**Annexure- II A**

**“Programs On All Addressing Modes of 8086 MIC”**

1. **Brief Introduction:**

In the 8086 microprocessor, an addressing mode refers to the method or technique used to specify the operand (data) for an instruction. The addressing mode determines how the processor should access or compute the location of the operand, whether it's a value stored in a register, in memory, or embedded within the instruction itself. The mode also influences how the instruction interacts with the memory or registers to perform operations like addition, subtraction, or data transfer.

There are several addressing modes in the 8086 microprocessor, each designed to provide flexibility in accessing data for different types of operations. These modes allow for efficient and diverse handling of data across memory and registers**.**

1. **Immediate Addressing Mode**

In this mode, the operand is a constant value specified directly in the instruction. This value is immediately used by the CPU without accessing memory.

* The operand is part of the instruction itself.
* Example: MOV AX, 5 moves the value 5 directly into the AX register.
* Used when constants are required as operands.

1. **Register Addressing Mode**

In register addressing mode, the operand is stored in a register. The instruction specifies which register holds the operand.

* Fastest mode as it involves no memory access.
* Example: MOV AX, BX moves the contents of the BX register into the AX register.
* Common in operations that only need to manipulate registers.

1. **Direct Addressing Mode**

In direct addressing, the instruction specifies the memory address where the operand is located.

* The operand is located at a specific address in memory.
* Example: MOV AX, [1234h] moves the contents of memory at address 1234h into AX.
* Useful when working with known memory locations.

1. **Indirect Addressing Mode**

Here, the operand’s address is stored in a register or memory location. The instruction specifies the register that contains the memory address of the operand.

* More flexible, as the operand’s address can change dynamically.
* Example: MOV AX, [BX] uses the value in the BX register as the memory address and moves the contents into AX.
* Often used with pointers.

1. **Register Indirect Addressing Mode**

In this mode, a register points to the memory location where the operand is located. It’s a subset of indirect addressing where only registers are used to store the memory address.

* A memory address is held in a register, and it points to the data.
* Example: MOV AX, [SI] where the SI register contains the address of the operand.
* Efficient for operations on data structures like arrays.

1. **Based Addressing Mode**

This mode combines a base register with an offset to calculate the effective memory address.

* It uses a base register (like BX or BP) along with an offset to form the address.
* Example: MOV AX, [BX+05h] adds an offset (05h) to the address in the BX register.
* Often used for data structures where the base register points to a block of memory.

1. **Indexed Addressing Mode**

In indexed addressing mode, the address of the operand is formed by adding an index register to a constant offset.

* This mode is often used in accessing arrays.
* Example: MOV AX, [SI+100h] moves the value at the memory location pointed by SI + 100h into AX.
* The index register provides dynamic access to a memory range.

1. **Based Indexed Addressing Mode**

This mode combines both the base register and an index register, plus a constant offset to form the effective memory address.

* The operand’s address is calculated by adding the base register, index register, and an optional offset.
* Example: MOV AX, [BX+SI+05h] uses both BX and SI as part of the address calculation.
* Commonly used in complex data structures.

1. **Relative Addressing Mode**

In relative addressing mode, the operand is at a memory location specified by an offset added to the instruction pointer (IP).

* The address is given relative to the current instruction pointer.
* Example: JMP 0x10 where the target address is calculated relative to the current IP.
* Used mainly for control flow instructions (like jumps).

1. **Memory Addressing Mode**

Memory addressing mode refers to any addressing where the operand is located in memory, and the instruction accesses memory directly, using various forms such as direct, indirect, or indexed modes.

* Involves accessing operands directly from memory rather than registers.
* Provides flexibility to access large amounts of data.
* Used in a variety of situations where large or dynamic data needs to be processed.

These addressing modes provide flexibility in how the 8086 accesses and manipulates data, supporting various programming tasks efficiently.

* **Syntex of each Adderessing Modes in 8086 Microprocessor**

**1. Immediate Addressing Mode:**

In **immediate addressing**, the operand is a constant value that is directly specified in the instruction.

**Syntax:**

MOV destination, immediate\_value

**Explanation:**

* The operand is the value itself, and it is part of the instruction.
* The immediate value is used directly by the CPU, with no memory access involved.

**Example:**

MOV AX, 5

This instruction moves the immediate value 5 directly into the AX register.

**2. Register Addressing Mode:**

In **register addressing**, the operand is stored in a register, and the instruction specifies which register to use.

**Syntax:**

MOV destination\_register, source\_register

**Explanation:**

* Both the source and destination are registers.
* No memory access is involved.

**Example:**

MOV AX, BX

This instruction moves the contents of register BX into register AX.

**3. Direct Addressing Mode:**

In **direct addressing**, the operand is located at a specific memory address that is given directly in the instruction.

**Syntax:**

MOV destination, [address]

**Explanation:**

* The address is a direct memory location.
* The square brackets ([ ]) indicate that it's a memory address.

**Example:**

MOV AX, [1234h]

This instruction moves the contents of the memory at address 1234h into register AX.

**4. Indirect Addressing Mode**

In **indirect addressing**, the operand’s address is stored in a register, and the instruction specifies the register containing the memory address of the operand.

**Syntax:**

MOV destination, [register]

**Explanation:**

* The memory address of the operand is held in the specified register, and the square brackets indicate that the contents of the register point to the operand’s memory address.

**Example:**

MOV AX, [BX]

This instruction moves the contents of memory at the address contained in register BX into AX.

**5. Register Indirect Addressing Mode**

In **register indirect addressing**, a register (such as SI, DI, BX, or BP) contains the address of the operand, and no explicit memory address is provided.

**Syntax:**

MOV destination, [register]

**Explanation:**

* Similar to indirect addressing, but the register contains the address of the operand.

**Example:**

MOV AX, [SI]

This instruction moves the contents of the memory location pointed to by the SI register into AX.

**6. Based Addressing Mode**

In **based addressing**, the operand’s address is calculated by adding an offset to the contents of a base register (such as BX or BP).

**Syntax:**

MOV destination, [base\_register + offset]

**Explanation:**

* The base register holds a starting address, and the offset is added to that address to compute the final memory address.
* Used for addressing arrays and data structures.

**Example:**

MOV AX, [BX + 10h]

This instruction moves the contents of memory at the address BX + 10h into AX.

**7. Indexed Addressing Mode**

In **indexed addressing**, the operand’s address is formed by adding an index register to a constant offset.

**Syntax:**

MOV destination, [index\_register + offset]

**Explanation:**

* The index register contains a value that, when added to the offset, forms the operand’s memory address.
* Commonly used to access elements in arrays.

**Example:**

MOV AX, [SI + 100h]

This instruction moves the contents of memory at the address SI + 100h into AX.

**8. Based Indexed Addressing Mode**

In **based indexed addressing**, both a base register and an index register, plus an optional offset, are used to calculate the effective address.

**Syntax:**

MOV destination, [base\_register + index\_register + offset]

**Explanation:**

* This mode combines both the base register and the index register to determine the operand's address.
* Often used for more complex data structures like multidimensional arrays.

**Example:**

MOV AX, [BX + SI + 10h]

This instruction moves the contents of memory at the address BX + SI + 10h into AX.

**9. Relative Addressing Mode**

In **relative addressing**, the operand’s address is given relative to the current value of the instruction pointer (IP), used mainly for jump and branch instructions.

**Syntax:**

JMP label

**Explanation:**

* The address is calculated relative to the current value of the instruction pointer (IP), typically used in control flow instructions like jumps and loops.

**Example:**

JMP 100h

This instruction causes a jump to the address 100h relative to the current IP.

**10. Memory Addressing Mode**

In **memory addressing mode**, the operand is located in memory, and the address is determined using various addressing modes, such as direct, indirect, or indexed.

**Syntax:**

MOV destination, [address\_or\_register]

**Explanation:**

* Operand is located in memory, accessed directly or indirectly through registers or base indices.
* The square brackets ([ ]) indicate memory access.

**Example (Direct):**

MOV AX, [1234h]

This instruction moves the contents of memory at address 1234h into AX.

**Example (Indirect):**MOV AX, [BX].

This instruction moves the contents of the memory at the address pointed to by BX into AX.

**2.0 Aim of the Micro-Project :**

## This Micro-Project aims at:

* The main objective of the micro-project is “**Programs On All Addressing Modes of 8086 MIC**”.

1. **Flexibility in Data Access**:
   * Addressing modes allow the 8086 processor to access operands in various ways, making it versatile in how data is retrieved or stored. This flexibility is essential for performing different types of operations on data located in registers, memory, or both.
2. **Efficient Memory Utilization**:
   * By supporting modes like indirect, indexed, and based addressing, the 8086 can access large amounts of data efficiently, minimizing the need for hardcoding specific memory addresses. This allows the processor to access data dynamically and makes better use of available memory.
3. **Code Compactness**:
   * Addressing modes help in reducing the size of instructions by enabling the use of registers or pointers to refer to memory locations, rather than explicitly specifying full memory addresses in every instruction. This results in shorter, more efficient code.
4. **Support for Complex Operations**:
   * Some addressing modes, like based-indexed or indirect, support complex operations like accessing arrays, tables, and data structures. This is particularly useful in tasks like looping through arrays or manipulating data that requires dynamic address computation.
5. **Improved Program Modularity and Portability**:
   * Using flexible addressing modes, programs can be more modular. For example, programs can access different parts of memory or registers without having to change the code each time. This aids in writing reusable code and enhances portability across different memory layouts or configurations.
6. **Efficient Execution of Control Flow**:
   * The **relative addressing mode** allows for more efficient implementation of branch operations, such as jumps or loops, by specifying offsets relative to the current instruction pointer. This reduces the need for absolute memory addresses in branch instructions.

Overall, the addressing modes in the 8086 microprocessor provide the means for efficient, flexible, and powerful data handling, which is crucial for complex programs and efficient execution.

**3.0 Course Outcomes Integrated**

* + Manage database using sql command.
  + Implement advanced sql concept on datadase.
  + Write PL/SQl code for given database application.

**4.0 Actual Procedure Followed**

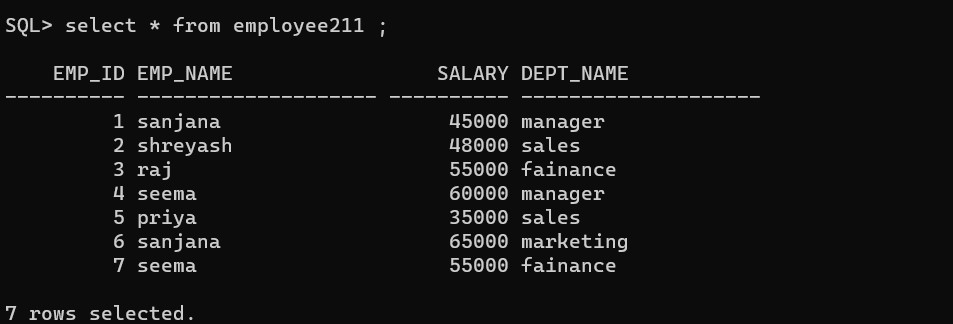
* In initial phase, we started to search the topics of micro-projects from different resources. We were listed number of topic as follows:
  + - Assembler directives.
    - Instructions in 8086 MIC.
    - Addressing Modes.
    - Pipelining.
    - Memory segmentation.
* We discussed on above topic with our micro-project guide, course guide suggested us to prepare Proposal and develop micro-project on “Programs On All Addressing Modes of 8086 MIC” and provide some idea about micro project topic.
* We divided our task in different phase. We go through reference books from our library also we Search information through various websites on browsers.
* we started to prepare documentation.
* Finally, we completed our micro-project task. And then we submitted it to our guide

**5.0 Actual Resource Used** :

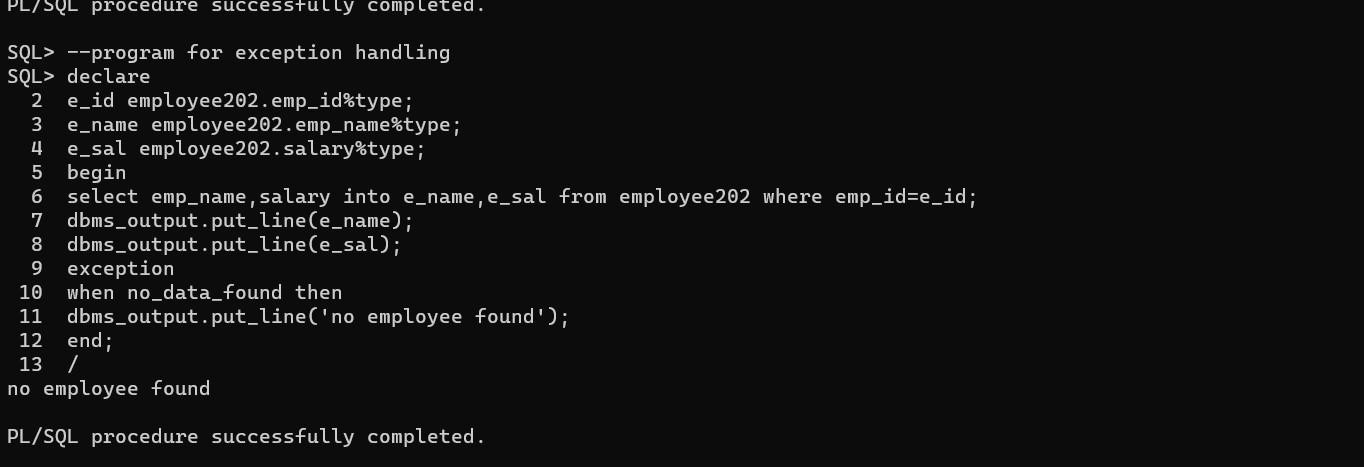
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sr.  No. | Name of Resource / Material | Specifications | Qty. | Remarks |
| 1 | Operating system | Windows 11 | - | - |
| 2 | Reference Book | 1.Microprocessor 8086: Architecture, Programming and Interfacing" by Sunil Mathur,  2.8086 Microprocessor Bharat Acharya Education: Architecture and Interfacing" by Bharat Acharya | - | - |
| 3 | Website | [Udemy's free Microprocessors Tutorial](https://www.udemy.com/course/8086-microprocessor/)  [NPTEL's Microprocessors and Interfacing course](https://onlinecourses.nptel.ac.in/noc20_ee11/preview) | - | - |
| 4 | software | EMU8086: [Softonic's EMU8086 emulator](https://emu8086-microprocessor-emulator.en.softonic.com/) | - | - |

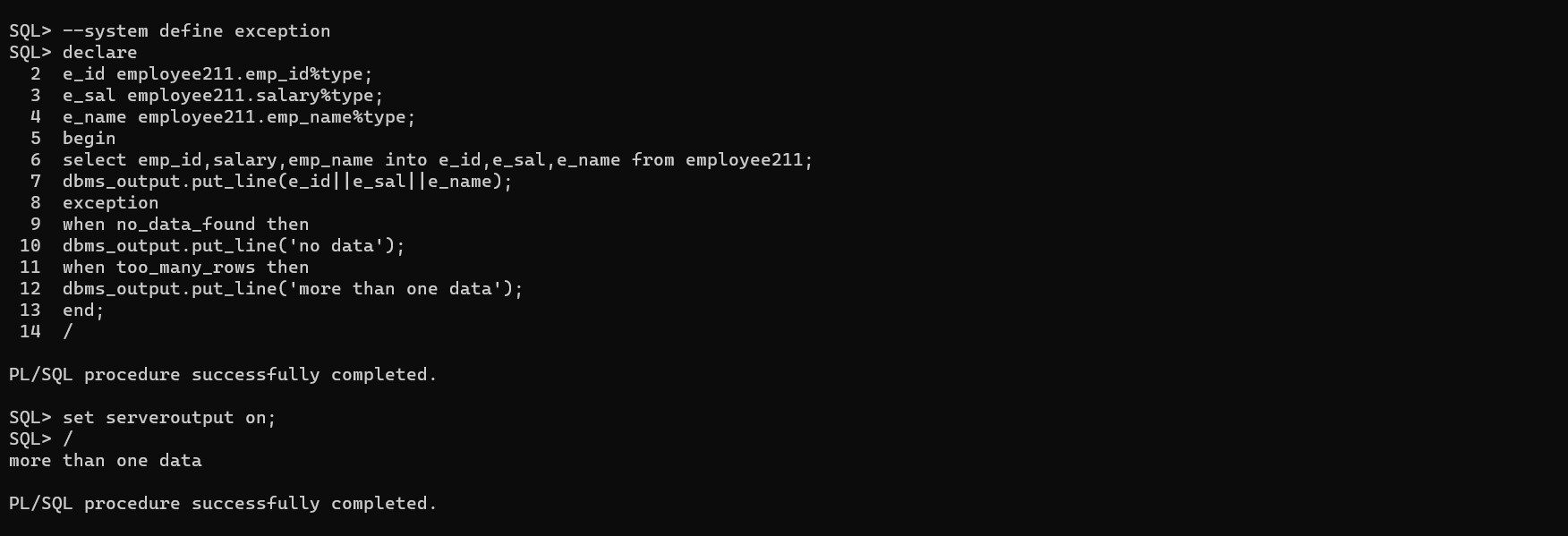
**6.0 Outputs of the Micro-Projects :**

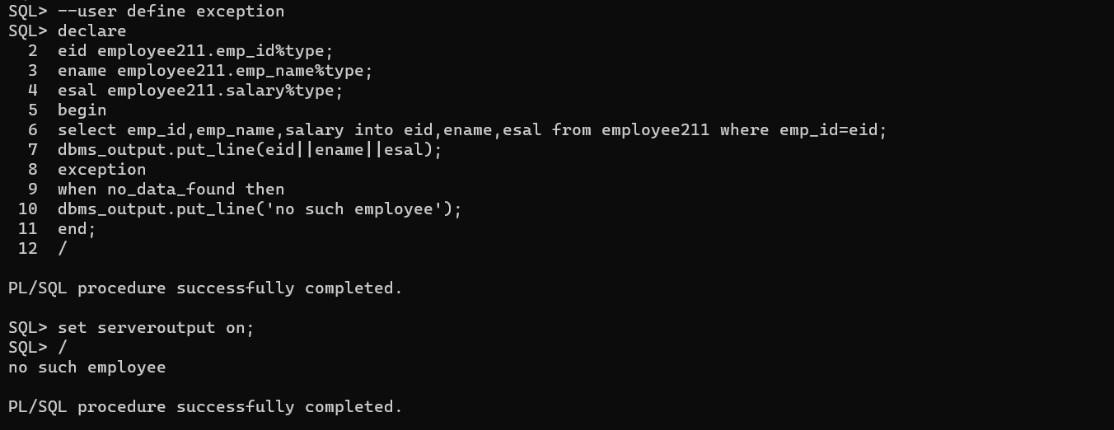
Consider the employee table.



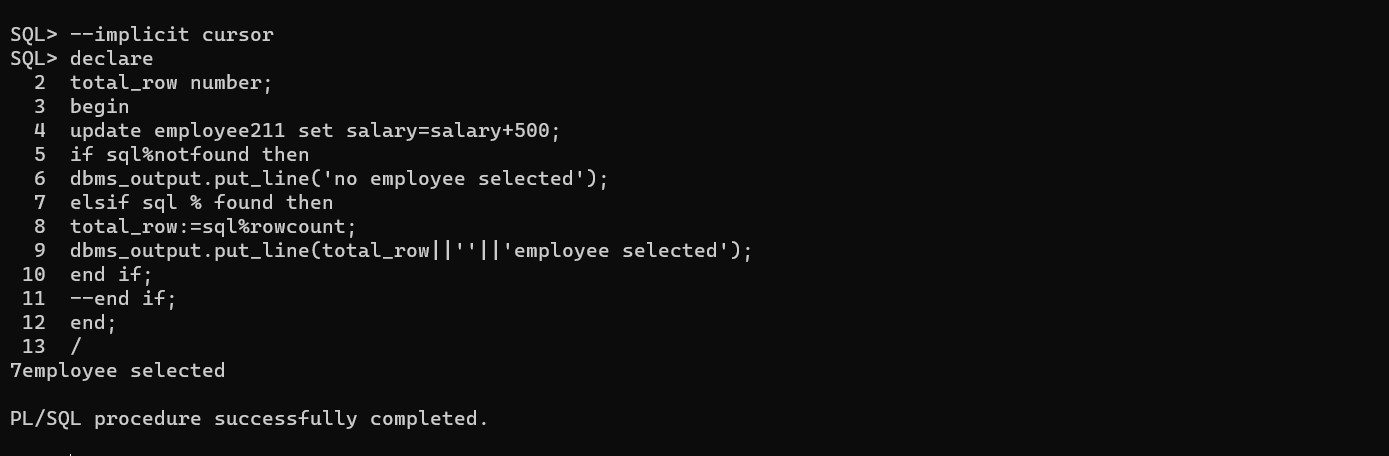
* + - Program for exception handling



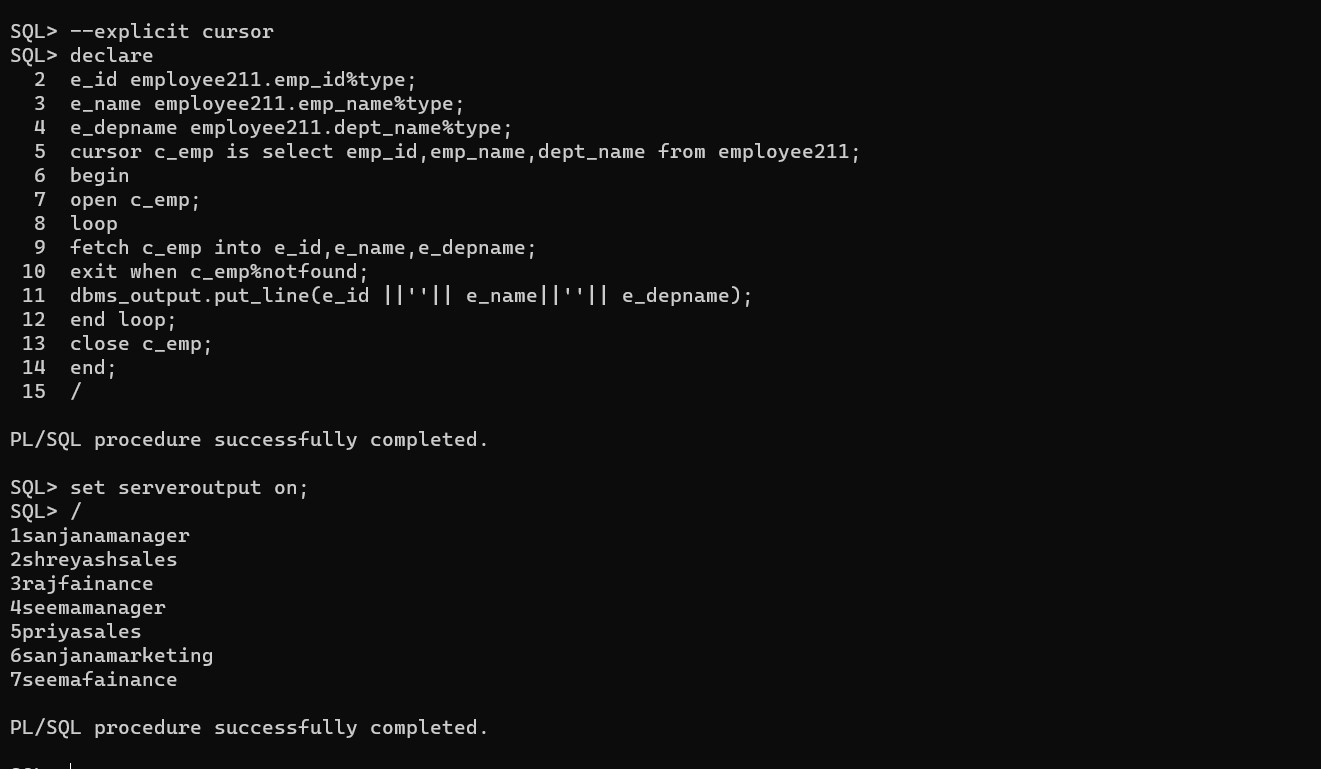
* Program for system define exeption
* Program for user define exception:



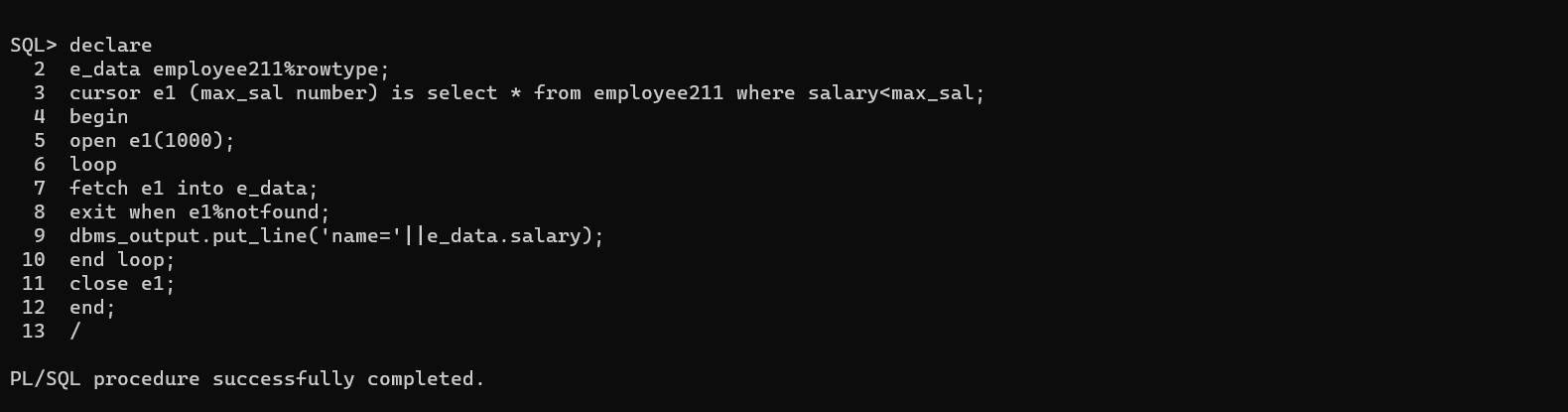
* + - Program for implicit cursor:



* + - Program for explicit cursor :



* Program for parametrized cursor:



**7.0 Skill Developed / learning out of this Micro –Project:**

* In this project we are able to understand the concept of Adderessing modes of 8086 Micrpprocessor.
* We are able to understand the different types of Adderess modes .
* We increase our communication skills.
* We learnt how to handle exceptions/error.
* We are able to use Adderess modes in programs.

**8.0 Any other :**

* + - **Conclusion :**

In conclusion, the addressing modes of the 8086 microprocessor play a crucial role in providing flexibility, efficiency, and versatility in data access. They allow the processor to work with data in various ways, such as directly, indirectly, or through indexed and based methods. These modes enable the 8086 to:

* **Access data dynamically** from registers, memory, or computed addresses, improving the efficiency of code execution.
* **Simplify complex tasks** like handling arrays, loops, and data structures through indexed and based addressing.
* **Enhance program portability** and modularity by enabling indirect references and memory-relative addressing.
* **Minimize code size**, since many addressing modes allow for more compact instructions, especially when working with memory addresses indirectly or through registers.

By offering a variety of addressing modes, the 8086 processor can efficiently handle a wide range of applications, making it a powerful microprocessor for both simple and complex computing tasks. The diverse addressing modes not only optimize memory usage but also facilitate easier development of flexible and efficient software.

### **Reference**:

<https://www.grafiati.com/en/literature-selections/intel-8086-microprocessor/>

<https://www.geeksforgeeks.org/architecture-of-8086/>