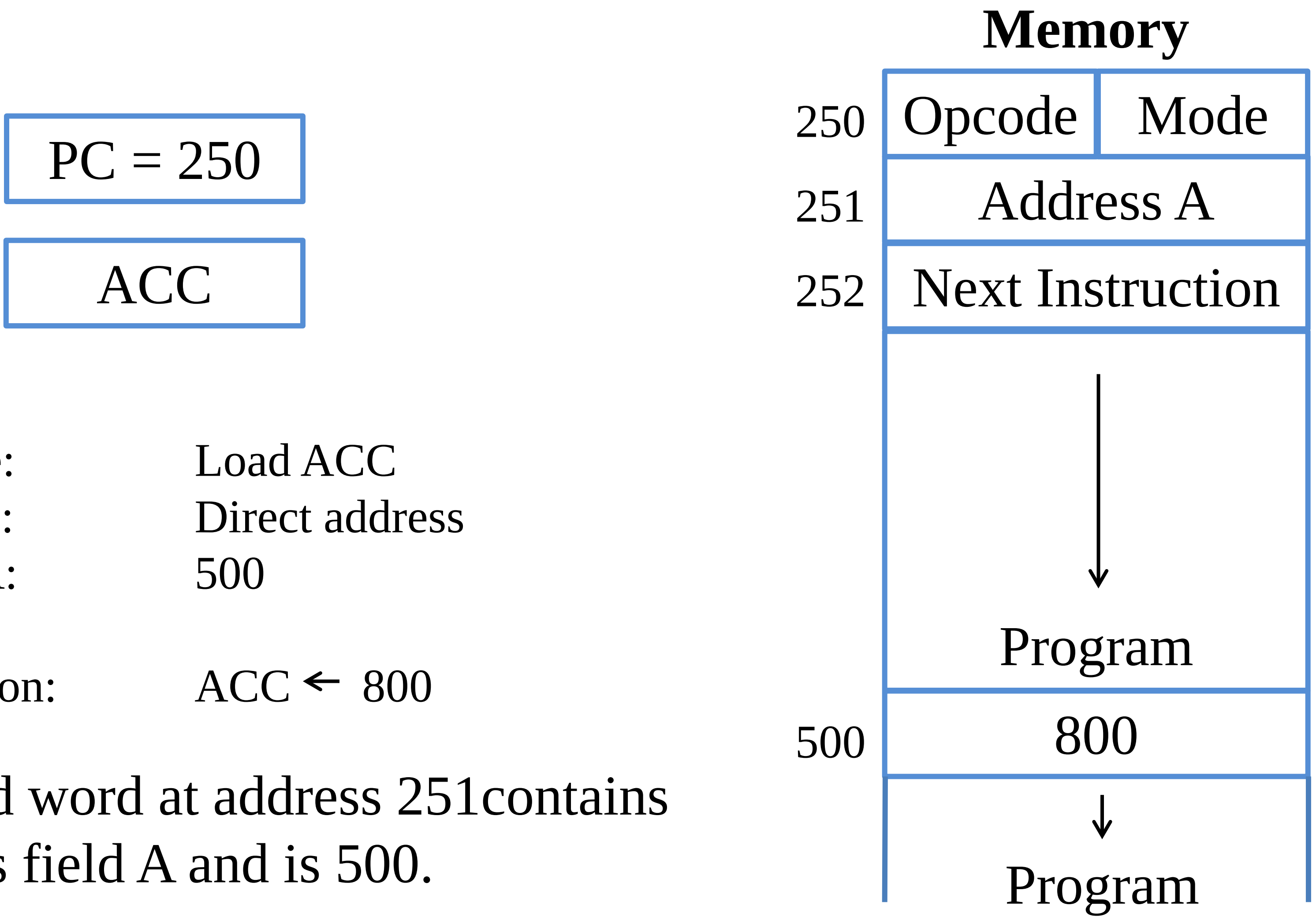
Advantages:

- Single memory access to retrieve data
- No calculations to work out the effective address of the data
- Fast access for global variables which are widely used

Disadvantages:

- Can only be used to access global variables whose address is known when the program is compiled
- Instruction will always access the same memory location. The value can change but the location cannot
- Limited address space





Opcode:	Load ACC
Mode:	Direct address
A:	500
Operation:	ACC ← 800

The second word at address 251contains the address field A and is 500.

Instruction $\longrightarrow$

Opcode	Mode	Address A

Two memory accesses are required to retrieve the complete instruction and the PC is then incremented to point to the next instruction at 252.

Execution of the instruction results in the operation:

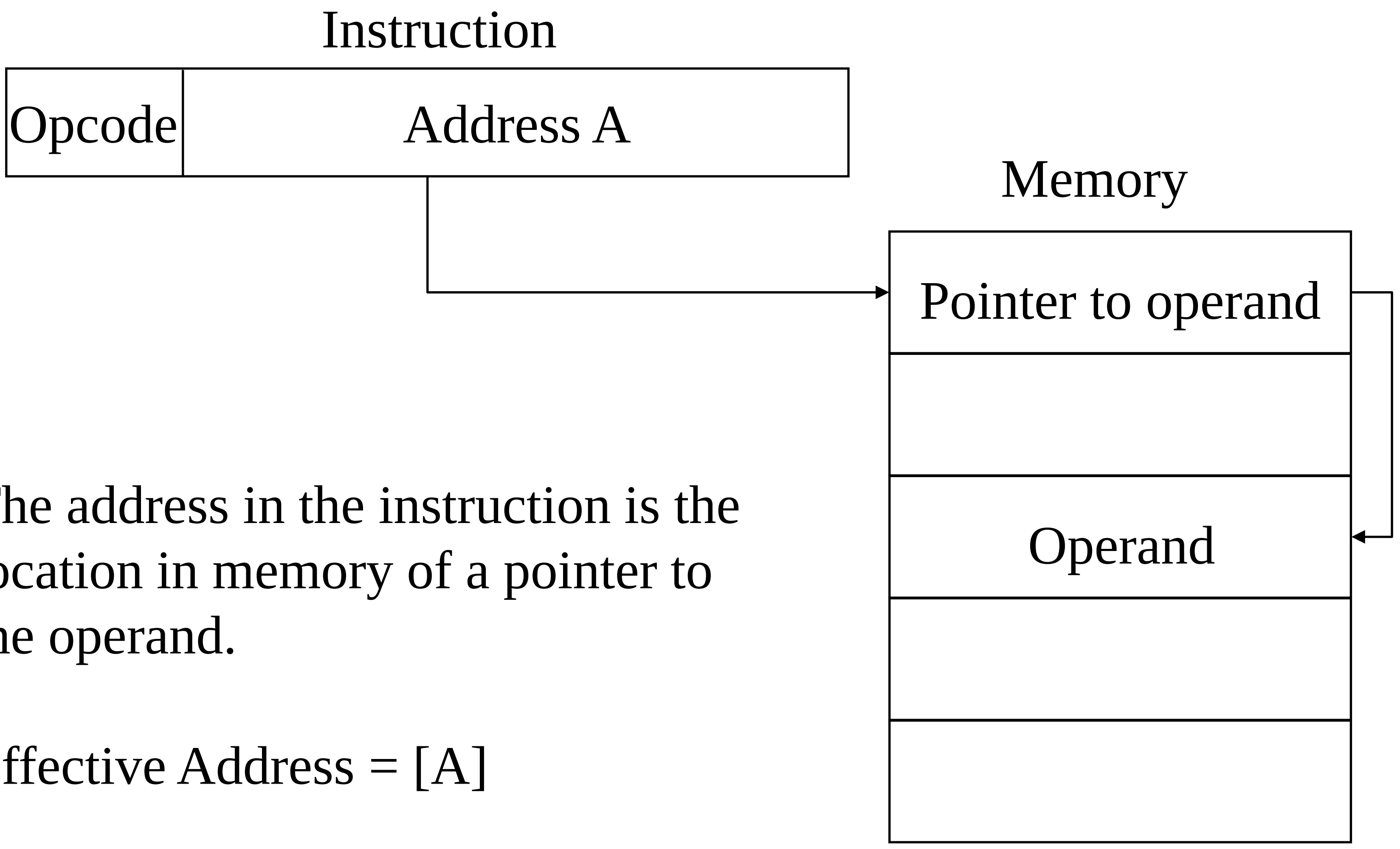
$$ACC \leftarrow M[A]$$

As **A = 500** and **M[500] = 800**, the accumulator receives the value **800**.

$$\boxed{PC = 252}$$

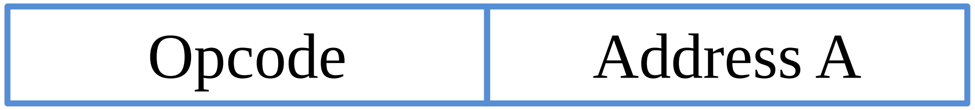
$$\boxed{ACC = 800}$$

Indirect Memory Addressing



The address in the instruction is the location in memory of a pointer to the operand.

Effective Address = [A]



Advantages:

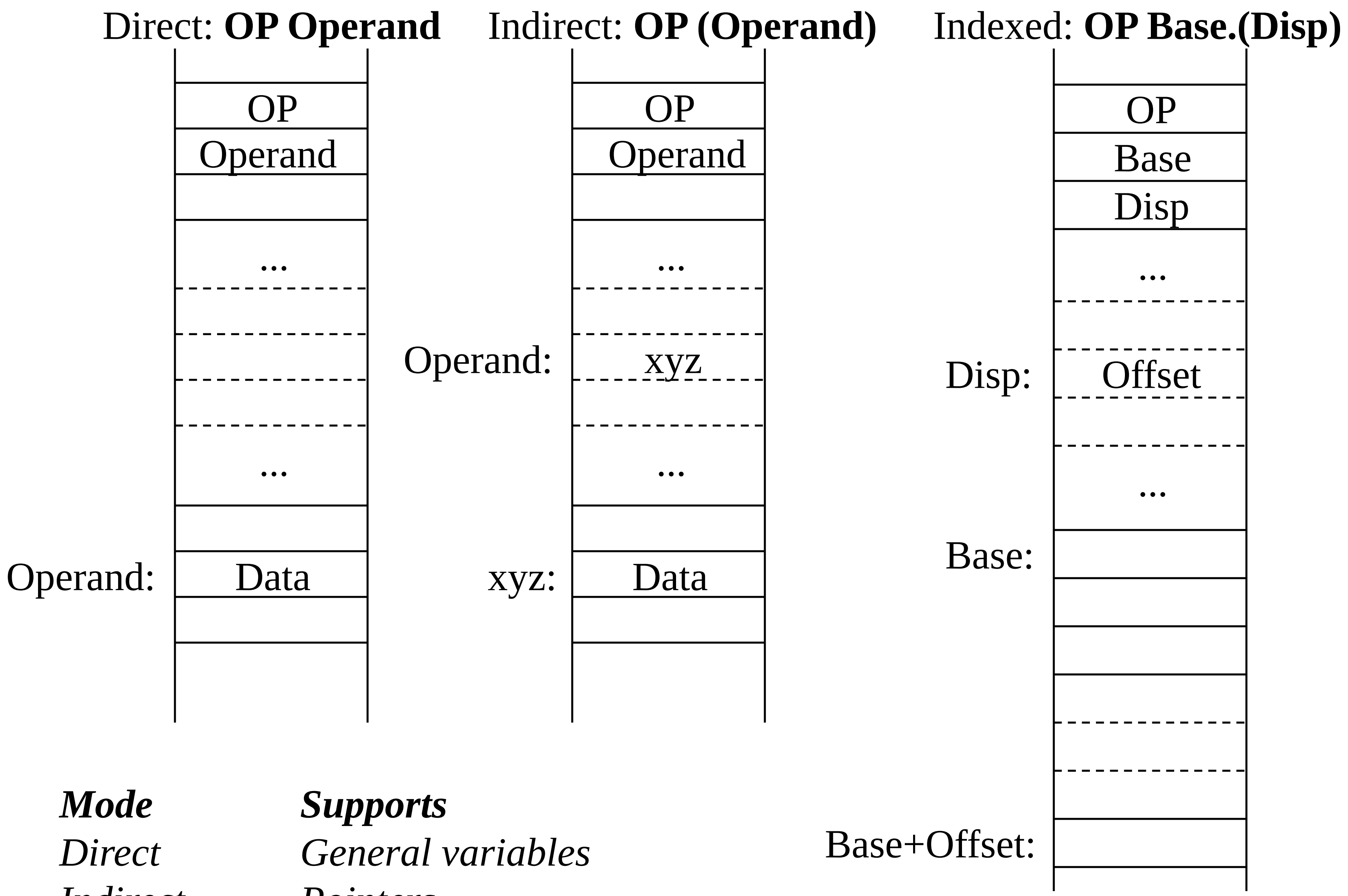
- Large address space
- May be nested – effective address = [[[A]]]

Disadvantages:

- Multiple memory accesses are required to obtain the operand
- Thus, slower than direct addressing

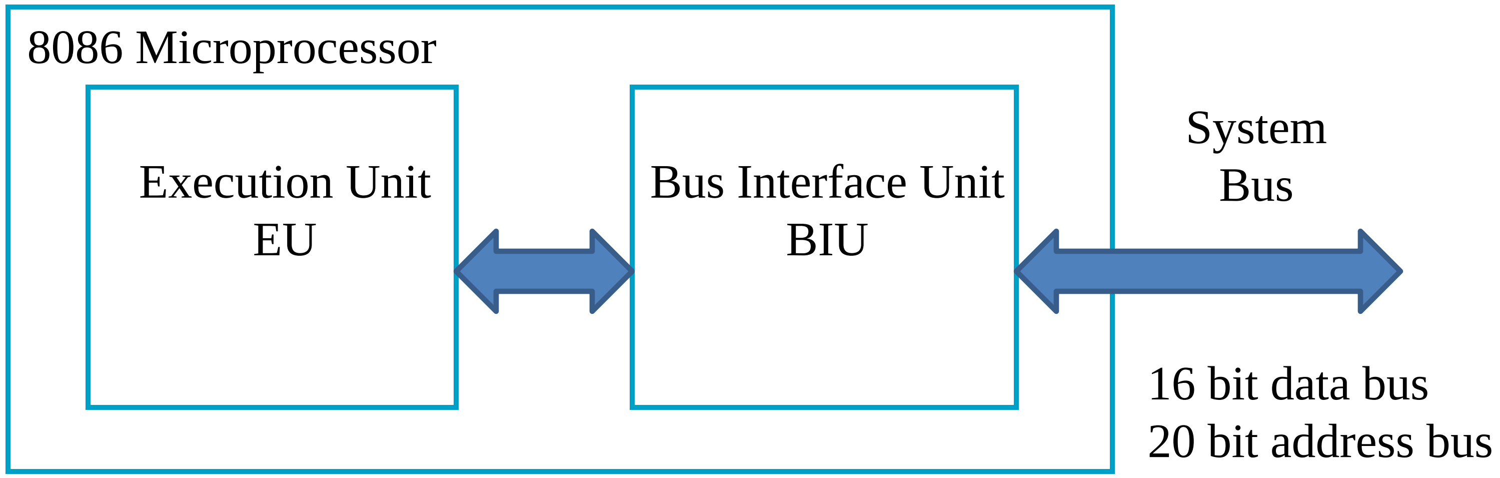
Indexed Addressing

Works in a similar way to Base-Register Addressing. The instruction contains a base address and a displacement.



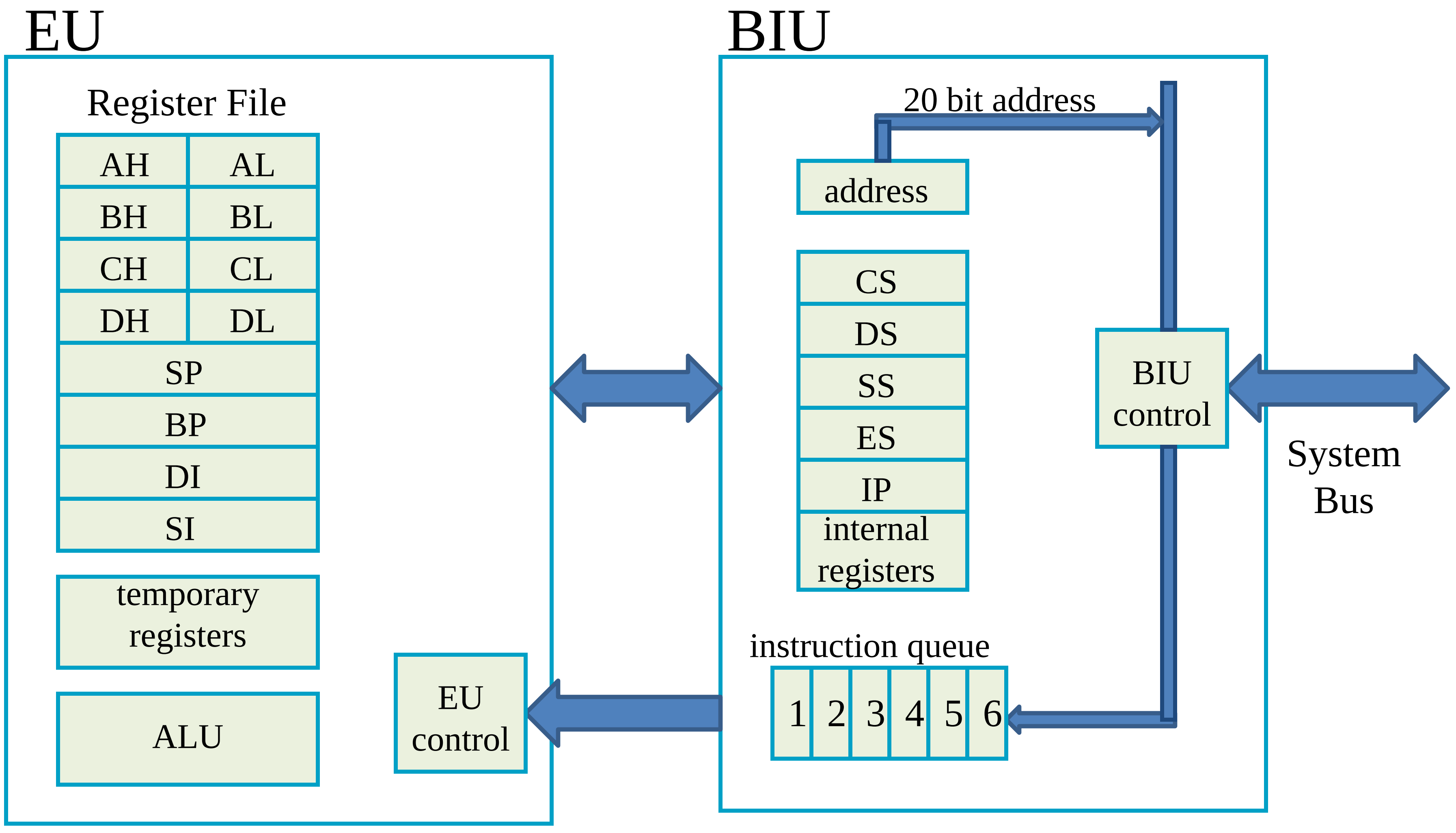
Mode	***Supports***
Direct	*General variables*
Indirect	*Pointers*
Indexed	*Data Arrays*

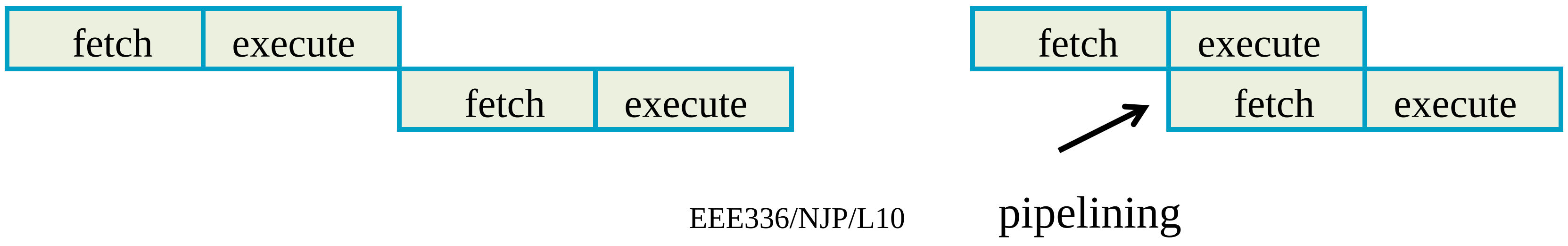
Intel X86 Architecture



EU – executes instructions
BIU – fetches instructions, reads operands, writes results

While instructions are executing, the BIU can fetch more instructions from memory which it holds in a queue. This improves performance because the processor can execute an instruction at the same time as fetching an instruction from memory. This is known as pipelining.


EU
Register File
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
DI
SI
temporary registers
ALU
EU control
BIU
20 bit address
address
CS
DS
SS
ES
IP
internal registers
BIU control
System Bus
instruction queue
1
2
3
4
5
6

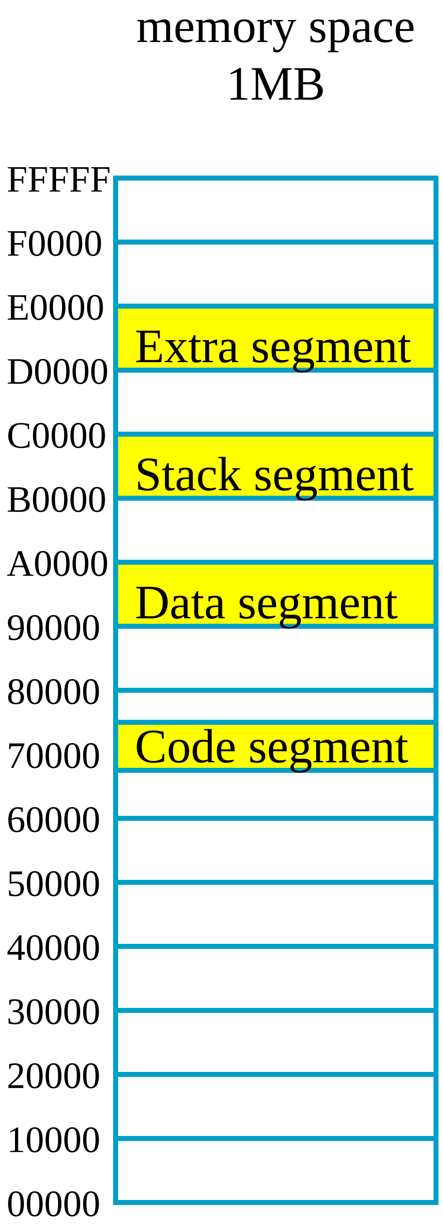

fetch
execute
fetch
execute
fetch
execute
fetch
execute
pipelining

Segment Registers

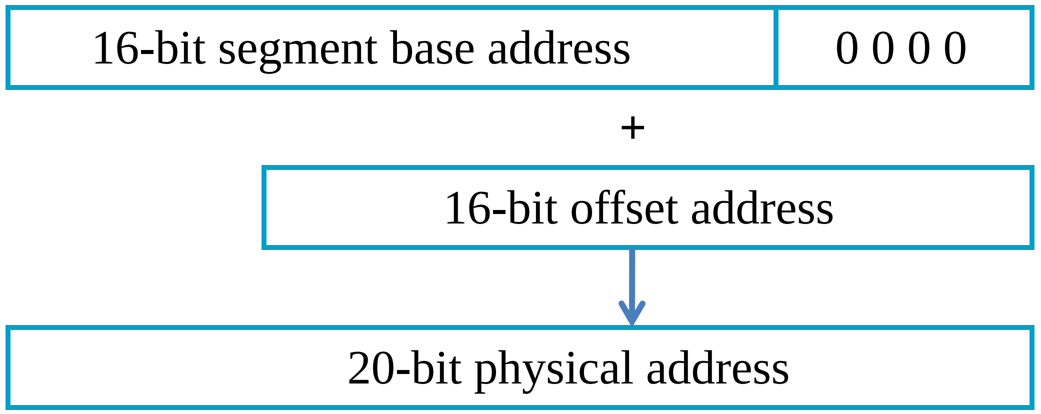
In the 8086, 20 address bits can address 1MB of data.
It has 16 bit registers and data, so how is the 20 bit address formed?

There are four 16 bit segment registers (CS, DS, SS, ES)
Each represents the starting address of a 64K block of memory.

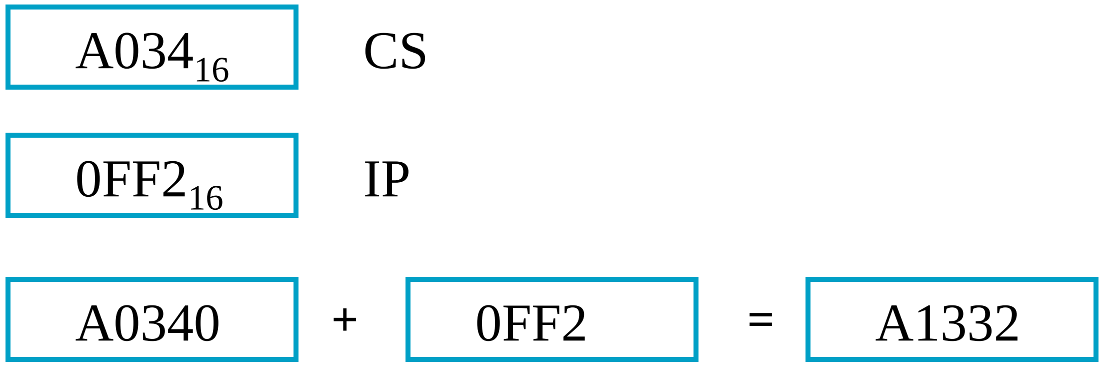
To form an address, a segment address is combined with an offset address contained in the instruction pointer (IP)



The segment address is shifted four bits to the left and added to the offset address.



eg



The address is shown in assembly language as CS:IP
This strategy makes it easy to relocate code within the memory space.