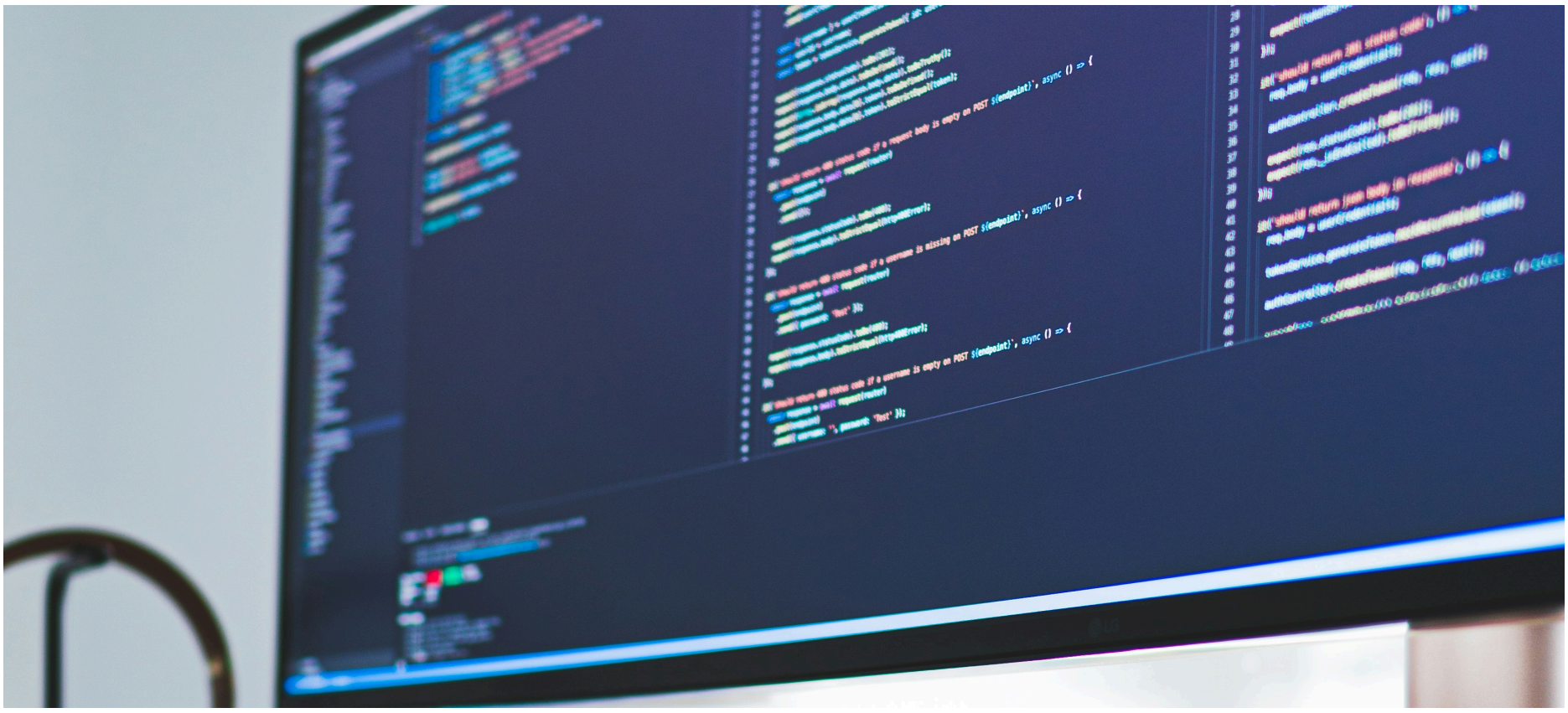

Phase 2

Components

Physical (hardware)

- Main Memory : 300x4 bytes
- Virtual Memory : 100x4 bytes
- Buffer : 40 bytes
- Instruction Register [IR] : 4 bytes
- Instruction Counter [IC] : 2 bytes
- General Purpose Register [R] : 4 bytes
- Toggle Register [C] : 1 byte
- Page Table Register [PTR] : 4 bytes

Logical

- Process Control Block [PCB] : a data structure that stores information about a process like id, time, etc.
- Total Time Counter [TLC] : a counter we run to check if Time Limit is exceeded or not.
- Line Limit Counter [LLC] : a counter we run to check if Line Limit is exceeded or not.
- System Interrupt [SI]
  - SI=1 → read() [GD]
  - SI=2 → write() [PD]
  - SI=3 → terminate() [H]
- Program Interrupt [PI]
  - PI=1 → Opcode Error
  - PI=2 → Operand Error
  - PI=3 → Page Fault
- Time Interrupt [TI]
  - TI=2 → Time Limit Exceeded

Key Concepts

Paging

It is a non contiguous memory management scheme.

💡 **1 frame = 10 lines of memory**

Our program is divided into logical pages

1 page = 10 instructions max, or 40 bytes max

1 instruction takes up 1 line in the memory

Our main memory has 300 lines, labelled 0 - 299. We will have 30 frames, labelled 0 - 29

Instead of storing instructions from address 0 of main memory, we randomly select a page frame and store the page in that frame. Example : our random frame number generator function gives us 3, so we store our page from address 30 - 39 in the main memory.

💡 **Now how do we know which page is in which frame?**

Page Table stores this information.

Page Table itself is stored in a frame in the main memory. How to decide where to store the page table? Our random frame number generator function gives us the

Address Translation

The CPU will always generate logical or virtual address. We have to convert this virtual address into the real address, this conversion is known as Address Translation.

$$PTE = PTR + VA/10$$

$$RA = PTE * 10 + VAmod(10)$$

PTE gives us the frame number to visit.

RA is the real address.

Error Handling

In this phase we handle the following errors.

1. Opcode Error
   - Using any instruction other than specified 7. Example : **BD30**
2. Operand Error
   - Invalid Operands provided. Example : **GDA8**
3. Time Limit Exceeded
   - Time taken by job to execute exceeds Total Time Limit.
4. Line Limit Exceeded
   - Number of line printed to the output file exceeds Total Line Limit.
5. Out of Data
   - Trying to fetch data that does not exist in the input file.
6. Page Fault
   - Looking for an entry in the Page Table that does not exist. The page we are looking for does not exist in the main memory.
   - Types of Page Faults
     - Valid : Caused by **GD** and **SR**. We handle this fault by updating the Page Table and adding those pages to the main memory.
     - Invalid : Caused by remaining instructions apart from **GD** and **SR**. We do not intervene.

Description of each function

allocate()

- Returns a unique random number from 0 to 29.
- An `ArrayList` is filled with numbers from 0 to 29, shuffled, and we withdraw an element one by one.

load()

- Reads the input file line by line
- Calls `initialize()` upon encountering **$AMJ,** as it signals the beginning of a new job.
- Calls `allocate()` to get a frame number to store the instructions in.
- Updates the Page Table after storing the instructions in the main memory.
- **Loads the instructions from the input file to the main memory.**
- Calls `executeProgram()` upon encountering **$DTA,** as it means that all the instructions that were present between $AMJ and $DTA are stored in the main memory.
- Calls `displayMemoryContents()` upon encountering **$END**.

initialize()

- Initializes SI, IC to 0.
- Initializes the whole memory array, and registers to `_` (underscore) to depict empty memory space.
- Calls `allocate()` to get a random frame to assign to Page Table.
- Updates the Page Table Register.

executeProgram()

- Calls `addressTranslate()` to convert the value of IC to the corresponding real address.
- Loads the instruction present at the calculated real address into the IR.
- Calls `masterMode()` after setting **SI=1**, if it is **GD**.
- Calls `masterMode()` after setting **SI=2**, if it is **PD**.
- Calls `masterMode()` after setting **SI=3**, if it is **H**.
- Loads the R register with the content of memory location specified along with **LR**.
- Stores the value in R register to the memory location specifies along with **SR**.
- Compares the content present in the R register and at the memory location specified along with **CR**. Sets the toggle register if true.
- Updates the value of IC with the address given with **BT,** if the toggle register was set.
- If none of the above instructions match, calls `masterMode()` after setting PI=1.
- Increments IC to fetch the next instruction.
- Increments Time Limit Counter.

addressTranslate()

- Converts the String Virtual Address to Integer Address, calls `masterMode()` after setting PI=2, if there was an error in converting String to Integer.
- Calculate the Page Table Entry using the integer virtual address. If no entry found, calls `masterMode()` after setting PI=3.
- If a valid entry is found or the page fault was valid and handled, it returns the real address corresponding the the given virtual address.

masterMode()

- Calls `read()` if SI=1 & TI=0.
- Calls `write()` if SI=2 & TI=0.

- Calls `terminate()` if SI=3 & TI=0, with message **Process exited with status 0**.
- Calls `terminate()` if SI=1 & TI=2, with error message **Time Limit Exceeded**.
- Calls `write()` first, then `terminate()` if SI=2 & TI=2, with error message **Time Limit Exceeded**.
- Calls `terminate()` if SI=3 & TI=2, with message **Process exited with status 0**.
- Calls `terminate()` if PI=1 & TI=0, with message **Opcode Error**.
- Calls `terminate()` if PI=2 & TI=0, with message **Operand Error**.
- Calls `terminate()` if PI=3 & TI=0, with message **Invalid Page Fault**, if the page fault is not caused by GD or SR.
- Calls `terminate()` if PI=1 & TI=2, with message **Time Limit Exceeded, Opcode Error**.
- Calls `terminate()` if PI=2 & TI=2, with message **Time Limit Exceeded, Operand Error.**
- Calls `terminate()` if PI=3 & TI=2, with message **Time Limit Exceeded, Invalid Page Fault**.
- Handles Valid Page Faults. Checks whether the page fault is caused by GD or SR, if it is caused by either of those, `allocate()` is called to get a random frame number.

read()

- Reads the data card from the input file and stores it at the location specified with the instruction GD.
- If the data read begins with **$END**, calls `terminate()` with error message **Out of Data Error.**

write()

- Increments Line Limit Counter.
- Checks if Line Limit Counter is greater than Total Line Limit, if yes, calls `terminate()` with message **Line Limit Exceeded**.
- If Line Limit Counter is not greater than Total Line Limit, writes the data from the memory location specified with the instruction PD, to the output file.

terminate()

- Print the appropriate error message.
- Prints two blank lines in the output file.

displayMemoryContents()

- Displays the content of our main memory.

Jobs

Jobs without errors

1. **Print “Hello World”**

   $AMJ000900040001
   GD10PD10H
   $DTA
   Hello World
   $END0009

2. **Check whether two strings are same or not**

   $AMJ000800150003
   GD20GD30GD40GD50LR20CR30BT09PD50HPD40
   H
   $DTA
   VIT
   VIIT
   SAME
   NOT SAME
   $END0008

3. **Diamond Pattern**

   $AMJ000300250006
   GD30LR30SR46SR54SR55SR56SR63SR64SR65SR66

4. **Replacing one word with other given word**

   $AMJ000200480002
   GD60GD70GD80LR70CR60BT18CR61BT23CR62BT28

PD40PD50PD60PD50PD40H
$DTA
*
$END0003

5. **Hollow Square pattern**

$AMJ000400200004
GD30LR30SR40SR41SR42SR43SR50SR55PD40PD50
PD50PD40H
$DTA
*
$END0004

6. **Print all non zero elements in the 2x2 matrix**

$AMJ000500330002
GD30GD40GD50LR30CR40BT09LR40SR60LR30CR41
BT14LR41SR61LR30CR50BT19LR50SR70LR30CR51
BT23LR51SR71PD60PD70H
$DTA
0
1 0
2 4
$END0005

7. **Print all the occurrences of a given word**

$AMJ000100280005
GD40GD50GD60GD70GD80GD90LR40CR50BT17CR60
BT20CR70BT23CR80BT26PD90HPD50CR40BT09
PD60CR40BT11PD70CR40BT13PD80H
$DTA
Semester
Word1
Word2
Semester
Semester
No Match Found
$END0001

Jobs with errors

1. **Print “Hello World”**

$AMJ000900010001
GD10PD10H
$DTA
Hello World
$END0009

Time Limit Exceeded

2. **Check whether two strings are same or not**

$AMJ000800150003
GD20GD30GD40GD50LR20CR30BT09
PD60HPD40
H
$DTA
VIT
VIIT
SAME
NOT SAME
$END0008

Invalid Page Fault

3. **Diamond Pattern**

$AMJ000300250006

GDA0LR30SR46SR54SR55SR56SR63SR64SR65SR66

4. **Replacing one word with other given word**

$AMJ000200000002
GD60GD70GD80LR70CR60BT18CR61BT23CR62BT28
CR63BT33CR64BT38CR65BT43PD60HLR80SR60

PD40PD50PD60PD50PD40H
$DTA
$END0003

Operand Error

5. **Hollow Square pattern**

$AMJ000400200002
GD30LR30SR40SR41SR42SR43SR50SR55PD40PD50
PD50PD40H
$DTA
*
$END0004

Line Limit Exceeded

CR80LR70BT06LR80SR61CR80LR70BT08LR80SR62
CR80LR70BT10LR80SR63CR80LR70BT12LR80SR64
CR80LR70BT14LR80SR65CR80LR70PD60H
$DTA
A apple
A
An
$END0002

Time Limit Exceeded

6. **Print all non zero elements in the 2x2 matrix**

$AMJ000500330002

BD30GD40GD50LR30CR40BT09LR40SR60LR30CR41
BT14LR41SR61LR30CR50BT19LR50SR70LR30CR51
BT23LR51SR71PD60PD70H
$DTA
0
1 0
2 4
$END0005

Opcode Error

7. **Print all the given occurrences of a word**

$AMJ000100280005
GD40GD50GD60GD70GD80
GD90LR40CR50BT17CR60
BT20CR70BT23CR80BT26PD90HPD50CR40BT09
PD60CR40BT11PD70CR40BT13PD80H
$DTA
Semester
Word1
Word2
Semester
Semester
$END0001

Out of Data Error

Source Code

```
import java.io.*;
import java.util.*;

public class phase2{

    public static char[][] memory=new char[300][4];
    public static char[][] virtualMemory=new char[100][4];
    public static int IC;
    public static char[] R=new char[4];
    public static char[] IR=new char[4];
    public static char[] PTR=new char[4];
    public static int SI;
    public static int TI;
    public static int PI;
```

```
	public static int C;
	public static char[] buffer=new char[40];
	public static int numberOfInstructions;
	public static int TLC;
	public static int TTL;
	public static int LLC;
	public static int TLL;
	public static int calledByRead;
	public static int calledBySR;
	public static int frameAssignedDuringValid;
	public static ArrayList<Integer> frameNumbers=new ArrayList<Integer>();
	public static int frameIndex;
	public static int pageNumber;
	public static int errorDetected;
	public static int pageTableLoc;
	public static int pageTableAT;
	public static int errorLineLimit;
	public static int errorOutOfData;

	static PCB pro=new PCB();
	public static void main(String[] args) throws Exception{
		int i;
		load();
	}

	public static void displayMemoryContents(){
		int i;
		System.out.println();
		for(i=0; i<300; i++){
			if(i%10==0 && i!=0)
				System.out.println("------------------------------------------------------------------");
			System.out.println(i+"  "+memory[i][0]+memory[i][1]+memory[i][2]+memory[i][3]);
		}
		System.out.println();
		System.out.println();
		System.out.println("===========PCB==========");
		System.out.println("Process ID : "+pro.processID);
		System.out.println("Total Time Limit : "+pro.totalTimeLimit);
		System.out.println("Total Line Limit : "+pro.totalLineLimit);
		System.out.println("Time Given : "+pro.completionTime);
		System.out.println("Lines Printed : "+pro.linesPrinted);
		System.out.println("Current Status : "+pro.currentStatus);
		System.out.println("\n\n\n\n\n\n");
	}

	public static void load() throws Exception{
		String line;
		int i;
		pageNumber=49;

		String subline;
		int linePointer=0;
		int insCounter=0;
		int frameToFill;
		File f=new File("withErrors.txt");
		Scanner sc=new Scanner(f);
		while(sc.hasNextLine()){
			line=sc.nextLine();
```

```
                line=line.trim();
                System.out.println(line);
                if(line.length()>4)
                        subline=line.substring(0,4);
                else
                        subline=line;
                if(subline.equals("$AMJ")){
                        linePointer=0;
                        insCounter=0;
                        pro.processID=Integer.parseInt(line.substring(4,8));
                        TTL=Integer.parseInt(line.substring(8,12));
                        pro.totalTimeLimit=TTL;
                        TLL=Integer.parseInt(line.substring(12));
                        pro.totalLineLimit=TLL;
                        initialize();
                }
                else if(subline.equals("$DTA")){
                        pro.currentStatus="EXECUTE";
                        executeProgram(sc);
                }
                else if(subline.equals("$END")){
                        pro.currentStatus="FINISHED";
                        displayMemoryContents();
                        continue;
                }
                else{
                        pro.currentStatus="FETCH";
                        insCounter=0;
                        line=line.trim();
                        char[] instructions=line.toCharArray();
                        for(i=0; i<instructions.length; i++)
                                buffer[i]=instructions[i];

                        frameToFill=allocate();
                        String idk=Integer.toString(frameToFill);
                        if(frameToFill<10)
                                idk="0"+idk;
                        char[] frameAsChar=idk.toCharArray();
                        memory[pageTableLoc][0]='P';
                        memory[pageTableLoc][1]=(char)pageNumber;
                pageNumber++;
                memory[pageTableLoc][2]=frameAsChar[0];
                memory[pageTableLoc][3]=frameAsChar[1];
                pageTableLoc++;
                linePointer=frameToFill*10;
                for(i=0; i<instructions.length; i++){
                        if(insCounter!=0 && insCounter%4==0){
                                linePointer++;
                                numberOfInstructions++;
                        }
                        if(instructions[i]=='H'){
                                memory[linePointer][insCounter%4]=buffer[i];
                                numberOfInstructions++;
                                insCounter=((insCounter/4)+1)*4;
                        }
                        else{
                                memory[linePointer][insCounter%4]=buffer[i];
                                insCounter++;
```

```
                        if(i==(instructions.length-1))
                            numberOfInstructions++;
                    }
                }
                for(i=0; i<40; i++)
                    buffer[i]='_';
            }
        }
    }

    public static int allocate(){
        int n=frameNumbers.get(frameIndex);
        frameIndex++;
        return n;
    }

    public static void initialize() throws Exception{
        int i;
        IC=0;
        SI=0;
        TI=0;
        PI=0;
        TLC=0;
        LLC=0;
        pageNumber=49;
        errorDetected=0;
        numberOfInstructions=0;
        errorLineLimit=0;
        errorOutOfData=0;
        C=0;
        calledByRead=0;
        calledBySR=0;
        frameAssignedDuringValid=0;
        R[0]='_';
        R[1]='_';
        R[2]='_';
        R[3]='_';
        IR[0]='_';
        IR[1]='_';
        IR[2]='_';
        IR[3]='_';
        frameIndex=0;


        frameNumbers.clear();

        for(i=0; i<30; i++)
            frameNumbers.add(i);
        Collections.shuffle(frameNumbers);
        int temp=allocate();
        pageTableLoc=temp*10;
        String frame=Integer.toString(temp);
        if(temp<10)
            frame="00"+frame+"0";
        else
            frame="0"+frame+"0";

        char[] r=frame.toCharArray();
```

```
        PTR[0]=r[0];
        PTR[1]=r[1];
        PTR[2]=r[2];
        PTR[3]=r[3];
        System.out.println("PTR : "+PTR[0]+PTR[1]+PTR[2]+PTR[3]);

        for(i=0; i<300; i++){
                memory[i][0]='_';
                memory[i][1]='_';
                memory[i][2]='_';
                memory[i][3]='_';
        }
        for(i=0; i<100; i++){
                virtualMemory[i][0]='_';
                virtualMemory[i][1]='_';
                virtualMemory[i][2]='_';
                virtualMemory[i][3]='_';
        }
        for(i=0; i<40; i++)
                buffer[i]='_';
        File of=new File("output.txt");
        if(of.createNewFile())
                System.out.println("Created an output file : output.txt");
    }

    public static int addressTranslate(String va, Scanner sc) throws Exception{
        int virtualAddress=0;
        try{
                virtualAddress=Integer.parseInt(va);
        }
        catch(Exception e){
                PI=2;
                masterMode(sc);
        }

        String pageTable=""+PTR[0]+PTR[1]+PTR[2]+PTR[3];
        int ptrValue=Integer.parseInt(pageTable);
        int pte=ptrValue+(virtualAddress/10);
        pageTableAT=pte;
        if(memory[pte][0]!='P'){
                PI=3;
                masterMode(sc);
                if(PI!=3){
                        int realAddress=frameAssignedDuringValid*10;
                        realAddress=realAddress+virtualAddress%10;
                        return realAddress;
                }
                else
                        return -1;
        }
        else{
                String temp=""+memory[pte][2]+memory[pte][3];
                int frameNo=Integer.parseInt(temp);
                int realAddress=(frameNo*10)+(virtualAddress%10);
                return realAddress;
        }
    }
```

```
public static void executeProgram(Scanner sc) throws Exception{
	IC=0;
	int i,address;
	int realAddress;
	String temporary;
	while(IC!=numberOfInstructions){
		pro.completionTime=TLC;
		if(TLC>=TTL)
			TI=2;
		System.out.println("\n\nValue of IC : "+IC);
		temporary="";
		address=0;
		String temporary2=""+IC;
		realAddress=addressTranslate(temporary2,sc);
		System.out.println("Real Address corresponding to IC "+IC+" is : "+realAddress);

		for(i=0; i<4; i++)
		IR[i]=memory[realAddress][i];
	if(IR[0]=='G' && IR[1]=='D'){
		System.out.println(""+IR[0]+IR[1]+IR[2]+IR[3]);
		IC++;

		IR[3]='0';
		SI=1;
		masterMode(sc);
		TLC++;
	}
	else if(IR[0]=='P' && IR[1]=='D'){
		System.out.println(""+IR[0]+IR[1]+IR[2]+IR[3]);
		IC++;
		IR[3]='0';
		SI=2;
		masterMode(sc);
		TLC++;
	}
	else if(IR[0]=='H' && IR[1]=='_'){
		System.out.println(""+IR[0]+IR[1]+IR[2]+IR[3]);
		IC++;
		SI=3;
		TLC++;
		masterMode(sc);
		break;
	}
	else if(IR[0]=='L' && IR[1]=='R'){
		System.out.println(""+IR[0]+IR[1]+IR[2]+IR[3]);
		IC++;
		temporary=""+IR[2]+IR[3];
		address=addressTranslate(temporary,sc);
		for(i=0; i<4; i++)
			R[i]=memory[address][i];
		TLC++;
	}
	else if(IR[0]=='S' && IR[1]=='R'){
		System.out.println(""+IR[0]+IR[1]+IR[2]+IR[3]);
		IC++;
		temporary=""+IR[2]+IR[3];
		calledBySR=1;
```

```
                address=addressTranslate(temporary,sc);
                for(i=0; i<4; i++)
                        memory[address][i]=R[i];
                TLC++;
        }
        else if(IR[0]=='C' && IR[1]=='R'){
                System.out.println(""+IR[0]+IR[1]+IR[2]+IR[3]);
                IC++;
                temporary=""+IR[2]+IR[3];
                address=addressTranslate(temporary,sc);
                for(i=0; i<4; i++){
                        if(R[i]==memory[address][i]){
                                C=1;
                        }
                        else{
                                C=0;
                                break;
                        }
                }
                TLC++;
        }
        else if(IR[0]=='B' && IR[1]=='T'){
                System.out.println(""+IR[0]+IR[1]+IR[2]+IR[3]);
                if(C==1){
                        temporary=""+IR[2]+IR[3];
                        address=Integer.parseInt(temporary);
                        IC=address;
                }
                else{
                        IC++;
                }
                TLC++;
        }
        else{
                System.out.println(""+IR[0]+IR[1]+IR[2]+IR[3]);
                System.out.println("No opcode matches");

                PI=1;
                masterMode(sc);
        }
}


public static void masterMode(Scanner sc) throws Exception{
        System.out.println("Entered masterMode()");
        System.out.println("SI : "+SI);
        System.out.println("TI : "+TI);
        System.out.println("PI : "+PI);
        if(TLC>=TTL)
                TI=2;
        if(TI==0 && SI==1){
                SI=0;
                read(sc);
        }
        else if(TI==0 && SI==2){
                SI=0;
                write(sc);
```

```
        }
        else if(TI==0 && SI==3){
            terminate("Process exited with status 0",sc);
        }

        else if(TI==2 && SI==1){
            terminate("Time Limit Exceeded",sc);
        }
        else if(TI==2 && SI==2){
            write(sc);
            terminate("Time Limit Exceeded",sc);
        }
        else if(TI==2 && SI==3){
            terminate("Process exited with status 0",sc);
        }



        else if(TI==0 && PI==1){
            terminate("Opcode Error",sc);
        }
        else if(TI==0 && PI==2){
            terminate("Operand Error",sc);
        }
        else if(TI==0 && PI==3){
            if(calledByRead==1){
                calledByRead=0;
                int frameToFill=allocate();
                System.out.println("Frame assigned while handling valid page fault caused by GD : "+frameToFill);
                String idk=Integer.toString(frameToFill);
                if(frameToFill<10)
                    idk="0"+idk;
                char[] frameAsChar=idk.toCharArray();
                memory[pageTableAT][0]='P';
                memory[pageTableAT][1]=(char)pageNumber;
                pageNumber++;
                memory[pageTableAT][2]=frameAsChar[0];
                memory[pageTableAT][3]=frameAsChar[1];
                frameAssignedDuringValid=frameToFill;
                TLC++;
                PI=0;
            }
            else if(calledBySR==1){
                calledBySR=0;
                int frameToFill=allocate();
                System.out.println("Frame assigned while handling valid page fault caused by SR : "+frameToFill);
                String idk=Integer.toString(frameToFill);
                if(frameToFill<10)
                    idk="0"+idk;
                char[] frameAsChar=idk.toCharArray();
                memory[pageTableAT][0]='P';
                memory[pageTableAT][1]=(char)pageNumber;
                pageNumber++;
                memory[pageTableAT][2]=frameAsChar[0];
                memory[pageTableAT][3]=frameAsChar[1];
                frameAssignedDuringValid=frameToFill;
                TLC++;
                PI=0;
```

```
                }
                else{
                        terminate("Invalid Page Fault",sc);
                }
        }

        else if(TI==2 && PI==1){
                terminate("Time Limit Exceeded\nOpcode Error",sc);
        }
        else if(TI==2 && PI==2){
                terminate("Time Limit Exceeded\nOperand Error",sc);
        }
        else if(TI==2 && PI==3){
                terminate("Time Limit Exceeded\nInvalid Page Fault",sc);
        }
}

public static void read(Scanner sc) throws Exception{
String data=sc.nextLine();
data=data.trim();
System.out.println("Data read : "+data);
String subdata;
if(data.length()>4)
        subdata=data.substring(0,4);
else
        subdata=data;
if(subdata.equals("$END")){
        errorOutOfData=1;
        terminate("Out of data",sc);
}
else{
        char[] dataChar=data.toCharArray();
        int length=dataChar.length;
        String temp2=""+IR[2]+IR[3];
        calledByRead=1;
        int block=addressTranslate(temp2,sc);
        int counterForArray=0;
        int counterForMemory=0;
        while(counterForArray!=length){
                if(dataChar[counterForArray]==' ' && counterForMemory!=0){
                        block++;
                        counterForArray++;
                        counterForMemory=((counterForMemory/4)+1)*4;
                }
                else if(counterForMemory%4==0 && counterForMemory!=0){
                        block++;
                }
                memory[block][counterForMemory%4]=dataChar[counterForArray];
                counterForArray++;
                counterForMemory++;
            }
        }
}

public static void write(Scanner sc) throws Exception{
        LLC++;
        pro.linesPrinted=LLC;
        if(LLC>TLL){
```

```
                errorLineLimit=1;
                terminate("Line Limit Exceeded",sc);
            }
            else{
                String temp=""+IR[2]+IR[3];
                int block=addressTranslate(temp,sc);
                if(block!=-1 && errorOutOfData!=1){
                    System.out.println("Writing from block : "+block+" to output file");
                    int i,j;
                    String dataToWrite="";
                    for(i=block; i<block+10; i++){
                        for(j=0; j<4; j++){
                            if(memory[i][j]!='_'){
                                dataToWrite=dataToWrite+memory[i][j];
                            }
                        }
                        if(memory[i][j-1]=='_')
                            dataToWrite=dataToWrite+" ";
                        }
                        FileWriter fw=new FileWriter("output.txt",true);
                        dataToWrite=dataToWrite+"\n";
                        fw.write(dataToWrite);
                        fw.close();
                    }
                }
            }

            public static void terminate(String message, Scanner sc) throws Exception{
                FileWriter fw=new FileWriter("output.txt",true);
                String temp="SI : "+SI+"  PI : "+PI+"  TI : "+TI;
                if(errorOutOfData==1){
                    fw.write(message+"\n"+temp+"\n\n\n");
                    IC=numberOfInstructions;
                }
                else if(errorLineLimit==1 || PI!=0 || TI==2){
                    fw.write(message+"\n"+temp+"\n\n\n");
                    String card;
                    String subcard;
                    do{
                        card=sc.nextLine();
                        if(card.length()>4)
                            subcard=card.substring(0,4);
                        else
                            subcard=card;
                        IC=numberOfInstructions;
                    }while(!subcard.equals("$END"));
                }
                else
                    fw.write(message+"\n"+temp+"\n\n\n");
                fw.close();
                System.out.println("==========PCB==========");
                System.out.println("Process ID : "+pro.processID);
                System.out.println("Total Time Limit : "+pro.totalTimeLimit);
                System.out.println("Total Line Limit : "+pro.totalLineLimit);
                System.out.println("Time Given : "+pro.completionTime);
                System.out.println("Lines Printed : "+pro.linesPrinted);
                System.out.println("Current Status : "+pro.currentStatus);
```

```
            System.out.println("\n\n\n\n\n\n");
        }
```

Testing

```
Hello World
Process exited with status 0
SI : 3  PI : 0  TI : 2


NOT SAME
Process exited with status 0
SI : 3  PI : 0  TI : 0


     *
    * * *
   * * * *
    * * *
     *
Process exited with status 0
SI : 3  PI : 0  TI : 0


An apple
Process exited with status 0
SI : 3  PI : 0  TI : 0


* * * *
*     *
*     *
* * * *
Process exited with status 0
SI : 3  PI : 0  TI : 0


1
2 4
Process exited with status 0
SI : 3  PI : 0  TI : 0


Semester
Semester
Process exited with status 0
SI : 3  PI : 0  TI : 0


~
```

Output file generated when jobs are error free.

```
Hello World
Time Limit Exceeded
SI : 2  PI : 0  TI : 2


Invalid Page Fault
SI : 0  PI : 3  TI : 0


Operand Error
SI : 0  PI : 2  TI : 0


Time Limit Exceeded
SI : 1  PI : 0  TI : 2


* * * *
*     *
Line Limit Exceeded
SI : 0  PI : 0  TI : 0


Opcode Error
SI : 0  PI : 1  TI : 0


Out of data
SI : 0  PI : 0  TI : 0


~
```

Output file generated when jobs have errors.

```
abhi@LAPTOP-63MDO3JQ: ~,
abhi@LAPTOP-63MDO3JQ:~/java_progs/phase2$ java -jar noErrors.jar
$AMJ000900040001
PTR : 0190
GD10PD10H
$DTA


Value of IC : 0
Real Address corresponding to IC 0 is : 230
GD10
Entered masterMode()
SI : 1
TI : 0
PI : 0
Data read : Hello World
Entered masterMode()
SI : 0
TI : 0
PI : 3
Frame assigned while handling valid page fault caused by GD : 27


Value of IC : 1
Real Address corresponding to IC 1 is : 231
PD10
Entered masterMode()
SI : 2
TI : 0
PI : 0
Writing from block : 270 to output file


Value of IC : 2
Real Address corresponding to IC 2 is : 232
H___
Entered masterMode()
SI : 3
TI : 0
PI : 0
==========PCB==========
Process ID : 9
Total Time Limit : 4
Total Line Limit : 1
Time Given : 3
Lines Printed : 1
Current Status : EXECUTE




$AMJ000800150003
PTR : 0260
GD20GD30GD40GD50LR20CR30BT09PD50HPD40
H
```

```
abhi@LAPTOP-63MDO3JQ: ~,
$AMJ000800150003
PTR : 0260
GD20GD30GD40GD50LR20CR30BT09PD50HPD40
H
$DTA


Value of IC : 0
Real Address corresponding to IC 0 is : 290
GD20
Entered masterMode()
SI : 1
TI : 0
PI : 0
Data read : VIT
Entered masterMode()
SI : 0
TI : 0
PI : 3
Frame assigned while handling valid page fault caused by GD : 28


Value of IC : 1
Real Address corresponding to IC 1 is : 291
GD30
Entered masterMode()
SI : 1
TI : 0
PI : 0
Data read : VIIT
Entered masterMode()
SI : 0
TI : 0
PI : 3
Frame assigned while handling valid page fault caused by GD : 11


Value of IC : 2
Real Address corresponding to IC 2 is : 292
GD40
Entered masterMode()
SI : 1
TI : 0
PI : 0
Data read : SAME
Entered masterMode()
SI : 0
TI : 0
PI : 3
Frame assigned while handling valid page fault caused by GD : 7


Value of IC : 3
Real Address corresponding to IC 3 is : 293
GD50
Entered masterMode()
SI : 1
```

```
abhi@LAPTOP-63MDO3JQ: ~,
Value of IC : 3
Real Address corresponding to IC 3 is : 293
GD50
Entered masterMode()
SI : 1
TI : 0
PI : 0
Data read : NOT SAME
Entered masterMode()
SI : 0
TI : 0
PI : 3
Frame assigned while handling valid page fault caused by GD : 5


Value of IC : 4
Real Address corresponding to IC 4 is : 294
LR20


Value of IC : 5
Real Address corresponding to IC 5 is : 295
CR30


Value of IC : 6
Real Address corresponding to IC 6 is : 296
BT09


Value of IC : 7
Real Address corresponding to IC 7 is : 297
PD50
Entered masterMode()
SI : 2
TI : 0
PI : 0
Writing from block : 50 to output file


Value of IC : 8
Real Address corresponding to IC 8 is : 298
H___
Entered masterMode()
SI : 3
TI : 0
PI : 0
==========PCB==========
Process ID : 8
Total Time Limit : 15
Total Line Limit : 3
Time Given : 12
Lines Printed : 1
Current Status : EXECUTE
```

```
----------------------------------------------------------------
50  NOT_
51  SAME
52  ____
53  ____
54  ____
55  ____
56  ____
57  ____
58  ____
59  ____
----------------------------------------------------------------
60  H___
61  ____
62  ____
63  ____
64  ____
65  ____
66  ____
67  ____
68  ____
69  ____
----------------------------------------------------------------
70  SAME
71  ____
72  ____
73  ____
74  ____
75  ____
76  ____
77  ____
78  ____
79  ____
----------------------------------------------------------------
80  ____
81  ____
82  ____
83  ____
84  ____
85  ____
86  ____
87  ____
88  ____
89  ____
----------------------------------------------------------------
90  ____
91  ____
92  ____
93  ____
94  ____
95  ____
96  ____
97  ____
98  ____
99  ____
----------------------------------------------------------------
100  ____
```

```
260  P129
261  P206
262  P328
263  P411
264  P507
265  P605
266  ____
267  ____
268  ____
269  ____
------------------------------------------------------------------
270  ____
271  ____
272  ____
273  ____
274  ____
275  ____
276  ____
277  ____
278  ____
279  ____
------------------------------------------------------------------
280  VIT_
281  ____
282  ____
283  ____
284  ____
285  ____
286  ____
287  ____
288  ____
289  ____
------------------------------------------------------------------
290  GD20
291  GD30
292  GD40
293  GD50
294  LR20
295  CR30
296  BT09
297  PD50
298  H___
299  PD40


==========PCB==========
Process ID : 8
Total Time Limit : 15
Total Line Limit : 3
Time Given : 12
Lines Printed : 1
Current Status : FINISHED
```

References

https://stackoverflow.com/questions/8115722/generating-unique-random-numbers-in-java