Dr. D.Y. Patil Institute of Engineering, Management and Research, Akurdi, Pune
Department of Computer Engineering

Department of Computer Engineering

LAB MANUAL

Laboratory Practice I

(Third Year Computer Engineering)

A.Y. 2022-23 Semester – I

Prepared By:
Mrs. Ketaki Bhoyar
Mrs. Swati Rajput
Mrs. Priyanka Patil
Mrs. Maheshwari M C

Institute Vision

To strive for excellence by providing quality technical education and facilitate research for the welfare of society

Institute Mission

1. To educate students with strong fundamentals by providing conducive environment
2. To inculcate research with creativity & innovation
3. To strengthen leadership, team work, professional & communication skills and ethical standards
4. To promote Industry Institute collaboration & prepare students for life long learning in context of technological change.

Department Vision

To produce quality computer professionals and fostering research aptitude for dispensing service to society

Department Mission

1. To promote growth of an individual by imparting comprehensive knowledge of tools and technologies.
2. To facilitate research and innovation by engaging faculty and students in research activities.
3. To enrich industry-institute interaction in order to provide a platform to know industry demands and motivation for self-employment.
4. To bring forth a conducive environment to enhance soft skills and professional skills to cater needs of society

Program Specific Outcomes

1. Professional Skills: The ability to comprehend, analyze and develop software and hardware systems and amp; applications through research, in varying domains.

2. Problem-Solving Skills: The ability to apply standard paradigms and strategies in software project development using open-ended programming environments to deliver a quality product.

3. Successful Career and Entrepreneurship: Adaptation of modern practical and systematic approaches in creating innovative solutions for a successful career, entrepreneurship, and a zest for higher studies.

Course Objectives:

- To learn system programming tools
- To learn modern operating system
- To learn various techniques, tools, applications in IoT and Embedded Systems /Human Computer Interface/Distributed Systems/ Software Project Management
- Understand, Execute and Monitor different project plan.
- Understand GOMS modelling techniques
- Understand the working and Raspberri pi

Course Outcomes:

Systems Programming and Operating System

- CO1: Implement language translators
- CO2: Use tools like LEX and YACC
- CO3: Implement internals and functionalities of Operating System

Internet of Things and Embedded Systems

- CO4: Design IoT and Embedded Systems based application
- CO5: Develop smart applications using IoT
- CO6: Develop IoT applications based on cloud environment

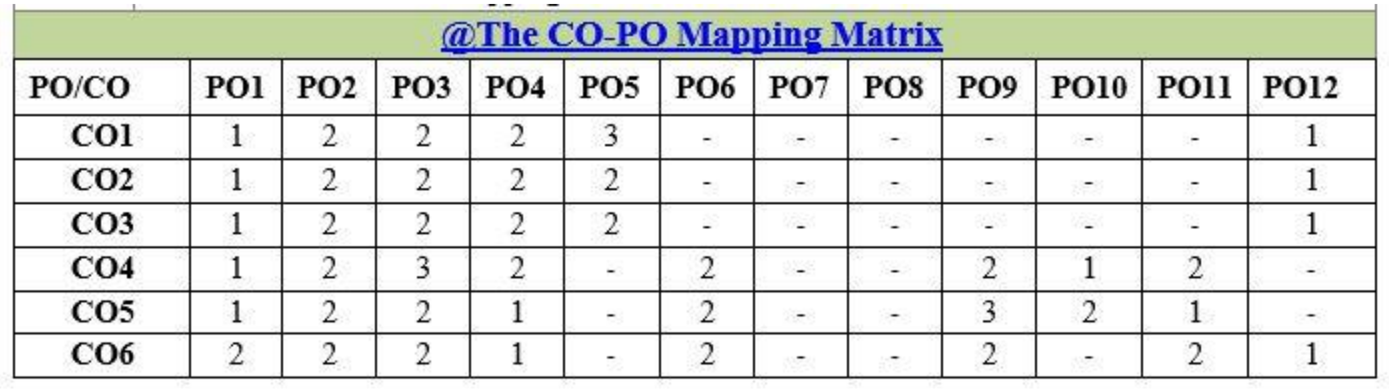
Human Computer Interface

- CO4:Implement the interactive designs for feasible data search and retrieval
- CO5:Analyze the scope of HCI in various paradigms like ubiquitous computing, virtual Reality and ,multi-media, World wide web related environments
- CO6:Analyze and identify user models, user support, socio-organizational issues, and stakeholder requirements of HCI systems

Software Project Management

- CO4:Apply Software Project Management tools

- CO5:Implement software project planning and scheduling
- CO6:Analyse staffing in software project



@The CO-PO Mapping Matrix

PO/CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	1	2	2	2	3	-	-	-	-	-	-	1
CO2	1	2	2	2	2	-	-	-	-	-	-	1
CO3	1	2	2	2	2	-	-	-	-	-	-	1
CO4	1	2	3	2	-	2	-	-	2	1	2	-
CO5	1	2	2	1	-	2	-	-	3	2	1	-
CO6	2	2	2	1	-	2	-	-	2	-	2	1

Guidelines for Student's Laboratory Journal

The laboratory assignments are to be submitted by student in the form of journal. Journal consists of Certificate, table of contents, and handwritten write-up of each assignment (Title, Date of Completion, Objectives, Problem Statement, Software and Hardware requirements, Assessment grade/marks and assessor's sign, Theory- Concept in brief, algorithm, flowchart, test cases, Test Data Set(if applicable), mathematical model (if applicable), conclusion/analysis. Program codes with sample output of all performed assignments are to be submitted as softcopy. As a conscious effort and little contribution towards Green IT and environment awareness, attaching printed papers as part of write-ups and programlisting to journal must be avoided. Use of DVD containing students programs maintained by Laboratory In-charge is highly encouraged. For reference one or two journals may be maintained with program prints in the Laboratory.

The instructor is expected to frame the assignments by understanding the prerequisites, technological aspects, utility and recent trends related to the topic. The assignment framing policy need to address the average students and inclusive of an element to attract and promote the intelligent students. Use of open source software is encouraged. Based on the concepts learned. Instructor may also set one assignment or mini-project that is suitable to respective branch beyond the scope of syllabus

Guidelines for Laboratory /Term Work Assessment

Continuous assessment of laboratory work should be based on overall performance of Laboratory assignments by a student. Each Laboratory assignment assessment will assign grade/marks based on parameters, such as timely completion, performance, innovation, efficient codes, punctuality

Virtual Laboratory:

- http://cse18- iiith.vlabs.ac.in/Introduction.html?domain=Computer%20Scie nce
- http://vlabs.iitb.ac.in/vlabs-dev/labs/cglab/index.php

Table of Contents

ASSIGNMENT NO: 1 – Group A

Title of Assignment:

Design suitable Data structures and implement Pass-I of a two-pass assembler for pseudo-machine.

Problem Statement:

Implement one pass-I of TWO Pass assembler with hypothetical Instruction set using Java language. Instruction set should include all types of assembly language statements such as Imperative, Declarative and Assembler Directive. While designing stress should be given on

a) How efficiently Mnemonic opcode could be implemented so as to enable faster retrieval on op-code.

b) Implementation of symbol table for faster retrieval.

Objectives: learn how pass-1 and pass-2 data structure works in assembly programs input.

Theory: Assembler:

1. An assembler is a program that accepts as input an assembly language program and produces as output its machine language equivalent i.e. It produces bit configuration of each of the mnemonic in the assembly language as shown. This machine language information from assembler is given to loader for further processing.
2. Note that in std. Cases Assembler not only produce this bit configuration; but also produces information useful for loader like- The externally defined symbols etc. are noted and these symbols are passed on to the loader for further resolution of their addresses.
3. The main reason for existence of assembler was to shift the burdens of calculating specific addresses from programmer to computer.

Tasks performed by the pass2 assembler

Pass 1 :

1. Separate symbol, Mnemonic and Operand fields
2. Build the symbol table
3. Perform IC processing
4. Construct intermediate code

Pass 2:

1. Synthesis of target program
2. Evaluate fields and generate code
3. Process pseudo opcodes

Pass 1 of assembler:

1. Pass1 uses following data structures:

   a. Symbol table (st).

   b. Literal table(lt)

   c. Machine opcode table(mot)

   d. Location counter

   e. Copy of source program

2. In some designs, assembler uses following table

   (i)Base Table BT:

   (a) To store the contents of base register

   (b) This table is used when we encounter the USING and BALR directives

   (ii)Pool tabl e(POOL TAB):

   a) Awareness of different literal pools is maintained using auxiliary table POOL TAB

   b) This table contains literal number of the starting literal of each pool

   c) At any stage the current literal pool is last pool in the literal table

   d) In encountering an LTORG statement literals in the current pool are allocated addresses starting with current value in LC and LC is appropriately incremented

3. Now the principle activities of pass 1 are:

   (i) Find length of machine instruction from MOT

   (ii) Keep track of location counter

   (iii) Remember values of symbols until pass II

   (iv) Process some pseudo-op-code

   (v)Remember literals

Algorithm for Pass I :

```
a. oc_cntr := 0; (default value)
        pooltab_ptr :=1; POOLTAB[1]:=1;
        littab_ptr:=1;
b. While next statement is not an END statement
        a) If label is present then
        {
        this_label:= symbol in label field;
        Enter(this_label, loc_cntr) in SYMTAB.
        }
        b) If an LTORG statement then
        {
                i Process literals
                LITTAB[POOLTAB[pooltab_ptr]…LITTAB[lit_tab_ptr-1] to
                allocate memory and put the address in the address field.
                Update location counter accordingly.
                ii pooltab_ptr := pooltab_ptr +1;
                iii POOLTAB[pooltab_ptr]:=littab_ptr;
        }
        c) If START or ORIGIN statement then
        {
        loc_cntr := value specified in the operand field;
        }
        d) If an EQU statement then
        {
                i. this_addr := value of <address_spec>;
                ii. Correct the symbtab entry for this_label to
                (this_label,this_addr).
        }
        e) If a declaration statement then
        {
                i. code:= code of the declaration statement;
```

```
            ii. size := size of memory are required by DC/DS
            iii. loc_cntr := loc_cntr + size;
            iv. Generate IC ‘(DL, code)…’
        }
        f) If an imperative statement then
            i. code:= machine opcode from OPTAB;
            ii. loc_cntr := loc_cntr + instruction length from OPTAB;
            iii. If operand is a literal then
            {
            this_literal := literal in operand field;
            LITTAB[littab_ptr]:= this_literal;
            littab_ptr= littab_ptr +1;
            }
            else (i.e. operand is a symbol)
            {
            this_entry := SYMTAB entry number of operand
            Generate IC ‘(IS,code)(S,this_entry)’;
            }
    c.
        a) Perform step 2(b).
        b) Generate IC’(AD, 02)’.
        c) Go to Pass II.
```

SAMPLE PROGRAM Input

```
START 200
READ A
READ B
MOVER AREG, ='5'
MOVER AREG, A
ADD AREG, B
SUB AREG, ='6'
```

```
MOVEM AREG, C
PRINT C
LTORG
MOVER AREG, ='15'
MOVER AREG, A
ADD AREG, B
SUB AREG, ='16'
DIV AREG, ='26'
MOVEM AREG, C
A DS 1
B DS 1
C DS 1
STOP
END Symbol Table
------------------------------------------------------------
Symb Addr Decl Used Val Len
------------------------------------------------------------
A 216 1 1 0 1
B 217 1 1 0 1
C 218 1 1 0 1
------------------------------------------------------------
Total Errors: 0
Total Warnings: 0
Literal Table
------------------------------------
Lit# Lit Addr
------------------------------------
00 ='5' 208
01 ='6' 209
02 ='15' 220
03 ='16' 221
```

```
04 ='26' 222
------------------------------------
Pool Table
--------------------
Pool# Pool Base
--------------------
00 0
01 2
------------------------
```

INTERMEDIATE CODE

```
Intermediate Code
(AD, 00) (C, 200)
(IS, 09) (S, 00)
(IS, 09) (S, 01)
(IS, 04) (0) (L, 00)
(IS, 04) (0) (S, 00)
(IS, 01) (0) (S, 01)
(IS, 02) (0) (L, 01)
(IS, 05) (0) (S, 02)
(IS, 10) (S, 02)
(AD, 04)
(IS, 04) (0) (L, 02)
(IS, 04) (0) (S, 00)
(IS, 01) (0) (S, 01)
(IS, 02) (0) (L, 03)
(IS, 08) (0) (L, 04)
(IS, 05) (0) (S, 02)
(DL, 01) (C, 01)
(DL, 01) (C, 01)
(DL, 01) (C, 01)
```

(IS, 00)

(AD, 01)

Conclusion:

ASSIGNMENT NO: 2 – Group A

Title of Assignment:

Design suitable Data structures and implement Pass-II of a two-pass assembler for pseudo-machine.

Problem Statement:

Implement pass-II of TWO Pass assembler with hypothetical Instruction set using Java language. Instruction set should include all types of assembly language statements such as Imperative, Declarative and Assembler Directive. While designing stress should be given on

a) How efficiently Mnemonic opcode table could be implemented so as to enable faster retrieval on op-code.

b) Implementation of symbol table, pool tables for faster retrieval.

Objectives: learn how pass-1 and pass-2 data structure works in assembly programs input.

Theory: Assembler:

ALGORITHM:

1. Open and read the first line from the intermediate file.

2. If the first line contains the opcode “START”, then write the label, opcode and operand field values of the corresponding statement directly to the final output file.

3. Do the following steps, until an “END” statement is reached.

    3.1 Start writing the location counter, opcode and operand fields of the corresponding statement to the output file, along with the object code.

    3.2 If there is no symbol/label in the operand field, then the operand address is assigned as zero and it is assembled with the object code of the instruction.

    3.3 If the opcode is BYTE, WORD, RESB etc convert the constants to the object code.

4. Close the files and exit.

Forward Referencing: - In forward referencing, variable or label is referenced before it is declared. Different problems can be solved using **One Pass** or **Two Pass** forward referencing. In One Pass forward referencing source program is translated instruction by instruction. Assembler leave address space for label when it is referenced and when assembler found the declaration of label, it uses *back patching*.

Two Pass forward referencing consist of two passes.

During first pass ***symbol table***, ***op-code table*** and ***label table*** are maintained.

- In op-code table instruction size and address is stored.
- Label and label's address is stored in label table. When label is encountered, its name is stored in label table when label declaration is found then its location is also stored in label table.

During 2nd Pass, translation from source language to machine language takes place. Instruction addresses and label addresses are used from symbol table instead of their names.

Compiler does not know where program will be executed in the memory so compiler generated logical addresses instead of absolute address. Loader also uses the ***Relocation Constant*** to solve the problem of relocation. **External Referencing** problem is resolved by the linker during compilation. Linker connects the object program to the code for standard library functions.

The assembler implements the back patching technique as follows:

- It builds a table of incomplete instructions (TII) to record information about instructions whose operand fields left blank.
- Each entry in this table contains a pair of the form (instruction address, symbol) to indicate that the address of symbol should put in the operand field of the instruction with the address instruction address.
- By the time the END statement processed, the symbol table would contain the addresses of all symbols defined in the source program and TII would contain information describing all forward references.
- The assembler can now process each entry in TII to complete the concerned instruction.
- Alternatively, entries in TII can process on the fly during normal processing.
- In this approach, all forward references to a symbol i would be processed when the statement that defines symbol encountered. also …
- The instruction corresponding to the statement
  MOVER BREG, ONE contains a forward reference to ONE.
- Hence the assembler leaves the second operand field blank in the instruction that assembled to reside in location 101 of memory and makes an entry (101, ONE) in the table of incomplete instructions (TII).
- While processing the statement
  ONE DC ‘1’address of ONE, which 115, entered in the symbol table.

- After the END statement processed, the entry (101, ONE) would be processed by obtaining the address of ONE from the symbol table and inserting it in the second operand field of the instruction with assembled address 101.

Pass II Algorithm

It has been assumed that the target code is to be assembled in the named code_area.

```
1. code_area_address := address of code_area;
        Pooltab_ptr :=1;
        Loc_cntr:=0;
2. While next statement is not an END statement
        (a) Clear machine_code_buffer;
        (b) If an LTORG statement
                (i) Process literals in LITTAB[POOLTAB[pooltab_ptr]]…
                LITTAB[POOLTAB[pooltab_ptr+1]]-1 similar to processing of constants in a DC statement i.e.
                assemble the literals in machine_code_buffer.
                (ii) size := size of memory area required for literals;
                (iii) pooltab_ptr:= pooltab_ptr +1;
        (c) If a START or ORIGIN statement then
                (i) loc_cntr := value specified in operand field;
                (ii) size:=0;
        (d) If a declaration statement
                (i) If a DC statement then
                Assemble the constant in machine_code_buffer.
                (ii) size: = size of memory area required by DC/DS;
        (e) If an imperative statement
                (i) Get operand address from SYMTAB or LITTAB.
                (ii) Assemble instruction in machine_code_buffer.
                (iii) size: = size of instruction;
        (f) if size not equal to 0 then
                (i) Move contents of Machine_code_buffer to the address
                code_area_address + loc_cntr;
                (ii) loc_cntr := loc_cntr + size;
3. (Processing of END statement)
        (a) Perform steps 2(b) and 2(f).
        (b) Write code_area into output file.
```

Example:

SAMPLE PROGRAM Input

```
        START 200
        READ A
        READ B
        MOVER AREG, ='5'
        MOVER AREG, A
        ADD AREG, B
        SUB AREG, ='6'
        MOVEM AREG, C
        PRINT C
        LTORG
        MOVER AREG, ='15'
        MOVER AREG, A
        ADD AREG, B
        SUB AREG, ='16'
        DIV AREG, ='26'
        MOVEM AREG, C
        A DS 1
        B DS 1
        C DS 1
        STOP
        END Symbol Table
------------------------------------------------------------
Symb   Addr Decl      Used    Val      Len
------------------------------------------------------------
```

```
A       216             1        10      1
B       217             1        1 0     1
C       218             1        1 0      1
----------------------------------------------------------
Total Errors: 0
Total Warnings: 0
Literal Table
----------------------------------
Lit# Lit Addr
----------------------------------
00 ='5' 208
01 ='6' 209
02 ='15' 220
03 ='16' 221
04 ='26' 222
----------------------------------
Pool Table
-------------------
Pool# Pool Base
-------------------
00 0
01 2
-------------------
```

INTERMEDIATE CODE

```
Intermediate Code
(AD, 00) (C, 200)
(IS, 09) (S, 00)
(IS, 09) (S, 01)
(IS, 04) (0) (L, 00)
(IS, 04) (0) (S, 00)
(IS, 01) (0) (S, 01)
```

```
(IS, 02) (0) (L, 01)

(IS, 05) (0) (S, 02)

(IS, 10) (S, 02)

(AD, 04)

(IS, 04) (0) (L, 02)

(IS, 04) (0) (S, 00)

(IS, 01) (0) (S, 01)

(IS, 02) (0) (L, 03)

(IS, 08) (0) (L, 04)

(IS, 05) (0) (S, 02)

(DL, 01) (C, 01)

(DL, 01) (C, 01)

(DL, 01) (C, 01)

(IS, 00)

(AD, 01)
```

PASS II OUTPUT

Target Code

```
200) + 09 0 216

201) + 09 0 217

202) + 04 0 208

203) + 04 0 216

204) + 01 0 217

205) + 02 0 209

206) + 05 0 218

207) + 10 0 218

208) + 00 0 005

209) + 00 0 006

210) + 04 0 220

211) + 04 0 216

212) + 01 0 217

213) + 02 0 221
```

```
214) + 08 0 222
215) + 05 0 218
216)
217)
218)
219) + 00 0 000
220) + 00 0 015
221) + 00 0 016
222) + 00 0 026
```

Conclusion:

ASSIGNMENT NO: 3 – Group A

Title of Assignment:

Design suitable data structures and implement pass-I of a two-pass macro-processor.

Problem Statement: Write a program in C for a pass-II of two pass macro processor for Implementation of Macro Processor. Following cases to be considered

a) Macro without any parameters

b) Macro with Positional Parameters

c) Macro with Key word parameters

d) Macro with positional and keyword parameters.

(Conditional expansion, nested macro implementation not expected)

Objectives: learn how pass-1 and pass-2 Macro processor works.

Theory: Macro Processor: A macro processor is a program that copies a stream of text from one place to another, making a systematic set of replacements as it does so. Macro processors are often embedded in other programs, such as assemblers and compilers. Sometimes they are standalone programs that can be used to process any kind of text. “A macro processor is a program that reads a file (or files) and scans them for certain keywords. When a keyword is found, it is replaced by some text. The keyword/text combination is called a macro.”

Two new assembler directives are used in macro definition are MACRO and MEND.

MACRO: identify the beginning of a macro definition

MEND: identify the end of a macro definition

Prototype for the macro Each parameter begins with ‘&’

```
name MACRO parameters
:
body
:
```

```
MEND
```

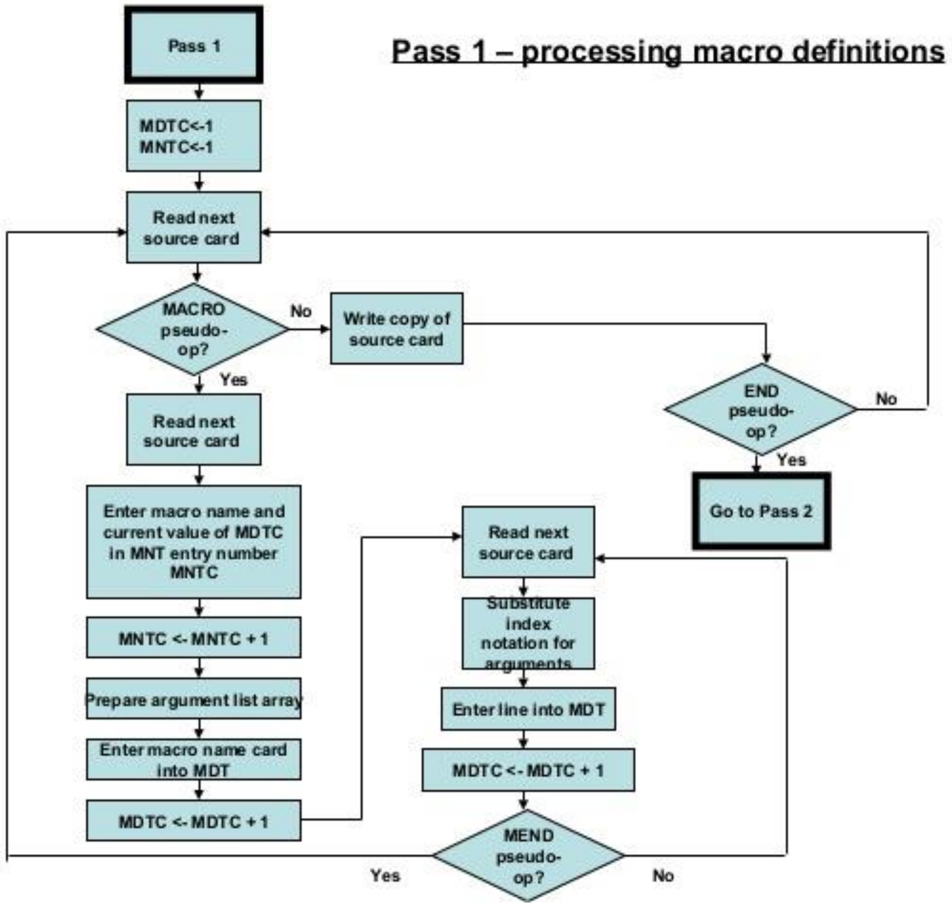
Body: the statements that will be generated as the expansion of the macro.

Macro facility:

An assembly language macro facility is to extend the set of operations provided in an assembly language. In order that programmers can repeat identical parts of their program macro facility can be used. This permits the programmer to define an abbreviation for a part of program & use this abbreviation in the program. This abbreviation is treated as macro definition & saved by the macro processor. For all occurrences the abbreviation i.e. macro call, macro processor, substitutes the definition.

Data Structure required for macro definition processing:

- Macro Name Table (MNT):- Fields Name of macro, #PP(Number of positional parameters),#KP(Number of keyword parameters), MDTP(Macro Definition Table Pointer), KPDTP (Keywords Parameters Default Table Position).
- Parameter Name Table (PNTAB) :- Feilds parameter name
- Keywords Parameters Default Table (KPDTAB) :- Feilds-parameter name, default value.
- Macro Definition Table (MDT) :- Model statements are stored in inetrmediate code form as : opcode and operands.


Pass 1 – processing macro definitions
Pass 1
MDTC<-1
MNTC<-1
Read next source card
MACRO pseudo-op?
No
Write copy of source card
Yes
Read next source card
END pseudo-op?
No
Yes
Go to Pass 2
Enter macro name and current value of MDTC in MNT entry number MNTC
Read next source card
MNTC <- MNTC + 1
Substitute index notation for arguments
Prepare argument list array
Enter line into MDT
Enter macro name card into MDT
MDTC <- MDTC + 1
MDTC <- MDTC + 1
MEND pseudo-op?
Yes
No

```
------PARAMETER NAME TABLE--------
#       PName
----------------------------------
0       X       N1
1       Y       N2
2       REG     CREG
----------------------------------


------PARAMETER NAME TABLE--------
#       PName
----------------------------------
0       A       N1
1       B       N2
```

```
2      REG     BREG
----------------------------------


----------MACRO NAME TABLE---------------------
#      MName   #MDTP
-----------------------------------------------
0      INCR    0
1      DECR    5
------------------------------------------------


-----------MACRO DEFINITION TABLE--------------
Opcode  Rest
-----------------------------------------------
0      INCR     &X, &Y,&REG=AREG
1      MOVER   #2,#0
2      ADD     #2,#1
3      MOVEM   #2,#0
4      MEND
5      DECR     &A,&B,&REG=BREG
6      MOVER   #2,#0
7      SUB     #2,#1
8      MOVEM   #2,#0
9      MEND
-----------------------------------------------
```

Contents of test.asm

```
--------------------
MACRO
INCR &X, &Y,&REG=AREG
MOVER &REG,&X
ADD &REG,&Y
```

```
MOVEM &REG,&X
MEND
MACRO
DECR &A,&B,&REG=BREG
MOVER &REG,&A
SUB &REG,&B
MOVEM &REG,&A
MEND
START 100
READ N1
READ N2
INCR N1,N2,REG=CREG
DECR N1,N2
STOP
N1 DS 1
N2 DS 1
END
```

Contents of test.ini

```
--------------------
START 100
READ N1
READ N2
+MOVER  CREG,N1
+ADD  CREG,N2
+MOVEM  CREG,N1
+MOVER  BREG,N1
+SUB  BREG,N2
+MOVEM  BREG,N1
STOP
N1 DS 1
```

N2 DS 1

END

Conclusion: Thus we have implemented pass-I of two pass Macro processor.

ASSIGNMENT NO: 4 – Group A

Title of Assignment:

Write a Java program for pass-II of a two-pass macro-processor.

Problem Statement:

Implement pass-II of TWO Pass assembler with hypothetical Instruction set using Java language. Instruction set should include all types of assembly language statements such as Imperative, Declarative and Assembler Directive. While designing stress should be given on

a) How efficiently Mnemonic opcode table could be implemented so as to enable faster retrieval on op-code.

b) Implementation of symbol table, pool tables for faster retrieval.

Objectives: learn how pass-1 and pass-2 data structure works in assembly programs input.

Theory: Macros are compiled programs that you can invoke (or **call**) in a submitted SAS program or from a SAS command prompt. Like macro variables, you generally use macros to generate text. However, macros provide additional capabilities:

- Macros can contain programming statements that enable you to control how and when text is generated.
- Macros can accept parameters. This allows you to write generic macros that can serve a number of uses.

To compile a macro, you must submit a macro definition. The general form of a macro definition is

%MEND <*macro_name*>;

%MACRO *macro-name*;
<*macro_text*>

where **macro_name** is a unique SAS name that identifies the macro and **macro_text** is any combination of macro statements, macro calls, text expressions, or constant text.

When you submit a macro definition, the macro processor compiles the definition and produces a member in the session catalog. The member consists of compiled macro program statements and text. The distinction between compiled items and noncompiled (text) items is important for macro execution. Examples of text items include:

- macro variable references
- nested macro calls
- macro functions, except %STR and %NRSTR
- arithmetic and logical macro expressions
- text to be written by %PUT statements
- field definitions in %WINDOW statements
- model text for SAS statements and SAS Display Manager System commands.

When you want to call the macro, you use the form

Macro expansion:- A macro name is an abbreviation, which stands for some related lines of code. Macros are useful for the following purposes:

· To simplify and reduce the amount of repetitive coding

· To reduce errors caused by repetitive coding

· To make an assembly program more readable.

A macro consists of name, set of formal parameters and body of code. The use of macro name with set of actual parameters is replaced by some code generated by its body. This is called macro expansion.

Macros allow a programmer to define pseudo operations, typically operations that are generally desirable, are not implemented as part of the processor instruction, and can be implemented as a sequence of instructions. Each use of a macro generates new program instructions, the macro has the effect of automating writing of the program.

Macros can be defined used in many programming languages, like C, C++ etc. Example macro in C programming. Macros are commonly used in C to define small snippets of code. If the macro has parameters, they are substituted into the macro body during expansion; thus, a C macro can mimic a C function. The usual reason for doing this is to avoid the overhead of a function call in simple cases, where the code is lightweight enough that function call overhead has a significant impact on performance.

For instance, #define max (a, b) a>b? A: b

Defines the macro max, taking two arguments a and b. This macro may be called like any C function, using identical syntax. Therefore, after preprocessing

z = max(x, y); Becomes z = x>y? X:y;

While this use of macros is very important for C, for instance to define type-safe generic data-types or debugging tools, it is also slow, rather inefficient, and may lead to a number of pitfalls.

C macros are capable of mimicking functions, creating new syntax within some limitations, as well as expanding into arbitrary text (although the C compiler will require that text to be valid C source code, or else comments), but they have some limitations as a programming construct. Macros which mimic functions, for instance, can be called like real functions, but a macro cannot be passed to another function using a function pointer, since the macro itself has no address.

In programming languages, such as C or assembly language, a name that defines a set of commands that are substituted for the macro name wherever the name appears in a program (a process called macro expansion) when the program is compiled or assembled. Macros are similar to functions in that they can take arguments and in that they are calls to lengthier sets of instructions. Unlike functions, macros are replaced by the actual commands they represent when the program is prepared for execution. function instructions are copied into a program only once.

Macro Expansion:- A macro call leads to macro expansion. During macro expansion, the macro statement is replaced by sequence of assembly statements. In the above program a macro call is shown in the middle of the figure. i.e. INITZ. Which is called during program execution? Every macro begins with MACRO keyword at the beginning and ends with the ENDM (end macro).whenever a macro is called the entire is code is substituted in the program where it is called. So the resultant of the macro code is shown on the right most side of the figure. Macro calling in high level programming languages

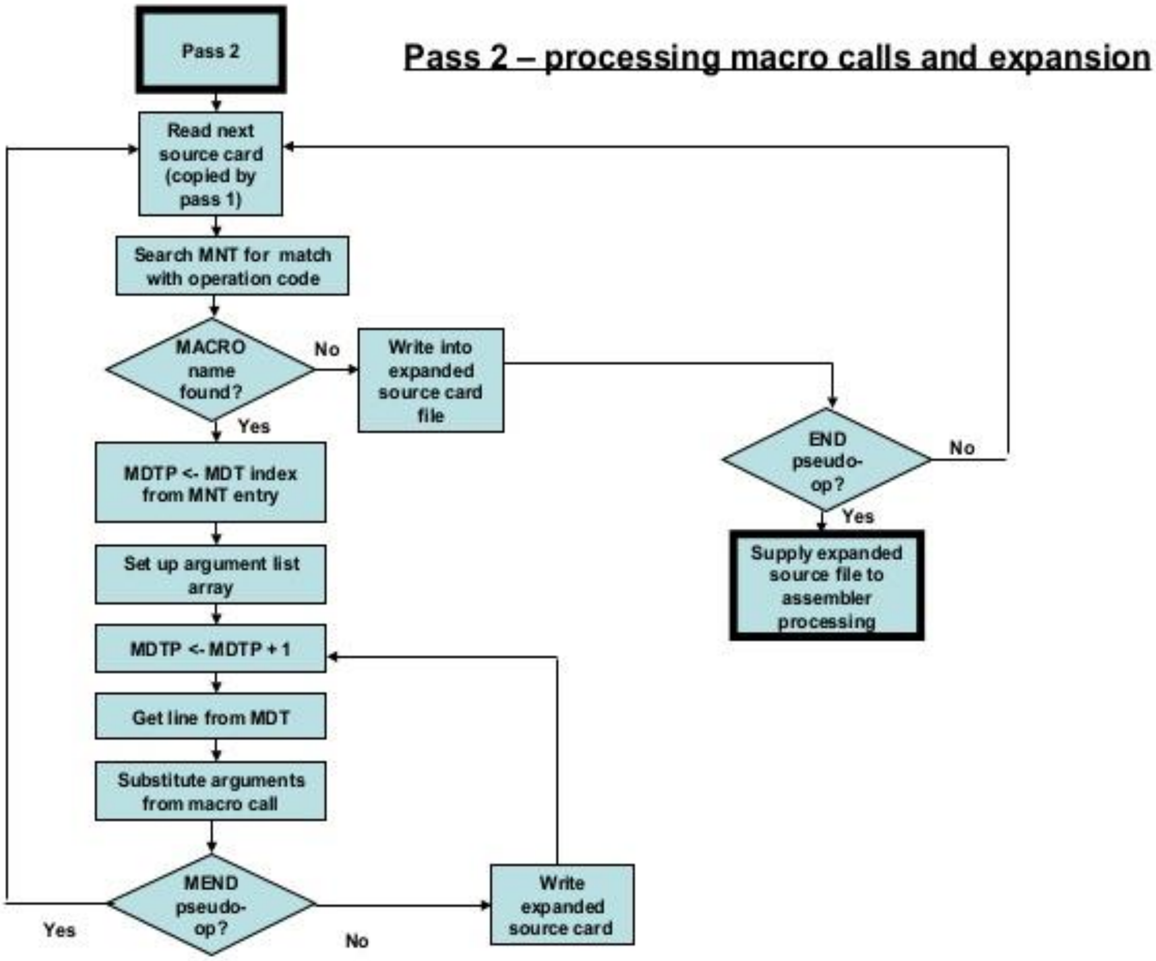
```
#define max(a,b) a>b?a:b
        Main ()
        {
```

```
        int x , y;
        x=4; y=6;
        z = max(x, y);
        }
```

The above program was written using C programming statements. Defines the macro max, taking two arguments a and b. This macro may be called like any C function, using identical syntax. Therefore, after preprocessing Becomes z = x>y ? x: y;

After macro expansion, the whole code would appear like this.

```
#define max(a,b) a>b?a:b
        main()
        {
         int x , y;
        x=4; y=6;
        z = x>y?x:y;
        }
```


Pass 2
Pass 2 – processing macro calls and expansion
Read next source card (copied by pass 1)
Search MNT for match with operation code
MACRO name found?
No
Yes
Write into expanded source card file
END pseudo-op?
No
Yes
Supply expanded source file to assembler processing
MDTP <- MDT index from MNT entry
Set up argument list array
MDTP <- MDTP + 1
Get line from MDT
Substitute arguments from macro call
MEND pseudo-op?
Yes
No
Write expanded source card

Conclusion:- Thus we have implemented pass-II of two pass Macro processor

ASSIGNMENT NO: 5 – Group A

Title of Assignment: Write a program to create Dynamic Link Library for any mathematical operation and write an application program to test it. (Java Native Interface / Use VB or VC++).

Problem Statement: Write a program to create Dynamic Link Library for Arithmetic Operation in VB.net

Objectives:

- To understand Dynamic Link Libraries Concepts
- To implement dynamic link library concepts
- To study about Visual Basic

Theory:

Dynamic Link Library: A dynamic link library (DLL) is a collection of small programs that can be loaded when needed by larger programs and used at the same time. The small program lets the larger program communicate with a specific device, such as a printer or scanner. It is often packaged as a DLL program, which is usually referred to as a DLL file. DLL files that support specific device operation are known as device drivers. A DLL file is often given a ".dll" file name suffix. DLL files are dynamically linked with the program that uses them during program execution rather than being compiled into the main program.

The advantage of DLL files is space is saved in random access memory (RAM) because the files don't get loaded into RAM together with the main program. When a DLL file is needed, it is loaded and run. For example, as long as a user is editing a document in Microsoft Word, the printer DLL file does not need to be loaded into RAM. If the user decides to print the document, the Word application causes the printer DLL file to be loaded and run.

A program is separated into modules when using a DLL. With modularized components, a program can be sold by module, have faster load times and be updated without altering other parts of the program. DLLs help operating systems and programs run faster, use memory efficiently and take up less disk space

Feature of DLL

- DLLs are essentially the same as EXEs, the choice of which to produce as part of the linking process is for clarity, since it is possible to export functions and data from either.

- It is not possible to directly execute a DLL, since it requires an EXE for the operating system to load it through an entry point, hence the existence of utilities like RUNDLL.EXE or RUNDLL32.EXE which provide the entry point and minimal framework for DLLs that contain enough functionality to execute without much support.

- DLLs provide a mechanism for shared code and data, allowing a developer of shared code/data to upgrade functionality without requiring applications to be re-linked or re-compiled. From the application development point of view Windows and OS/2 can be thought of as a collection of DLLs that are upgraded, allowing applications for one version of the OS to work in a later one, provided that the OS vendor has ensured that the interfaces and functionality are compatible.

- DLLs execute in the memory space of the calling process and with the same access permissions which means there is little overhead in their use but also that there is no protection for the calling EXE if the DLL has any sort of bug.

Executable file links to DLL:

An executable file links to (or loads) a DLL in one of two ways:

Implicit linking: - Implicit linking is sometimes referred to as static load or load-time dynamic linking. Explicit linking is sometimes referred to as dynamic load or run-time dynamic linking. With implicit linking, the executable using the DLL links to an import library (.lib file) provided by the maker of the DLL. The operating system loads the DLL when the executable using it is loaded. The client executable calls the DLL's exported functions just as if the functions were contained within the executable

Explicit linking:-With explicit linking, the executable using the DLL must make function calls to explicitly load and unload the DLL and to access the DLL's exported functions. The client executable must call the exported functions through a function pointer. An executable can use the same DLL with either linking method. Furthermore, these mechanisms are not mutually exclusive, as one executable can implicitly link to a DLL and another can attach to it explicitly.

Calling DLL function from Visual Basic Application:

For Visual Basic applications (or applications in other languages such as Pascal or Fortran) to call functions in a C/C++ DLL, the functions must be exported using the correct calling convention without any name decoration done by the compiler.

__stdcall creates the correct calling convention for the function (the called function cleans up the stack and parameters are passed from right to left) but decorates the function name differently. So, when **__declspec(dllexport)** is used on an exported function in a DLL, the decorated name is exported.

The __stdcall name decoration prefixes the symbol name with an underscore (_) and appends the symbol with an at sign (@) character followed by the number of bytes in the argument list (the required stack space). As a result, the function when declared as:

int __stdcall func (int a, double b)

is decorated as:

_func@12

The C calling convention (__cdecl) decorates the name as _func.

To get the decorated name, use /MAP. Use of **__declspec(dllexport)** does the following:

- If the function is exported with the C calling convention (**_cdecl**), it strips the leading underscore (_) when the name is exported.
- If the function being exported does not use the C calling convention (for example, __stdcall), it exports the decorated name.

Because there is no way to override where the stack cleanup occurs, you must use __stdcall. To undecorate names with __stdcall, you must specify them by using aliases in the EXPORTS section of the .def file. This is shown as follows for the following function declaration:

int __stdcall MyFunc (int a, double b);

void __stdcall InitCode (void);

In the .DEF file:

EXPORTS

MYFUNC=_MyFunc@12

INITCODE=_InitCode@0

For DLLs to be called by programs written in Visual Basic, the alias technique shown in this topic is needed in the .def file. If the alias is done in the Visual Basic program, use of aliasing in the .def file is not necessary. It can be done in the Visual Basic program by adding an alias clause to the Declare statement.

Conclusion:- Thus, I have studied visual programming and implemented dynamic link library application for arithmetic operation

Questions:

1. What Is Dll And What Are Their Usages And Advantages?

2. What Are The Sections in a DLL Executable/binary?

3. Where Should We Store DLL?

4. Who Loads and Links the DLL?

5. How Many Types Of Linking Are There?

6. What Is Implicit And Explicit Linking In Dynamic Loading?

ASSIGNMENT NO: 06 – Group B

Assignment Title: Implement a program to solve Classical Problems of Synchronization using Mutex and Semaphore

Problem Statement: Write a program to solve Classical Problems of Synchronization using Mutex and Semaphore

Objectives:

1. To understand concept of semaphore, critical section.
2. Learn implementation of Semaphore.

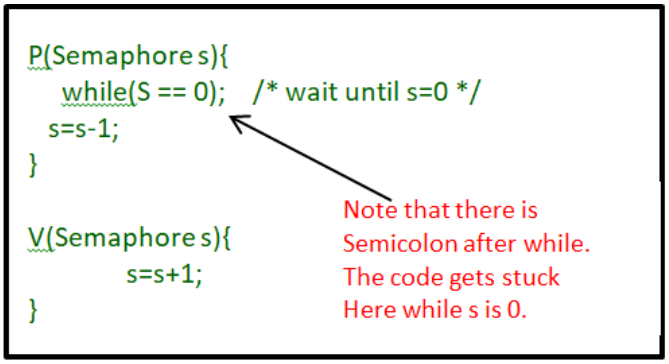
Theory:

- There are a number of processes that only read the data area (readers) and a number that only write to the data area (writers).
- The conditions that must be satisfied are
  - Any number of readers may read simultaneously read the file.
  - Only one write at a time may write to the file.
  - If a writer is writing to the file, no reader may read it.
- The data area could be a file, a block of main memory, or even a bank of processor registers.
- There is a data area shared among a number of processor registers

Semaphore: Semaphores are system variables used for synchronization of process .

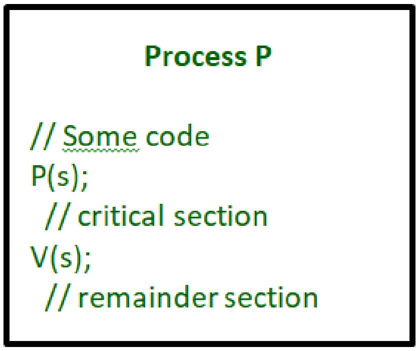
- **Binary Semaphore –**
  This is also known as mutex lock. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problems with multiple processes.

- **Counting Semaphore –**
  Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.



```
P(Semaphore s){
    while(S == 0);   /* wait until s=0 */
   s=s-1;
}

V(Semaphore s){
           s=s+1;
}
```

Note that there is Semicolon after while. The code gets stuck Here while s is 0.



Process P

```
// Some code
P(s);
  // critical section
V(s);
  // remainder section
```

- P operation is also called wait, sleep, or down operation, and V operation is also called signal, wake-up, or up operation.
- Both operations are atomic and semaphore(s) is always initialized to one. Here atomic means that variable on which read, modify and update happens at the same time/moment with no pre-emption i.e. in-between read, modify and update no other operation is performed that may change the variable.
- A critical section is surrounded by both operations to implement process synchronization. See the below image. The critical section of Process P is in between P and V operation.

Algorithm for Reader Writer:

1. **import java.util.concurrent.Semaphore;**
2. **Create a class RW**
3. **Declare semaphores – mutex and wrt**
4. **Declare integer variable readcount = 0**
5. **Create a nested class Reader implements Runnable**
    a. **Override run method (Reader Logic)**
        i. wait(mutex);
        ii. readcount := readcount +1;
        iii. if readcount = 1 then
        iv. wait(wrt);
        v. signal(mutex);
        vi. …
        vii. reading is performed
        viii. …

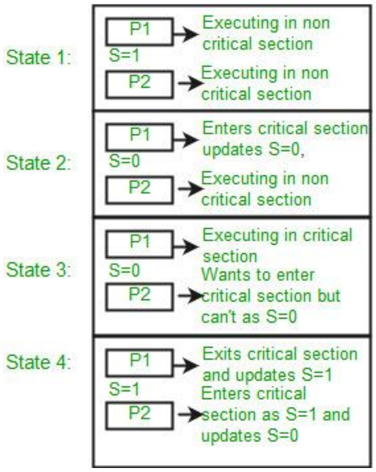
ix. wait(mutex);

x. readcount := readcount – 1;

xi. if readcount = 0 then signal(wrt);

xii. signal(mutex):

6. **Create a nested class Writer implements Runnable**
    a. **Override run method (Writer Logic)**
        i. wait(wrt);
        ii. …
        iii. writing is performed
        iv. …
        v. signal(wrt);
7. **Create a class main**
    a. **Create Threads for Reader and Writer**
    b. **Start these thread**


State 1:
P1
Executing in non critical section
S=1
P2
Executing in non critical section
State 2:
P1
Enters critical section updates S=0,
S=0
P2
Executing in non critical section
State 3:
P1
Executing in critical section
S=0
P2
Wants to enter critical section but can't as S=0
State 4:
P1
Exits critical section and updates S=1
S=1
P2
Enters critical section as S=1 and updates S=0

Limitations:

- One of the biggest limitations of semaphore is priority inversion.
- Deadlock, suppose a process is trying to wake up another process which is not in a sleep state. Therefore, a deadlock may block indefinitely.
- The operating system has to keep track of all calls to wait and to signal the semaphore.

Problem in this implementation of semaphore:

- The main problem with semaphores is that they require busy waiting,
- If a process is in the critical section, then other processes trying to enter critical section will be waiting until the critical section is not occupied by any process. Whenever any process waits then it continuously checks for semaphore value (look at this line while (s==0); in P operation) and waste CPU cycle.
- There is also a chance of "spinlock" as the processes keep on spins while waiting for the lock.

Conclusion: Thus we have learned basic concept of semaphore and mutex.

ASSIGNMENT NO: 07 – Group B

Assignment Title: implement following process scheduling algorithms: FCFS , SJF (Preemptive), Priority (Non-Preemptive) .

Problem Statement: Write a Java program (using OOP features) to implement following scheduling algorithms: FCFS, SJF (Preemptive), Priority (Non-Preemptive) .

Objectives:

1. To understand concept of scheduling.
2. To learn and use scheduling algorithms.

Theory:

CPU scheduling is the basis of multi programmed operating systems. By switching the CPU among processes, the operating system can make the computer more productive. The objective of multiprogramming is to have some process running at all times, in order to maximize CPU utilization. In a uniprocessor system, only one process may run at a time; any other processes must wait until the CPU is free and can be rescheduled.

The idea of multiprogramming is relatively simple. A process is executed until it must wait, typically for the completion of some I/O request. In a simple computer system, the CPU would then sit idle; all this waiting time is wasted. With multiprogramming, the time is to be used productively. Several processes are kept in memory at one time. When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process. This pattern continues.

Scheduling is a fundamental operating system function. Almost all computer resources are scheduled before use. The CPU is one of the primary computer resources. Thus, its scheduling is central to operating system design.

CPU-I/O Burst Cycle:

The success of CPU scheduling depends on the following observed property of processes: Process execution consists of a cycle of CPU execution and I/O wait. Processes alternate between these two states. Process execution begins with a CPU burst i.e. by an I/O burst, and then another CPU burst will end with a system request to terminate execution, rather than with another I/O burst.

CPU Scheduler

Whenever the CPU becomes idle, the operating system must select one of the processes in thread queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

The ready queue is not necessarily a first-in, first-out (FIFO) queue. A ready queue may be implemented as a FIFO queue, a priority queue, a tree, or simply an unordered linked list. All the processes in the ready queue are lined up waiting for a chance to run on the CPU. The records in the queue are generally process control blocks (PCBs) of the processes.

Preemptive Scheduling

CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state (for example, I/O request, or invocation of wait for the termination of one of the child processes)
2. When a process switches from the running state to the ready state (for example, when an interrupt occurs)
3. When a process switches from the waiting state to the ready state(for example, completion of I/O)
4. When a process terminates

In circumstances 1 and 4, there is no choice in terms of scheduling. A new process must be selected for execution. There is a choice in circumstances 2 and 3.

When the scheduling takes place only under circumstances 1 and 4, the scheduling is non-preemptive; otherwise, the scheduling scheme is preemptive. Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state. This scheduling method is used by the Microsoft Windows 3.1 and by the Apple Macintosh operating systems. It is the only method that can be used on certain hardware platforms, because it does not require the special hardware needed for preemptive scheduling.

Preemptive scheduling incurs a cost. Consider the case of two processes sharing data. One may be in the midst of updating the data when it is preempted and the second process is run. The second process may try to read the data, which are currently in an inconsistent state. New mechanisms are thus needed to coordinate access to shared date.

Preemption also has an effect on the design of the operating system kernel. During the processing of a system call, the kernel may be busy with an activity on behalf of a process. Such activities may involve changing important kernel data. If the process is preempted in the middle of these changes, and the kernel needs to read or modify the structure, chaos could ensue. Some operating systems, including most version of UNIX, deal with this problem by waiting either for a system call to complete, or for an I/O block to take place, before doing a context switch. This scheme ensures that the kernel structure is simple, since the kernel will not preempt a process while the kernel data structures are in an inconsistent state. Unfortunately, this kernel execution model is a poor one for supporting real-time computing and multiprocessing.

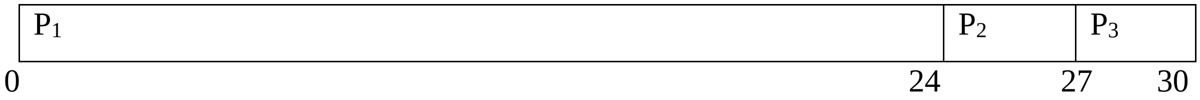
First-Come, First-Served Scheduling

The simplest CPU scheduling algorithm is the first-come, first-served (FCFS) scheduling algorithm. With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCFS policy is easily managed with a FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. The running process is then removed from the queue. The code for FCFS scheduling is simple to write and understand.

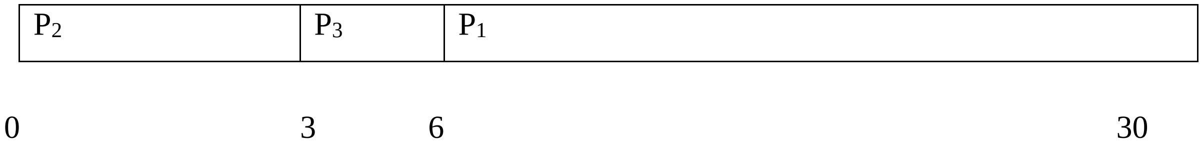
The average waiting time under the FCFS policy, is often quite long. Consider the following set of processes that arrive at time 0, with the length of the CPU burst time given in milliseconds.

Process	*Burst Time*
P_1	24
P_2	3
P_3	3

If the processes arrive in the order P_1,P_2,P_3, and are served in FCFS order, the result is obtained as shown in the Gantt chart:



The waiting time is 0 milliseconds for process P_1, 24 milliseconds for process P_2, and 27 milliseconds for process P_3. Thus, the average waiting time is (0+24+27)/3=17 milliseconds. If the processes arrive in the order P_2,P_3,P_1, the results will be as shown in the following Gantt chart:



The average waiting time is now (0+3+6)/3=3 milliseconds. This reduction is substantial. Thus, the average waiting time under FCFS policy is generally not minimal, and may vary substantially if the process CPU burst times vary greatly.

In addition, consider the performance of FCFS scheduling in a dynamic situation. Assume only one CPU bound process and many I/O bound processes are there. As the processes flow around the system, the following scenario may result. The CPU bound process will get the CPU and hold it. During this time, all the other processes will finish their I/O and move into the ready queue, the I/O devices are idle. Eventually, the CPU bound process finishes its CPU burst and moves to an I/O device. All the I/O bound processes, which have very short CPU bursts, execute quickly and move back to the I/O queues. At this point, the CPU sits idle. The CPU bound process will then move back to the ready queue and be allocated to the CPU. Again, all the I/O processes end up waiting in the ready queue until the CPU bound process is done. There is a convoy effect, as all the other processes wait for the one big process to get off the CPU. This effect results in lower CPU and device utilization than might be possible if the shorter processes were allowed to go first.

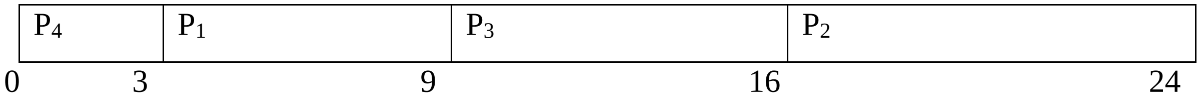
Shortest-Job-First Scheduling

A different approach to CPU scheduling is the shortest-job-first (SJF) scheduling algorithm. This algorithm associates with each process the length of the latter’s next CPU burst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If two processes have the same length next CPU burst, FCFS scheduling is used to break the tie. The appropriate term would be the *shortest next CPU burst*, because the scheduling is done by examining the length of the next CPU burst of a process, rather than its total length.

Consider the following set of processes, with length of the CPU burst time given in milliseconds as an example:

Process	*Burst Time*
P_1	6
P_2	8
P_3	7
P_4	3

Using SJF scheduling, these processes are scheduled according to the following Gantt chart:



The waiting time is 3 milliseconds for process P_1, 16 milliseconds for process P_2, 9 milliseconds for process P_3, and 0 milliseconds for process P_4. Thus, the average waiting time is (3+16+9+0)/4=7 milliseconds. If it was FCFS scheduling scheme, then the waiting time would have been 10.25 milliseconds.

The SJF scheduling algorithm is provably optimal, in that it gives the minimum average waiting time for a given set of processes. By moving a short process before a long one, the waiting time of the short process decreases more than it increases the waiting time of the long process. Consequently, the average waiting time decreases.

The real difficulty with the SJF algorithm is to know the length of the next CPU request. For long term (or job) scheduling in a batch system, we can use as the length the process time limit that a user specifies when he submits the job. Thus, users are motivated to estimate the process time limit accurately, since a lower value may mean faster response. SJF scheduling is used frequently in long-term scheduling.

Although the SJF algorithm is optimal, it cannot be implemented at the level of short term CPU scheduling. There is no way to know the length of the next CPU burst. One approach is to try to approximate SJF scheduling. The length of the next CPU burst may not be known, but it can be predicted. It is expected that the next CPU burst will be similar in length to the previous ones. Thus, by computing an approximation of the length of the next CPU burst, the process with the shortest predicted CPU burst can be picked.

The next CPU burst is generally predicted as an exponential average of the measured lengths of previous CPU bursts. Let t_n be the length of the nth CPU burst, and let T_{n+1} be the predicted value for the next CPU burst. Then, for α, $0 \leq \alpha \leq 1$, define

$$T_{n+1}= \alpha\, t_n+(1- \alpha)T_n$$

This formula defines an exponential average. The value of t_n contains most recent information; T_n stores the past history in the prediction. The parameter α controls the relative weight of recent and past history in the prediction. If $\alpha=0$, then $T_{n+1}=t_n$, and only the most recent CPU burst matters.

More commonly, $\alpha=1/2$, so recent history and past history are equally weighted. The initial T_0 can be defined as a constant or as an overall system average. Figure 6.1 shows an exponential average with $\alpha=1/2$ and $T_0=10$.To understand the behavior of the exponential average, the formula for T_{n+1} can be expanded by substituting for T_n, to find

$$T_{n+1}= \alpha\, t_n+ (1- \alpha)\, \alpha\, t_{n-1}+\ldots+ (1- \alpha)^j\, \alpha\, t_{n-j}+\ldots+ (1- \alpha)^{n+1}T_0$$

Since both α and $(1- \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor.

The SJF algorithm may be either preemptive or non-preemptive. The choice arises when a new process arrives at the ready queue while a previous process is executing. The new process may have a shorter next CPU burst than what is left of the currently executing process. A preemptive SJF algorithm will preempt the currently executing process, whereas a non-preemptive SJF algorithm will allow the currently running process to finish its CPU burst. Preemptive SJF scheduling is sometimes called shortest-remaining-time-first scheduling.

Consider the following four processes, with length of the CPU burst time given in milliseconds:

Process	*Arrival Time*	*Burst Time*
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

If the processes arrive at the ready queue at the times shown and need the indicated burst times, then the resulting preemptive SJF schedule is as depicted in the following Gantt chart:

P_1	P_2	P_4	P_1	P_3

0 1 5 10 17 26

Process P_1 is started at time 0, since it is the only process in the queue. Process P_2 arrives at time 1. The remaining time for process P_1 (7 milliseconds) is larger than the time required by process P_2 (4 milliseconds), so process P_1 is preempted, and process P_2 is scheduled. the average waiting time for this example is ((10-1)+(1-1)+(17-2)+(5-3))/4=26/4=6.5 milliseconds. A non-preemptive SJF scheduling would result in an average waiting time of 7.75 milliseconds.

Priority Scheduling

The SJF algorithm is a special case of the general priority-scheduling algorithm. A priority is associated with each process, and the CPU is allocated to the process with the highest priority. Equal-priority processes are scheduled in FCFS order.

An SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst. The larger the CPU burst, the lower the priority, and vice versa.

Priorities are generally some fixed range of numbers, such as 0 to 7, or 0 to 4, 095. However, there is no general agreement on whether 0 is the highest or lowest priority. Some systems use low numbers to represent low priority; others use low numbers for high priority. This difference can lead to confusion. In this description, low priority numbers are used to represent high priority.

Consider the following set of processes, assumed to have arrived at time 0, in the order P_1, P_2, …, P_5, with the length of the CPU burst time given in milliseconds as an example:

Process	*Burst Time*	*Priority*
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Using priority scheduling, these processes are scheduled according to the following Gantt chart:

P_2	P_5	P_1	P_3	P_4

0 1 6 16 18 19

The average waiting time is 8.2 milliseconds.

Priorities can be defined either internally or externally. Internally defined priorities use some measurable quantity or quantities to compute the priority of a process. For example, time limits, memory requirements, the number of open files, and the ratio of average I/O burst to average CPU burst have been used in computing priorities. External priorities are set by criteria that are external to the operating system, such as the importance of the process, the type and amount of funds being paid for computer use, the department sponsoring the work, and other, often political factors.

Priority scheduling can be either preemptive or non-preemptive. When a process arrives at the ready queue, its priority is compared with priority of the currently running process. A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process. A non-preemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.

Round Robin Scheduling

The round robin (RR) scheduling algorithm is designed especially for time-sharing systems. It is similar to FCFS scheduling, but preemption is added to switch between processes. A small unit of time called a time quantum (or time slice) is defined. A time quantum is generally from 10 to 100 milliseconds. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.

To implement RR scheduling, the ready queue is kept as a FIFO queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process.

One of two things will then happen. The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in thread queue. Otherwise, if the CPU burst of the currently running process is longer than 1 time quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will be executed, and the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue.

Consider the following set of processes that arrive at time 0, with the length of the CPU burst time given in milliseconds:

Process	*Burst Time*
P_1	24
P_2	3
P_3	3

If a time quantum of 4 milliseconds is used, then process P_1 gets the first 4 milliseconds. Since it requires another 20 milliseconds, it is preempted after the first time quantum, and the CPU is given to the next process in the queue, process P_2. Since process P_2 does not need 4 milliseconds, it quits before its time quantum expires. The CPU is then given to the next process, process P_3.Once each process has received 1 time quantum, the CPU is returned to process P_1 for an additional time quantum. The resulting RR schedule is

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

0 4 7 10 14 18 22 26 30

The average waiting time is 17/3=5.66 milliseconds.

In RR scheduling algorithm, no process is allocated the CPU for more than 1 time quantum in a row. If a process's CPU burst exceeds 1 time quantum, that process is preempted and is put back in the ready queue. The RR scheduling algorithm is preemptive.

If there are n processes in the ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units. Each process must wait no longer than (n-1)×q time units until its next time quantum. For example, then each process will get up to 20 milliseconds every 100 milliseconds.

Conclusion: Thus, we have studied various scheduling algorithms.

QUESTIONS:

1. Scheduling? List types of scheduler & 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

ASSIGNMENT NO:08 – Group B

Assignment Title: Implement a program to simulate Memory placement strategies – best fit, first fit, next fit and worst fit.

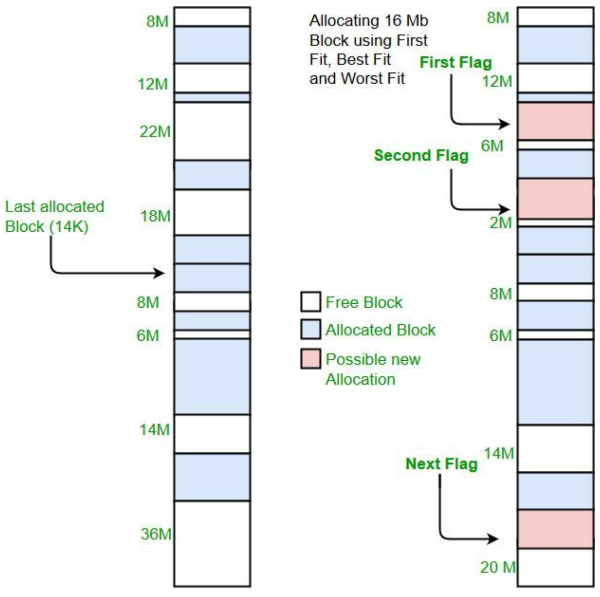
Problem Statement: Write a program to simulate Memory placement strategies – best fit, first fit, next fit and worst fit.

Objectives:

1. To understand concept of memory placement technique.
2. To implement concept of memory placement technique.

Theory:

- First Fit:-
    - Advantages
    - It is the fastest search as it searches only the first block i.e. enough to assign a process.
    - It may have problems of not allowing processes to take space even if it was possible to allocate. Consider the above example; process number 4 (of size 426) does not get memory. However it was possible to allocate memory if we had allocated using best fit allocation [block number 4 (of size 300) to process 1, block number 2 to process 2, block number 3 to process 3 and block number 5 to process 4].
    - **Implementation:**
    - Input memory blocks with size and processes with size.
    - Initialize all memory blocks as free.
    - Start by picking each process and check if it can, be assigned to current block.
    - If size-of-process <= size-of-block if yes then assign and check for next process.
    - If not then keep checking the further blocks.



(a) Before (a) After

Next Fit

Next fit is a modified version of 'first fit'. It begins as the first fit to find a free partition but when called next time it starts searching from where it left off, not from the beginning. This policy makes use of a roving pointer. The pointer moves along the memory chain to search for a next fit. This helps in, to avoid the usage of memory always from the head (beginning) of the free block chain.

- **Advantages:-**

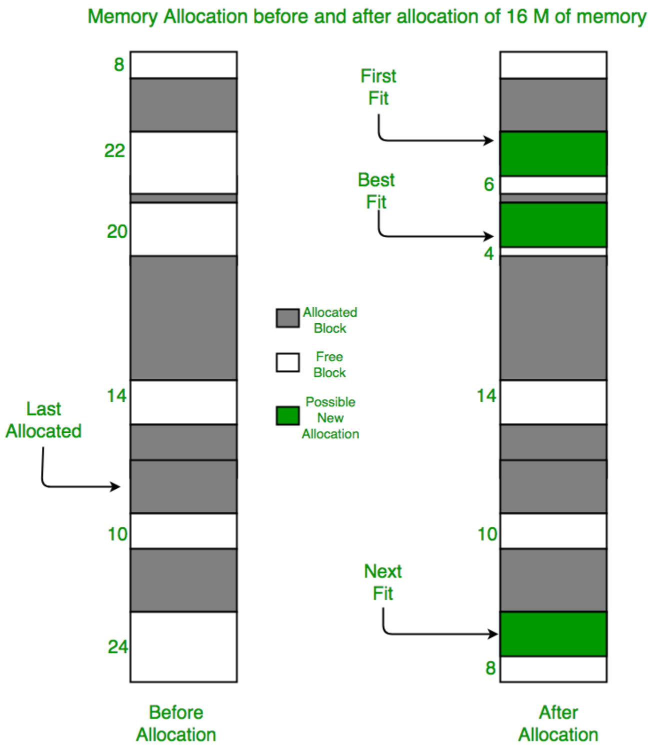
- First fit is a straight and fast algorithm, but tends to cut large portion of free parts into small pieces due to which, processes that need a large portion of memory block would not get anything even if the sum of all small pieces is greater than it required which is so-called external fragmentation problem.
- Another problem of the first fit is that it tends to allocate memory parts at the beginning of the memory, which may lead to more internal fragments at the beginning. Next fit tries to address this problem by starting the search for the free portion of parts not from the start of the memory, but from where it ends last time.
- Next fit is a very fast searching algorithm and is also comparatively faster than First Fit and Best Fit Memory Management Algorithms.

- **Implementation**

  Input the number of memory blocks and their sizes and initializes all the blocks as free.

- Input the number of processes and their sizes.

- Start by picking each process and check if it can be assigned to the current block, if yes, allocate it the required memory and check for next process but from the block where we left not from starting.
- If the current block size is smaller then keep checking the further blocks.

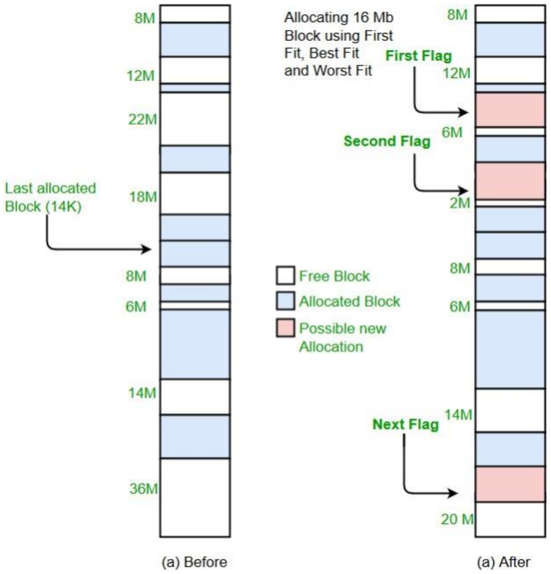


- Worst Fit

  Allocates a process to the partition which is largest sufficient among the freely available partitions available in the main memory. If a large process comes at a later stage, then memory will not have space to accommodate it.

  - **Implementation:**
    - Input memory blocks and processes with sizes.
    - Initialize all memory blocks as free.
    - Start by picking each process and find the maximum block size that can be assigned to current process i.e., find max(bockSize[1], blockSize[2], ...blockSize[n]) > processSize[current], if found then assign it to the current process.

- If not then leave that process and keep checking the further processes.



(a) Before (a) After

Conclusion: Processes and files allocated to free blocks. List of processes and files which are not allocated memory. The remaining free space list left out after performing allocation..

ASSIGNMENT NO: 09 – Group B

Assignment Title: Implementation of concept called Paging, using simulation.

Problem Statement: Write a Java Program (using OOP features) to implement paging simulation using

1. Least Recently Used (LRU)
2. Optimal algorithm

1.2 Objectives:

1. To understand concept of paging.
2. To learn and paging techniques.

1.3 Theory: Paging: Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non – contiguous.

- Logical Address or Virtual Address (represented in bits): An address generated by the CPU
- Logical Address Space or Virtual Address Space(represented in words or bytes): The set of all logical addresses generated by a program
- Physical Address (represented in bits): An address actually available on memory unit
- Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses

Example:

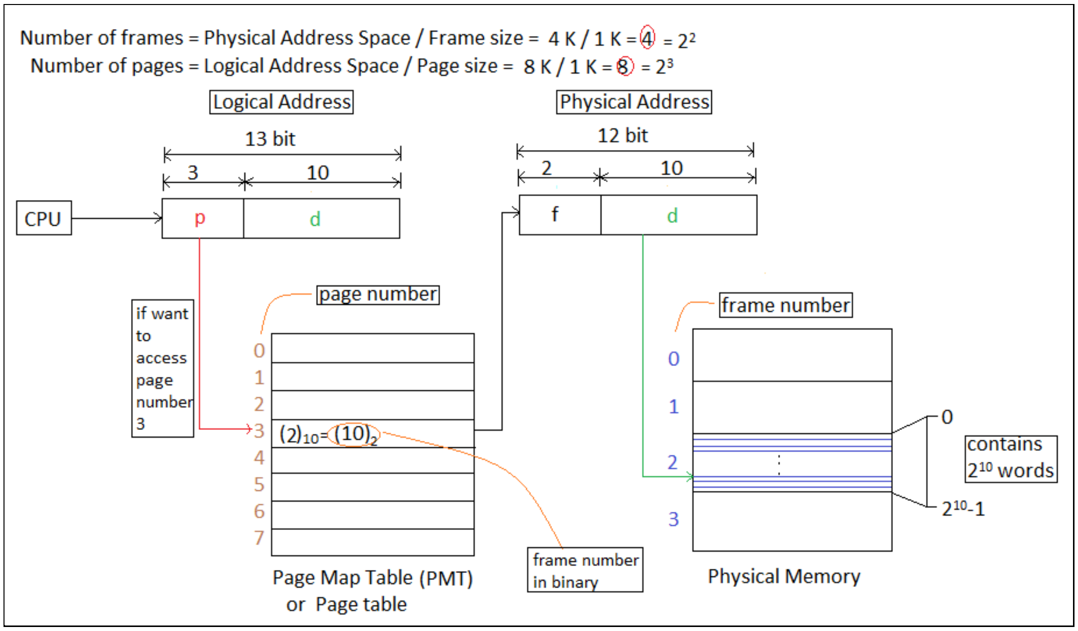
- If Logical Address = 31 bit, then Logical Address Space = 2^{31} words = 2 G words (1 G = 2^{30})
- If Logical Address Space = 128 M words = $2^7 * 2^{20}$ words, then Logical Address = $\log_2 2^{27}$ = 27 bits
- If Physical Address = 22 bit, then Physical Address Space = 2^{22} words = 4 M words (1 M = 2^{20})
- If Physical Address Space = 16 M words = $2^4 * 2^{20}$ words, then Physical Address = $\log_2 2^{24}$ = 24 bits

The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device and this mapping is known as paging technique.

- The Physical Address Space is conceptually divided into a number of fixed-size blocks, called **frames**.
- The Logical address Space is also splitted into fixed-size blocks, called **pages**.
- Page Size = Frame Size

Let us consider an example:

- Physical Address = 12 bits, then Physical Address Space = 4 K words
- Logical Address = 13 bits, then Logical Address Space = 8 K words
- Page size = frame size = 1 K words (assumption)


Number of frames = Physical Address Space / Frame size = 4 K / 1 K = 4 = 2^2
Number of pages = Logical Address Space / Page size = 8 K / 1 K = 8 = 2^3
Logical Address
13 bit
3
10
CPU
p
d
Physical Address
12 bit
2
10
f
d
if want to access page number 3
page number
0
1
2
3
4
5
6
7
$(2)_{10} = (10)_2$
Page Map Table (PMT) or Page table
frame number in binary
frame number
0
1
2
3
0
contains 2^{10} words
2^{10}-1
Physical Memory

Address generated by CPU is divided into

- **Page number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
- **Page offset(d):** Number of bits required to represent particular word in a page or page size of Logical Address Space or word number of a page or page offset.

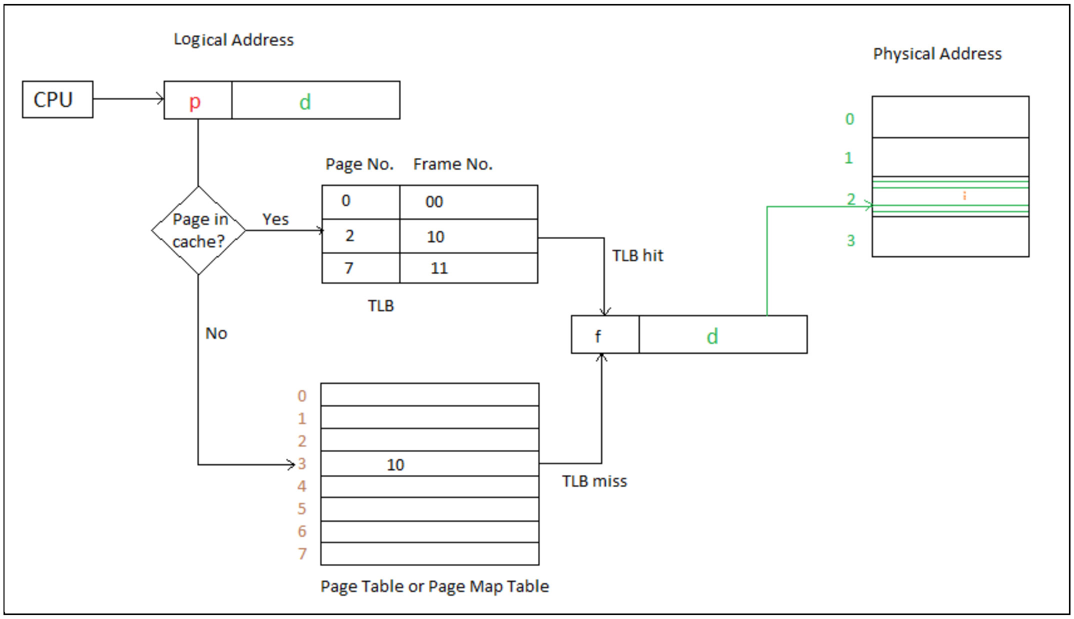
Physical Address is divided into

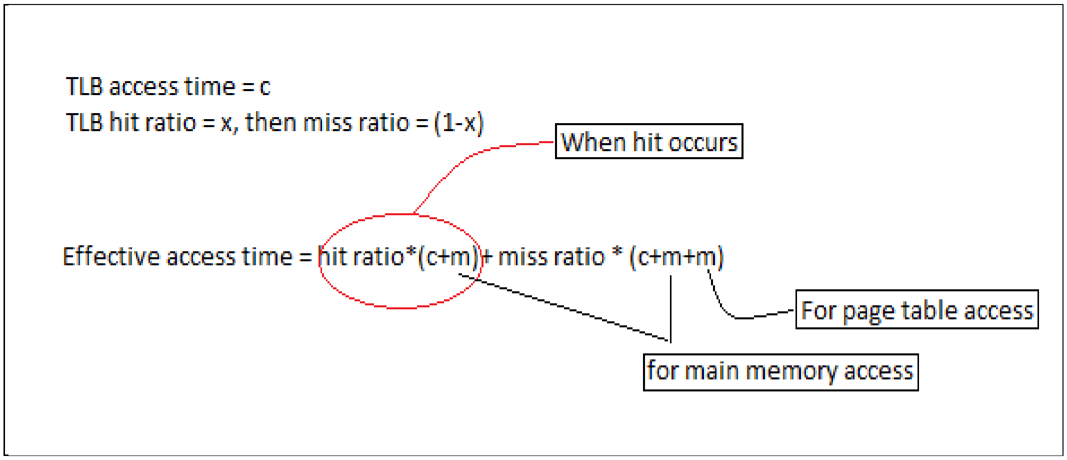
- **Frame number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number.
- **Frame offset(d):** Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

The hardware implementation of page table can be done by using dedicated registers. But the usage of register for the page table is satisfactory only if page table is small. If page table contain large number of entries then we can use TLB(translation Look-aside buffer), a special, small, fast look up hardware cache.

- The TLB is associative, high speed memory.
- Each entry in TLB consists of two parts: a tag and a value.
- When this memory is used, then an item is compared with all tags simultaneously.If the item is found, then corresponding value is returned.

Main memory access time = m
If page table are kept in main memory,
Effective access time = m(for page table) + m(for particular page in page table)




TLB access time = c
TLB hit ratio = x, then miss ratio = (1-x)
When hit occurs
Effective access time = hit ratio*(c+m)+ miss ratio * (c+m+m)
For page table access
for main memory access

Page Replacement Algorithms:

In a operating systems that use paging for memory management, page replacement algorithm are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.

Page Fault – A page fault is a type of interrupt, raised by the hardware when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Page Replacement Algorithms:

1. **Least Recently Used:**–In this algorithm page will be replaced which is least recently used.

Algorithm:

Let **capacity** be the number of pages that memory can hold. Let **set** be the current set of pages in memory.

1- Start traversing the pages.

i) **If set holds less pages than capacity.**

a) Insert page into the set one by one until the size of**set** reaches **capacity** or all page requests are processed.

b) Simultaneously maintain the recent occurred index of each page in a map called **indexes**.

c) Increment page fault

ii) **Else**

If current page is present in **set**, do nothing.

Else

a) Find the page in the set that was least recently used. We find it using index array.

We basically need to replace the page withminimum index.

b) Replace the found page with current page.

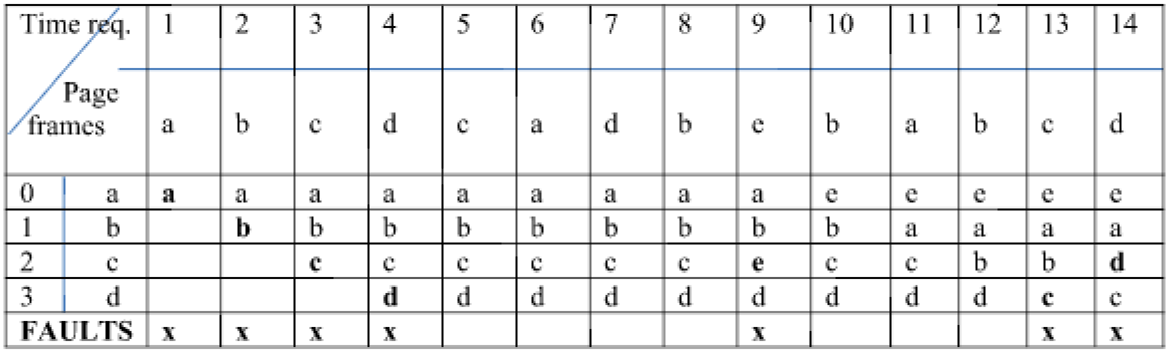
c) Increment page faults.

d) Update index of current page.

2. Return page faults.

Let say the page reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 . Initially we have 4 page slots empty. Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults** 0 is already their so —>**0 Page fault.** When 3 came it will take the place of 7 because it is least recently used —>**1 Page fault** 0 is already in memory so —> **0 Page fault**. 4 will takes place of 1 —> **1 Page Fault** Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

Example-, Let's have a reference string: a, b, c, d, c, a, d, b, e, b, a, b, c, d and the size of the frame be 4.



Time req.		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Page frames		a	b	c	d	c	a	d	b	e	b	a	b	c	d
0	a	**a**	a	a	a	a	a	a	a	a	e	e	e	e	e
1	b		**b**	b	b	b	b	b	b	b	b	a	a	a	a
2	c			**c**	c	c	c	c	c	**e**	c	c	b	b	**d**
3	d				**d**	d	d	d	d	d	d	d	d	**c**	c
FAULTS		**x**	**x**	**x**	**x**					**x**				**x**	**x**

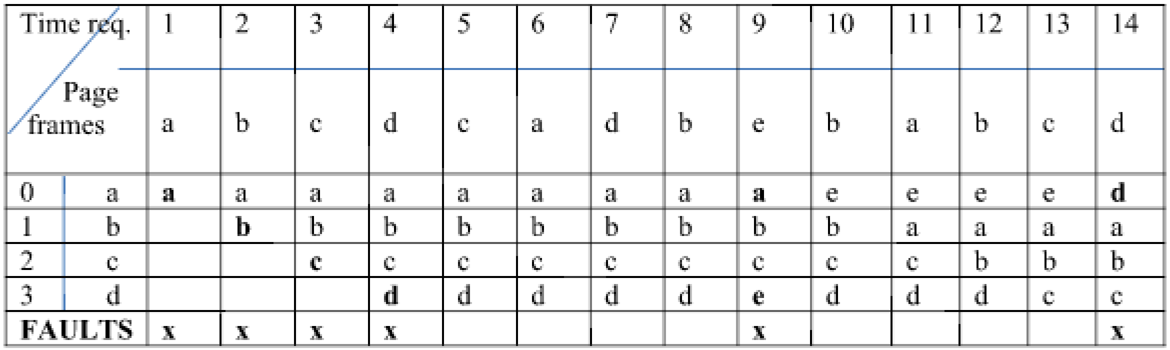
There are 7 page faults using LRU algorithm.

2. **Optimal Page replacement:** – In this algorithm, pages are replaced which are not used for the longest duration of time in the future. Let us consider page reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 and 4 page slots.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already there so —>**0 Page fault.**
When 3 came it will take the place of 7 because it’s not used for longest duration in future. ->**1 Page fault.**
0 is already there so —> **0 Page fault.**.
4 will take place of 1 —> **1 Page Fault.**

Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

Example-2, Let’s have a reference string: a, b, c, d, c, a, d, b, e, b, a, b, c, d and the size of the frame be 4.



Time req.		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Page frames		a	b	c	d	c	a	d	b	e	b	a	b	c	d
0	a	**a**	a	a	a	a	a	a	a	**a**	e	e	e	e	**d**
1	b		**b**	b	b	b	b	b	b	b	b	a	a	a	a
2	c			**c**	c	c	c	c	c	c	c	c	b	b	b
3	d				**d**	d	d	d	d	**e**	d	d	d	c	c
FAULTS		x	x	x	x					x					x

There are 6 page faults using optimal algorithm.

Optimal page replacement is perfect, but not possible in practice as operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

Conclusion: Thus, we have studied various paging techniques.

ASSIGNMENT NO: 10 – Group B

Assignment Title: Perform Implementation of Deadlock avoidance algorithm, i.e Bankers Algorithms.

Problem Statement: Write a Java program to implement Banker's Algorithm.

Objectives:

1. To understand concept of resource allocation and deadlock avoidance.
2. To learn and use Bankers algorithms.

Theory: Bankers Algorithm:-

The resource-allocation graph algorithm is not applicable to a resource-allocation system with multiple instances of each resource type. The deadlock-avoidance algorithm that we describe next is applicable to such a system, but is less efficient than the resource-allocation graph scheme. This algorithm is commonly known as the banker's algorithm. The name was chosen because this algorithm could be used in a banking system to ensure that the bank never allocates its available cash such that it can no longer satisfy the needs of all its customers.

When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system. When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If it will, the resources are allocated; otherwise, the process must wait until some other process releases enough resources.

Algorithm

Several data structures must be maintained to implement the banker's algorithm. These data structures encode the state of the resource-allocation system. Let n be the number of processes in the system and m be the number of resource types. We need the following data structures:

1. **Available:** A vector of length m indicates the number of available resources of each type. If $Available[j]=k$, there are k instances of resource type $R_j$available.
2. **Max:** An $n\times m$ matrix defines the maximum demand of each process. If $Max\ [i,j]=k$, then process $P_i$may request at most k instances of resource type R_j.

3. **Allocation:** An n×$m$matrix defines the number of resources of each type currently allocated to each process. If *Allocation [i,j]=k*, then process P_i is currently allocated *k* instances of resource type R_j.
4. **Need:** An n×$m$matrix indicates the remaining resource need of each process. If *Need [i,j]=k,* then process $P_i$may need *k* more instances of resource type $R_j$to complete its task.

   Need [i,j]=Max[i,j]-Allocation[i,j]

These data structures vary over time in both size and value.

To simplify the presentation of the banker's algorithm, let us establish some notation. Let X and Y be vectors of length n, X≤Y if and only if X[i]≤Y[i] for all i=1,2,…,n. For example, if X=(1,7,3,2) and Y=(0,3,2,1), then Y≤X. Y<X if Y≤X and Y≠X.

We can treat each row in the matrices Allocation and Need as vectors and refer to them as $Allocation_i$and $Need_i$, respectively. The vector $Allocation_i$ specifies the resources currently allocated to process P_i; the vector $Need_i$ specifies the additional resources that process $P_i$may still request to complete its task.

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be:

Let Work and Finish be vectors of length m and n, respectively. Initialize *Work:=Availableand Finish* [*i*]*:=false* for *i=1,2,…,n*

1. Find an i such that both

   a. *Finish*[*i*]*=false*

   b. *Needi≤Work*

   If no such i exists, go to step 4.

2. *Work:=Work+*$Allocation_i$

   Finish[*i*]*:=true*

   go to step 2.

3. If *Finish*[*i*]*=true* for all *i*, then the system is in a safe state.

This algorithm may require an order of $m \times n^2$ operations to decide whether a state is safe.

Resource-Request Algorithm

Let *Request*$_i$ be the request vector for process *P*$_i$. If *Request*$_i$[*j*]=*k*, then process *P*$_i$wants *k* instances of resource type *R*$_j$. When a request for resources is made by process *Pi*, the following actions are taken:

1. If *Requesti≤Needi*, go to step2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.
2. If *Requesti≤Available*, go to step3. Otherwise, *Pi* must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process *Pi* by modifying the state as follows:

$Available:=Available-Request_i;$

$Allocation_i:=Allocation_i+Request_i;$

$Need_i:=Need_i-Request_i;$

If the resulting resource-allocation state is safe, the transaction is completed and process *P*$_i$ is allocated its resources. If the new state is unsafe, then *P*$_i$ must wait for *Request*$_i$ and the old resources-allocation state is restored.

Example: The system with five processes P_0 through P_4 and three resource types *A*,*B*,*C*. Resource type A has 10 instances, resource type B has 5 instances, and

resource type C has 7 instances. Suppose that, at time T_0, the following snapshot of the system has been taken:

	Allocation			*Max*			*Available*		
	A	*B*	*C*	*A*	*B*	*C*	*A*	*B*	*C*
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

The content of the matrix *Need* is defined to be *Max- Allocation* and is

Need

	A	*B*	*C*
P_0	7	4	3

P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

We claim that the system is currently in a safe state. The sequence <P_1, P_3, P_4,P_2, P_0> satisfies the safety criteria. Suppose now that process P_1 requests one additional instance of resource type A and two instances of resource type C, so Request$_1$=(1,0,2). To decide whether this request can be immediately granted, we first check that Request$_1$ ≤Available which is true. Then this request has been fulfilled, and the following new state is arrived:

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

We must determine whether this new system state is safe. To do so, safety algorithm should be executed and the sequence is found to be <P_1, P_3, P_4, P_0, P_2> satisfies our safety requirement. Hence, the request of process P1 can be granted immediately.

Conclusion: Thus, we have studied deadlock avoidance using Bankers algorithms.

QUESTIONS:

1. Which one of the following is the deadlock avoidance *algorithm*?

a) Banker's algorithm b) round-robin algorithm c) elevator algorithm d) karn's algorithm.

2. What is deadlock? Why it occurs in system?

3. What is Deadlock prevention

4. What is Deadlock Avoidance
5. Explain Bankers algorithm with one example.

Assignment -11

Problem Statement:

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

LEARNING OBJECTIVES:

● Understanding the connectivity of Raspberry Pi IR sensor.

LEARNING OUTCOMES:

● The students will be able to To interface IR sensor to Raspberry pi.
● Detect the obstacle with IR sensor.
● Can perform actuation.

H/W AND S/W REQUIREMENTS:

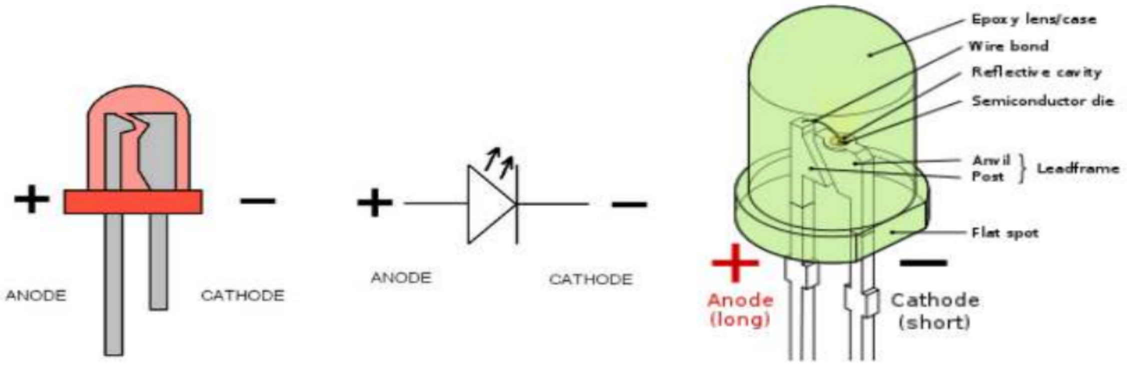
Raspberry Pi/Beagle board Development Boards PC / Monitor/Keyboard IR (Infrared) Sensor, 1 LED, 1 Resistor (330 Ω) Few jumper cables,1 Breadboard Raspbian (OS), Debian LINUX and Python.

Theory:

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the
Raspberry Pi

The raspberry pi comes with a set of open source technologies, i.e. communication and multimedia web technologies. In the year 2014, the foundation of the raspberry pi board launched the computer module, that packages a model B raspberry pi board into module for use as a part of embedded systems, to encourage their use.IR module: Infrared radiation is an electromagnetic wave with wavelength of 700nm to 1 mm. It is emitted by objects with temperature above 0 kelvins. Furthermore intensity and wavelength of infrared radiation depends on the temperature of the object. The infrared sensors are the sensors that detect/measure infrared radiation or change in the radiation from outer source or inbuilt source. Also sensors that use the property of infrared radiations to detect the changes in surrounding are termed as infrared sensors. 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.

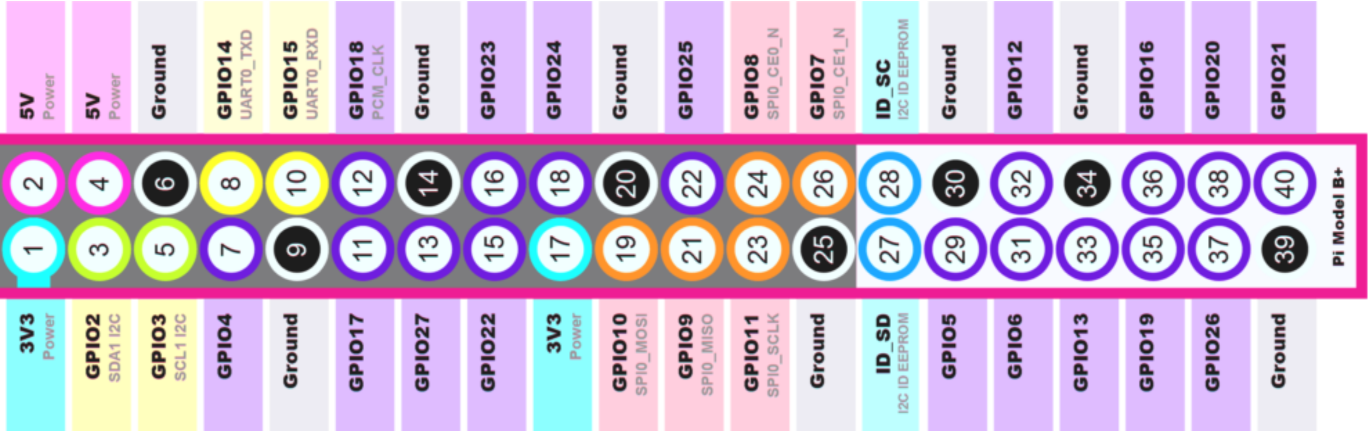
FC-51


+
−
ANODE
CATHODE
+
−
ANODE
CATHODE
Epoxy lens/case
Wire bond
Reflective cavity
Semiconductor die
Anvil
Post
Leadframe
Flat spot
+
−
Anode
(long)
Cathode
(short)

LED:

A light-emitting diode (LED) is a two-lead semiconductor light source. It is a p–n junction diode that emits light when activated When a suitable current is applied to the leads; electrons are able to recombine with electron holes within the device, releasing energy in the form of photons. This effect is called electroluminescence, and the color of the light (corresponding to the energy of the photon) is determined by the energy band gap of the semiconductor. LEDs are typically small (less than 1 mm2) and integrated optical components may be used to shape the radiation pattern. Raspberry Pi GPIO Pin and interfacing: A simple LED circuit consists of a LED and resistor. The resistor is used to limit the current that is being drawn and is called a current limiting resistor. Without the resistor the LED would run at too high of a voltage, resulting in too much current being drawn which in turn would instantly burn the LED, and likely also the GPIO port on the Raspberry Pi.

Raspberry Pi GPIO Pin and interfacing:



With the circuit created we need to write the Python script to blink the LED. Before we start writing the software we first need to install the Raspberry Pi GPIO Python module. This is a library that allows us to access the GPIO port directly from Python.

To install the Python library open a terminal and execute the following

```
$ sudo apt-get install python-rpi.gpio python3-rpi.gpio
```

With the library installed now open your favorite Python IDE (recommend Thonny Python IDE more information about using it here).Script needs to do the following:

Initialize the GPIO ports Turn the LED on and off in 1 second intervals

Procedure:

Note: The Raspbian operating system comes with Python already installed in it.

1.Setting up the circuit:

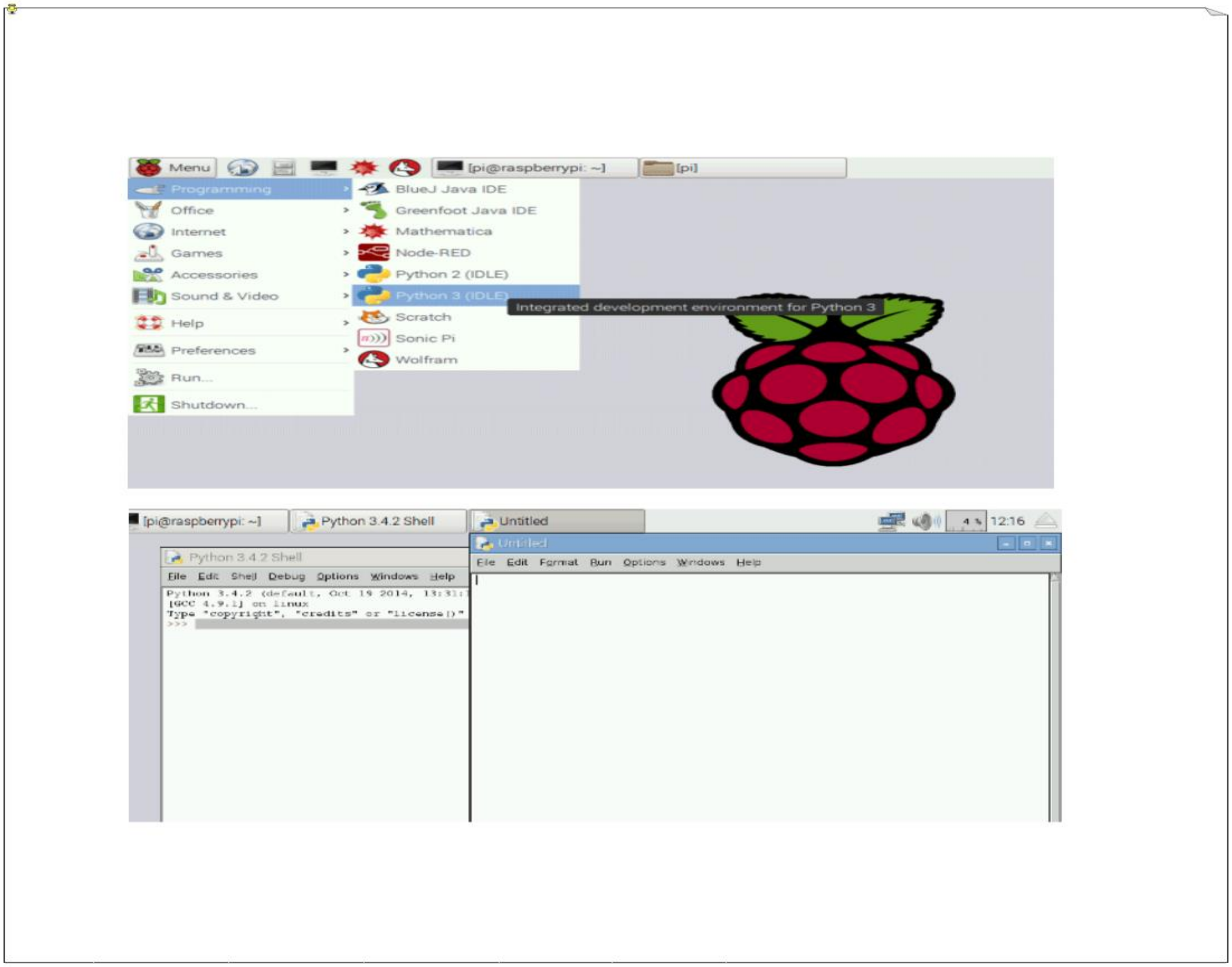
a.Turn OFF the Raspberry Pi while building/connecting the circuitboard/components according to diagram.

b.Then turn the Raspberry Pi and check the Raspbian OS loaded properly or not. If not then check the circuit connection again.

2.Python Programming: Python via IDLE

a.Start Raspberry Pi in desktop mode, open the Applications Menu in the top left of your screen, and navigate to Programming > Python 3 (IDLE) /. This will open the Python shell.

Program4: Python program to detect object motion using IR Sensor with LED indicator



```
Python 3.4.2 (default, Oct 19 2014, 13:31:1
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()"
>>>
```

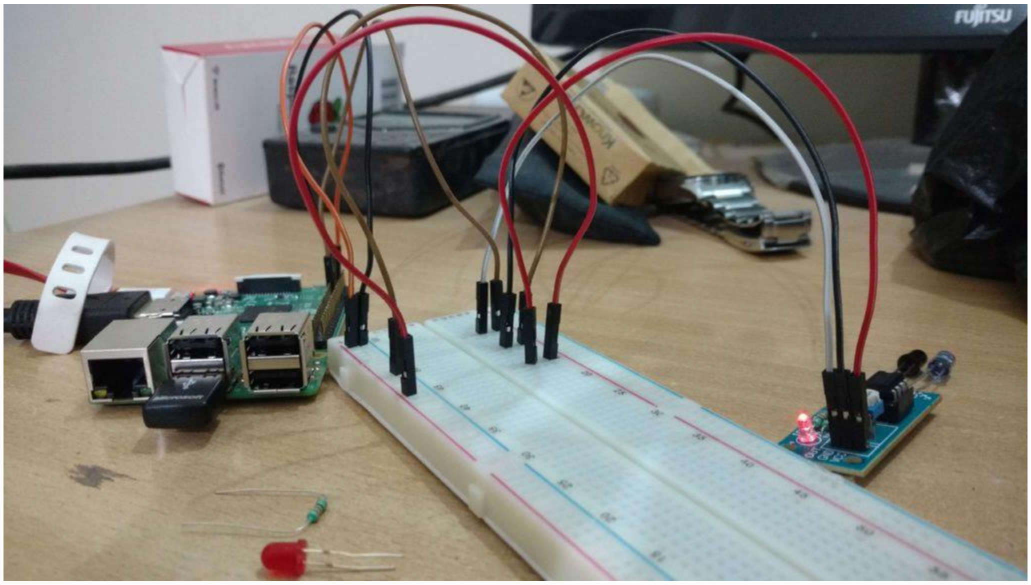
Circuit: To detect obstacles

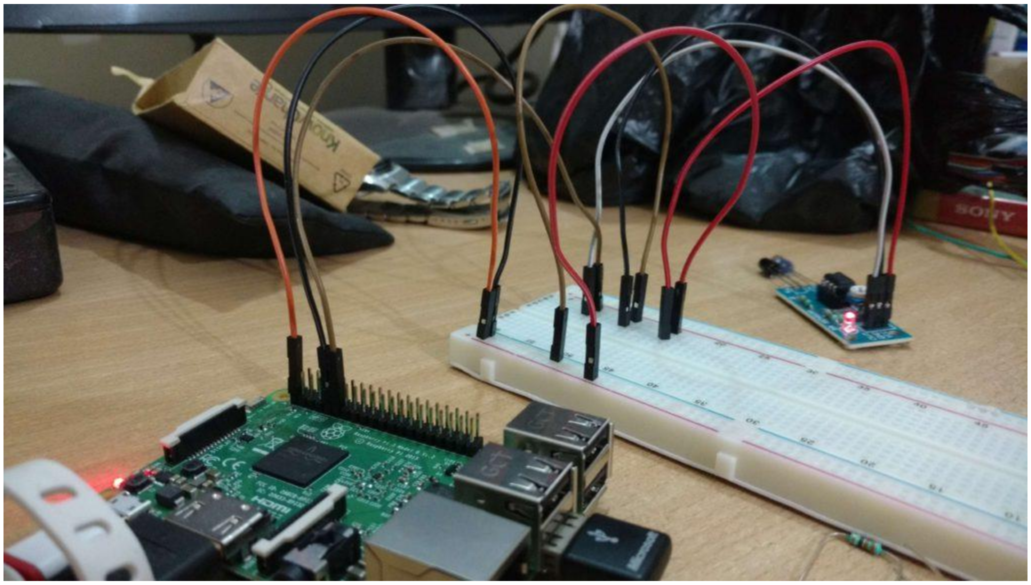
We will be creating a circuit which will turn on the LED when an obstacle is detected. And, as soon as the obstacle is removed from the way the LED will turn off. In order to achieve that follow the steps to create required circuit.

Part 1: Connecting IR Sensor

IR Sensor has 3 pins, viz VCC, GND and OUT. We will use GPIO 17 (do not get confused with pin number 17) for receiving input from the sensor.

1. Connect GPIO 17 from the Raspberry Pi to Breadboard (5a)
2. Connect OUT pin of the sensor with the Breadboard (5c) This will send input received from sensor to GPIO 17, which will be processed further.
3. Connect GND (any pin from board will work, in this post we are using pin number 9) with negative line on left side of the breadboard
4. Connect GND of the IR Sensor to Breadboard (10c)
5. Connect GND from Step 3 to Breadboard (10a)
6. Connect VCC of the IR Sensor to Breadboard (15c)
7. Connect 3v3 (Pin #1) to positive line on left side of the breadboard
8. Connect 3v3 (connected in Step 7) to the Breadboard (15a)



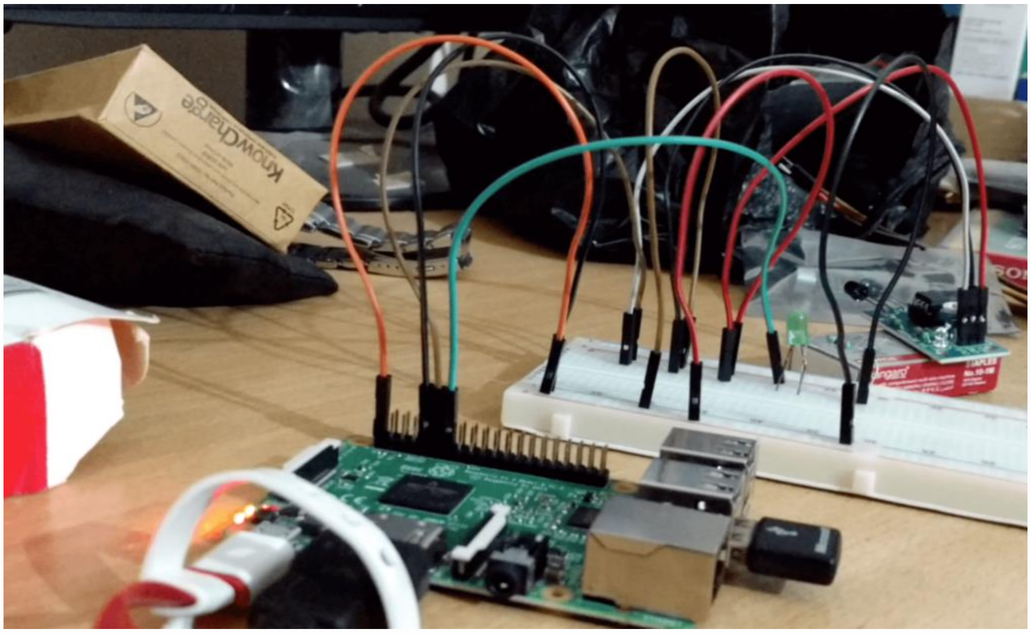


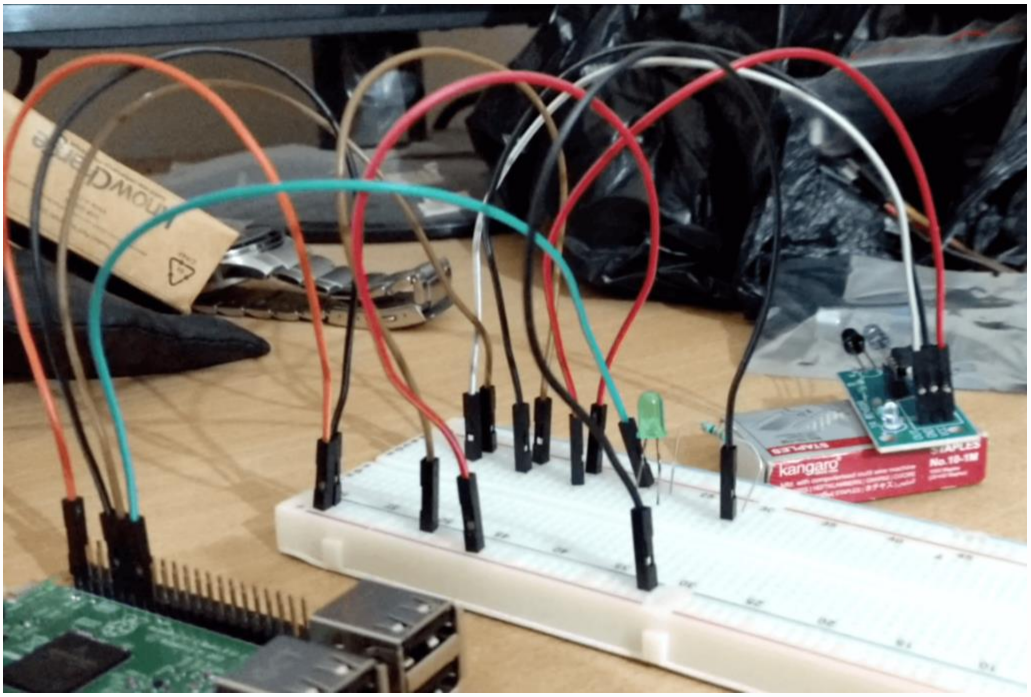
Now the circuit is complete and sensor will detect the obstacle. It can be tested by putting anything in front of the IR Sensor. On-board LED will be on if obstacle is put in front of the sensor, else it will be off.

Part 2: Connecting LED

Objective is to turn on the LED when obstacle is detected.

1. Connect GPIO 4 from the board to the Breadboard(20a)
2. Connect positive point of the LED (longer pin of the LED) to the Breadboard (20c)
3. Connect negative point of the LED (smaller pin of the LED) to the Breadboard (22c)
4. Use resistor (330 Ω) to connect negative (row from Part 1: Step 3) to the negative point of the LED(22a)





Now we are ready to send signal based on the input received from IR Sensor to turn on/off the LED.

Part 3: Code to Connect IR Sensor I/P with LED status

```
from gpiozero import LED
from signal import pause
import RPi.GPIO as GPIO
import time
```

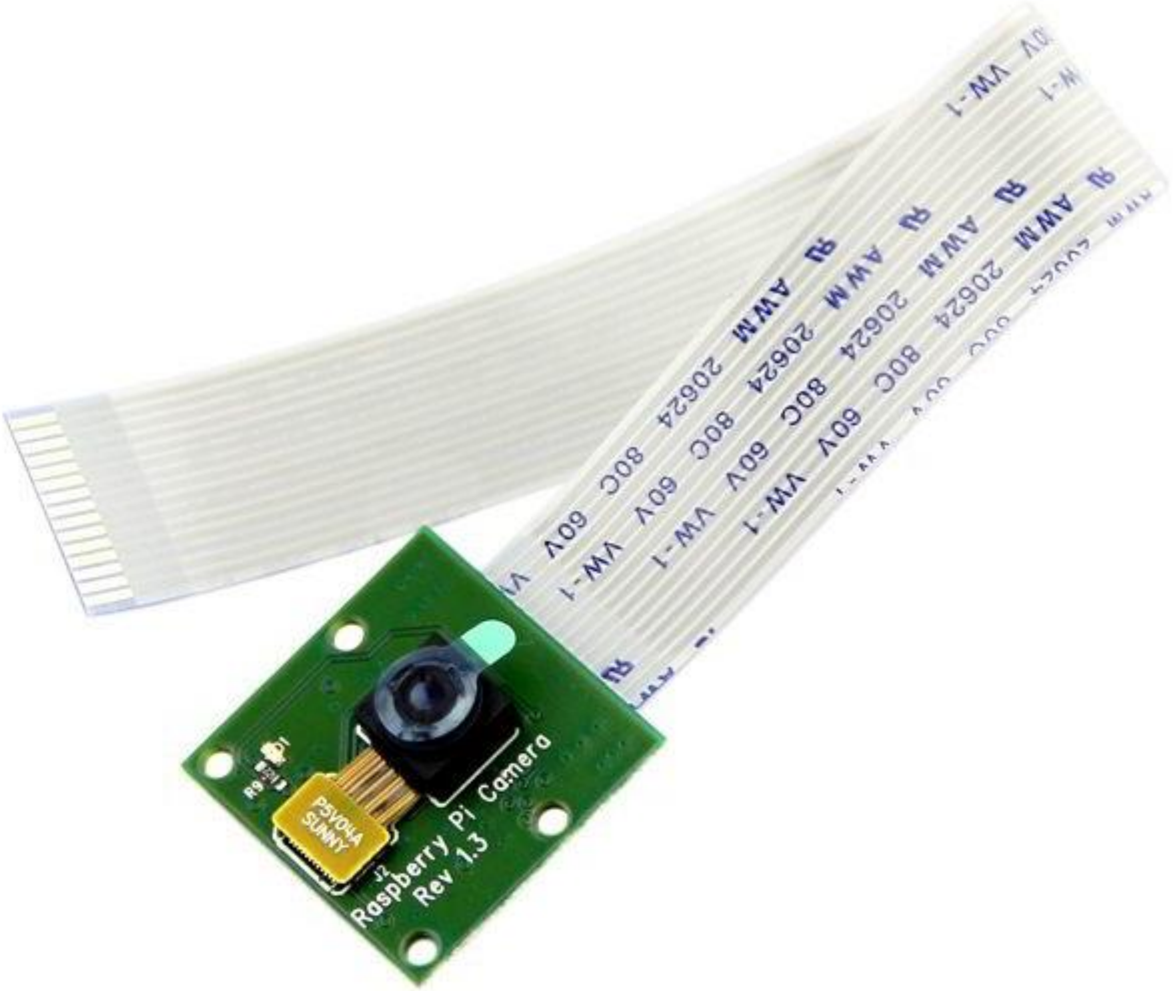
```
GPIO.setmode(GPIO.BCM)

LED_PIN = 27
IR_PIN = 17
indicator = LED(LED_PIN)
GPIO.setup(IR_PIN, GPIO.IN)
count = 1
while True:
  got_something = GPIO.input(IR_PIN)
  if got_something:
    indicator.on()
    print("{:>3} Got something".format(count))
  else:
    indicator.off()
    print("{:>3} Nothing detected".format(count))
  count += 1 time.sleep(0.2)
```

Conclusion: LED blinks when the IR sensor notify the obstacle

Assignment -12

Problem Statement : Understanding and connectivity of Raspberry-Pi /Beagle board with camera. Write an application to capture and store the image.

Theory :



Pi Camera Module (v1.3)

Pi Camera module is a camera which can be used to take pictures and high definition video.

Raspberry Pi Board has CSI (Camera Serial Interface) interface to which we can attach PiCamera module directly.

This Pi Camera module can attach to the Raspberry Pi’s CSI port using 15-pin ribbon cable.

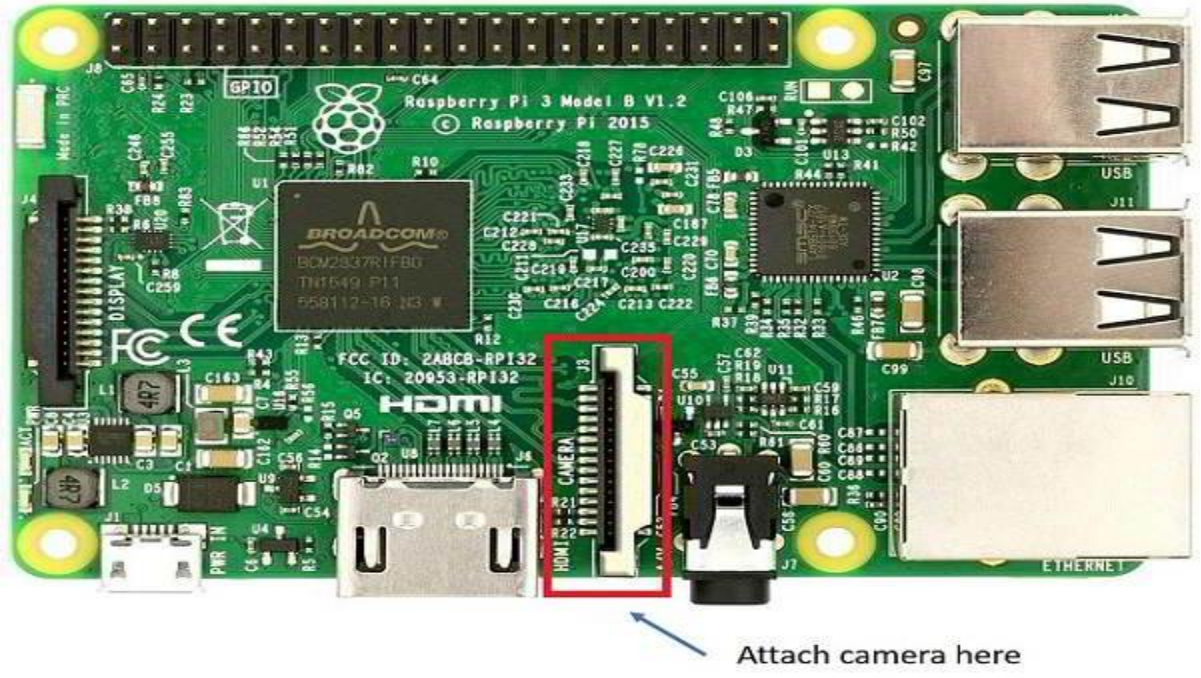
Features of Pi Camera

Here, we have used Pi camera v1.3. Its features are listed below,

- Resolution – 5 MP
- HD Video recording – 1080p @30fps, 720p @60fps, 960p @45fps and so on.
- It Can capture wide, still (motionless) images of resolution 2592x1944 pixels
- CSI Interface enabled.

How to attach Pi Camera to Raspberry Pi?

Connect Pi Camera to CSI interface of Raspberry Pi board as shown below,



Attach camera here

Now, we can use Pi Camera for capturing images and videos using Raspberry Pi.

Before using Pi Camera, we need to enable camera for its working.

How to Enable Camera functionality on Raspberry Pi

For enabling camera in Raspberry Pi, open raspberry pi configuration using following command,

```
sudo raspi-config
```

then select **Interfacing options** in which select **camera** option to enable its functionality.

reboot Raspberry Pi.

Now we can access camera on Raspberry Pi.

Now we can capture images and videos using Pi Camera on Raspberry Pi.

Example

Capture images and save it to the specified directory.

We can capture images using Python. Here, we will write a Python program to capture images using Pi Camera on Raspberry Pi.

Here, we have used picamera package(library) which provides different classes for Raspberry Pi. Out of which we are mainly interested in PiCamera class which is for camera module.

Python Program for Image Capture

```
'''
import picamera
from time import sleep

#create object for PiCamera class
camera = picamera.PiCamera()
#set resolution
camera.resolution = (1024, 768)
camera.brightness = 60
camera.start_preview()
#add text on image
camera.annotate_text = 'Hi Pi User'
sleep(5)
#store image
camera.capture('image1.jpeg')
camera.stop_preview()
```

Functions Used

To use picamera python based library we have to include it in our program as given below

```
import picamera
```

This picamera library has PiCamera class for camera module. So, we have to create object for **PiCamera** class.

PiCamera Class

To use Pi Camera in Python on Raspberry Pi, we can use PiCamera class which has different APIs for camera functionality. We need to create object for PiCamera class.

E.g. Camera = picamera.PiCamera()

The above PiCamera class has different member variables and functions which we can access by simply inserting a dot (.) in between object name and member name.

E.g. Camera.resolution = (1080, 648)

capture()

It is used to capture images using Pi Camera.

E.g. Camera.capture(“/home/pi/image.jpeg”)

The capture() function has different parameters which we can pass for different operations like resize, format, use_video_port, etc.

E.g. Camera.capture(“/home/pi/image.jpeg”, resize=(720, 480))

resolution= (width,height)

It sets the resolution of camera at which image captures, video records and preview will display. The resolution can be specified as **(width, height)** tuple, as a string formatted **WIDTHxHEIGHT**, or as a string containing commonly recognised display resolution name e.g. “HD”, “VGA”, “1080p”, etc.

E.g.

```
Camera.resolution = (720, 480)

Camera.resolution = “720 x 480”

Camera.resolution = “720p”

Camera.resolution = “HD”
```

Assignment :13

Problem Statement : Implement GOMS (Goals, Operators, Methods and Selection rules) modeling technique to model user's behavior in given scenario.

Goals, Operators, Methods, Selection (GOMS)

Models for user-interface interactions must simulate both the user and the interface. Broadly there two types of models, generative and GOMS like models. Generative model, like EPIC or ACT-R, do not need a precise description of how the user will behave using the interface, rather they determine from a set of rules how the user will behave in general. (The rules are called production rules.) GOMS like models need a precise description of how the user will behave. Typically the description is determined from a Hierarchical Task Analysis. GOMS models assume that the user is an expert in the interface, meaning that the user does not make mistakes and does not have to search for the proper action to perform.

Models are useful for evaluating alternative design without usability testing. They are cheaper and quick to perform than usability test, but an expert is required to construct the model.

GOMS (Goals, Operators, Methods, Selection)

Card, Moran, and Newell developed GOMS in the 1980. They developed two versions:

CMN-GOMS: the original GOMS, “The Psychology of Human-Computer Interaction,” Lawrence Erlbaum Ass. Publishers, 1983. It is a top level task analysis. You can find a lecture GOMS-CMN in the old lecture.
KLM-GOMS: A key stroke level model, this lecture. This is the simplest GOMS.
Other GOMS models are:

Critical-Path Model GOMS (**CPM-GOMS**) or Cognitive Perceptual Motor GOMS (Yes the acronym means either name). Developed by Bonnie John and eliminates the restriction that actions are performed sequentially.
Natural GOMS language (**NGOMSL**) was developed by David Kieras and formalized the description of tasks and enable automated calculation of task execution time and learning time.

Keystroke-level Model GOMS (KLM-GOMS)

Card, Moran, and Newell (The Keystroke-level Model for User Performance with Interactive Systems, Communications of the ACM, 23:396-410, 1980) measured the time for users to perform a series of gestures on the computer. They discovered a fundamental principle:

The total time to perform a sequence of gestures is the sum on the individual gestures.

A lot is implied in this statement. The most important is that there are fundamental gestures. Individual users perform the fundamental gestures in different times; the researchers attempted to determine typical values:

K = 0.2 sec **Keying**: The time to perform a keystroke, or mouse click
P = 1.1 sec **Pointing**: The time to position the mouse pointer
H = 0.4 sec **Homing**: The time for user to move hands from keyboard to mouse
M = 1.35 sec **Mental**: The time for the user to prepare for the next step
R = ? **Responding**: The time for the computer to respond to the user inputs.
The variation of the timings across users can be as much as 100%, for example an expert typist can type 200 words per minute = 0.06 sec (Note that the measurement assumes 5 characters/words). So the model cannot accurately predicate the response time of an individual user. Chris Blazek and I have measured these variables for a web user and they are surprisingly accurate. Even without precise gesture times for a specific user, the model can be used to determine times for expert users and compare across interfaces.

We calculate the total response time by listing the individual gesture and summing their individual execution time. The difficult part is determining where a mental preparation, **M**, occurs. The researchers determined heuristics rules for placing mental operations:

Rule 0: *Initial insertion of candidate Ms:* Insert M before all Ks and Ps

Rule 1: *Deletion of anticipated Ms:* If P or K is fully anticipated by a preceding P or K then delete the middle M. For example moving the mouse to tap on the button; PMK => PK

Rule 2: *Deletion of Ms in cognitive units:* If a series of Ks represent a string then delete the middle Ks; for example type ‘1.2’ is a cognitive unit; MKMKMK => MKKK

Rule 3: *Deletion of Ms before consecutive terminators:* If several delimiters are typed only keep the first M. For example if ‘))’ is the terminator, use only one M.

Rule 4: *Deletion of Ms that are terminators of commands:* If the terminator is a frequently used, delete the M before the terminator; for example a command followed by “return,” so the M before the K representing the “return” is deleted. But if the terminator delimits arguments for a command string that vary then keep the M. This represents checking that the arguments are correct.

Rule 5: *Deletion of overlapped Ms:* Do not count any portion of an M that overlaps with a command response. (This is the reason that a responsive interface only needs to respond in a second.)

Assignment 14 :

Problem Statement : Design a User Interface in Python.

Creating a simple graphical user interface (GUI) that works across multiple platforms can be complicated. But it doesn't have to be that way. You can use Python and the PySimpleGUI package to create nice-looking user interfaces that you and your users will enjoy! PySimpleGUI is a new Python GUI library that has been gaining a lot of interest recently.

In this tutorial, you'll learn how to:

- **Install** the PySimpleGUI package
- Create basic **user interface** elements with PySimpleGUI
- Create applications, such as a PySimpleGUI **image viewer**
- Integrate PySimpleGUI with **Matplotlib**
- Use **computer vision** in PySimpleGUI
- Package your PySimpleGUI application for **Windows**

Getting Started With PySimpleGUI

PySimpleGUI was launched in 2018, so it's a relatively new package compared with the likes of wxPython or PyQt.

PySimpleGUI has four ports:

1. Tkinter
2. PyQt
3. wxPython
4. Remi

5. PySimpleGUI wraps portions of each of these other packages and makes them easier to use. However, each of the ports has to be installed separately.

6. PySimpleGUI wraps the entirety of Tkinter, which comes with Python. PySimpleGUI has wrapped most of PySide2, but only a small portion of wxPython. When you install PySimpleGUI, you get the **Tkinter** variant by default. For more information about Tkinter, check out Python GUI Programming With Tkinter.

7. Depending on which variant of PySimpleGUI you use, applications that you create with PySimpleGUI may not look native to their platform. But don't let this stop you from giving PySimpleGUI a try. PySimpleGUI is still quite powerful and can get most things done with a little work.

Installing PySimpleGUI

Installing PySimpleGUI is easy if you use pip. For the purposes of this tutorial, you'll learn how to install the regular PySimpleGUI port, which is the Tkinter variant.

Here's how to do it:

```
$ python -m pip install pysimplegui
```

This will install PySimpleGUI to whatever your system Python is set to. You can also install PySimpleGUI to a Python virtual environment. If you're unfamiliar with Python virtual environments, then you should read Python Virtual Environments: A Primer.

If you prefer to try the PyQt variant, then you can use `pip install PySimpleGUIQt` instead. Now that you have PySimpleGUI installed, it's time to find out how to use it!

Creating Basic UI Elements in PySimpleGUI

If you've ever used a GUI toolkit before, then you may have heard the term **widgets**. A widget is a generic term used to describe the elements that make up the user interface (UI), such as buttons, labels, windows, and more.

In PySimpleGUI, widgets are referred to as **elements**, which you may sometimes see capitalized elsewhere as **Elements**.

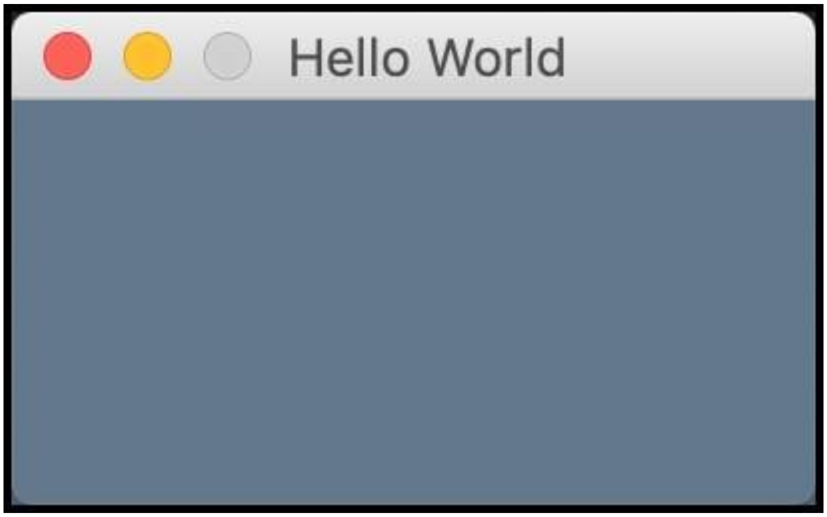
One of the basic building blocks of PySimpleGUI is the Window(). To create a Window(), you can do the following:

```
# hello_world.py

import PySimpleGUI as sg

sg.Window(title="Hello World", layout=[[]], margins=(100, 50)).read()
```

Window() takes lots of different arguments—too many to be listed here. However, for this example you can give the Window() a title and a layout and set the margins, which is how big the UI window will be in pixels.

read() returns any events that are triggered in the Window() as a string as well as a values dictionary. You'll learn more about these in later sections of this tutorial.

When you run this code, you should see something like this:



This example doesn't really do much of anything other than possibly displaying a message to the user.

Normally, you would have other elements besides a Window() in your application, so let's add some text and a button to the mix.

Create a new file named hello_psg.py and add this code:

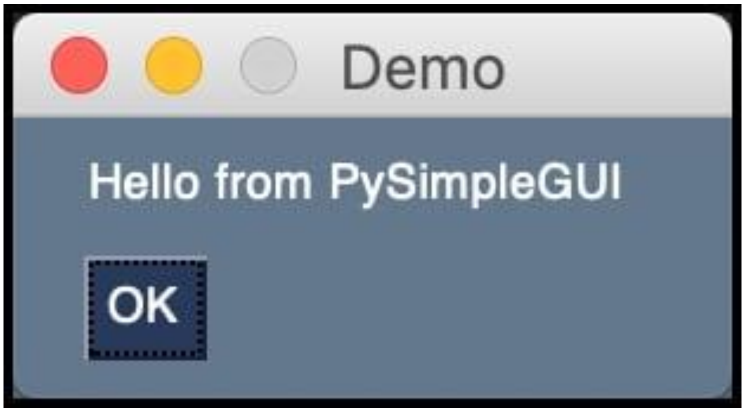
```
# hello_psg.py


import PySimpleGUI as sg


layout = [[sg.Text("Hello from PySimpleGUI")], [sg.Button("OK")]]


# Create the window
window = sg.Window("Demo", layout)


# Create an event loop
while True:
    event, values = window.read()
    # End program if user closes window or
    # presses the OK button
    if event == "OK" or event == sg.WIN_CLOSED:
        break


window.close()
```

Most GUI toolkits allow you to lay out the elements using **absolute positioning** or by allowing the GUI to lay them out **dynamically**. For example, wxPython uses Sizers to lay out elements dynamically. If you'd like to learn more about wxPython, then check out How to Build a Python GUI Application With wxPython.

PySimpleGUI uses nested Python lists to lay out its elements. In this case, you add a Text() element and a Button() element. Then you create the window and pass in your custom layout.

The last block of code is the **event loop**. A graphical user interface needs to run inside a loop and wait for the user to do something. For example, the user might need to press a button in your UI or type something with their keyboard. When they do that, those events are processed by the event loop.

When you use PySimpleGUI, you make an event loop by creating an infinite while loop that reads events from the window object. If the user presses the OK button or the Exit button, then you want the program to end. To accomplish that, you break out of the loop and close() the window.



Hello from PySimpleGUI

Conclusion : The GUI is ready. The code above creates an application that looks like this

Assignment No.: 15

Problem Statement:

Create Project Plan using Gantt chart

- Specify project name and start (or finish) date.
- Identify and define project tasks.
- Define duration for each project task.
- Define milestones in the plan
- Define dependency between tasks
- Define project calendar.
- Define project resources and specify resource type

Assign resources against each task and baseline the project plan

Objectives:

- Establish the initial project schedule
- Allocate resources
- Monitor and report progress

Theory: Gantt chart is a type of a bar chart that is used for illustrating project schedules. Gantt charts can be used in any projects that involve effort, resources, milestones and deliveries.

At present, Gantt charts have become the popular choice of project managers in every field. Gantt charts allow project managers to track the progress of the entire project. Through Gantt charts, the project manager can keep a track of the individual tasks as well as of the overall project progression.

In addition to tracking the progression of the tasks, Gantt charts can also be used for tracking the utilization of the resources in the project. These resources can be human resources as well as materials used.

Gantt chart was invented by a mechanical engineer named Henry Gantt in 1910. Since the invention, Gantt chart has come a long way. By today, it takes different forms from simple paper based charts to sophisticated software packages

The Use

As we have already discussed, Gantt charts are used for project management purposes. In order to use Gantt charts in a project, there are a few initial requirements fulfilled by the project.

First of all, the project should have a sufficiently detailed Work Breakdown Structure (WBS).

Secondly, the project should have identified its milestones and deliveries.

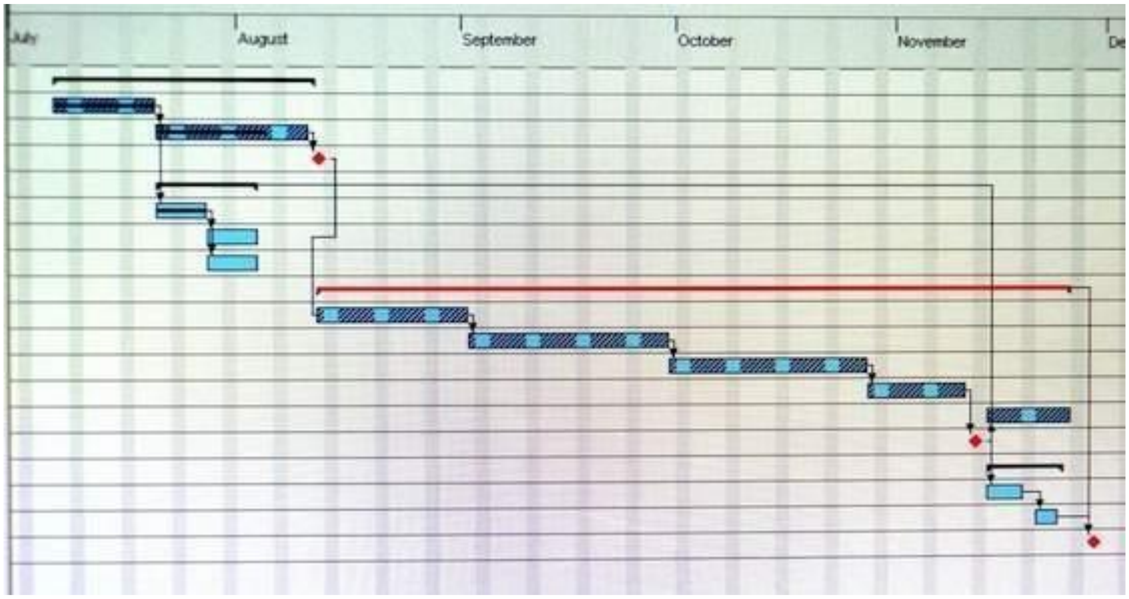
In some instances, project managers try to define the work break down structure while creating Gantt chart. This is one of the frequently practiced errors in using Gantt charts. Gantt charts are not designed to assist WBS process; rather Gantt charts are for task progress tracking.

Gantt charts can be successfully used in projects of any scale. When using Gantt charts for large projects, there can be an increased complexity when tracking the tasks.

This problem of complexity can be successfully overcome by using computer software packages designed for offering Gantt chart functionalities.

Tools Available

There are dozens of Gantt chart tools that can be used for successful project tracking. These tools usually vary by the feature offered.



This way, one may actually find the exact Gantt chart template (probably in Excel) required for the purpose. In case, if no match is found, then it is sensible to create one's own.

Excel is the most popular tool for creating custom Gantt charts. Of course, one can create a Gantt chart from scratch in Excel, but it is always advisable to use a Project Management add-on in Excel to create Gantt charts.

These project management add-ons are published by Microsoft and other third-party companies.

Advantages & Disadvantages:

The ability to grasp the overall status of a project and its tasks at once is the key advantage in using a Gantt chart tool. Therefore, upper management or the sponsors of the project can make informed decisions just by looking at the Gantt chart tool.

The software-based Gantt charts are able to show the task dependencies in a project schedule. This helps to identify and maintain the critical path of a project schedule.

Gantt chart tools can be used as the single entity for managing small projects. For small projects, no other documentation may be required; but for large projects, the Gantt chart tool should be supported by other means of documentation.

For large projects, the information displayed in Gantt charts may not be sufficient for decision making.

Although Gantt charts accurately represent the cost, time and scope aspects of a project, it does not elaborate on the project size or size of the work elements. Therefore, the magnitude of constraints and issues can be easily misunderstood.

Assignment No.: 16

Problem Statement:

Execute and Monitor Project Plan

- Update % Complete with current task status.
- Review the status of each task.
- Compare Planned vs Actual Status
- Review the status of Critical Path
- Review resources assignation status

Objectives:

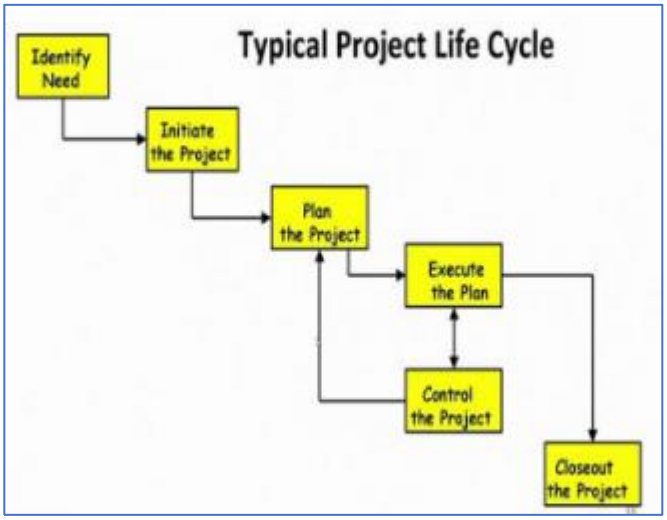
1. To learn various techniques, tools, applications in Software Project Management.
2. To understand efficient methods for planning the projects
3. To monitor and executing the projects with plan.

Theory:

Software project management is dedicated to the planning, scheduling, resource allocation, execution, tracking, and delivery of software and web projects.

The project management processes consist of initiating the project, here the project manager writes the business case and the project charter. The project manager also carries out the planning processes that is completes the work breakdown structure and performs the project scheduled cost estimation and so on. And then the project manager carries out the executing processes, which basically performing the necessary action to complete the work as outlined in the plan, but then during the execution process there may be deviations from the plan.

And the project manager needs to carry out the monitoring and control processes where the project manager checks, whether the project is proceeding as per the plan or there are deviations from the plan and when there are deviations the project manager takes corrective actions to match the progress of the project with the plan. And finally, the project manager needs to carry out the closing processes, where the closing documents are created and the customer finally gives the formal acceptance of the project.



The project manager has three main objectives during the execution phase:

- Managing people
- Managing processes
- Managing communication

The benefits of a well-executed project are threefold:

- The project can be completed on time and budget
- Team morale can be maintained
- Stakeholders are satisfied with overall project progress

The Execution & Monitoring phase produced:

1. Project deliverables are the tangible outputs of the project. They need to be reviewed, tested, and meet the acceptance criteria given by the clients.

2. Change Request: When client expectations change or there’s a disconnect between the team’s understanding of the client’s requirements, scope changes happen.
3. The execution stage produces a lot of data points that you can use to optimize your team’s performance.
4. Issue Log: Whenever there are bugs, issues, or defects, you document them in the issue log.
5. Documentation updates: Any changes to the documents created during the planning phase like the project scope and project schedule will be documented.