STES's

NBN SINHGAD SCHOOL OF ENGINEERING

Ambegaon(Bk), Pune.

Department of Computer Engineering

LABORATORY MANUAL

2022-23

Laboratory Practice-I (2019 Pattern)

TE-COMPUTER ENGINEERING

SEMESTER-I

Subject Code: 310248

TEACHING SCHEME
Practical: 2Hrs/Week

EXAMINATION SCHEME
Term Work: 25 Marks
Practical Exam: 25 Marks

HOD (Computer Engineering)

Assignment No.	A-01
Title	Design suitable Data structures and implement Pass-I and Pass-II of a two-pass assembler for pseudo-machine. Implementation should consist of a few instructions from each category and few assembler directives. The output of Pass-I (intermediate code file and symbol table) should be input for Pass-II.
Date	
Signature of Faculty	

Assignment No: A-01

Title: Design suitable Data structures and implement Pass-I and Pass-II of a two-pass assembler for pseudo-machine. Implementation should consist of a few instructions from each category and few assembler directives. The output of Pass-I (intermediate code file and symbol table) should be input for Pass-II.

Objectives:

- To study the design and implementation of 1st pass of two pass assembler.
- To study the design and implementation of 2nd pass of two pass assembler.
- To study the categorized instruction set of assembler.
- To study the data structure used in assembler implementation.

Theory:

Pass-I

General Design Procedure of Two Pass Assembler

1. Specify the problem
2. Specify data structures
3. Define format of data structures
4. Specify algorithm
5. Look for modularity [capability of one program to be subdivided into independent programming units.]
6. Repeat 1 through 5 on modules.

Specify the problem

Pass1: Define symbols & literals.

1. Determine length of m/c instruction [MOTGET1]
2. Keep track of Location Counter [LC]
3. Remember values of symbols [STSTO]
4. Process some pseudo ops[EQU,DS etc] [POTGET1]
5. Remember Literals [LITSTO]

Data structure of Pass1: Databases

1. Input source program
2. "LC" location counter used to keep track of each instructions addr.
3. M/c operation table (MOT) [Symbolic mnemonic & length]
4. Pseudo operation table [POT], [Symbolic mnemonic & action]
5. Symbol Table (ST) to store each lable & it's value.
6. Literal Table (LT), to store each literal (variable) & it's location.
7. Copy of input to used later by PASS-2.

Flowchart:

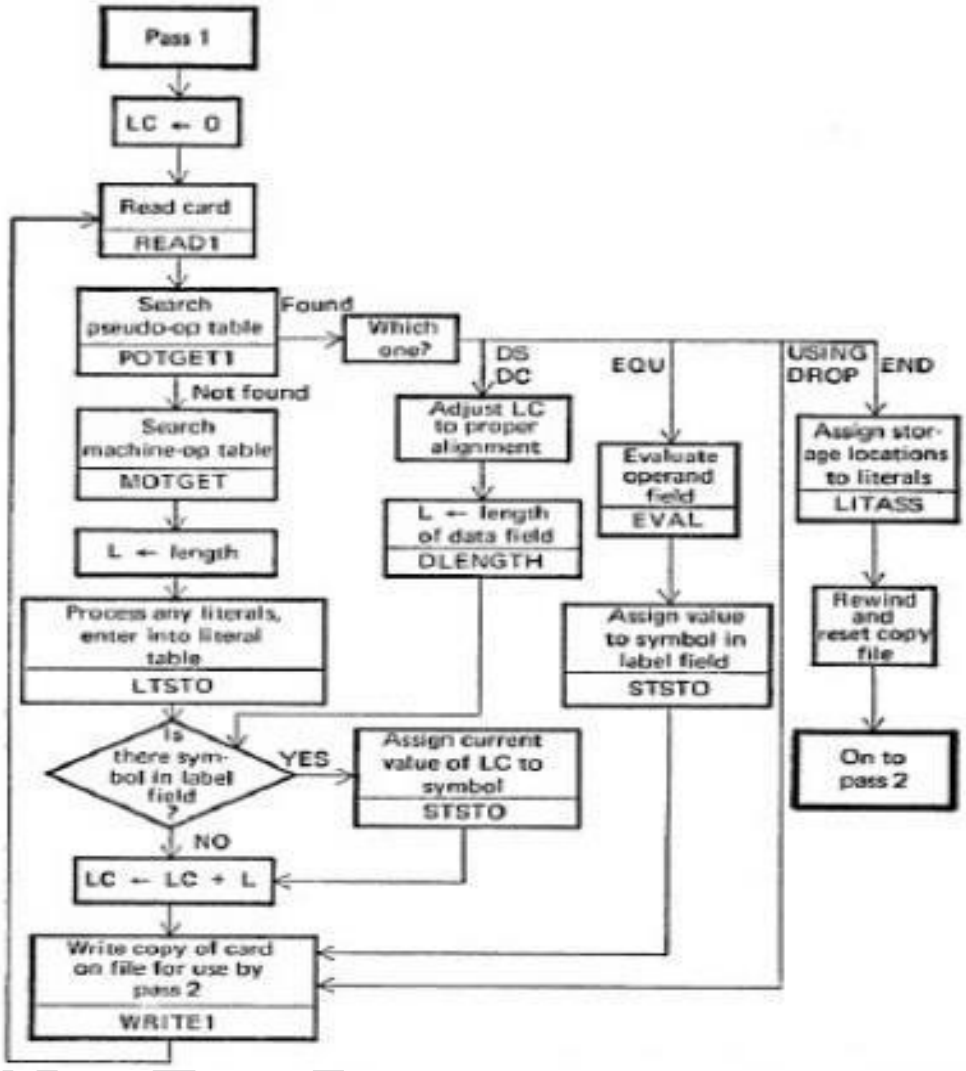


Fig. Pass-I of Two Pass Assembler

Input:

Source code of Assembly Language

```
SAMPLE    START     100
          USING     *, 15
          L         1, FOUR
          A         1, =F'3'
          ST        1, RESULT
          SR        1, 2
          LTORG
          L         2, FIVE
          A         2, =F'5'
```

```
                A       2, =F'7'
FIVE            DC      F'5'
FOUR            DC      F'4'
RESULT          DS      1F
                END
```

Output:

```
100     SAMPLE  START   100
100             USING   *, 15
100             L       1, FOUR
104             A       1, =F'3'
108             ST      1, RESULT
112             SR      1, 2
114             LTORG
124             L       2, FIVE
128             A       2, =F'5'
132             A       2, =F'7'
136     FIVE    DC      F'5'
140     FOUR    DC      F'4'
144     RESULT  DS      1F
152             5
156             7
160             END
```

Machine Opcode Table (MOT)

Mnemonic	Hex / Binary Code	Length (Bytes)	Format
L	5A	4	RX
A	1B	4	RX
ST	50	4	RX
SR	18	2	RR

Pseudo Opcode Table (POT)

Pseudo op	Address / Name of Procedure to implement pseudo operation
START	START
USING	USING
DC	DC
DS	DS
LTORG	LTORG

END	END

Symbol Table (ST)

Sr. No	Symbol name	Address	Value	Length	Relocation
1	SAMPLE	100	--	160	R
2	FIVE	136	5	4	R
3	FOUR	140	4	4	R
4	RESULT	144	___	4	R

Literal Table (LT)

Sr. No	Literal	Address	Length
1	3	120	4
2	5	152	4
3	7	156	4

Pass-II

Generate object program

1. Look up value of symbols [STGET]
2. Generate instruction [MOTGET2]
3. Generate data (for DS, DC & literals)
4. Process pseudo ops[POTGET2]

Pass2: Databases

1. Copy of source program input to Pass1.
2. Location Counter (LC)
3. MOT [Mnemonic, length, binary m/c op code, etc.]
4. POT [Mnemonic & action to be taken in Pass2
5. ST [prepared by Pass1, label & value]
6. Base Table [or register table] indicates which registers are currently specified using 'USING' pseudo op & what are contents.
7. Literal table prepared by Pass1. [Lit name & value].

Flowchart:

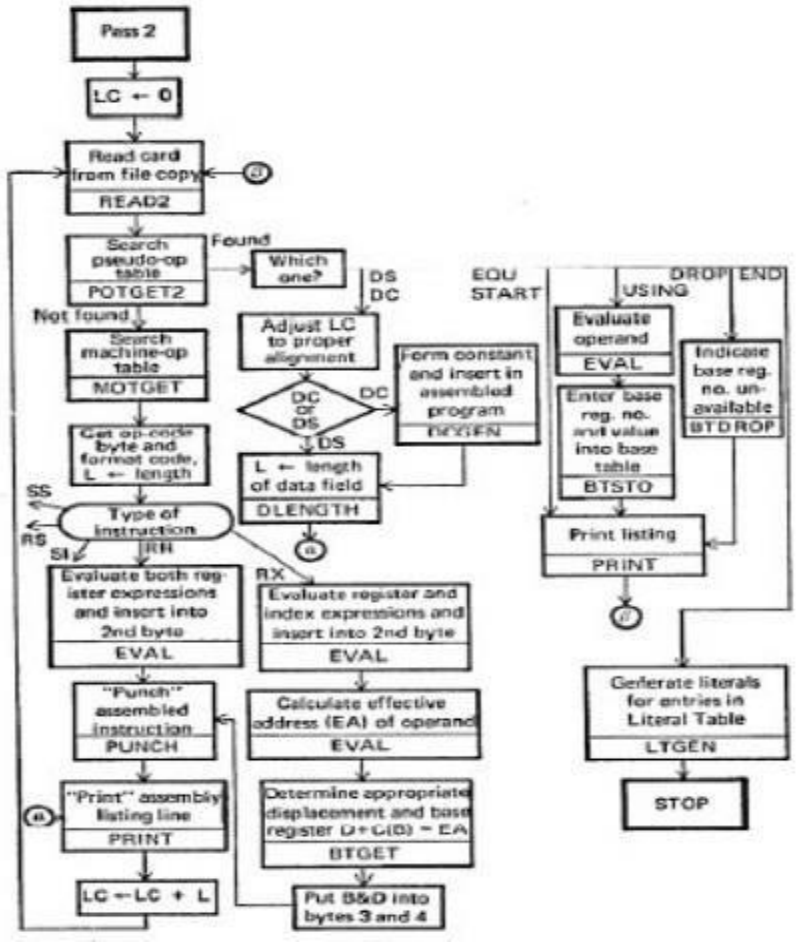


Fig. Pass-II of Two Pass Assembler

Input:

Intermediate code of pass-1.

LC	LABEL	INSTR.	OPERANDS
100	SAMPLE	START	100
100		USING	*, 15
100		L	1, FOUR
104		A	1, =F’3’
108		ST	1, RESULT
112		SR	1, 2
114		LTORG	
124		L	2, FIVE
128		A	2, =F’5’
132		A	2, =F’7’
136	FIVE	DC	F’5’
140	FOUR	DC	F’4’
144	RESULT	DS	1F

```
152         5
156         7
160         END
```

Output:

Object Code

```
        LC          OPCODE      OPERAND
-------------------------------------------------------
        100         5A          1,40(0,15)
        104         1B          1,20(0,15)
        108         50          1,44(0,15)
        112         18          1,2
        124         5A          2,36(0,15)
        128         1B          2,52(0,15)
        132         1B          2,56(0,15)
```

Machine Opcode Table (MOT)

Mnemonic	Hex / Binary Code	Length (Bytes)	Format
L	5A	4	RX
A	1B	4	RX
ST	50	4	RX
SR	18	2	RR

Pseudo Opcode Table (POT)

Pseudo op	Address / Name of Procedure to implement pseudo operation
START	START
USING	USING
DC	DC
DS	DS
LTORG	LTORG
END	END

Symbol Table (ST)

Sr. No	Symbol name	Address	Value	Length	Relocation

1	SAMPLE	100	--	160	R
2	FIVE	136	5	4	R
3	FOUR	140	4	4	R
4	RESULT	144	___	4	R

Conclusion:

Thus we have successfully implemented pass-I of a two-pass Assembler.

FAQ's:

1. What is Assembler?
2. What are declaration statements in Assembly language?
3. Which are the data structures in Pass-I Assembler?
4. What is POT, MOT, LT and ST?
5. What is intermediate code?

Assignment No.	A-02
Title	Design suitable data structures and implement Pass-I and Pass-II of a two-pass macro-processor. The output of Pass-I (MNT, MDT and intermediate code file without any macro definitions) should be input for Pass-II.
Date	
Signature of Faculty	

Assignment No: A-02

Title: Design suitable data structures and implement Pass-I and Pass-II of a two-pass macro-processor. The output of Pass-I (MNT, MDT and intermediate code file without any macro definitions) should be input for Pass-II.

Objectives:

- To Identify and create the data structures required in the design of macro processor.
- To implement Pass-I and Pass-II of macro processor

Theory:

MACRO

- Macro allows a sequence of source language code to be defined once & then referred to by name each time it is to be referred. Each time this name occurs is a program the sequence of codes is substituted at that point.
- A macro consist of
  1. Name of the macro
  2. Set of parameters
  3. Body of macro
- Macros are typically defined at the start of program. Macro definition consists of
  1. MACRO pseudo
  2. MACRO name
  3. Sequence of statements
  4. MEND pseudo opcode terminating
- A macro is called by writing the macro name with actual parameter is an assembly program. The macro call has following syntax <macro name>.

MACRO PROCESSOR

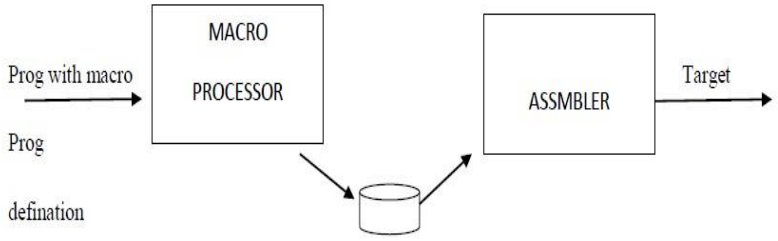


Fig. Assembly language program without macro

- Macro processor takes a source program containing macro definition & macro calls and translates into an assembly language program without any macro definition or calls. This program can now be handled over to a conventional assembler to obtain the target language.

MACRO DEINITION

Macros are typically defined at the start of a program. A macro definition consists of

1. MACRO pseudo code
2. MACRO name
3. Sequence of statement
4. MEND pseudo opcode terminating macro definition

Structure of a macro

Example

```
MACRO
INCR & ARG
ADD AREG,& ARG
ADD BRA,& ARG
ADD CREG, & ARG
MEND
```

MACRO Expansion:

- During macro expansion each statement forming the body of the macro as picked up one by one sequentially.
    a. Each statement inside macro may have as it is during expansion.
    b. The name of a formal parameter which is preceded by the character '&' during macro expansion an ordinary starting is retained without any modification. Formal parameters are replaced by actual parameters value.
- When a call is found the call processor sets a pointer the macro definition table pointer to the corresponding macro definition started in MDT. The initial value of MDT is obtained from MDT index.

Design of macro processor:

Pass-I:
Generate Macro Name Table (MNT)
Generate Macro Definition Table (MDT)
Generate IC i.e. a copy of source code without macro definitions.

MNT:

Sr.No	Macro Name	MDT Index

MDT:

Sr. No	MACRO STATEMENT

ALA:

Index	Argument

Specification of Data Bases

Pass-I data bases

1. The input macro source desk.
2. The output macro source desk copy for use by passes 2.
3. The macro definition table (MDT) used to store the names of defined macros.
4. Macro name table (MDT) used to stare the name of defined macros.
5. The Macro definition table counter used to indicate the next available entry in MNT.
6. The macro name table counter counter(MNTC) used to indicate next available entry in MNT.
7. The arguments list array (ALA) used to substitute index markers for dummy arguments before starting a macro definition.

Implementation of a 2 pass algorithm

1. We assume that our macro processor is functionally independent of the assembler and that the output text from the macro processor will be fed into the assembler.
2. The macro processor will make two independent scans or passes over the input text, searching first for macro definitions and then for macro calls
3. The macro processor cannot expand a macro call before having found and saved the corresponding macro definitions.
4. Thus we need two passes over the input text, one to handle macro definitions and other to handle macro calls.
5. The first pass examines every operation code, will save all macro definitions in a macro Definition Table and save a copy of the input text, minus macro definitions on the secondary storage.
6. The first pass also prepares a Macro Name Table along with Macro Definition Table as seen in the previous assignment that successfully implemented pass as seen in the previous assignment that successfully implemented pass-I of macro Pre-processor.
   The second pass will now examine every operation mnemonic and replace each macro name with the appropriate text from the macro definitions.

SPECIFICATION OF DATABASE

Pass-II databases:

1. The copy of the input source deck obtained from Pass- I
2. The output expanded source deck to be used as input to the assembler
3. The Macro Definition Table (MDT), created by pass 1

4. The Macro Name Table (MNT), created by pass 1
5. The Macro Definition Table Counter (MNTC), used to indicate the next line of text to be used during macro expansion
6. The Argument List Array (ALA), used to substitute macro call arguments for the index markers in stored macro definition

Flowchart:

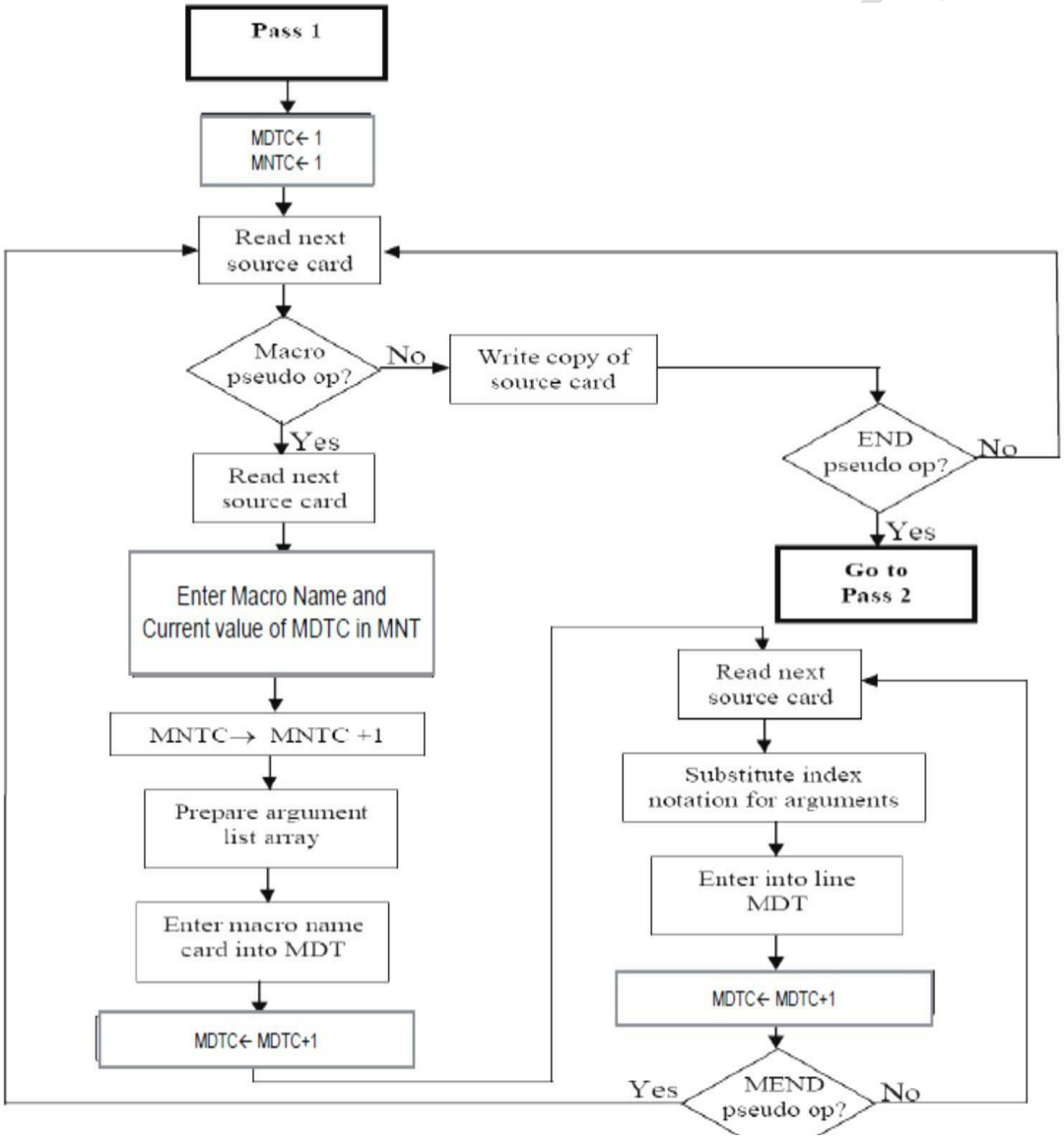


Fig. Pass-I of Two Pass Macro Processor

Flowchart:

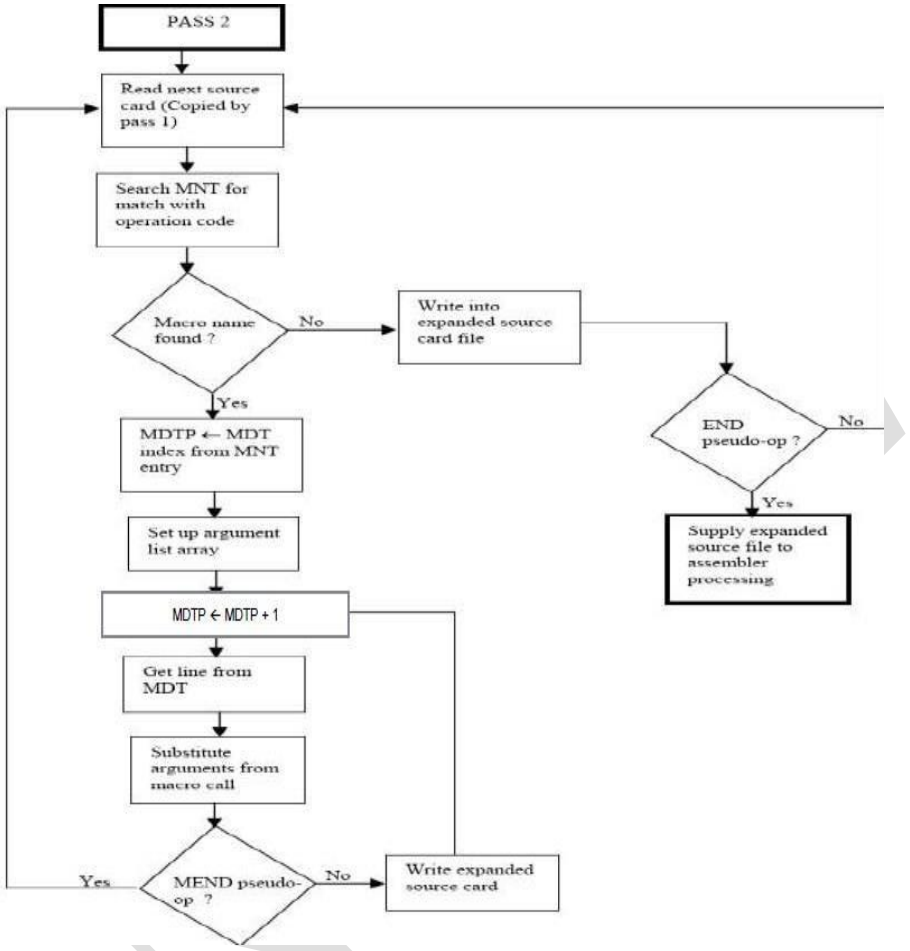


Fig. Pass-II of Two Pass Macro Processor

Conclusion:

Thus we have successfully implemented pass-I of a two-pass Macro-processor.

FAQ's:

1. What are macros? Why do we need macros?
2. Explain data structures that are used for implementing Pass I of a macro processor.
3. Explain the macro assembler facilities such as Nested Macro, Labels within Macro, Macro Parameters.
4. What are the contents of MDT and MNT?
5. Explain the algorithm of pass I of macro processor?

Assignment No.	B-01
Title	Write a program to simulate CPU Scheduling Algorithms: FCFS, SJF (Preemptive), Priority (Non-Preemptive) and Round Robin (Preemptive).
Date	
Signature of Faculty	

Assignment No: B-01

Title:
Write a program to simulate CPU Scheduling Algorithms: FCFS, SJF (Preemptive), Priority (Non-Preemptive) and Round Robin (Preemptive).

Objectives:
- To understand OS & SCHEDULLING Concepts
- To implement Scheduling FCFS, SJF, RR & Priority algorithms
- To study about Scheduling and scheduler

Theory:
Scheduling is a fundamental operating system function.
CPU scheduling is the basis of multi programming operating system. CPU scheduling algorithm determines how the CPU will be allocated to the process. These are of two types.
1. Preemptive scheduling algorithms
2. Non-Preemptive scheduling algorithms

1) **Preemptive Scheduling algorithms**: In this, the CPU can release the process even in the middle of execution. For example: the CPU executes the process p1, in the middle of execution the CPU received a request signal from process p2, then the OS compares the priorities of p1&p2. If the priority p1 is higher than the p2 then the CPU continue the execution of process p1.Otherwise the CPU preempt the process p1 and assigned to process p2.
2) **Non-Preemptive Scheduling algorithm**: In this, once the CPU assigned to a process the processor do not release until the completion of that process. The CPU will assign to some other job only after the previous job has finished.

Scheduling Criteria:
1. **Though put:** It means how many jobs are completed by the CPU with in a time period.
2. **Turn around time:** The time interval between the submission of the process and the time of the completion is the turn around time.
   Turn around time=Finished time – Arrival time
3. **Waiting time:** it is the sum of the periods spent waiting by a process in the ready queue
   Waiting time=Turn around time - Burst time
4. **Response time**: it is the time duration between the submission and first response
   Response time=First response-arrival time
5. **CPU Utilization:** This is the percentage of time that the processor is busy. CPU utilization may range from 0 to 100%.

a) FCFS SCHEDULING
First-come, first-serve scheduling (FCFS): In this, which process enter the ready queue first is served first. The OS maintains DS that is ready queue. It is the simplest CPU scheduling algorithm. If a process request the CPU then it is loaded into the ready queue, which process is the head of the ready queue, connect the CPU to that process.

Algorithm for FCFS scheduling:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Set the waiting of the first process as '0' and its burst time as its turn around time

Step 5: for each process in the Ready Q calculate

a. Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
b. Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

c. Average waiting time = Total waiting Time / Number of process
d. Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

Input:

Enter the number of processes: 3
Enter burst time for process P1: 10
Enter burst time for process P2: 5
Enter burst time for process P3: 8

Output:

Processes Burst time Waiting time Turn around time

1	10	0	10
2	5	10	15
3	8	15	23

Average waiting time = 8.33333
Average turn around time = 16

b) SJF SCHEDULING (Preemptive)

Shortest Job First: The criteria of this algorithm are which process having the smallest CPU burst, CPU is assigned to that next process. If two process having the same CPU burst time FCFS is used to break the tie.

Algorithm for SJF:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.

Step 5: Set the waiting time of the first process as '0' and its turnaround time as its burst time.

Step 6: For each process in the ready queue, calculate

a. Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
b. Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

c. Average waiting time = Total waiting Time / Number of process
d. Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

Input:

Enter the number of processes: 4
Enter burst time and arrival time for process P1: 8 0
Enter burst time for arrival time process P2: 4 1
Enter burst time for arrival time process P3: 9 2
Enter burst time for arrival time process P4: 5 3

Output:

Processes Arrival time Burst time Waiting time Turn around time

1	0	8	9	17
2	1	4	0	4
3	2	9	15	27
4	3	5	2	7

Average waiting time = 6.5
Average turn around time = 13

c) PRIORITY SCHEDULING

Priority Scheduling: These are of two types.
One is internal priority, second is external priority. The CPU is allocated to the process with the highest priority. Equal priority processes are scheduled in the FCFS order. Priorities are generally some fixed range of numbers such as 0 to 409. The low numbers represent high priority.

Algorithm for Priority Scheduling:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Sort the ready queue according to the priority number.
Step 5: Set the waiting of the first process as '0' and its burst time as its turn around time
Step 6: For each process in the Ready Q calculate

a. Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
b. Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 7: Calculate

c. Average waiting time = Total waiting Time / Number of process

d. Average Turnaround time = Total Turnaround Time / Number of process

Step 8: Stop the process

Input:

Enter the number of processes: 3
Enter burst time and priority for process P1: 10 2
Enter burst time and priority for process P2: 5 0
Enter burst time and priority for process P3: 8 1

Output:

Order in which processes gets executed
1 3 2

Processes	Burst time	Waiting time	Turn around time
1	10	0	10
3	8	10	18
2	5	18	23

Average waiting time = 9.33333
Average turn around time = 17

d) ROUND ROBIN SCHEDULING

Round Robin: It is a primitive scheduling algorithm it is designed especially for time sharing systems. In this, the CPU switches between the processes. When the time quantum expired, the CPU switches to another job. A small unit of time called a quantum or time slice. A time quantum is generally is a circular queue new processes are added to the tail of the ready queue.

If the process may have a CPU burst of less than one time slice then the process release the CPU voluntarily. The scheduler will then process to next process ready queue otherwise; the process will be put at the tail of the ready queue**.**

Algorithm for Round Robin:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue and time quantum (or) time slice
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time St.
Step 4: Calculate the no. of time slices for each process where
No. of time slice for process(n) = burst time process(n)/time slice
Step 5: If the burst time is less than the time slice then the no. of time slices =1.
Step 6: Consider the ready queue is a circular Q, calculate

a. Waiting time for process(n) = waiting time of process(n-1)+ burst time of process(n-1) + the time difference in getting the CPU from process(n-1)
b. Turn around time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).

Step 7: Calculate

c. Average waiting time = Total waiting Time / Number of process
d. Average Turnaround time = Total Turnaround Time / Number of process

Step 8: Stop the process

Input:
Enter the number of processes: 3
Enter burst time for process P1: 10
Enter burst time for process P2: 5
Enter burst time for process P3: 8
Enter time quantum: 2

Output:
Processes Burst time Waiting time Turn around time

1	10	13	23
2	5	10	15
3	8	13	21

Average waiting time = 12
Average turn around time = 19.6667

Conclusion:
Hence, we have studied and implemented Operating system scheduling algorithms.

FAQ's:
1. What is CPU Scheduling?
2. List and define scheduling criteria.
3. Define preemption & non-preemption.
4. State FCFS, SJF, Priority & Round Robin scheduling.
5. Compare FCFS, SJF, RR, Priority w.r.t. waiting time.

Assignment No.	B-02
Title	Write a program to simulate Page replacement algorithm.
Date	
Signature of Faculty	

Assignment No: B-02

Title: Write a program to simulate Page replacement algorithm.

Objectives:

- To understand Page replacement policies
- To understand paging concept
- To understand Concept of page fault, page hit, miss, hit ratio etc

Theory:

Paging

- Paging is a memory-management scheme that permits the physical-address space of a process to be noncontiguous.
- In paging, the physical memory is divided into fixed sized blocks called **page frames**

and logical memory is also divided into fixed size blocks called **pages** which are of same size as that of page frames. When a process is to be executed, its pages can be loaded into any unallocated frames (not necessarily contiguous) from the disk.

How a logical address is translated into a physical address:

- In paging, address translation is performed using a mapping table, called **Page Table**. The operating system maintains a page table for each process to keep track of which page frame is allocated to which page. It stores the frame number allocated to each page and the page number is used as index to the page table.

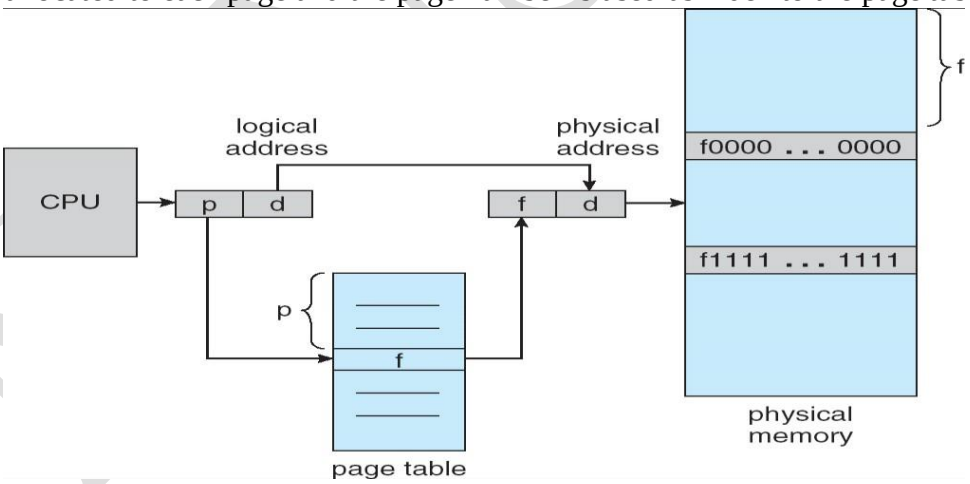


Fig1 paging

Page Replacement Algorithm :

- Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated.
- Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

- When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.
- A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself.
- There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults.

Page fault :

- A **page fault** (sometimes called #PF, PF or hard **fault**) is a type of exception raised by computer hardware when a running program accesses a memory **page**that is not currently mapped by the memory management unit (MMU) into the virtual address space of a process.

Page hit :

- A **hit** is a request to a web server for a file, like a web **page**, image, JavaScript, or Cascading Style Sheet. When a web **page** is downloaded from a server the number of "**hits**" or "**page hits**" is equal to the number of files requested.

Page frame :

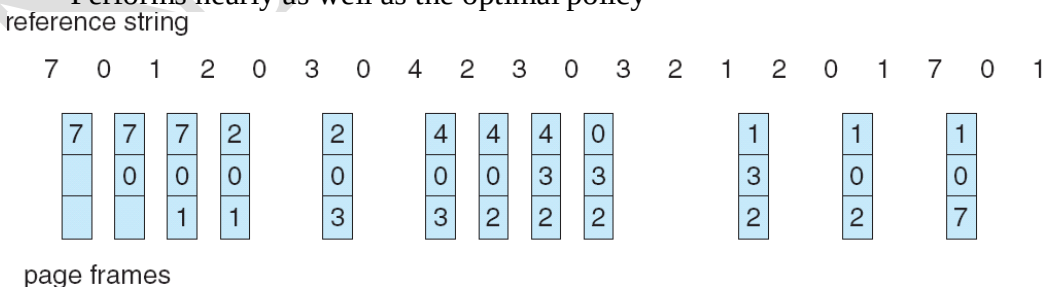
- The **page frame** is the storage unit (typically 4KB in size) whereas the **page** is the contents that you would store in the storage unit ie the **page frame**. For eg) the RAM is divided into fixed size blocks called **page frames** which is typically 4KB in size, and each **page frame** can store 4KB of data ie the **page**.

Page table :

- A **page table** is the data structure used by a virtual memory system in a computer operating system to store the mapping between virtual addresses and physical addresses.

LRU

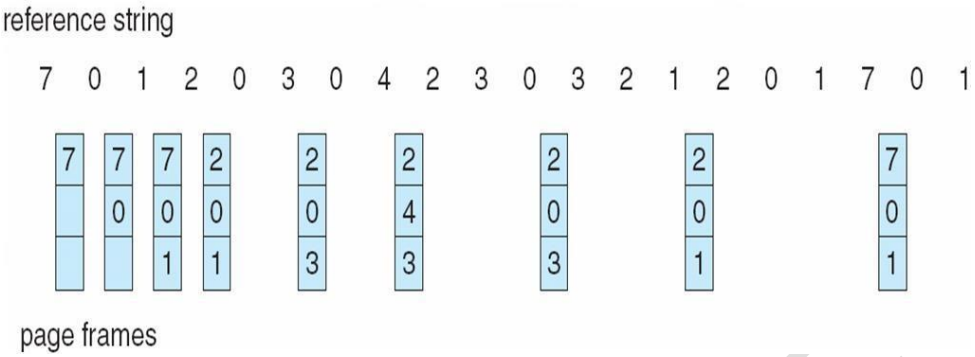
- Replaces the page that has not been referenced for the longest time:
  – By the principle of locality, this should be the page least likely to be referenced in the near future.
  – Performs nearly as well as the optimal policy



Example of LRU

Optimal

Optimal Page Replacement refers to the removal of the page that will not be used in the future, for the longest period of time.



Example of Optimal

Algorithm for LRU Page Replacement:
Step 1: Create a queue to hold all pages in memory
Step 2: When the page is required replace the page at the head of the queue
Step 3: Now the new page is inserted at the tail of the queue
Step 4: Create a stack
Step 5: When the page fault occurs replace page present at the bottom of the stack
Step 6: Stop the process.

Algorithm for Optimal(LFU) Page Replacement:
Here we select the page that will not be used for the longest period of time.
Step 1: Create an array.
Step 2: When the page fault occurs replace page that will not be used for the longest period of time.
Step 3: Stop

Conclusion:
The various memory management page replacement algorithms were studied and successfully implemented.

FAQ's:

1. What is paging.
2. What is page replacement policies.
3. Define page table, page hit, page fault, page reference.
4. What is FIFO page replacement.
5. What is LRU and OPT page replacement.
6. State virtual memory.
7. Define demand paging.
8. What is the difference between physical memory and logical memory?

Assignment No:	B-01
Title:	Understanding the connectivity of Raspberry-Pi / Adriano with IR sensor. Write an application to detect obstacle and notify user using LEDs.
Date:	
Signature of Faculty:	

Assignment No: B-01

Title: Understanding the connectivity of Raspberry-Pi / Adriano with IR sensor. Write an application to detect obstacle and notify user using LEDs.

Objectives:

- To understand the concept of Proximity sensor
- To interface Proximity sensor with Raspberry Pi model
- To program the Raspberry Pi model to detect the nearest object using proximity sensor and give indication through led.

Software:

- Raspbian OS (IDLE)

Hardware Modules:

- Raspberry Pi Board
- Proximity sensor, Led, 330 ohm register
- Monitor

Theory:

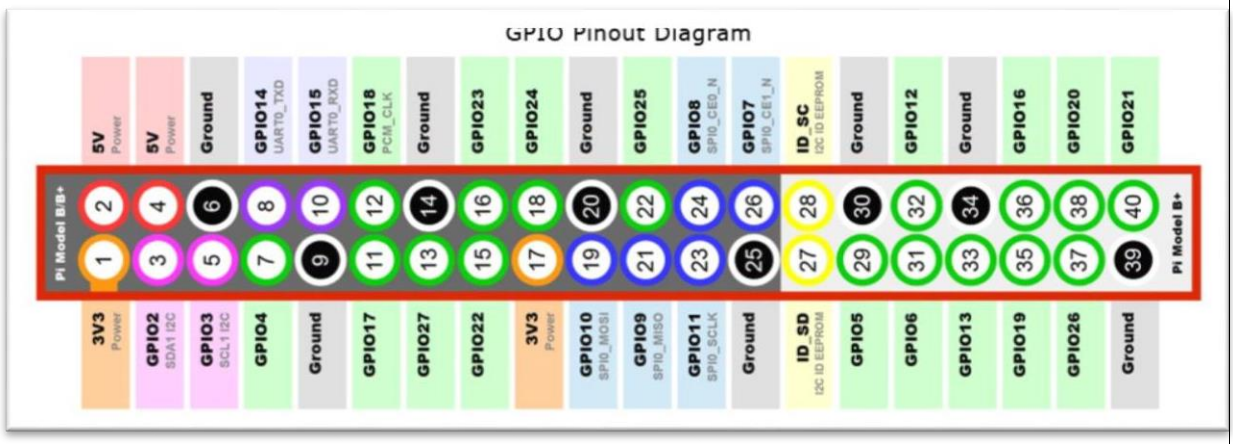
- Proximity IR sensor is a small board containing an IR transmitter, photodiode, IR Receiver
  and some processing circuitry.
- This is a discrete sensor that senses when an object comes near to the sensor face
  It works by detecting reflected light coming from its own infrared lights
- By measuring the amount of reflected infrared light & it can glow Onboard led when object is
  directly front of it.
- In Proximity, it consists of two leds, one is the transmitter (IR LED) and another is receiver
- (photodiode).
- The IR led transmits the infrared light signal which reaches till the object and deflects back.
- The Photo diode receives the deflected light.
- This signal is then amplified & status of this signal is checked by the microcontroller.
- Proximity sensor is more sensitive but it detects only object but cannot measure a distance
  value.
- By using a potentiometer, we can change sensitivity accordingly.
- When this sensor detects the object, it gives output as a digital value i.e. '1' and if not detected then the value is '0'

GPIO Modes:

GPIO Modes GPIO The **GPIO.BOARD** option specifies that you are referring to the pins by the number of the pin the the plug - i.e the numbers printed on the board (e.g. P1) and in the middle of the diagrams below.

• The **GPIO.BCM** option means that you are referring to the pins by the "Broadcom SOC channel" number, these are the numbers after "GPIO" in the green rectangles around the outside of the below diagrams:

• Unfortunately the **BCM** numbers changed between versions of the Pi1 Model B.
– The Model B+ uses the same numbering as the Model B r2.0, and adds new pins (board numbers 27-40).The Raspberry Pi Zero, Pi 2B and Pi 3B use the same numbering as the B+.



Resister:
You must ALWAYS use resistors to connect LEDs up to the GPIO pins of the Raspberry Pi. The Raspberry Pi can only supply a small current (about 60mA). The LEDs will want to draw more, and if allowed to they will burn out the Raspberry Pi. Therefore putting the resistors in the circuit will ensure that only this small current will flow and the Pi will not be damaged. Resistors are a way of limiting the amount of electricity going through a circuit; specifically, they limit the amount of 'current' that is allowed to flow. The measure of resistance is called the Ohm (Ω), and the larger the resistance, the more it limits the current. The value of a resistor is marked with coloured bands along the length of the resistor body.

Jumper Wires:
Jumper Wires Jumper wires are used on breadboards to 'jump'from one connection to another. The ones you will be using in this circuit have different connectors on each end. The end with the 'pin' will go into the Breadboard.The end with the piece of plastic with a hole in it will go onto the Raspberry Pi's GPIO pins.

IR Sensor:

An infrared sensor is an electronic instrument which is used to sense certain characteristics of its surroundings by either emitting and/or detecting infrared radiation. Infrared sensors are also capable of measuring the heat being emitted by an object and detecting motion. Infrared waves are not visible to the human eye. In the electromagnetic spectrum, infrared radiation can be found between the visible and microwave regions. The infrared waves typically have wavelengths between 0.75 and 1000μm.

Safety precautions:

1. Raspberry-Pi provides 3.3V and 5V VCC pins
2. Raspberry-Pi operates on 3.3V.
3. Various sensors and actuators operate on different voltages.
4. Read datasheet of a given sensor or an actuator and then use appropriate VCC pin to
5. connect a sensor or an actuator.
6. Ensure that signal voltage coming to the Raspberry-Pi from any sensor or actuator does not
7. exceed 3.3V.
8. If signal/data coming to Raspberry-Pi is greater than 3.3V then use voltage level shifter
9. module to decrease the incoming voltage.
10. The Raspberry-Pi is a costly device, hence you should show the circuit connections to your
11. instructor before starting your experiment.

Steps for assembling circuit:

1. Connect the VCC pin of Proximity sensor to 3.3 V (pin) of Raspberry Pi module
2. Connect the GND pin of Proximity sensor to GND pin of Raspberry Pi module
3. Connect the DATA pin of Proximity sensor to pin '15' of Raspberry Pi module
4. Connect the D0 pin of LED bar to pin '16' of Raspberry Pi module
5. Connect the GND pin of LED bar to GND pin of Raspberry Pi module

Algorithm:

1. Import GPIO and Time library
2. Set mode i.e. GPIO.BOARD
3. Set GPIO pin '15' as Input
4. Set GPIO pin '16' as Input
5. Read input from GPIO pin '15'
6. Store the input value in the variable 'i'
7. If (i==1) then print the message as "Object is detected" and make the LED ON
8. If (i==0) then print the message as "No object detected" and make the LED OFF

Conclusion:

Hence, we have studied and implemented IOT System for to detect obstacle and notify user using LEDs.

Assignment No:	B-02
Title:	Understanding the connectivity of Raspberry-Pi /Beagle board circuit with temperature sensor. Write an application to read the environment temperature. If temperature crosses a threshold value, generate alerts using LEDs.
Date:	
Signature of Faculty:	

Assignment No: B-02

Title: Understanding the connectivity of Raspberry-Pi /Beagle board circuit with temperature sensor. Write an application to read the environment temperature. If temperature crosses a threshold value, generate alerts using LEDs.

Objectives:

- To understand the concept of Temperature-Humidity sensor (DHT11)
- To interface Temperature-Humidity sensor with Raspberry Pi model
- To program the Raspberry Pi model to measure the real time Temperature and Humidity of the Environment

Software:

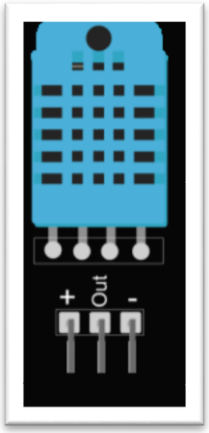
- Raspbian OS (IDLE)
-

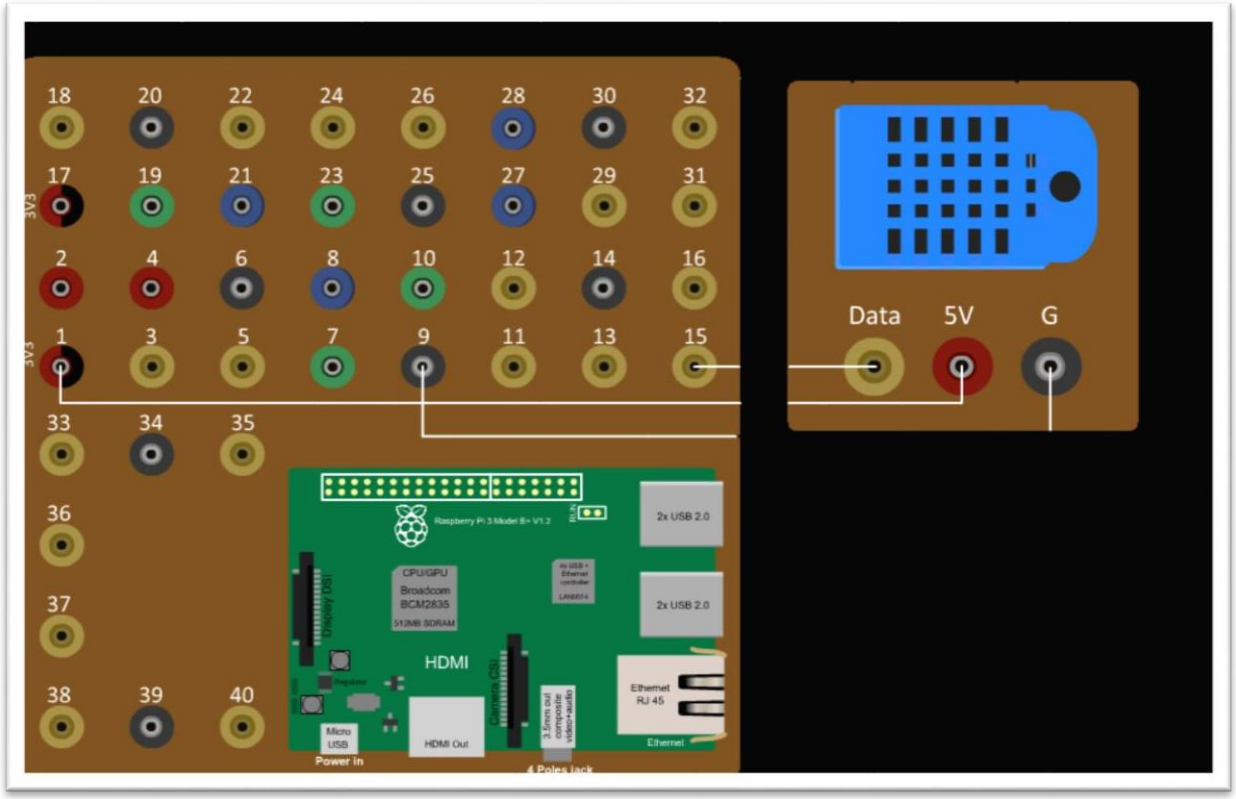
Hardware Modules:

- Raspberry Pi Board
- Temperature-Humidity sensor (DHT11) module
- Monitor

Theory:

Physical quantities like Humidity, temperature, pressure etc. are monitored to get information about the environmental conditions. Temperature is basically amount of heat present in environment. Humidity is the presence of water vapors in air. The Temperature & amount of water vapor in air can affect human comfort as well as many manufacturing processes in industries. The presence of water vapour also influences various physical, chemical, and biological processes. In our module we are using "DHT11 Temperature and Humidity Sensor". The features of this sensor are, calibrated digital signal output, and high reliability and excellent long-term stability. This sensor has a resistive-type humidity measurement component in which resistivity of semiconductor material changes as per humidity in environment changes. This sensor also includes NTC temperature measurement component which detects the change in temperature.





Temperature Sensor – DHT11:

The DHT11 temperature and humidity sensor is a nice little module that provides digital temperature and humidity readings. It's really easy to set up, and only requires one wire for the data signal.
These sensors are frequently used in remote weather stations, soil monitors, and home environment control systems. The programming is simple too, and many libraries and example code in both Python and C already exist. The DHT11 contains a surface mounted NTC thermistor and a resistive humidity sensor. An IC on the back of the module converts the resistance measurements from the thermistor and humidity sensor into digital outputs of degrees Celsius and Relative Humidity.

Fact sheet – DHT11:

- Ultra low cost
- 3 to 5V power and I/O
- 2.5mA max current use during conversion (while requesting data)
- Good for 20–80% humidity readings with 5% accuracy
- Good for 0–50°C temperature readings ±2°C accuracy
- No more than 1 Hz sampling rate (once every second)
- Body size 15.5mm x 12mm x 5.5mm
- 4 pins with 0.1" spacing

Safety precautions:

- Raspberry-Pi provides 3.3V and 5V VCC pins
- Raspberry-Pi operates on 3.3V.
- Various sensors and actuators operate on different voltages.
- Read datasheet of a given sensor or an actuator and then use appropriate VCC pin to
- connect a sensor or an actuator.
- Ensure that signal voltage coming to the Raspberry-Pi from any sensor or actuator does not exceed 3.3V.
- If signal/data coming to Raspberry-Pi is greater than 3.3V then use voltage level shifter
- module to decrease the incoming voltage.
- The Raspberry-Pi is a costly device, hence you should show the circuit connections to your
- instructor before starting your experiment.

Steps for assembling circuit:

1. Connect the VCC pin of Temperature & Humidity sensor (DHT11) to VCC pin of Raspberry Pi module
2. Connect the DATA pin of Temperature & Humidity sensor (DHT11) to GPIO pin 15 of
3. Raspberry Pi module
4. Connect the GND pin of Temperature & Humidity sensor (DHT11) to GND pin of Raspberry Pi module

Procedure to install dht11 library:

1. Open LXTerminal of Raspberry Pi and enter the following commands.
2. sudo apt-get update
3. sudo apt-get install git-core
4. cd ~
5. git clone https://github.com/szazo/DHT11_Python.git
6. cd DHT11_Python
7. sudo python setup.py install

Algorithm:

1. Import GPIO, time and dht11 libraries
2. Set all the warnings as False
3. Set mode i.e. GPIO.BOARD
4. Read data using GPIO pin number 7 (dhtPin)
5. Write ‘while loop’ for displaying Temperature and Humidity values continuously
6. First Read the GPIO pin and Store the data in dhtValue Variable.
7. Print the temperature value.
8. Print the Humidity value.
9. Give delay of 1 second

Conclusion:

Hence, we have studied and implemented application to read the environment temperature. If temperature crosses a threshold value, generate alerts using LEDs