INDRODUCTION

1 **SEQUENCE CONTROL FOR SUBPROGRAM**

2 EXCEPTION

Sequence Control for Subprogram:

There may two kind of subprogram:

- Simple Call-Return
- Recursive Calls

Simple Call return:

It supports the following properties:

- No Recursive Calls- There is no recursion in simple call return
- Explicit Calls- Each subprogram should be called explicitly
- Complete Execution- Each subprogram should be execute completely
- Immediate Control Transfer- It is happen when a subprogram call a another subprogram
- Single Execution Sequence-There must be a single execution sequence for a given condition.

Implementation:

1. There is a little a difference between definition and

Sequence Control for Subprogram:

Implementation:

Activation of a subprogram

2)An activation is implemented as a two part:

- **A Code Segment**
- **Activation Record**

3)The code segment contain the part of the subprogram that never change known as a static part

4)Each activation record is created each new time when subprogram is called it contains the that parts that is dynamic in nature

While dealing or implementing this subprogram we must take care this pointer:

- **Current Instruction Pointer(CIP):**

It is used to point the memory location where the at present a instruction is executing.

Sequence Control for Subprogram:

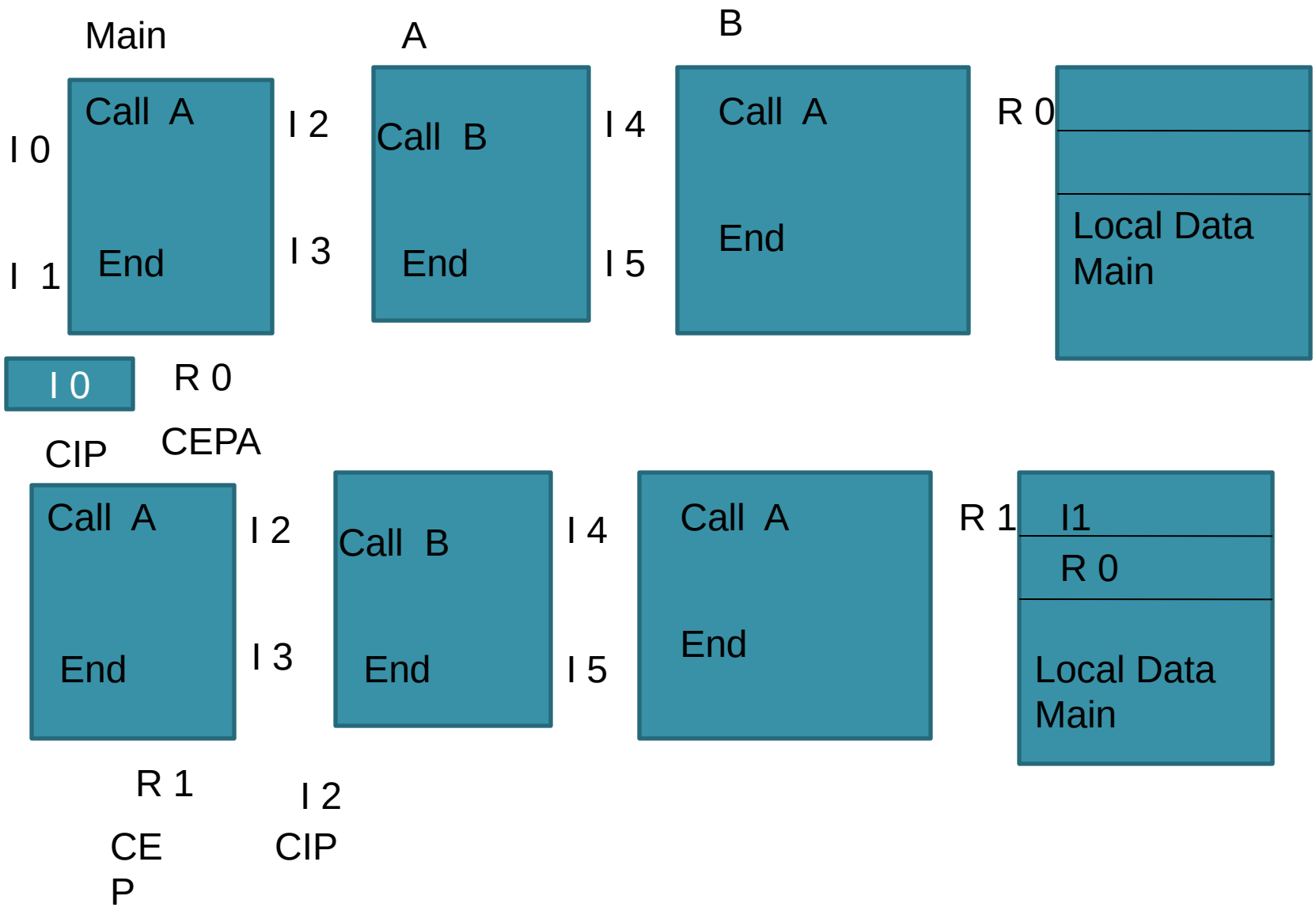
- **Current Environment Pointer(CEP):**

Because all activations of the same subprogram contains the same code segment but different activation records. So CIP is not enough to provide exact information so that's why we required the one more pointer is known as CEP. It will points to address the current activation record of a subprogram

Working:

At initial state there is only one activation of the main program. So there will be the only one activation record for the main prorgam.CEP is assigned a pointer to it. Where as CIP is assigned a pointer that points to the first instruction of the subprogram. If a subprogram calls a other subprogram then a CIP and CEP assign with new value. But the previous value of CIP & CEP are stored before calling of a next subprogram. When a return statement is reached that terminates the activation record of the subprogram again CIP and CEP assign with the old value of the main program and continue with it.

Sequence Control for Subprogram:



Exception:

An exception is hardware-detected run time error or unusual condition detected by the software.

Examples:

- ✓ **Arithmetic overflow**
- ✓ **End-of File Input**
- ✓ **Wrong Type of Input Data**
- ✓ **User Defined Conditions that may or may not recognized as error**

Requirement of Exception Handling Techniques:

1. As it is provide a uniform method without any confusion for a user
2. Allow user to check these errors without checking these conditions

Exception Handling:

When Exception occurs there lot of things that can be performed .The control may transfer to the following types of the function:

- **In-Built Function**
- **Programmer defined Function also known a the Handlers for the particular exception**

To catch exceptions we must place a portion of code under exception inspection. This is done by enclosing that portion of code in a try block. When the certain situation of a exception is arises in that block the control is suddenly transfer to the exception handler(the function that is particularly designed for handle the effect of exception). If there is no exception the flow of execution remain unchanged ignoring the handler.

Exception Handling:

A n exception is thrown by using the throw keyword from inside the **try block.** Exception handlers are declared with the keyword

catch that must be placed just after the try block.

Example:

```
# include <iostream>
int main() {
try
{
throw 20
}
catch(int e)
{
cout <<“An exception occurred.” <<e  <<end1;
}
return 0
}
```

Exception Handling:

Types of Exception: May be categorized as :

1. **Synchronous versus Asynchronous**

If the event occurs at the same place every time the program is executed with the same data is and memory location is known as **Synchronous**

With the of hardware malfunctions **Asynchronous** events are caused by devices external to the processor memory. These events can be handled after the completion of the current instruction, that makes them easier to handle

2. User Mask able versus User Non Mask able

If the event can be **masked** or **disabled** by a user task, it is user **mask able** .This mask simply controls the whether the hardware responds to the exception or not

If an event can't be masked or disabled by the user task , it is user **non mask able**

Exception Handling:

Advantages of Exception:

1. Separating Error-Handling Code from the "Regular code
2. Propagating Errors up to the Call Stack
3. Grouping and Differentiating Error Types

Co routines

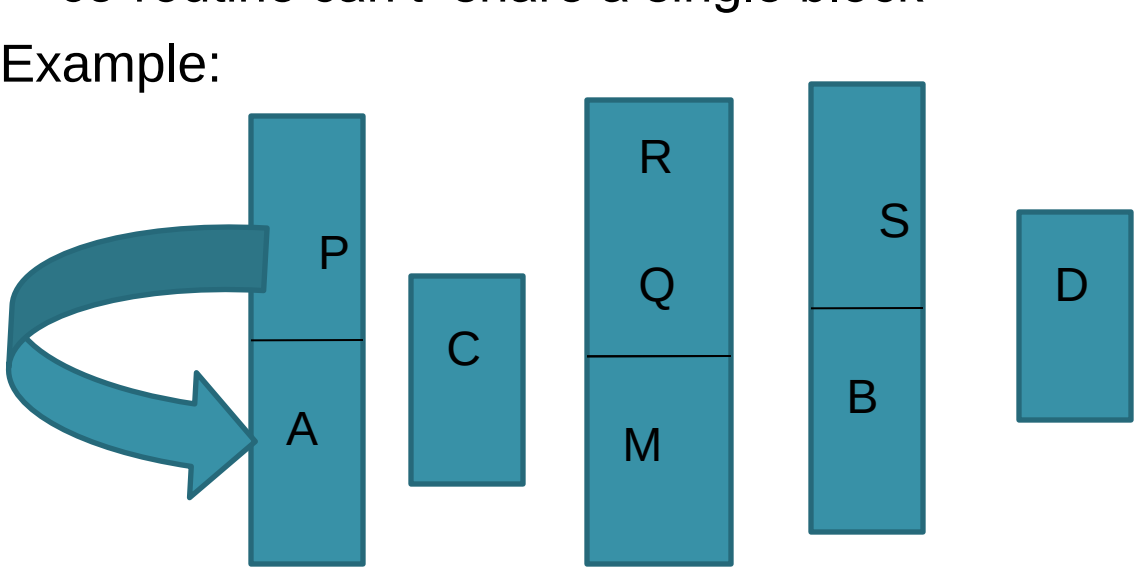
Co-routines are the execution contexts that exist concurrently, but that execute one at a time, and that transfer control to each other explicitly by **name**

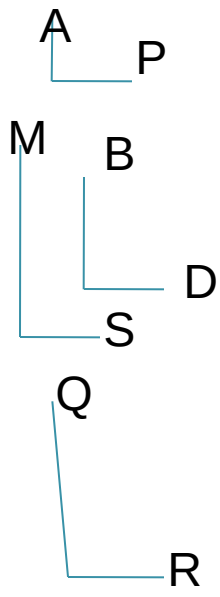
Co routine can be used to implement either :

- **Iterators**
- **Threads(in java)**

As they are concurrent(simultaneously started but not completed , co-routine can't share a single block

Example:





Co routines

Control transfer between Co-routine A and B