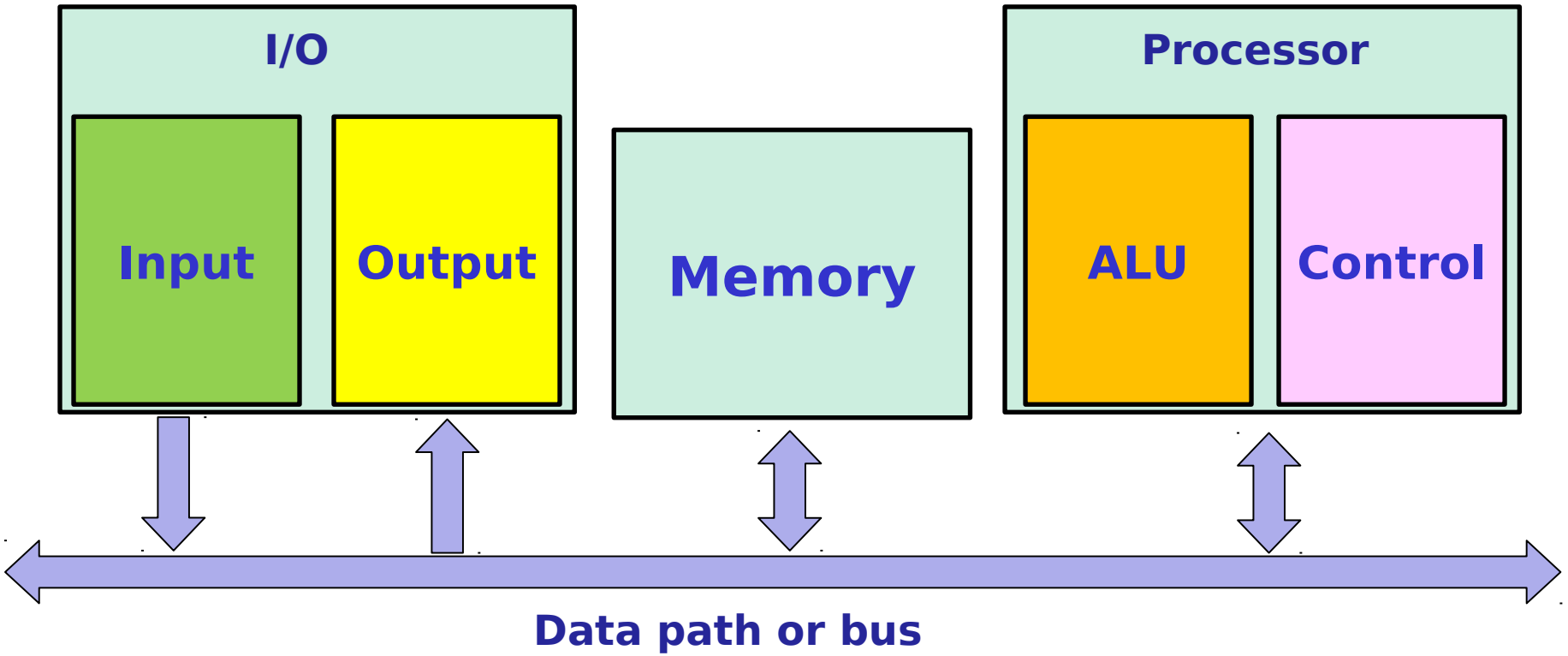
Instruction Set Architecture

Microarchitecture Level



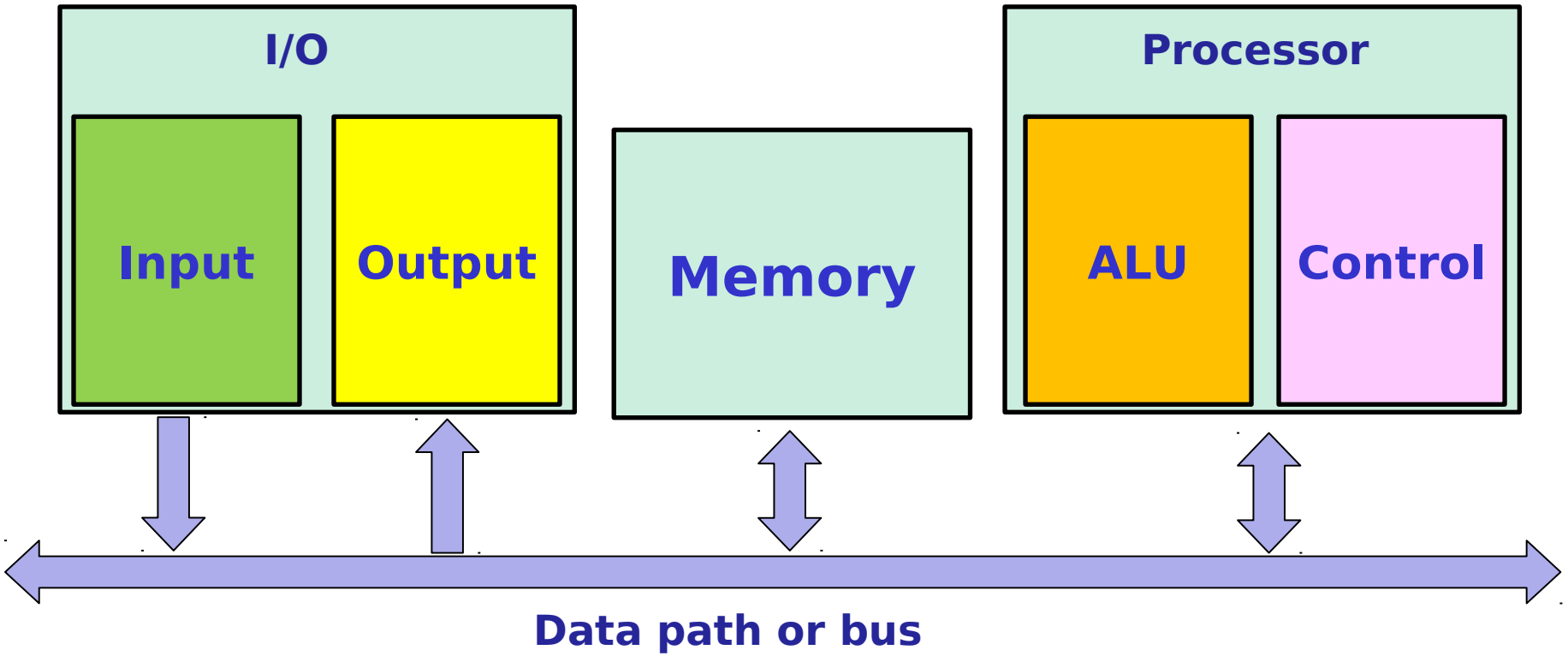
Instruction Set Architecture

- Compiler
  - Converts a high-level language program into a sequence of machine level instructions

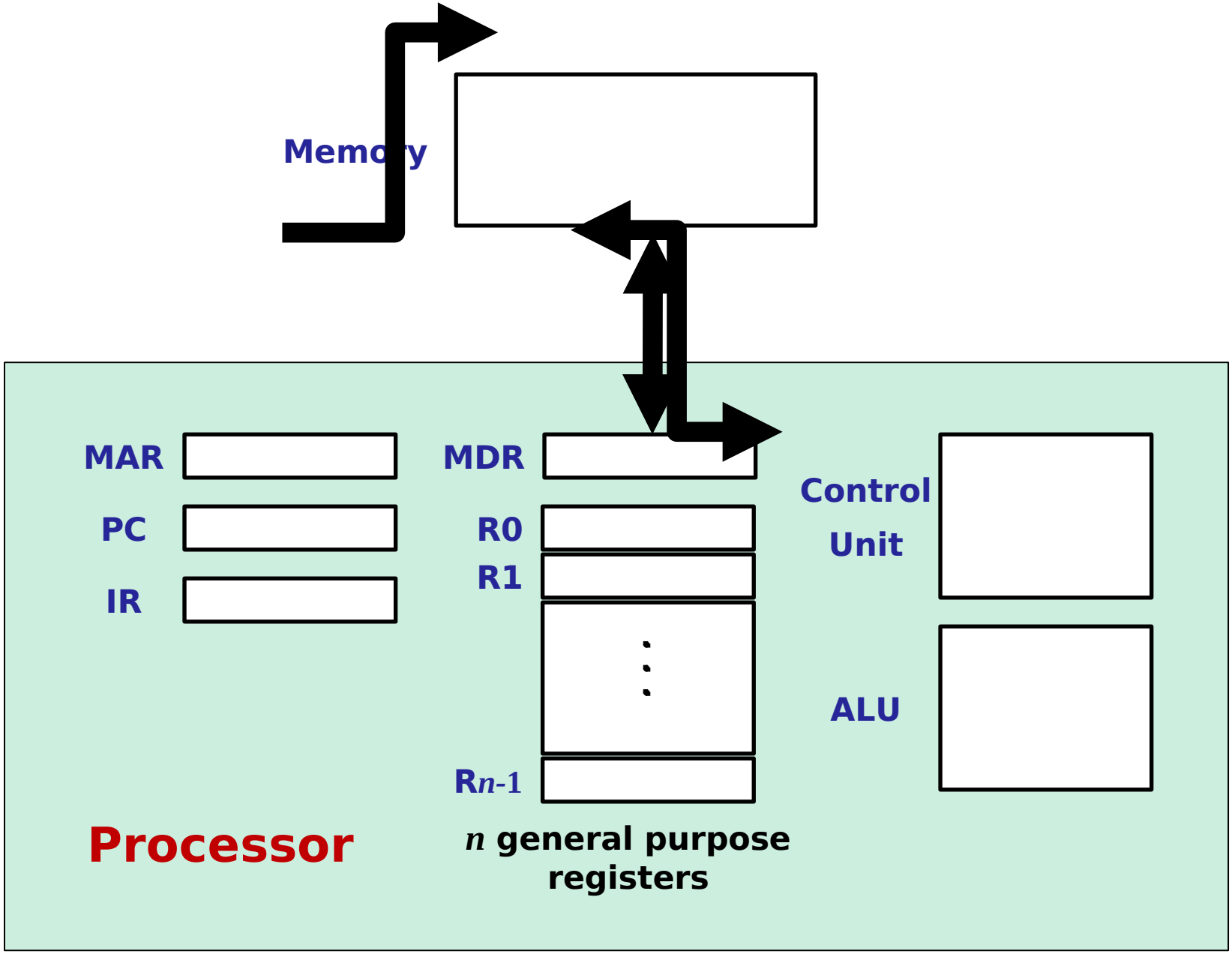
- Instruction set architecture
  - Interface between the high-level language and the machine language

  - Instruction set
  - Instruction formats
  - Addressing modes
  - Instruction representation

Microarchitecture Level



Operational Details



Instruction Set Architecture

- Instruction set architecture
  - **Instruction set**
  - Instruction formats
  - Addressing modes
  - Instruction representation

Instruction Set

- Instructions
  - Logical instructions
    - AND, OR, XOR, Shift
  - Arithmetic instructions
    - Data types
      - Integers: Unsigned, Signed, Byte, Short, Long
      - Real numbers: Single-precision (float), Double-precision (double)
    - Operations
      - Addition, Subtraction, Multiplication, Division
  - Data transfer instructions
    - Register transfer: Move
    - Memory transfer: Load, Store
    - I/O transfer: In, Out
  - Control transfer instructions
    - Unconditional branch
    - Conditional branch
    - Procedure call
    - Return

Instruction Set Architecture

- Instruction set architecture
  - Instruction set
  - **Instruction formats**
  - Addressing modes
  - Instruction representation

Instruction Format

- Instruction contains operation and operand
- 3-operand instructions
  - ADD op1, op2, op3; op1 ←op2 + op3
- 2-operand instructions
  - ADD op1, op2; op1 ← op1 + op2
- 1-operand instructions
  - INC op1; op1 ← op1 + 1
- 0-operand instructions
  - Operands in stack
- Effect of instruction format:
  - Instruction length
  - Number of instructions for a program
  - Complexity of instruction decoding (Control unit)

Instruction Set Architecture

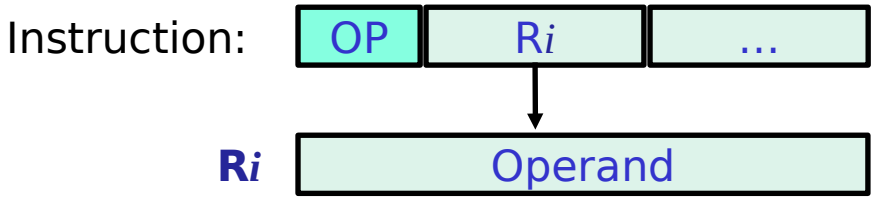
- Instruction set architecture
  - Instruction set
  - Instruction formats
  - **Addressing modes**
  - Instruction representation

Addressing Mode

- Specification of operands in instructions
- Different addressing modes:
  - Immediate: Value of operand

    - Example: ADD #3
    - Constant

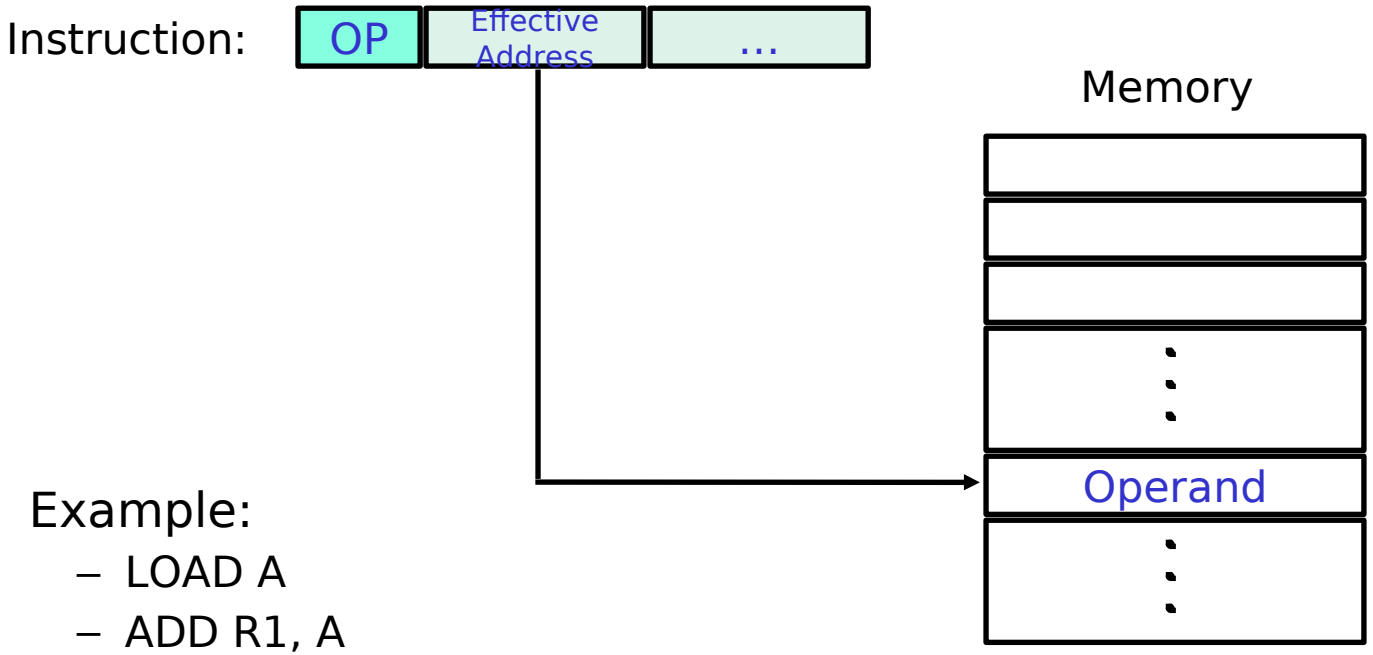
  - Register: Value of operand in a register



    - Example:
      - ADD R1, R2
      - ADD R1, #3
      - LOAD R1
    - Local variable

Addressing Mode

- Specification of operands in instructions
- Different addressing modes:
  - Memory direct: Address of operand

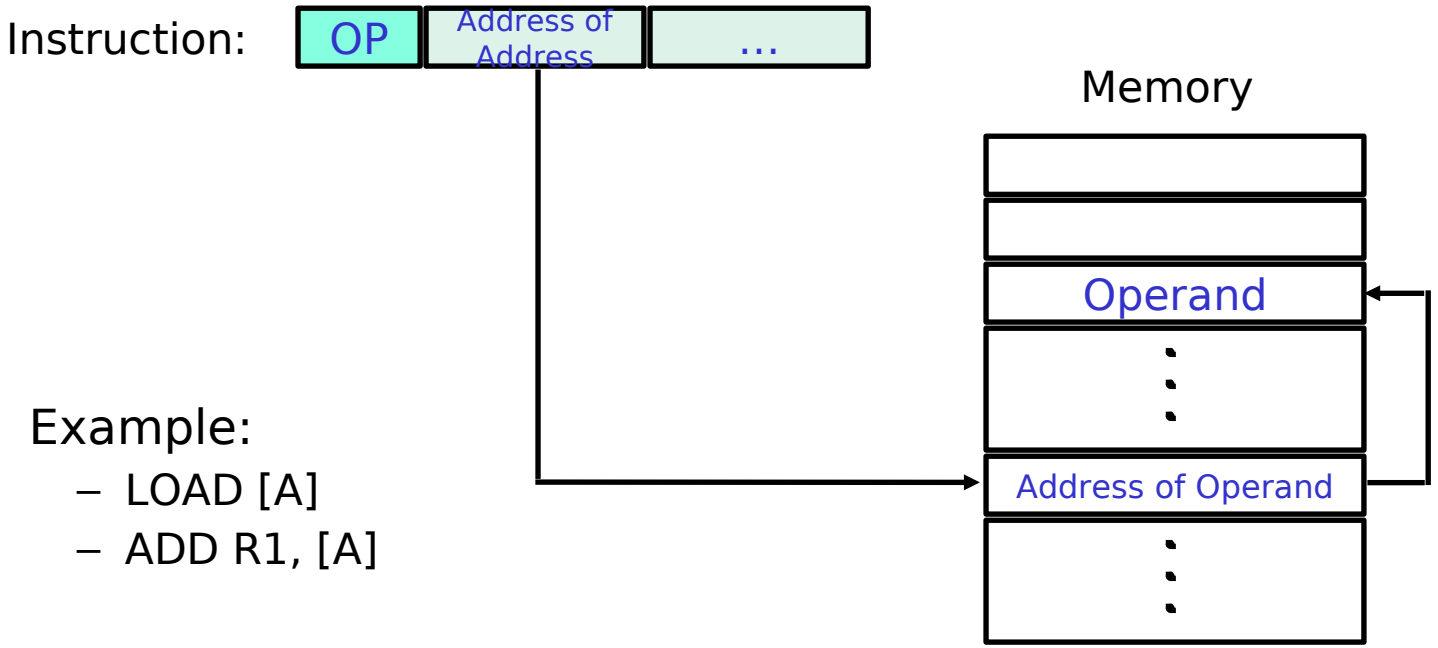


- Example:
  - LOAD A
  - ADD R1, A

- Global variable

Addressing Mode

- Specification of operands in instructions
- Different addressing modes:
  - Indirect: Address of the address of operand in instruction

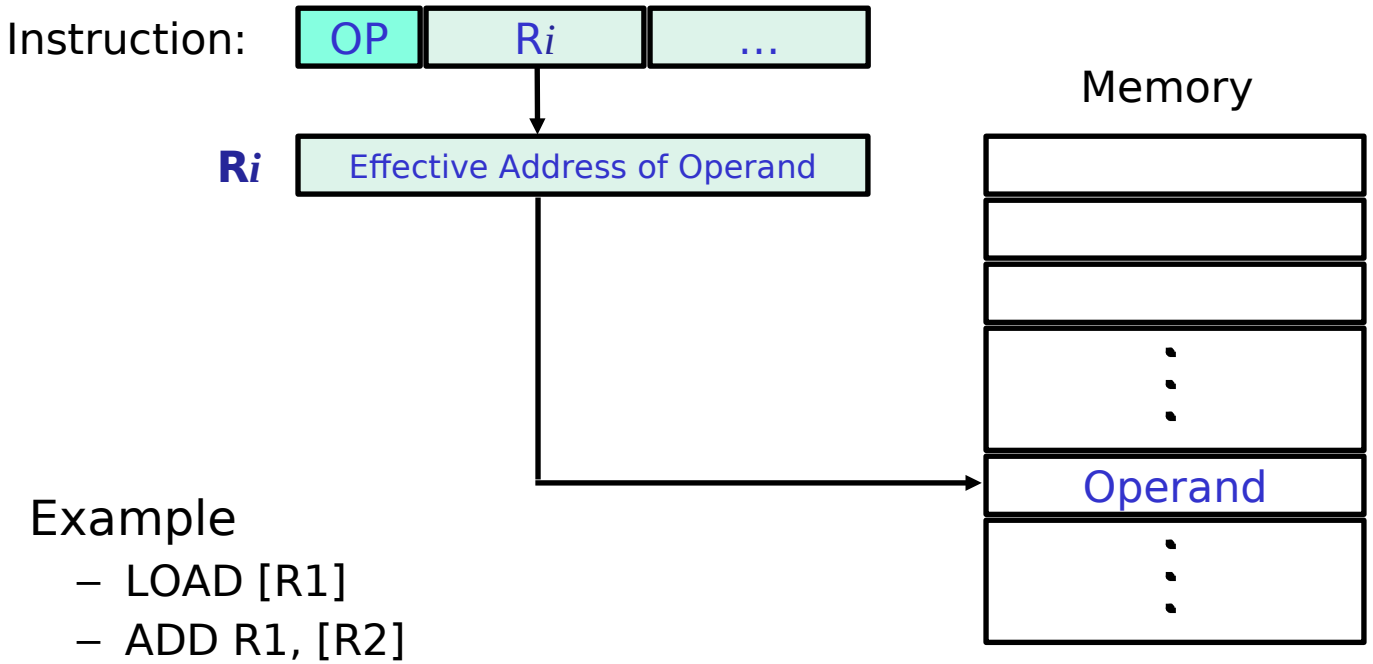


- Example:
  - LOAD [A]
  - ADD R1, [A]

- Pointer

Addressing Mode

- Specification of operands in instructions
- Different addressing modes:
  - Register indirect: Address of operand in a register



- Example
  - LOAD [R1]
  - ADD R1, [R2]

Compute Sum of marks obtained by students in three subjects

N	n
LIST	SID
	Test 1
	Test 2
	Test 3
	SID
	Test1
	Test 2
	Test 3

Number of student

Student 1

Student 2

Each field takes 4 bytes of memory

Instruction Set Architecture

Register Indirect

```
MOV R0, N          MOV R6, R7
CLEAR R4           L1 ADD  R1, (R4)
CLEAR R5          ADD R2, (R5)
CLEAR R6           ADD R3, (R6)
CLEAR R1          ADD  R4, #16
CLEAR R2          ADD R5, #16
CLEAR R3          ADD R6, #16
CLEAR R7          DEC R0
MOV R7, LIST      Branch > 0   L1
ADD R7, #4        MOV SUB1, R1
MOV R4, R7        MOV  SUB2, R2
ADD R7, #4        MOV SUB3,  R3
MOV R5, R7        END
ADD R7, #4
```

Addressing Mode

- Specification of operands in instructions
- Different addressing modes:
  - Indexed: Base register, Index register

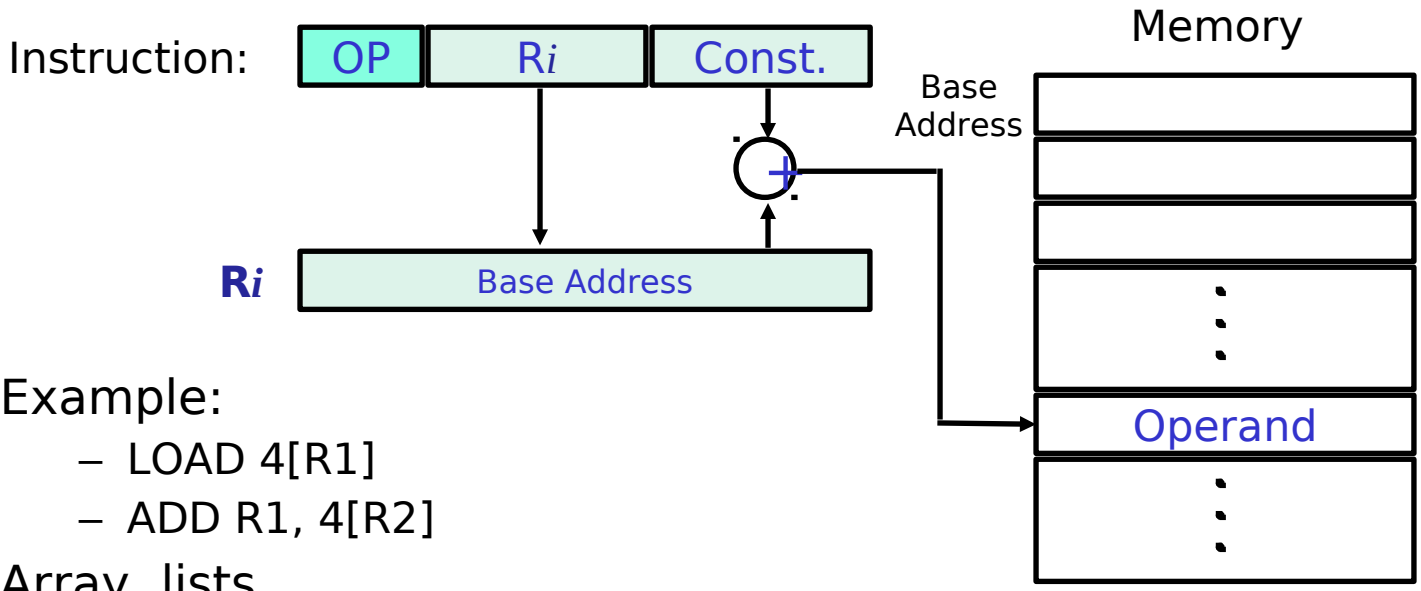
Effective Address (EA) of the operand = [Register] + Constant or Offset

EA = [R*i*] + Constant or offset

•Register Contains base address

•Register used may be special registers or any general purpose registers

- Base registers or Index registers



•Example:

- LOAD 4[R1]
- ADD R1, 4[R2]

•Array, lists

Instruction Set Architecture

Register Indirect

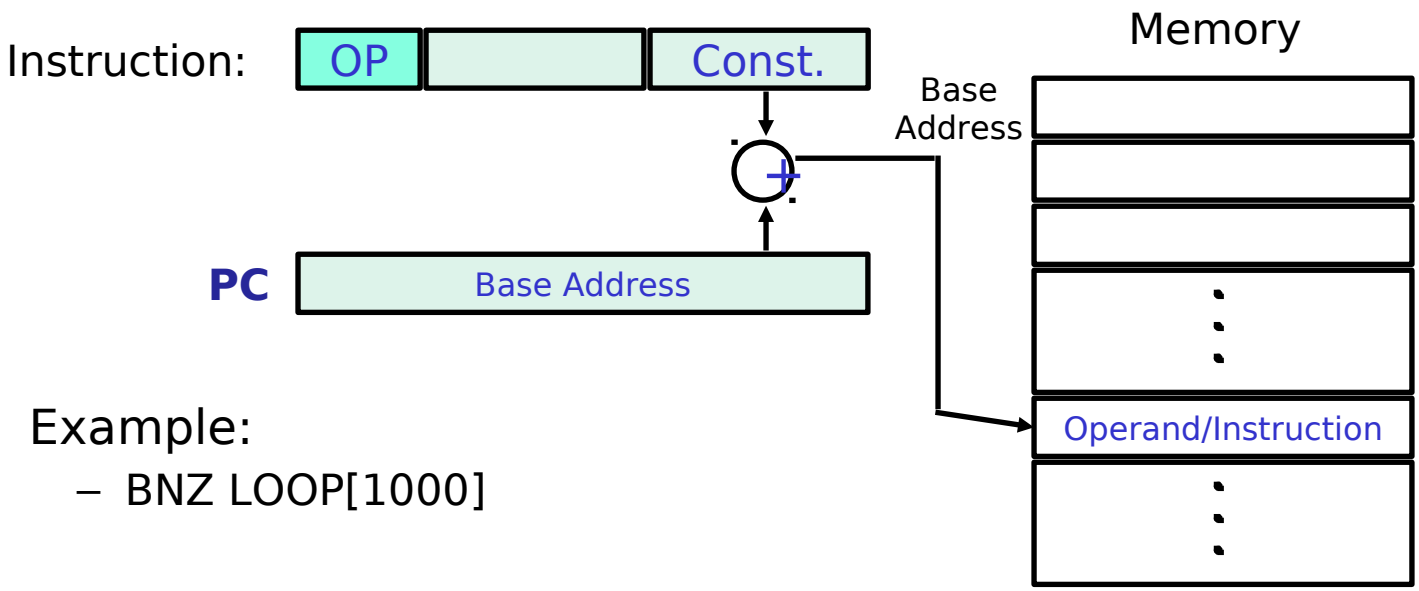
```
MOV R0, N
CLEAR R4
CLEAR R5
CLEAR R6
CLEAR R1
CLEAR R2
CLEAR R3
CLEAR R7
MOV R7, LIST
ADD R7, #4
MOV R4, R7
ADD R7, #4
MOV R5, R7
ADD R7, #4
MOV R6, R7
L1 ADD  R1, (R4)
ADD R2, (R5)
 ADD R3, (R6)
ADD  R4, #16
ADD R5, #16
ADD R6, #16
DEC R0
Branch > 0   L1
MOV SUB1, R1
MOV  SUB2, R2
MOV SUB3,  R3
END
```

Index

```
 MOV R0, N
     CLEAR R1
     CLEAR R2
     CLEAR R3
     MOV R4, LIST
L1   ADD R1,4(R4)
     MOV R2,8(R4)
     ADD R3, 12(R4)
     DEC R0
    Branch >0 L1
    MOV SUB1, R1
    MOV SUB2, R2
    MOV SUB3, R3
END
```

Addressing Mode

- Specification of operands in instructions
- Different addressing modes:
  - Relative: Base register, Displacement
    - Similar to indexed addressing mode
      - Program Counter (PC) is used in place of general purpose registers
    - To specify target address in branch instruction



- Example:
  - BNZ LOOP[1000]

Addressing Mode

- Specification of operands in instructions
- Different addressing modes:
  - Indexed Register: Base register, Index register

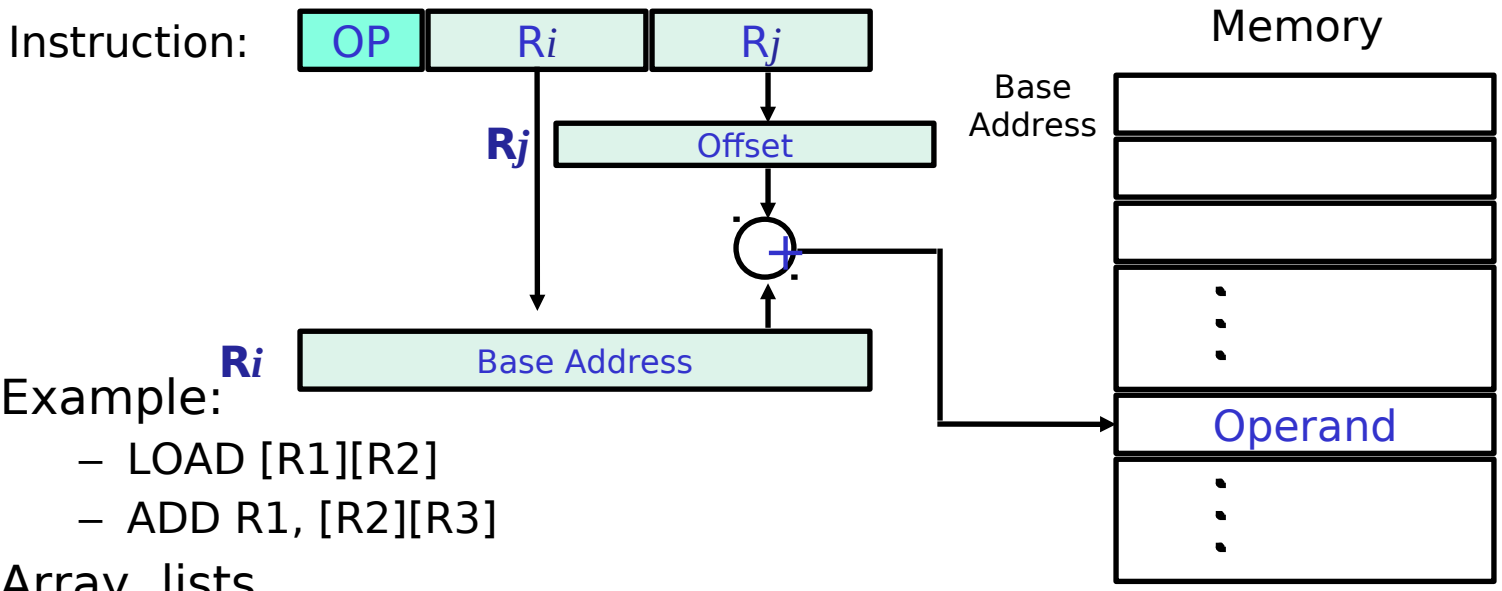
Effective Address (EA) of the operand = [Register] + [Register]

EA = [Ri] + [Rj]

•Register Contains base address

•Register used may be special registers or any general purpose registers

- Base registers or Index registers



•Example:

- LOAD [R1][R2]
- ADD R1, [R2][R3]

•Array, lists

Addressing Mode

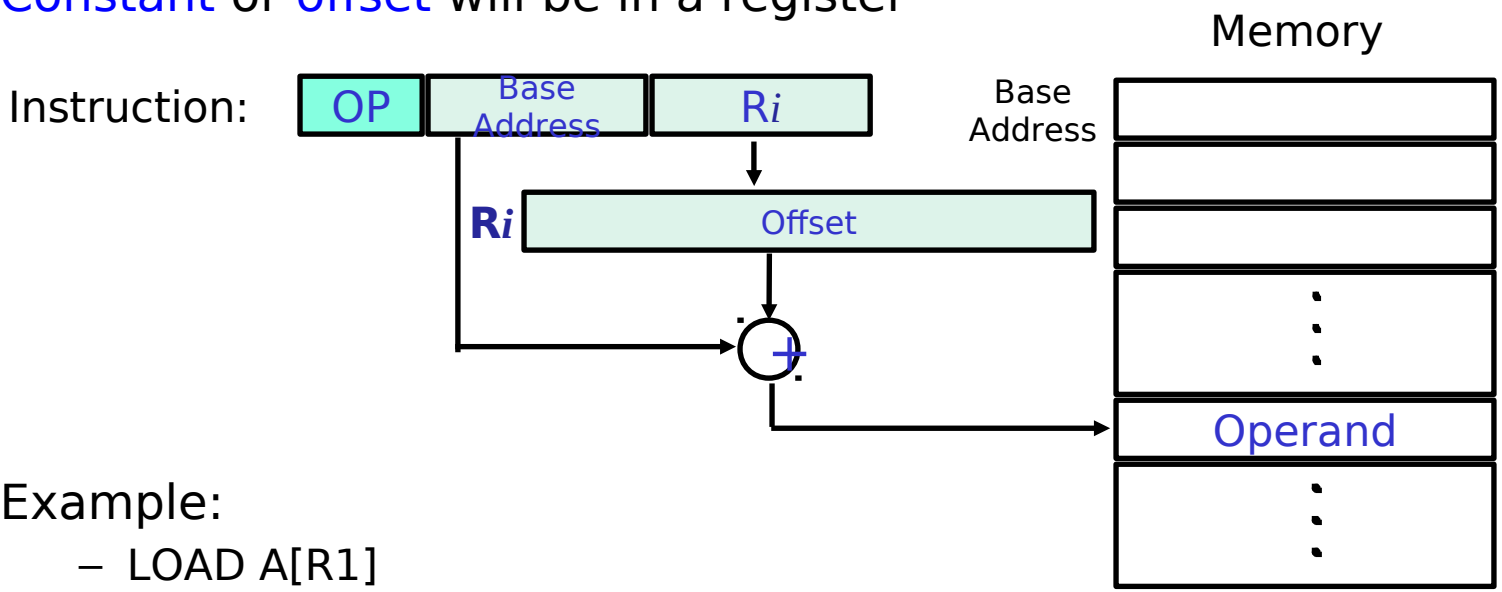
- Specification of operands in instructions
- Different addressing modes:
  - Memory Indexed Register:

Effective Address (EA) of the operand = Base Address + [Register].

EA = Base Address + [Ri]

•Memory address acts as base address

•Constant or offset will be in a register



•Example:

- LOAD A[R1]
- ADD R1, A[R2]

•Array, lists

Addressing Mode

- Specification of operands in instructions
- Different addressing modes:
  - Autoincrement/Autodecrement mode
    - (Ri)+, (Ri)-

Instruction Set Architecture

- Instruction set architecture
  - Instruction set
  - Instruction formats
  - Addressing modes
  - Instruction representation

Instruction Format

- Instruction word should have the complete information required to fetch and execute the instruction

- Fields of an instruction word
  - Opcode of the operation to be carried out
    - Varying length (CISC)
    - Fixed length (RISC)
  - Size of the operands:
    - Byte, Word, Longword, Quadword for integer operands
    - Float, Double for real operands
  - Addressing mode (AM) of each operand
  - Specification of each operand involves specifying **one or more** of the following:
    - General purpose register
    - Value of an immediate operand
    - Address of operand
    - Base register
    - Index register
    - Displacement

Instruction length

- Instruction: Opcode + operands/various methods of getting operands (addressing)
- The addressing bits
  - Number of addressing modes
  - Number of operands
  - Number of Registers
  - Number of Register set: Pentium has two more specialized sets
  - Address range
  - Address granualrity
  - Design principles: Orthogonality and Completness
    - an orthogonal instruction set is an instruction set architecture where all instruction types can use all addressing modes.
    - Orthogonal in practice: PDP-11, VAX-11, 8080
- Completness: Each arithmatic data type (integer, fixed point, real) should have complete and identical set of operations.

Instruction length

- Instruction length affects and affected by:
  - memory size
  - memory organization
  - bus structure
  - processor complexity
  - Processor speed.

- Programmers desire:
  - more opcodes
  - more operands
  - more addressing modes
  - greater address range
- Instruciton length should be equal or integral number of memor-transfer length

Instruction Set Architectures

- Complex Instruction Set Computer (CISC) processors:
  - 3-operand, 2-operand and 1-operand instructions
  - Any instruction can use memory operands
  - Many addressing modes
  - Complex instruction formats: Varying length instructions
  - Microprogrammed control unit (will be discussed in control unit)
  - Examples:
    - IBM System/360, VAX, PDP-11, Motorola 68000 family, AMD and Intel x86 CPUs.

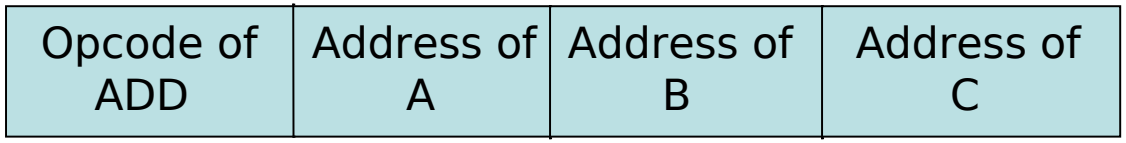
- Reduced Instruction Set Computer (RISC) processors:
  - 3-operand, 2-operand, and 1-operand instructions
  - Load-Store Architecture (LSA) processors:
    - Only memory transfer instructions (Load and Store) can use memory operands.
    - All other instructions can use register operands only.
  - A few addressing modes
  - Simple instruction formats: Fixed length instructions
  - Hardwired control unit (will be discussed in control unit)
  - Suitable for pipelining
  - Examples:
    - Alpha, ARC, ARM, AVR, MIPS, PA-RISC, PIC, Power Architecture, and SPARC.

Translation of High-Level Language Statements

Assignment statement: **A = B + C**

- CISC Architecture with 3-operand instructions:

ADD A, B, C

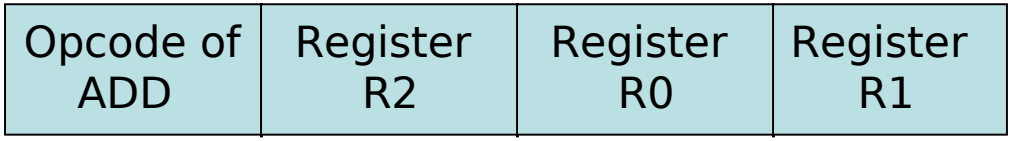


- CISC Architecture with 2-operand instructions:

```
LOAD    R0, B
ADD     R0, C
STORE   A,  R0
```

- RISC Architecture (Load-Store Architecture) with 3-operand instructions :

```
LOAD    R0, B
LOAD    R1, C
ADD     R2, R0, R1
STORE   A, R2
```



Translation of High-Level Language Statements

FOR (i = 0; i < N; i++)
A[i] = B[i] + C[i];

- CISC Architecture with 3-operand instructions:

```
                SUB  R0, R0, R0; R0 has the value of index i
Loop_Begin:     CMP  R0, N
                JEQ  Loop_End
                ADD A[R0], B[R0], C[R0]
                INC  R0
                JMP  Loop_Begin
Loop_End:
```

Opcode of ADD	Address of A	Register R0	Address of B	Register R0	Address of C	Register R0

Translation of High-Level Language Statements

FOR (i = 0; i < N; i++)
A[i] = B[i] + C[i];

- CISC Architecture with 2-operand instructions:

```
                SUB     R0, R0; R0 has value of loop index i
Loop_Begin:     CMP     R0, N
                JEQ     Loop_End
                LOAD    R1, B[R0]
                ADD     R1, C[R0]
                STORE   A[R0], R1
                INC     R0
                JMP     Loop_Begin
Loop_End:
```

Opcode of ADD	Register R1	Address of C	Register R0

Translation of High-Level Language Statements

FOR (i = 0; i < N; i++)
A[i] = B[i] + C[i];

- RISC Architecture (Load-Store Architecture) with 3-operand instructions :

```
                SUB     R0, R0, R0; R0 has value of loop index i
                LEA     R1, A;  Load the effective address of A in R1
                LEA     R2, B
                LEA     R3, C
                LOAD    R4, N
Loop_Begin:     CMP     R0, R4
                JEQ     Loop_End
                LOAD    R5, [R2][R0]
                LOAD    R6, [R3][R0]
                ADD     R7, R5, R6
                STORE   [R1][R0], R7
                INC     R0
                JMP     Loop_Begin
Loop_End:
```

Opcode of ADD	Register R7	Register R5	Register R6

Instruction Representation

- 3-operand CISC instruction format:

  ADD dst, src1, src2

Opcode	Size of operands	AM of dst	Specification of dst	AM of src1	Specification of src1	AM of src2	Specification of src2

- Example of CISC instruction representation:

  ADD R3, [R1][R0], 50[R2]

Opcode	Size of operands	AM of dst	Specification of dst	AM of src1	Specification of src1		AM of src2	Specification of src2	
0011	01	000	00011	101	00001	00000	110	0011 0010	00010

Instruction Representation

- Examples of RISC instructions:

ADD R2, R0, R1

Opcode	Size of operands	AM of dst	Specification of dst	AM of src1	Specification of src1	AM of src2	Specification of src2
000111	01	000	00010	000	00000	000	00001

LOAD R2, [R1][R0]

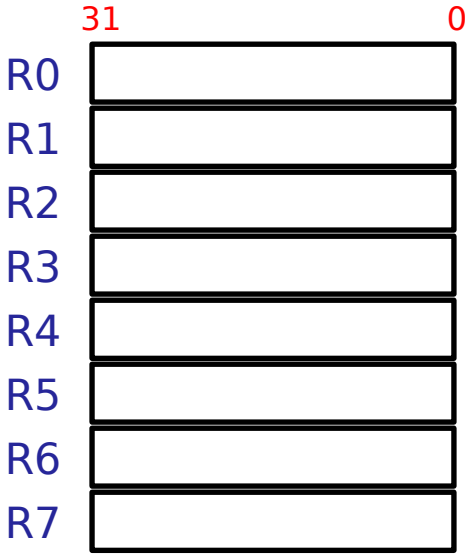
Opcode	Size of operands	Specification of dst	AM of src	Specification of src	
010011	01	00010	101	00001	00000

Pentium IA-32 Architecture

- Intel Corporation uses generic name Intel Architecture (IA) for its processors
- IA-32 operates with 32-bit memory addresses and 32-bit data bus
- IA-32 processors (i386) in the increasing order of their year of manufacturing are:
  - 80386
  - 80486
  - Pentium
  - Pentium pro
  - Pentium-II
  - Pentium-III
  - Pentium-IV
- Recent processors are x86 (64-bit) processors

IA-32 Register Structure

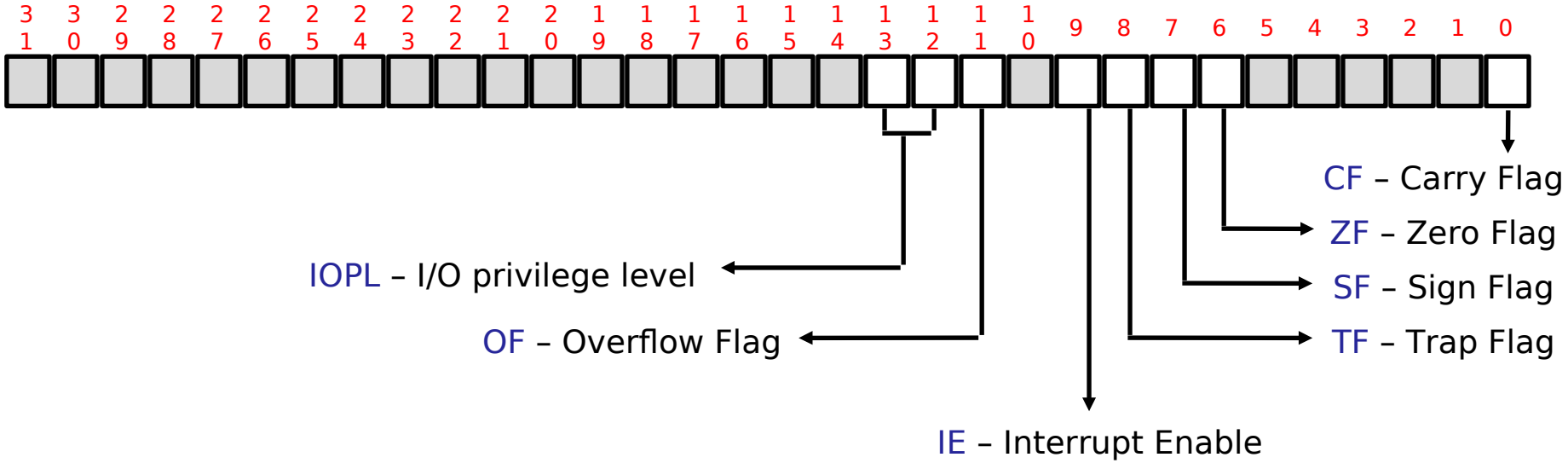
- General Purpose Registers (GPR)
  - **Eight**, 32-bit GPR
  - Hold operands or addressing information



IA-32 Register Structure (Contd.)

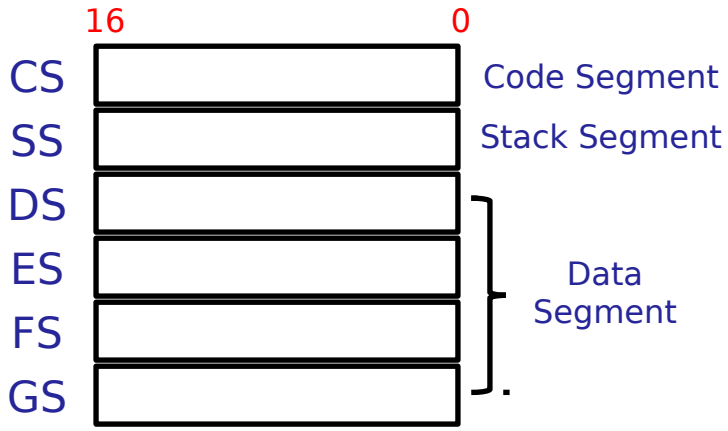
- Instruction Pointer (EIP)
  - 32-bit register serves as program counter (PC)
  - Contains the address of next instruction to be executed

- Status Registers
  - 32-bit register



IA-32 Register Structure (Contd.)

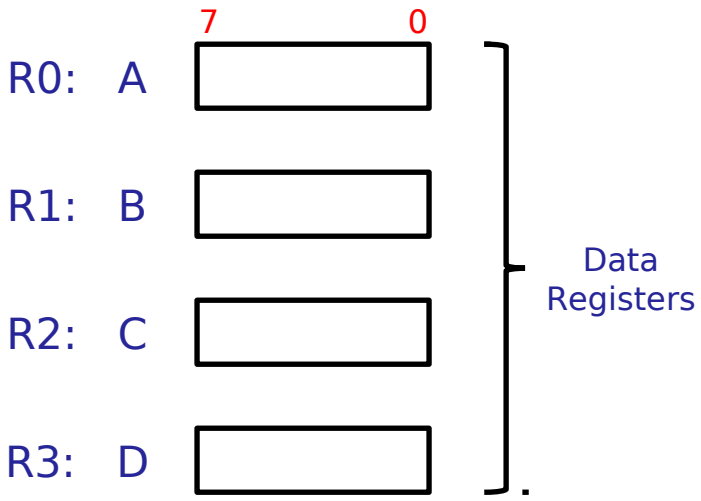
- Segments
  - IA-32 architecture based on a memory model that associates different areas of memory called segments
  - Code segment: Holds instructions of the program
  - Stack segment: Contain processor stack
  - Four Data segments: Provided for holding data operands
  - Each segment is a 16-bit register holding the selector values used for locating these segments in memory address space



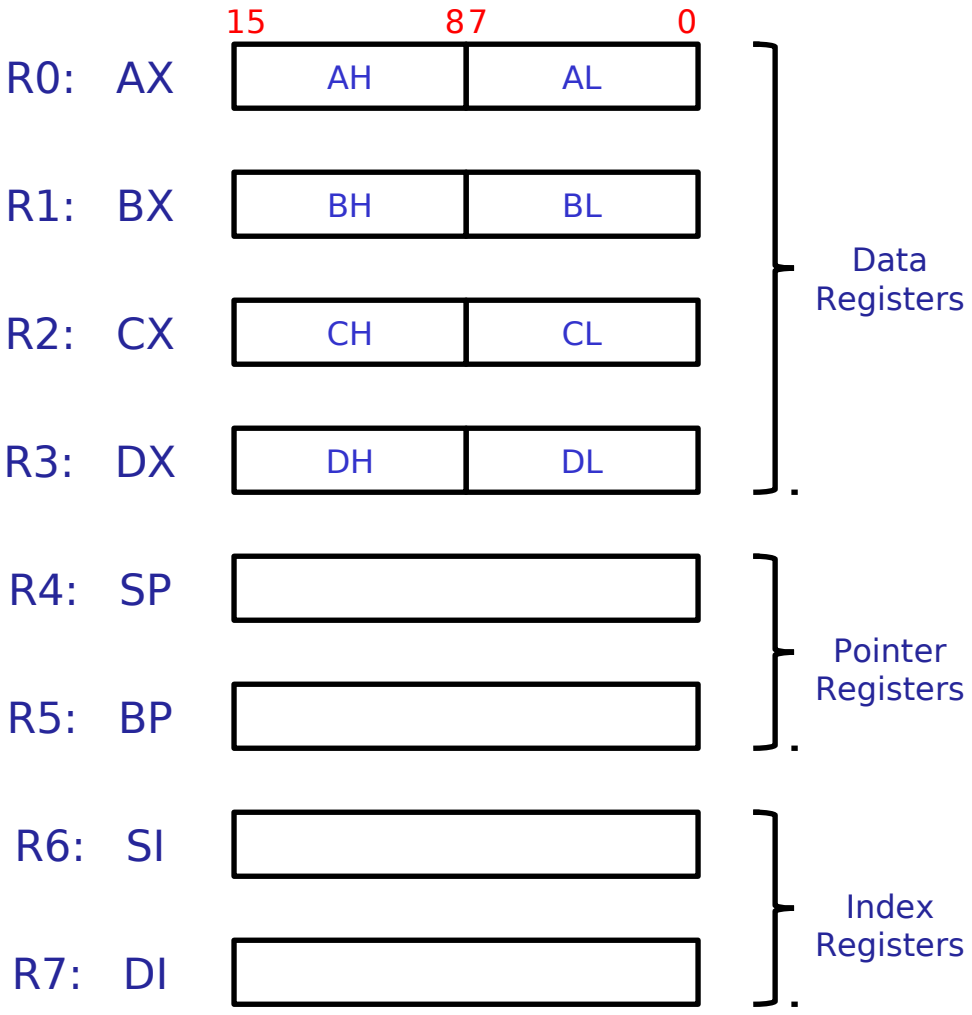
General Purposes Registers (GPRs)

- The eight GPRs are grouped into 3 different types:
  - Data registers: Holds operands
  - Pointer registers: Holds addresses
  - Index registers: Holds address indices
- 32-bit registers give compatibility for 8-bit and 16-bit Intel processors

GPRs for Intel 8-bit Processor (8085)



GPRs for Intel 16-bit Processor (8086)



GPRs for IA-32 Processors