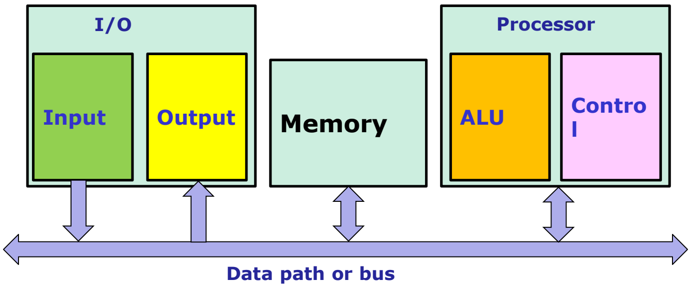
Microarchitecture Level

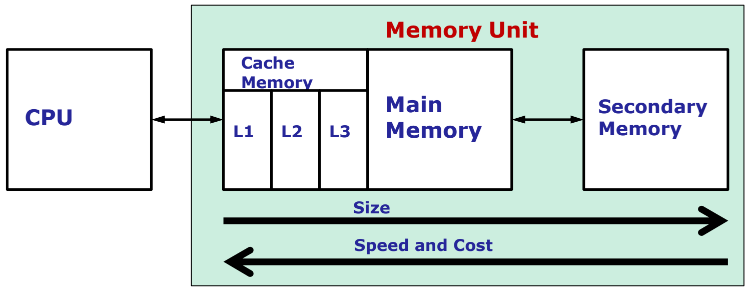
- **Functional Units**:
  - Input Unit
  - Memory Unit
  - Arithmetic and Logic Unit (ALU)
  - Output Unit
  - Control Unit



Recap

- Arithmetic unit design
- Integer arithmetic
  - Number representation
    - Signed
    - Unsigned
  - Adder/Subtractor
  - Multipliers
  - Division circuit
- Floating point Arithmetic
  - IEEE 754 single and double precision formats
  - Arithmetic operations

Memory Hierarchy



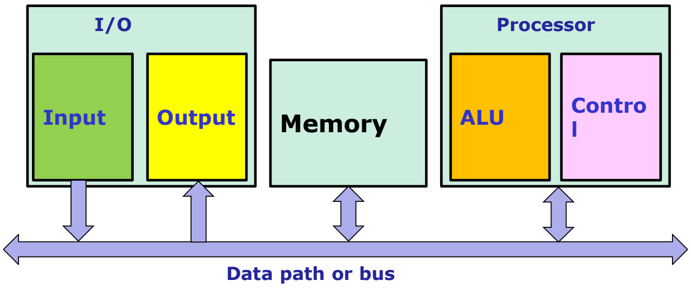
- Processor processes instructions and data faster than it can be fetched from memory unit
- Memory access time is the bottleneck
- One way to reduce memory access time is to use faster memory
  - A small and faster memory bridge the gap between processor and main memory
- **Virtual memory**

3

3

Microarchitecture Level

- **Functional Units**:
  - Input Unit
  - Memory Unit
  - Arithmetic and Logic Unit (ALU)
  - Output Unit
  - Control Unit



4

4

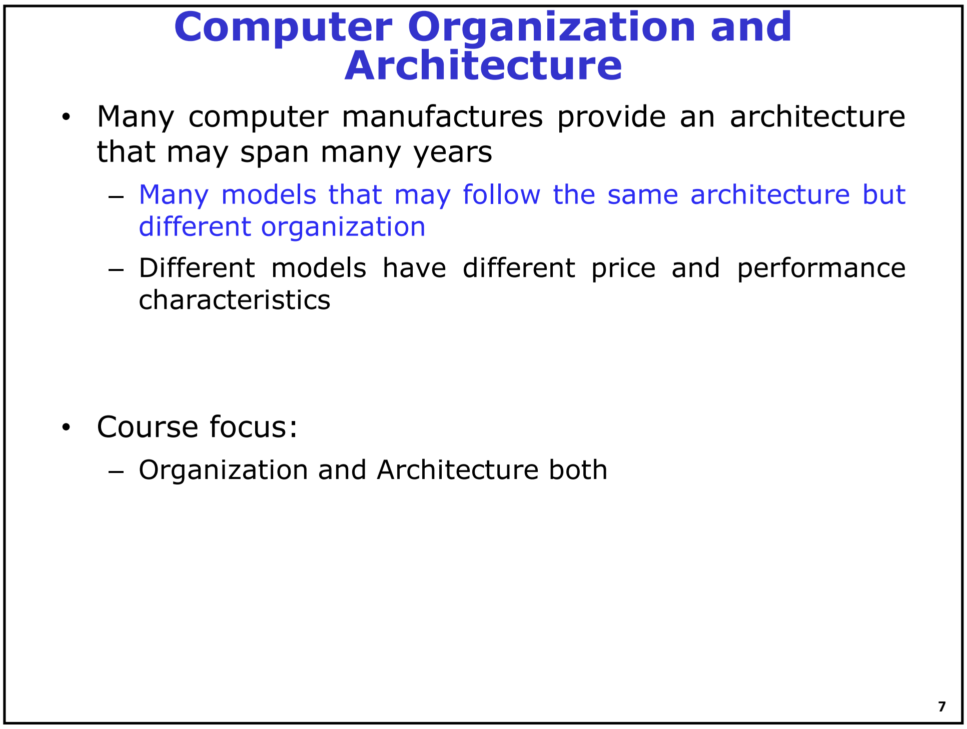
Instruction Set Architecture

5

Computer Organization and Architecture

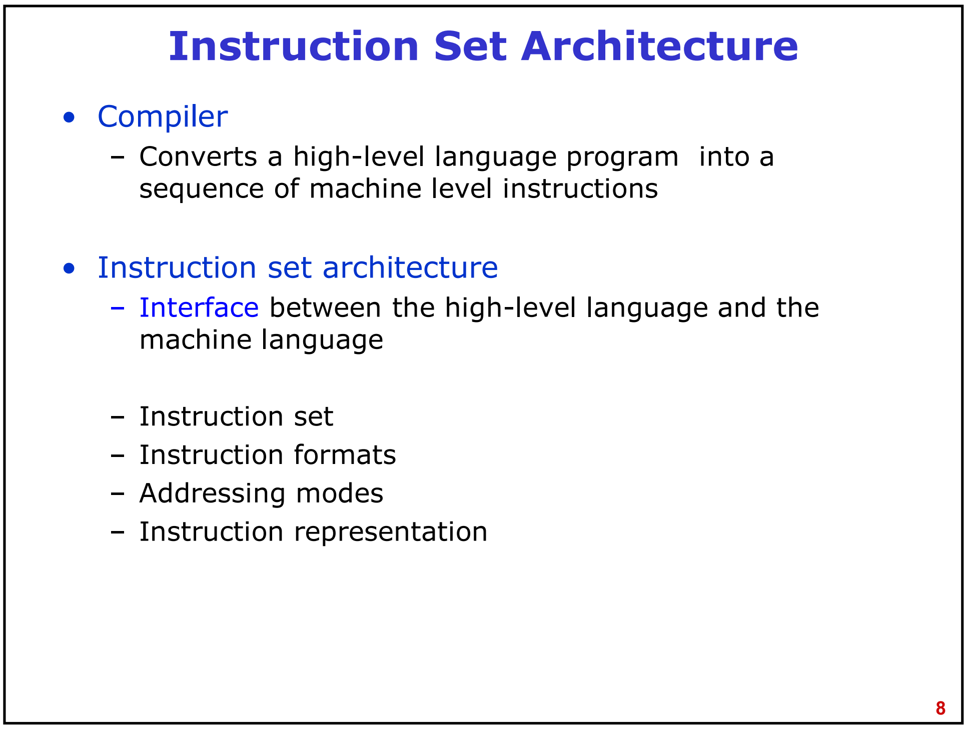
- **Computer Architecture**
  - Aspects that have direct impact on the logical execution of a program
  - Also called as **Instruction Set Architecture (ISA):** instruction formats, instruction opcodes, registers, memory
    - ADD instruction: A programmer is allowed to use
    - MUL instruction
- **Computer Organization**
  - Operational units and their interconnections to realise the architectural specifications
    - How the ADD operation should be internally realised?
      - Transparent to the programmer
    - Whether MUL instruction will be implemented by a special multiply unit or by repeated use of ADD instruction
      - Decision based on anticipated frequency of use of an instruction, relative speed of the approaches, cost and physical size of an a special multiply unit

6



Computer Organization and Architecture

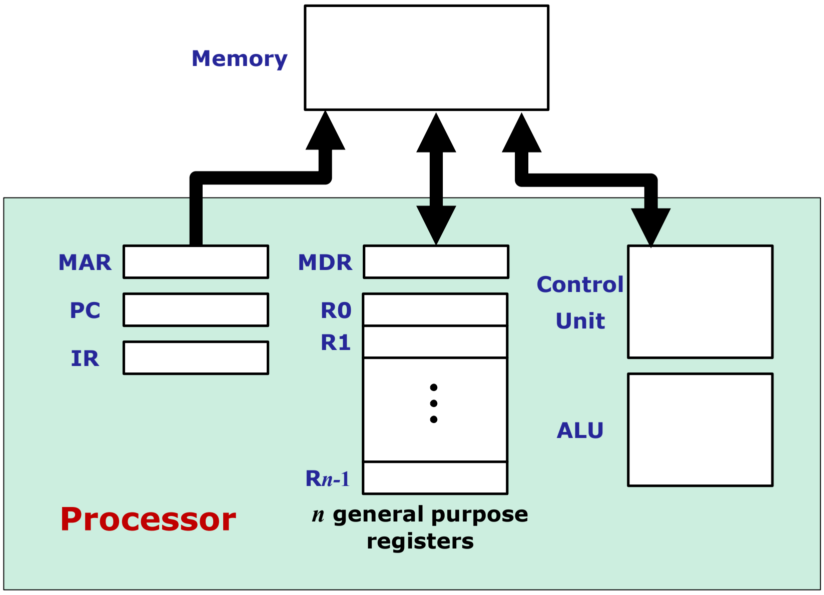
- Many computer manufactures provide an architecture that may span many years
  - Many models that may follow the same architecture but different organization
  - Different models have different price and performance characteristics
- Course focus:
  - Organization and Architecture both



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

Operational Details



9

9

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

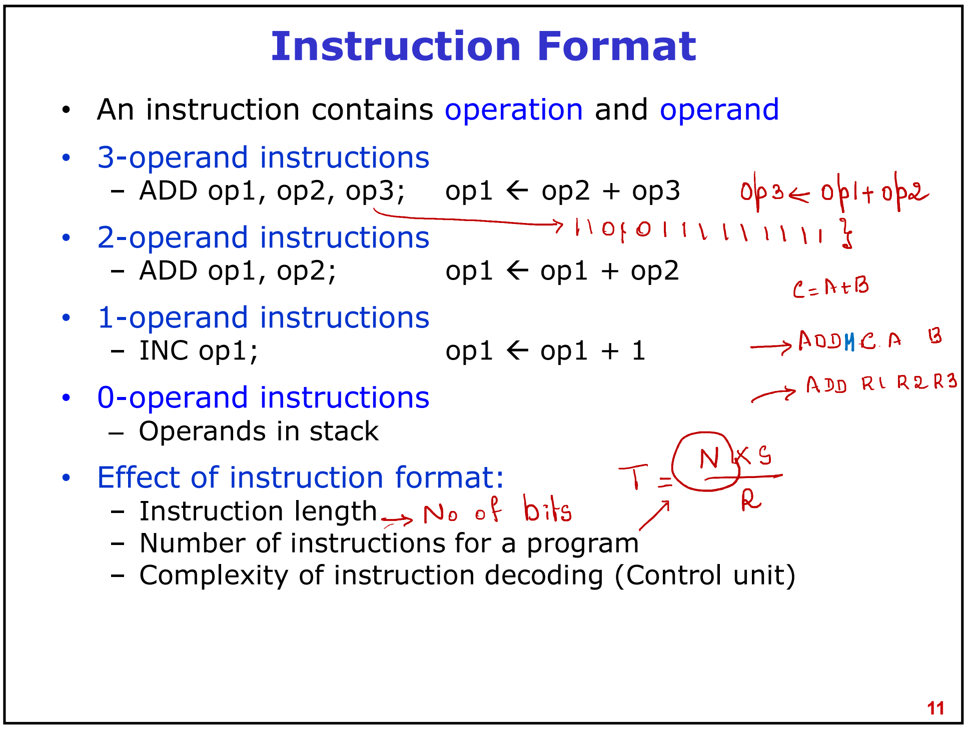
Mov R1, R2

While condition ← X
{

print result; ← JUMP X

10

10

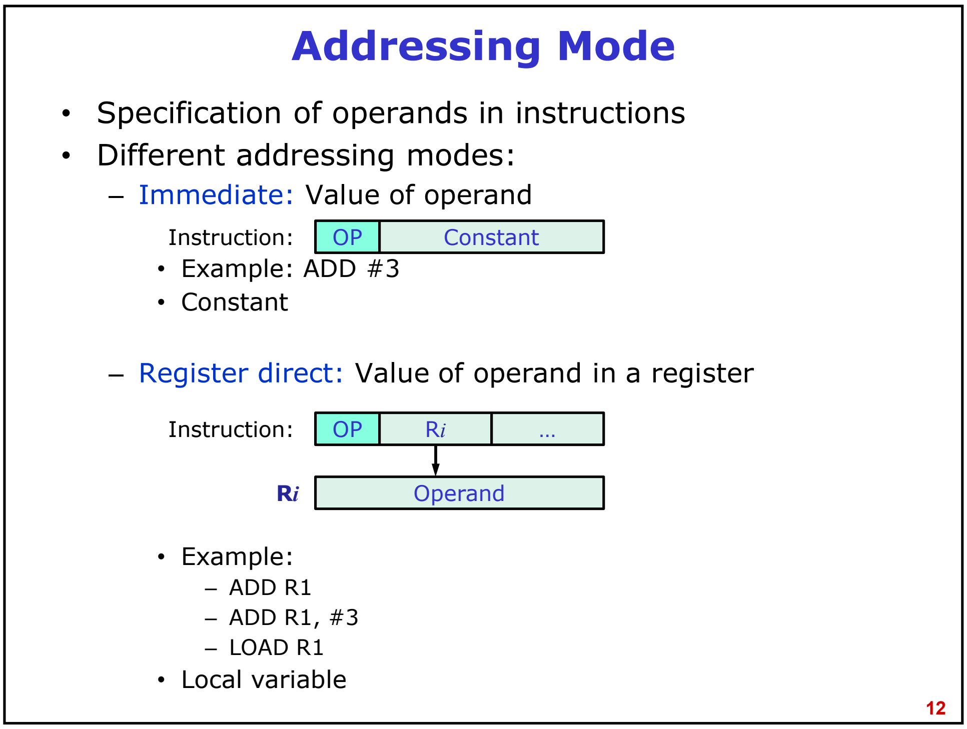


Instruction Format

- An instruction contains operation and operand
- 3-operand instructions
  - ADD op1, op2, op3; op1 ← op2 + op3 op3 ← op1 + op2
  - → 110|0111111111 }
- 2-operand instructions
  - ADD op1, op2; op1 ← op1 + op2 C = A+B
- 1-operand instructions
  - INC op1; op1 ← op1 + 1 → ADDMC A B
  - → ADD R1 R2 R3
- 0-operand instructions
  - Operands in stack
- Effect of instruction format: $T = \frac{N \times S}{R}$
  - Instruction length → No of bits
  - Number of instructions for a program
  - Complexity of instruction decoding (Control unit)

11

11



Addressing Mode

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

    Instruction: OP | Constant

    - Example: ADD #3
    - Constant
  - Register direct: Value of operand in a register

    Instruction: OP | R*i* | ...

    R*i* Operand

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

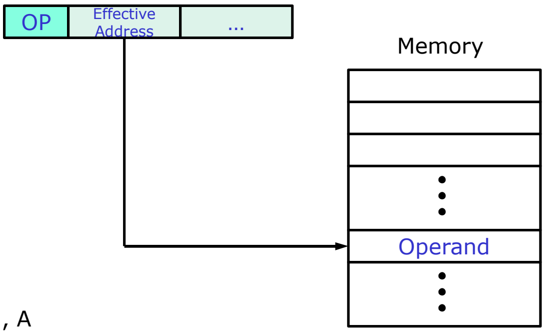
12

12

Addressing Mode

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

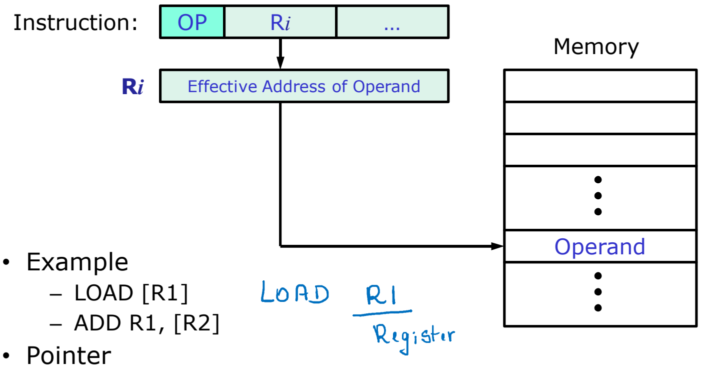
ADDM C, A, B



- Example:
  - LOAD A
  - ADD R1, A
- Global variable

Addressing Mode

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

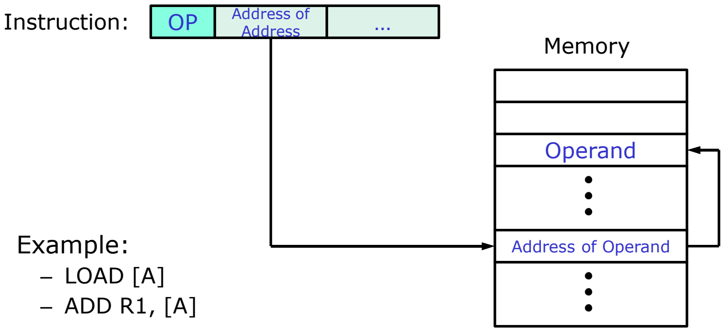


- Example
  - LOAD [R1]
  - ADD R1, [R2]
- Pointer

LOAD R1
Register

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

15

15

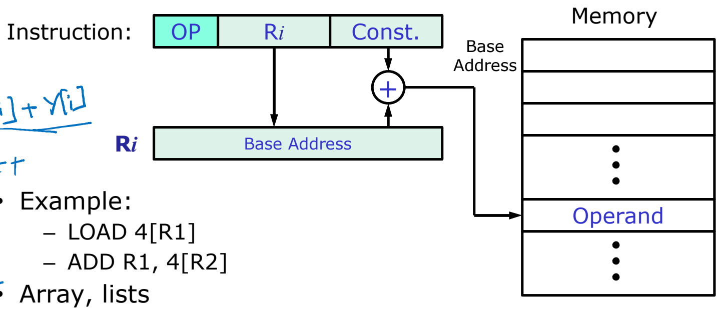
Addressing Mode

- 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



X, Y, Z

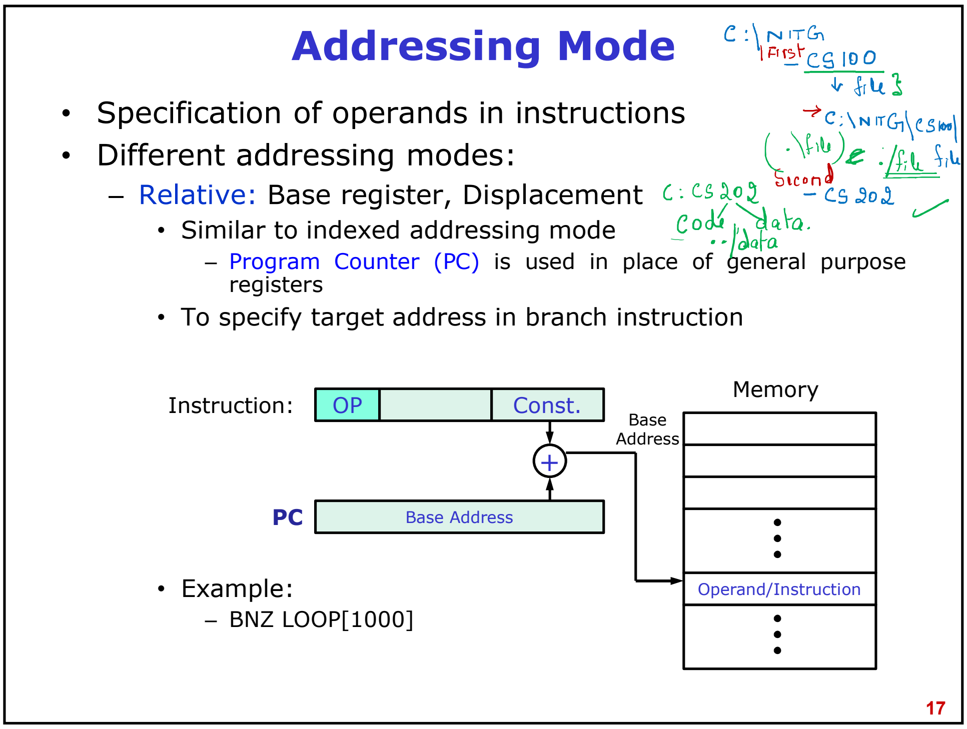
Z[i] = X[i] + Y[i]

X 10 20 30 : 100 i++

- Example:
  - LOAD 4[R1]
  - ADD R1, 4[R2]
- Array, lists

16

16



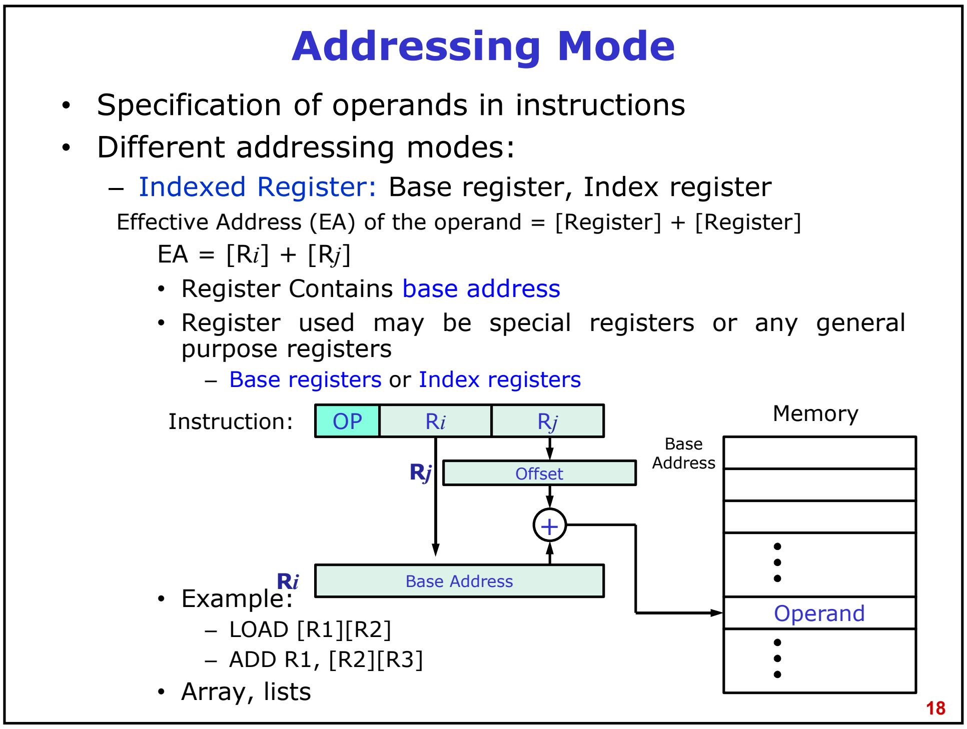
Addressing Mode

- 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

Instruction:
OP
Const.
+
PC
Base Address
Memory
Base Address
Operand/Instruction

- Example:
  - BNZ LOOP[1000]

C:\NITG
\First
CS100
↓ file }
→ C:\NITG\CS100\
(.\file) e .\file file
Second
C: CS202
– CS 202
Code data.
../data
✓



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 = [R*i*] + [R*j*]
    - Register Contains base address
    - Register used may be special registers or any general purpose registers
      - Base registers or Index registers

Instruction:
OP
R*i*
R*j*
R*j*
Offset
+
R*i*
Base Address
Memory
Base Address
Operand

- Example:
  - LOAD [R1][R2]
  - ADD R1, [R2][R3]
- Array, lists

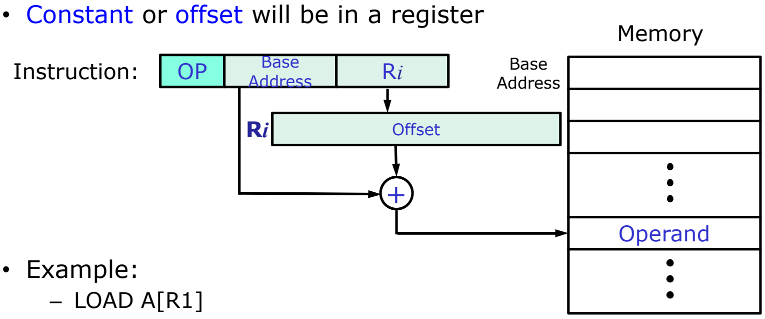
Addressing Mode

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

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

EA = Base Address + [R*i*]

- Memory address acts as base address
- Constant or offset will be in a register



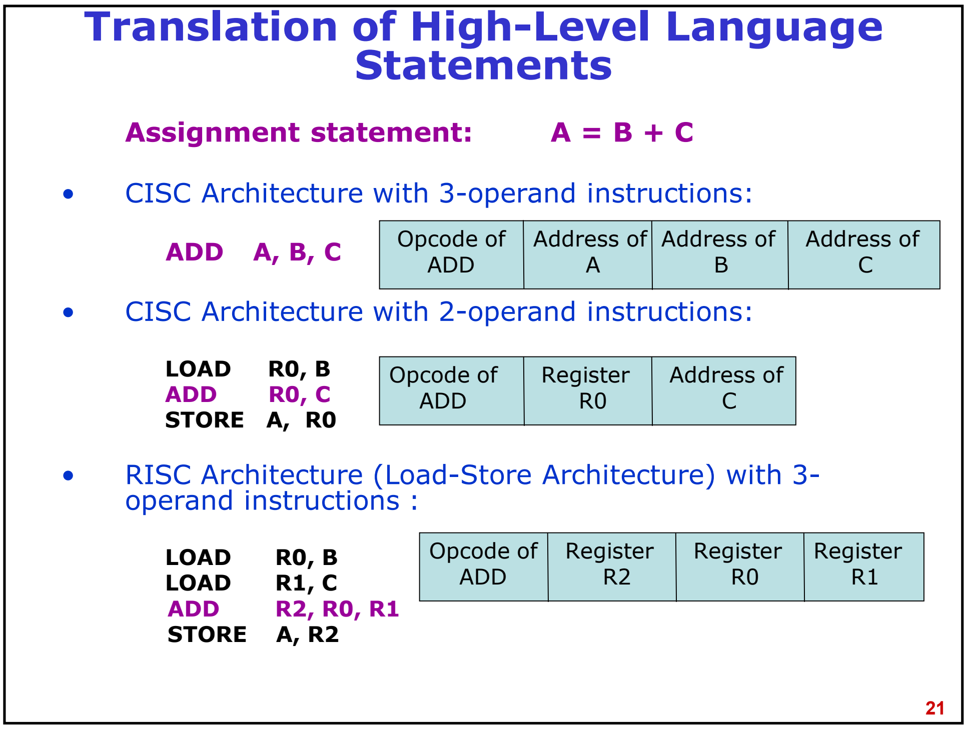
- Example:
  - LOAD A[R1]
  - ADD R1, A[R2]
- Array, lists

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
  - Examples:
    - 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
  - Suitable for pipelining ≈ 1
  - Examples:
    - Alpha, ARC, ARM, AVR, MIPS, PA-RISC, PIC, Power Architecture, and SPARC.

$T = \frac{N \times S}{R}$

$S \approx 5$



Translation of High-Level Language Statements

Assignment statement: A = B + C

- CISC Architecture with 3-operand instructions:

ADD A, B, C

Opcode of ADD	Address of A	Address of B	Address of C

- CISC Architecture with 2-operand instructions:

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

Opcode of ADD	Register R0	Address of C

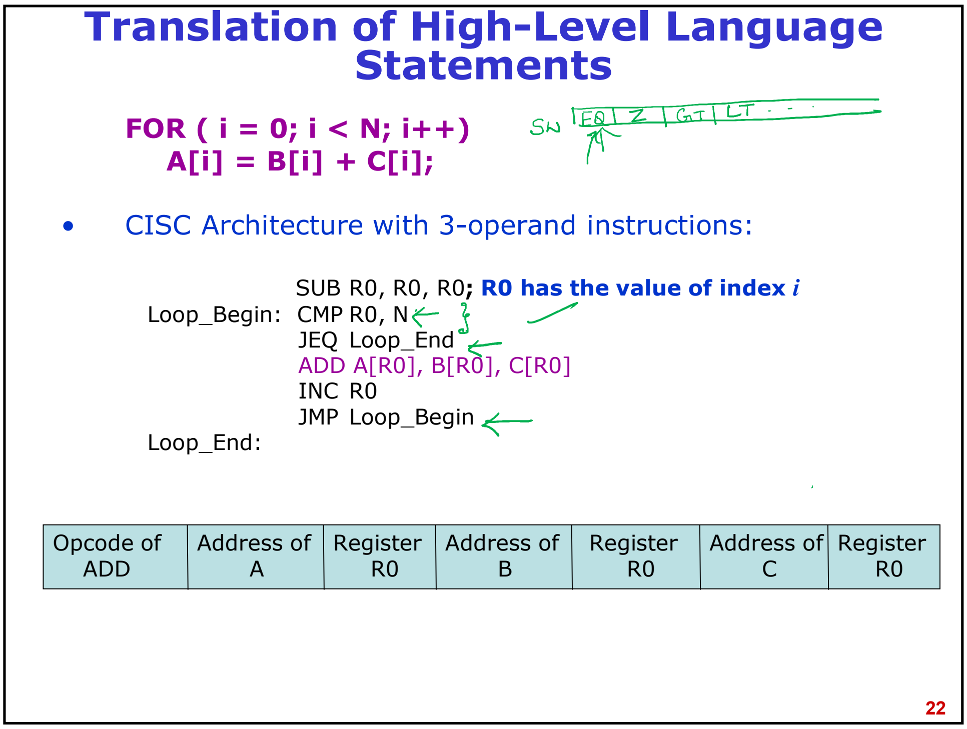
- RISC Architecture (Load-Store Architecture) with 3-operand instructions :

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

Opcode of ADD	Register R2	Register R0	Register R1

21

21



Translation of High-Level Language Statements

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

SW |EQ| Z |GT| LT . . .

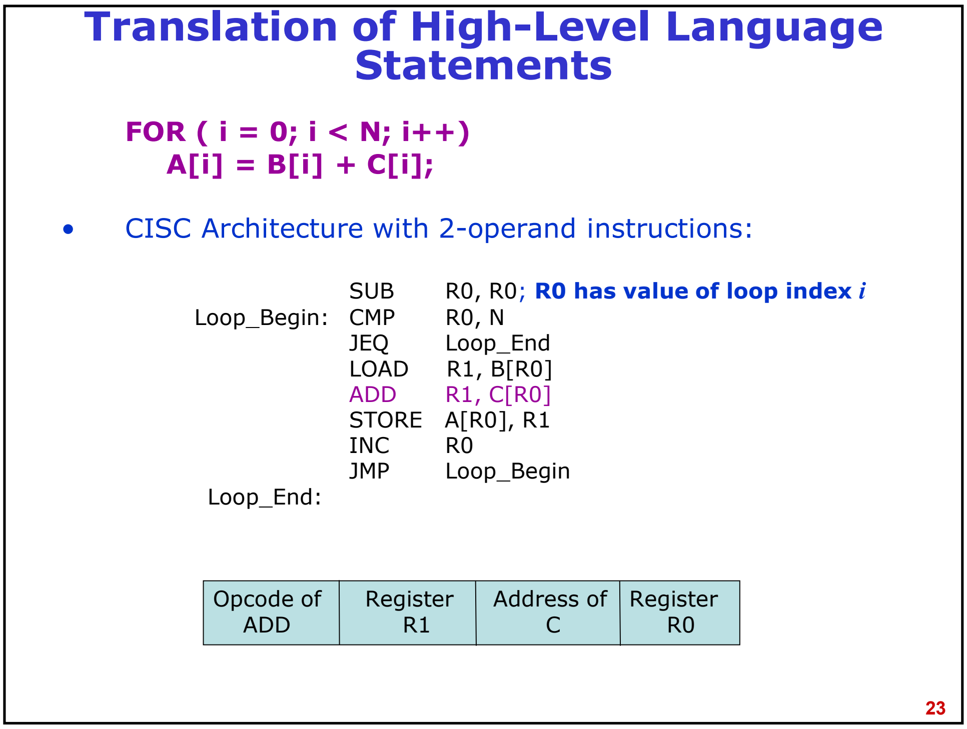
- CISC Architecture with 3-operand instructions:

```
             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

22

22



Translation of High-Level Language Statements

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

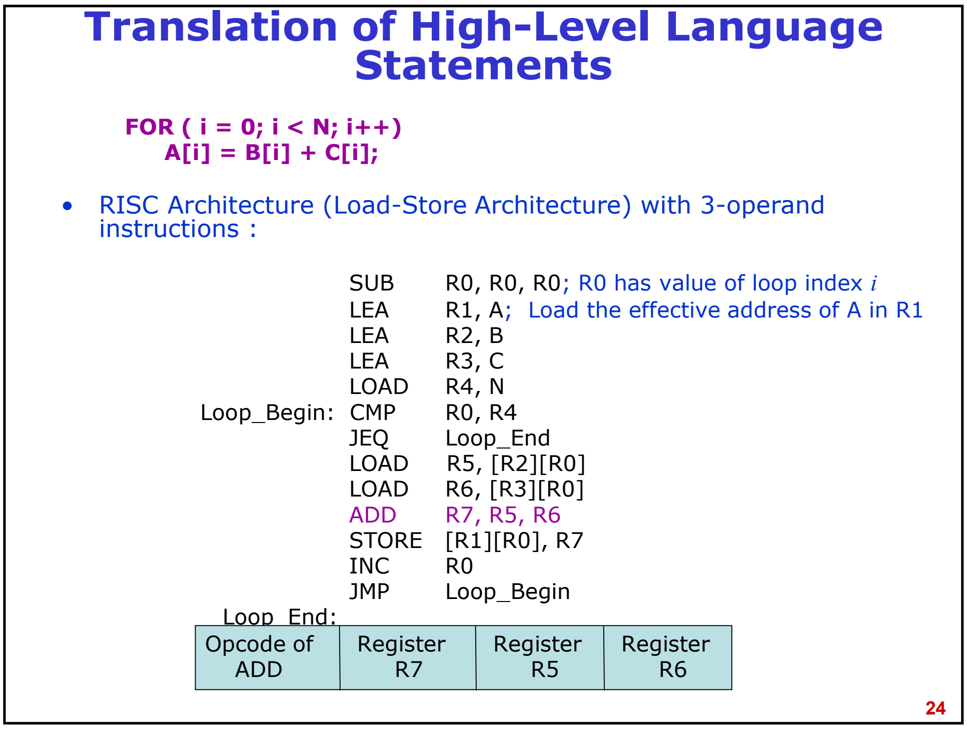
- CISC Architecture with 2-operand instructions:

```
                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

23

23



Translation of High-Level Language Statements

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

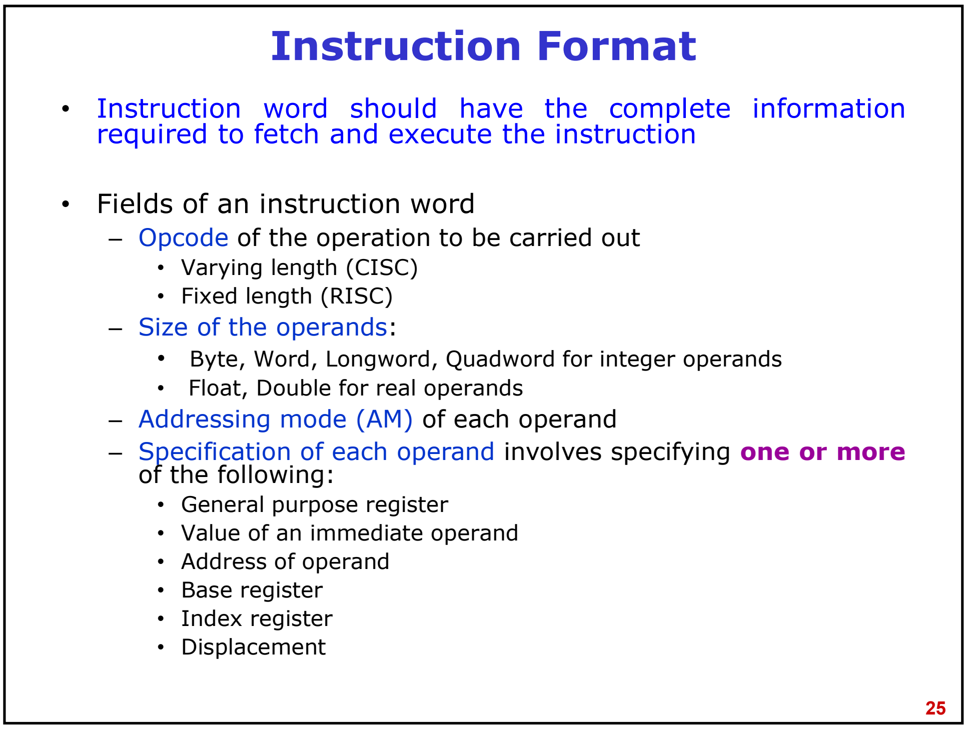
- RISC Architecture (Load-Store Architecture) with 3-operand instructions :

```
                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

24

24

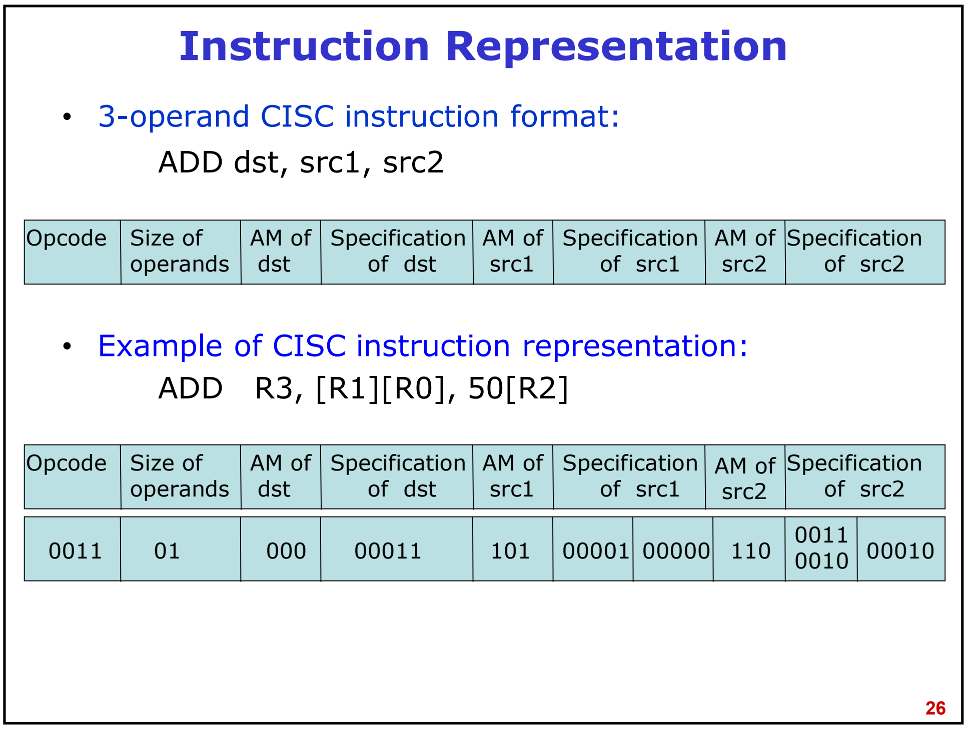


Instruction Format

- 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

25

25



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

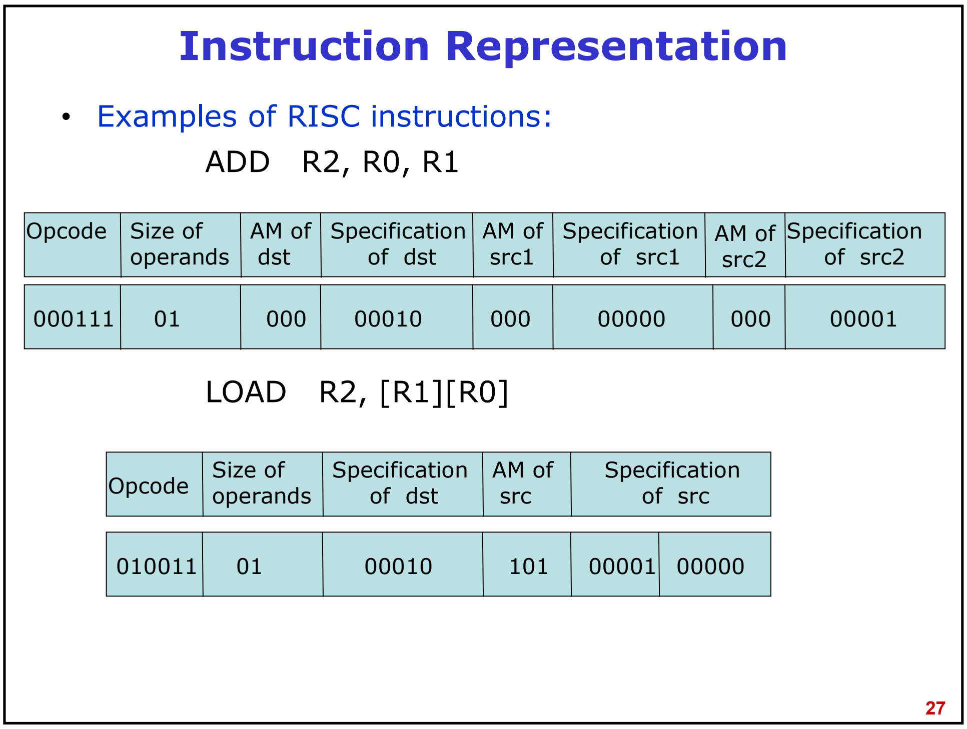
- Example of CISC instruction representation:

  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

26

26



Instruction Representation

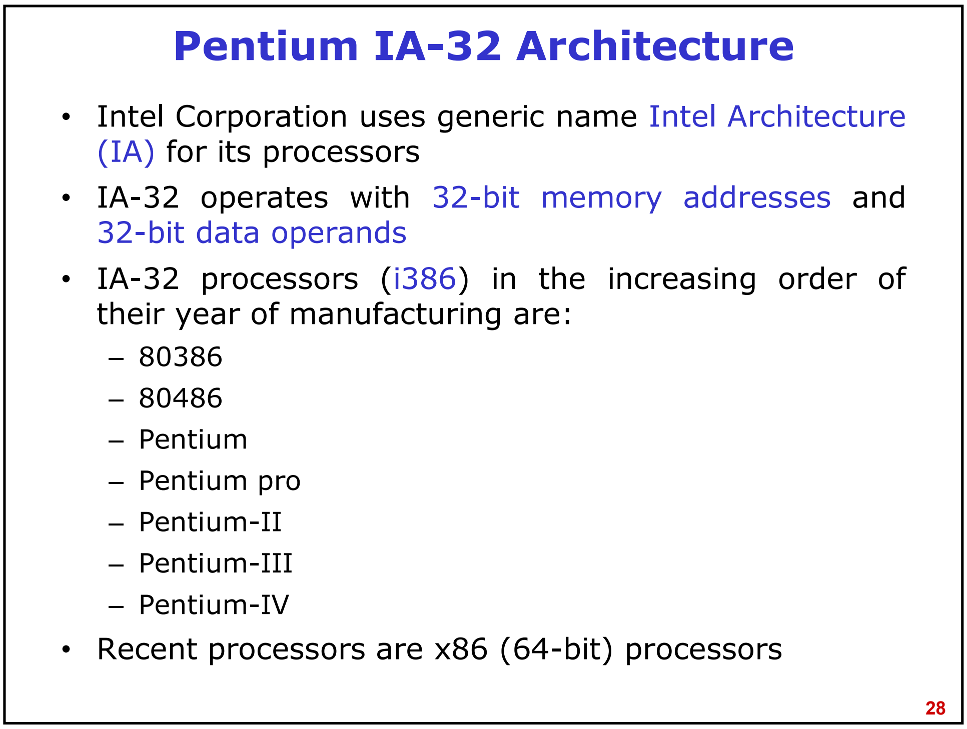
- Examples of RISC instructions:

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

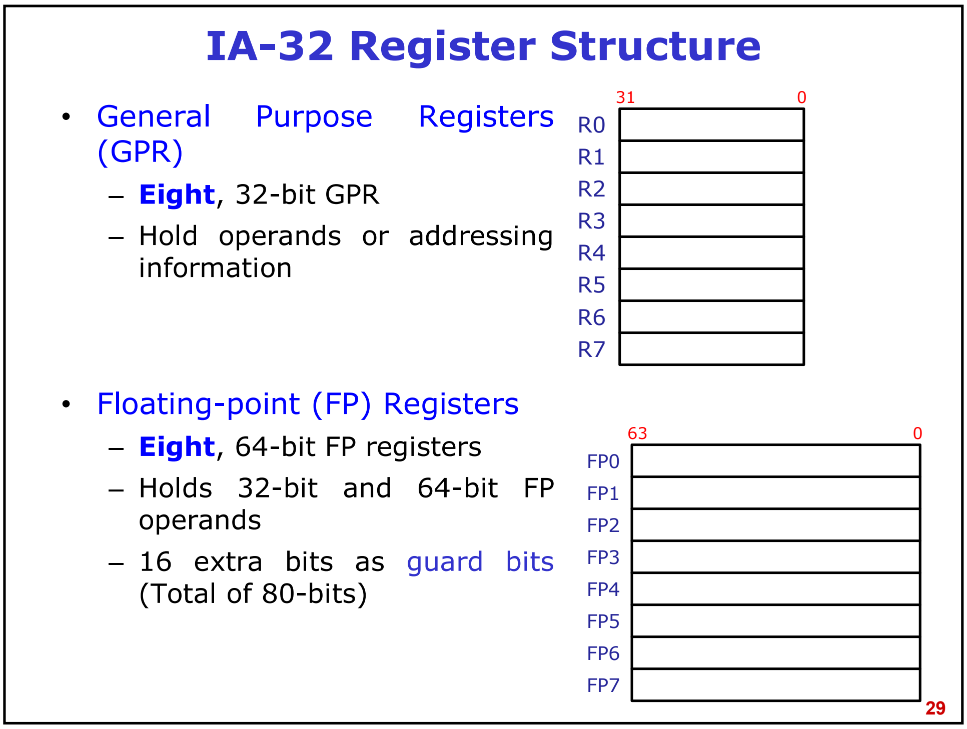
LOAD R2, [R1][R0]

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 operands
- 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

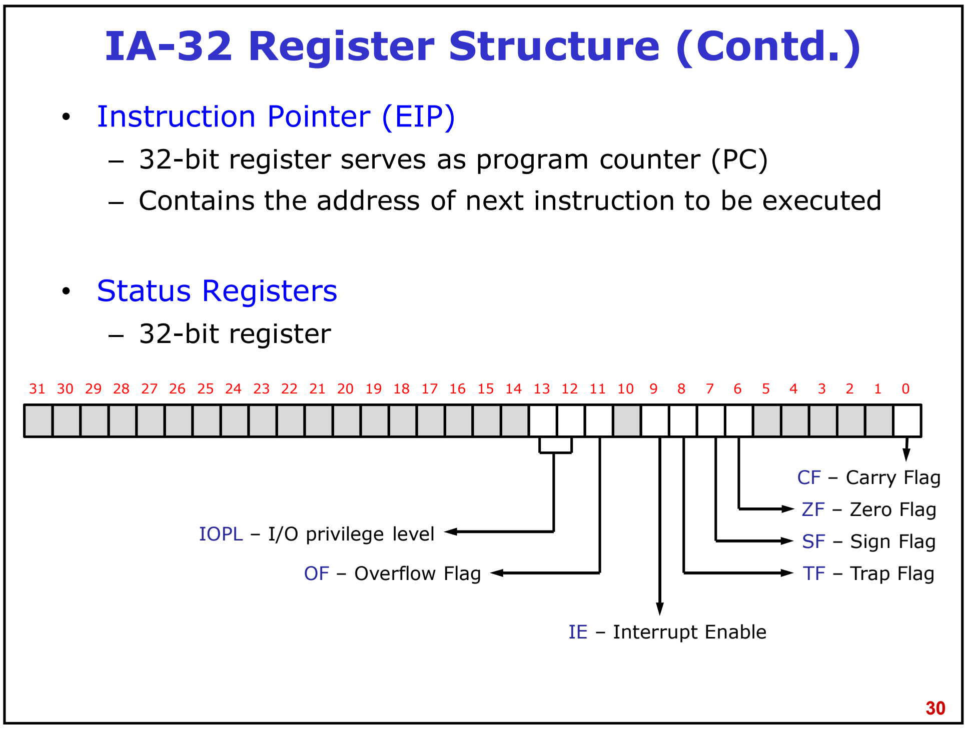
31 0

R0
R1
R2
R3
R4
R5
R6
R7

- Floating-point (FP) Registers
  - **Eight**, 64-bit FP registers
  - Holds 32-bit and 64-bit FP operands
  - 16 extra bits as guard bits (Total of 80-bits)

63 0

FP0
FP1
FP2
FP3
FP4
FP5
FP6
FP7

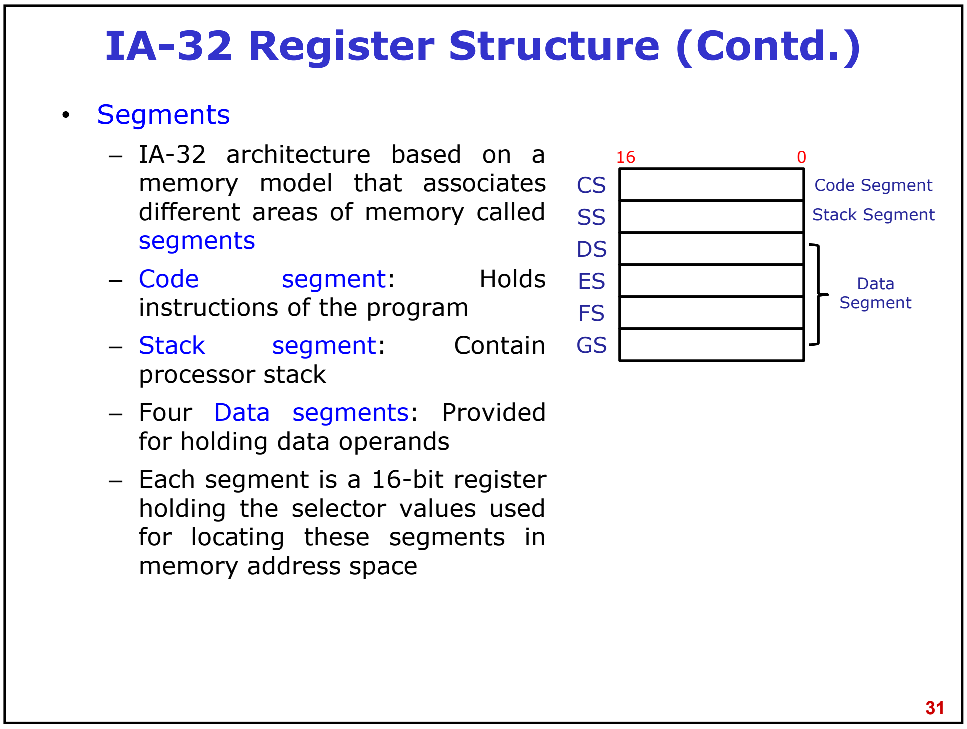


IA-32 Register Structure (Contd.)

- 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

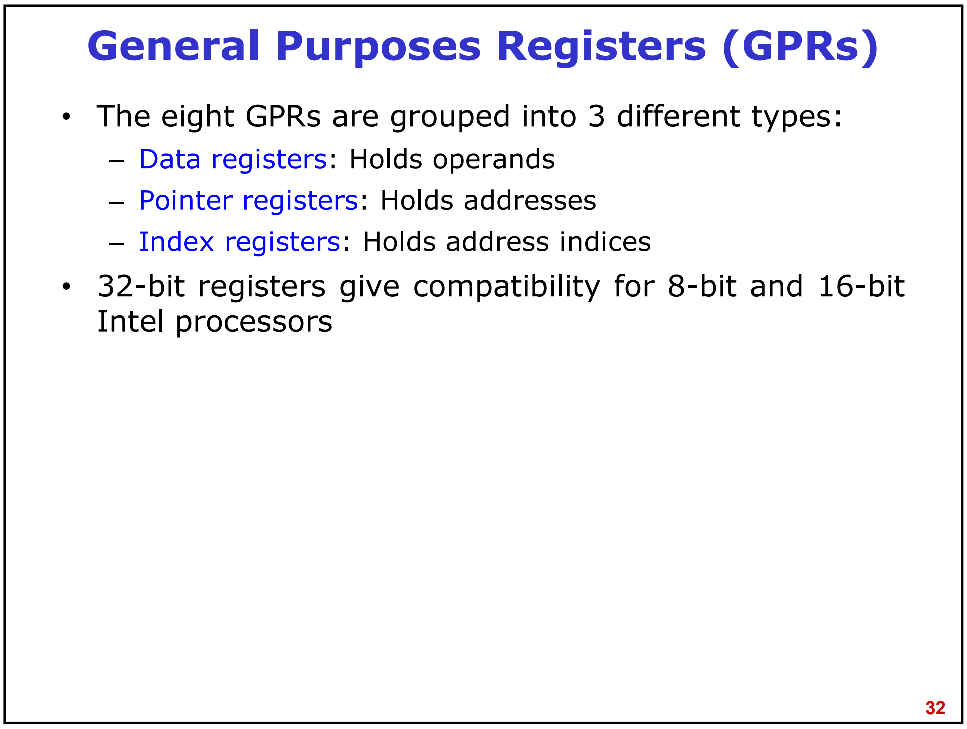
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

CF – Carry Flag
ZF – Zero Flag
IOPL – I/O privilege level
SF – Sign Flag
OF – Overflow Flag
TF – Trap Flag
IE – Interrupt Enable



IA-32 Register Structure (Contd.)

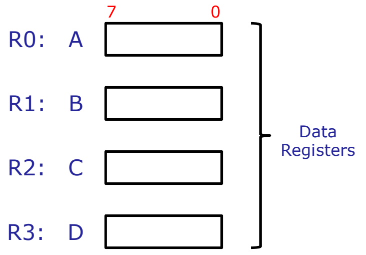
- Segments
  - IA-32 architecture based on a memory model that associates different areas of memory called segments
  - Code segment: Holds instructions of the program
  - Stack segment: Contain processor stack
  - Four Data segments: Provided for holding data operands
  - Each segment is a 16-bit register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

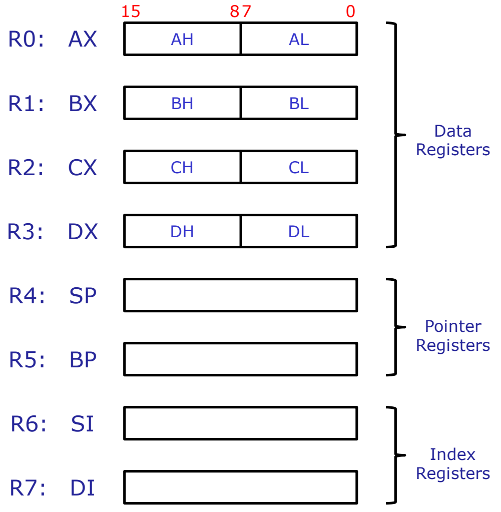
GPRs for Intel 8-bit Processor (8085)



33

33

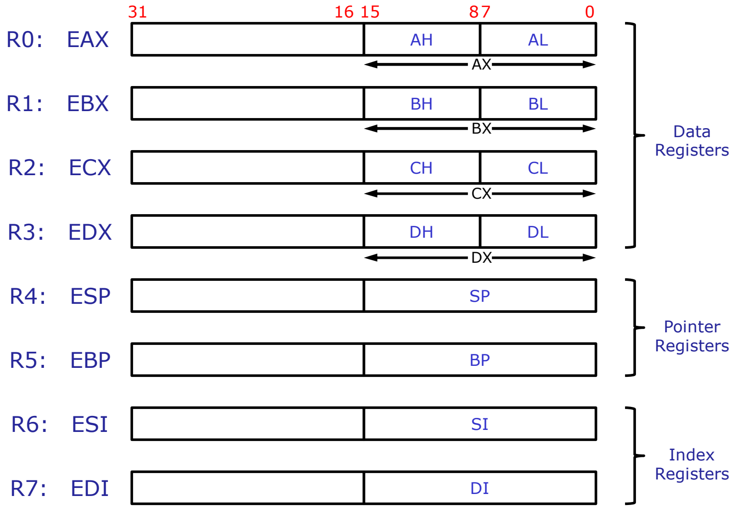
GPRs for Intel 16-bit Processor (8086)



34

34

GPRs for IA-32 Processors


31
16 15
8 7
0
R0: EAX
AH
AL
AX
R1: EBX
BH
BL
BX
R2: ECX
CH
CL
CX
R3: EDX
DH
DL
DX
Data Registers
R4: ESP
SP
R5: EBP
BP
Pointer Registers
R6: ESI
SI
R7: EDI
DI
Index Registers