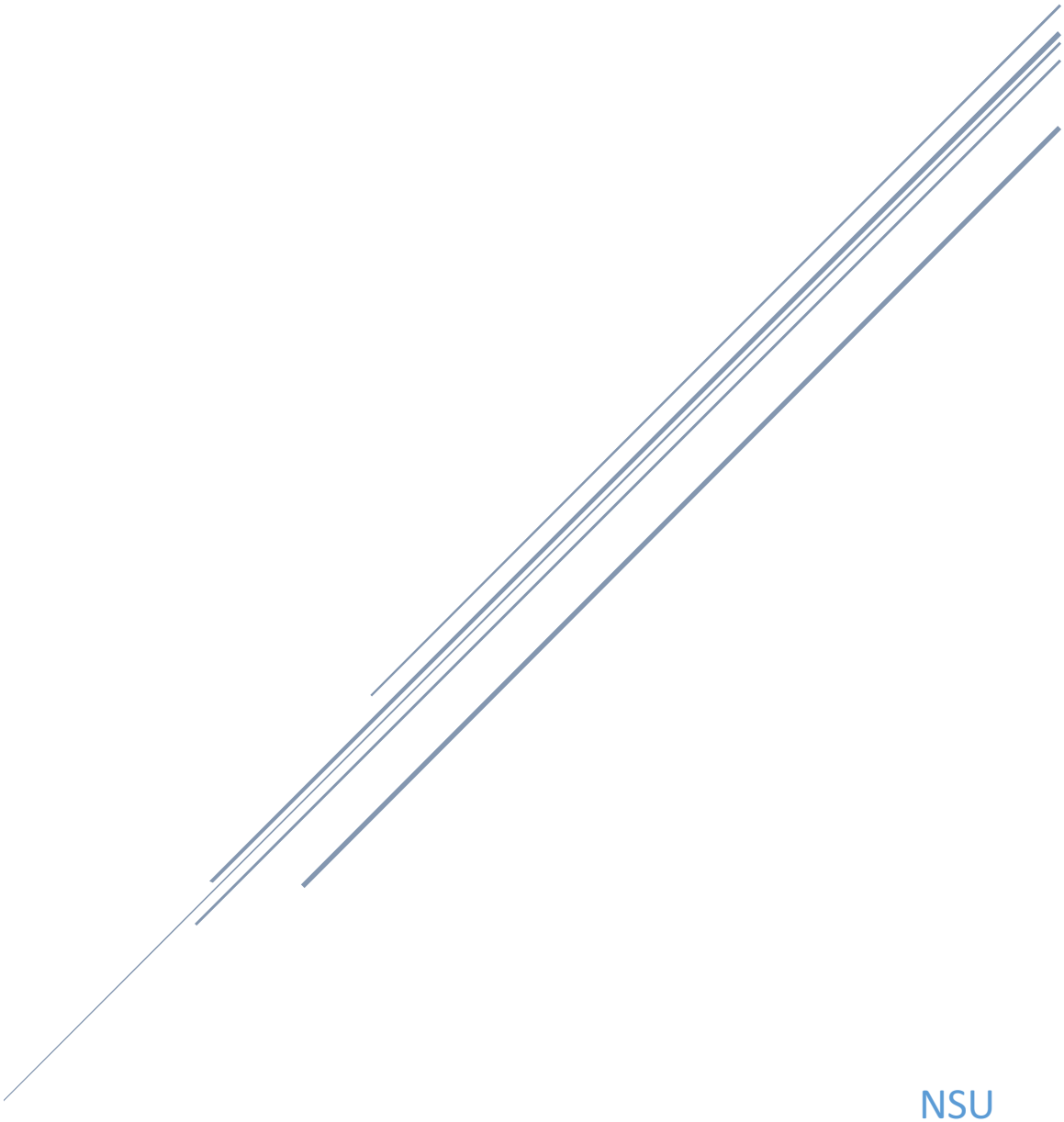
SIMMAC

Anabetsy Rivero



NSU

CISC640

Table of Contents:

Instructions for Terminal:

1- Unzip

2- Go into top level project directory from terminal

3- Cd into src

4- cd into com/example/java

5- Run javac *.java

6- Upon compilation, go back to src

7- Run java com/example/java/Main

8- Press enter

Introduction:

The present document specifies the requirements, architecture, design, and implementation of a multitasking operating system designed for SIMMAC. SIMMAC is a machine designed to simulate a multitasking operating system in a single queue using a round robin scheduling discipline.

SIMMAC Architecture:

SIMMAC is a machine that contains a word-addressable memory of size 512, and an ALU for its arithmetic operations. Each instruction of SIMMAC consists of a 32-bit word in which the leftmost 16 bits are reserved for the opcode and the rightmost 16 bits are reserved for the operand.

SIMMAC Registers:

- Accumulator(ACC) which is involved in all arithmetic operations
- Storage Address Register(SAR) which is involved in all references to primary storage. SAR holds the address of the data
- Storage Data Register(SDR) which is also involved in all references to primary storage. SDR holds th data specified in SAR
- Primary Storage Instruction Address Register(PSIAR) which points to the location of the next machine instruction in primary storage
- Temporary Register(TMPR) which is used (in my implementation) to hold the value of ACC while ACC is being used to increment PSIAR
- Control Storage Instruction Address Register(CSIAR) which points to the location of the next micro-instruction to be executed. I did not use this register
- Instruction Register(IR) which holds the current instruction being executed. I did not use this register
- Micro-instruction Register(MIR) which holds the current micro-instruction being executed. I did not use this register.

SIMMAC Instructions:

Instruction:	Format:
• ADD (Opcode = 10)	ADD <address>
• SUB (Opcode = 20)	SUB <address>
• LDA (Opcode = 30)	LDA <address>
• STR (Opcode = 40)	STR <address>
• BRH (Opcode = 50)	BRH <address>
• CBR (Opcode = 60)	CBR <address>
• LDI (Opcode = 70)	LDI <value>
• HALT (Opcode = 5)	HALT <0>

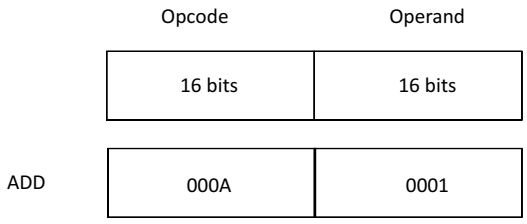


Fig 1: Representation of SIMMAC instructions. The leftmost 16 bits represent the opcode for the instruction, while the rightmost 16 bits represent the address in main memory of the operand. As an example, an ADD instruction is also shown.

SIMMAC Design:

SIMMAC was designed to read text files containing processes or programs written in its language. Each line of the text file contains a process or program starting with the process number, the arrival time, and cpu time the process needs. The cpu time is the number of instructions in each process, where each instruction is a clock cycle and the time quantum, specified by the user, is an integer multiple of a clock cycle. In this way, the round robin scheduler can execute each program for the time quantum and cycle through all the programs, thus allocating an equivalent cpu time to each program while making efficient use of the cpu at all times.

The language of SIMMAC specifies that each instruction is written as shown in figure 1. In my implementation, SIMMAC's language is written in 32 bit hexadecimal instructions, which are then split into opcode and operand before being executed. Because I separated the opcode from operand before execution, I did not use the micro-instructions where the TMPR register extracts the address portion of the instruction and places it in register SAR, then placing the integer corresponding to the location of SAR in main memory into register SDR. Although my design is a variation of the required design, it was advantageous because for every instruction that contains such a sequence of micro-instructions, I saved the time required to process those three micro-instructions.

In order to keep track of the locations of the start and end of each process, a tracker array was specified that stores the index of start and end for each process in main memory. This facilitates access to each process whenever they are executed.

SIMMAC Main Memory:

SIMMAC was designed with a main memory array of size 512 to hold words of 32 bits in size. Main memory was designed to hold instructions from index 0 to 200 and data from index 201 to 511. Data are loaded by assigning the integer values to the corresponding positions in main memory. The location of data was selected to include positions 201, 202, 301, and 302 to satisfy the requirements for programs 4 and 5 (processes 3 and 4 in SIMMAC).

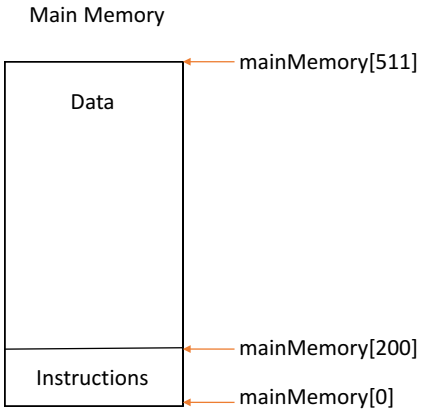


Fig 2: Representation of SIMMAC's main memory. Instructions are stored in mainMemory[0] to mainMemory[200]. Data are stored in mainMemory[201] to mainMemory[511].

SIMMAC Process Control Block:

Because some instructions request more cpu time than the time quantum in a multitasking system using a round robin, it is important that the status of the registers be saved until the sheduler comes back to finish the same process. The status of the process is usually stored in a process control block data structure. In SIMMAC, the status of each register is saved in array y, while array y is stored for each active process in a two-dimensional array pcb.

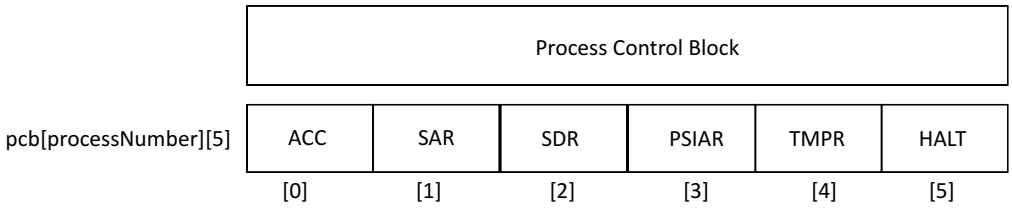


Fig 3: Representation of SIMMAC's process control block data structure. Array y holds the values of the registers after a time quantum has elapsed. Then, the scheduler loops back to the second part of the process, where the saved values are loaded again and the process continues. Pcb holds array y for each active process.

SIMMAC Process Scheduling:

SIMMAC uses a round robin algorithm for process scheduling. Processes are read from a text file, placed in main memory, instruction by instruction, and then retrieved and added to the job queue. Round robin then allocates an equal amount of cpu time for each process until a time quantum has passed and the number of instructions executed is equal to a quantum. At this point, array y stores the current state of the registers for the next round(if quantum is less than the length of the process) and round robin switches context to the next job. The algorithm keeps looping through the processes/jobs until all of them have been fully executed.

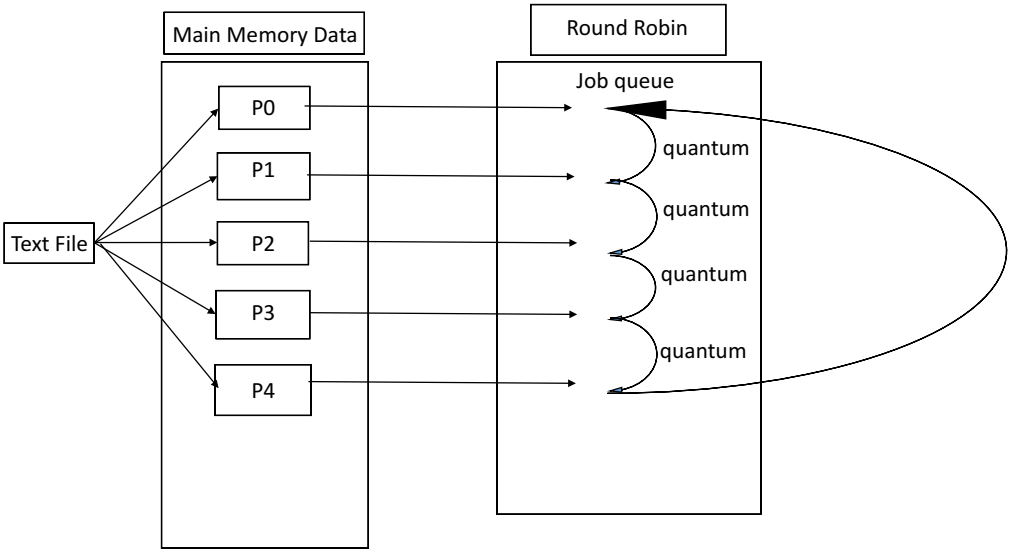


Fig 4: Representation of Round Robin scheduling in SIMMAC. Jobs/processes are read from text file, stored in SIMMAC main memory, and added to the job queue. Round robin then loops through each job, executing for a time quantum at a time. Process control block (Fig 3) was excluded for simplicity.

SIMMAC Processes/Programs:

A total of five SIMMAC programs or processes were written in SIMMAC's language. The execution of the five programs shows the functionality of the different instructions such as ADD, SUB, and CBR. The first three programs add twenty integers in different locations in main memory and store the sum in a different location in memory. The fourth process takes an integer (=> 100) in location 201 in main memory and decrements it by 1 until it is zero and the result is stored in location 202. This process tests SIMMAC's subtract, conditional branch, and store instructions.

The fifth process takes the integer in location 301 and increments its value by two until its value has increased by 200, and stores the result in location 302 in main memory. This is the most complex process because a counter (=100) is necessary to know when the integer has been increased by 200. By having a counter (=100) and decrementing it by 1 each time, while incrementing the value by 2, the conditional branch instruction "knows" if the value has been increased by 200 or not. If the counter is not zero, the value has not been increased by 200. This process tests addition, subtraction, conditional branch, and store in the same process.

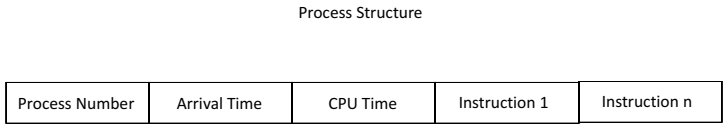


Fig 4: Depicts the structure of each SIMMAC program/process. Any process can have up to n instructions, which are executed in the order in which they appear. The final instruction for each process is a HALT, which I designed to be opcode 5.

Source Code:

```
package com.example.java;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;


public class Main {


    //Main memory to store data and instructions
    private static int [] mainMemory = new int[512];
    public static void main(String[] args) {

        System.out.println("Welcome to SIMMAC!");

        mainMemory[201] = 100;

        mainMemory[301] = 20;

        mainMemory[303] = 100;
        mainMemory[304] = 1;

        mainMemory[400] = 10;
        mainMemory[401] = 5;
        mainMemory[402] = 2;
        mainMemory[403] = 2;
        mainMemory[404] = 3;
        mainMemory[405] = 7;
        mainMemory[406] = 8;
        mainMemory[407] = 2;
        mainMemory[408] = 2;
        mainMemory[409] = 19;
        mainMemory[410] = 1;
        mainMemory[411] = 1;
        mainMemory[412] = 3;
        mainMemory[413] = 9;
        mainMemory[414] = 20;
        mainMemory[415] = 1;
        mainMemory[416] = 6;
        mainMemory[417] = 25;
        mainMemory[418] = 4;
        mainMemory[419] = 8;
        mainMemory[420] = 15;
        mainMemory[421] = 20;
        mainMemory[422] = 10;
        mainMemory[423] = 30;
        mainMemory[424] = 12;
        mainMemory[425] = 9;
        mainMemory[426] = 6;
        mainMemory[427] = 21;
        mainMemory[428] = 25;
        mainMemory[429] = 1;
        mainMemory[430] = 50;
        mainMemory[431] = 10;
```

```
mainMemory[432] = 5;
mainMemory[433] = 8;
mainMemory[434] = 4;
mainMemory[435] = 1;
mainMemory[436] = 11;
mainMemory[437] = 23;
mainMemory[438] = 4;
mainMemory[439] = 17;
mainMemory[440] = 35;


//Gets the quantum number from user
Scanner reader = new Scanner(System.in);
System.out.println("Please, enter the quantum number: ");
int quantum = reader.nextInt();
System.out.println("Quantum: " + quantum + "\n");

String filename = "testing.txt";
BufferedReader process = null;


//Tracker holds the start and end points of each process stored in main memory
int[] tracker;
int start = 0;
int end = 0;
try {
    String sCurrentLine;

    process = new BufferedReader(new FileReader(filename));

    tracker = new int[10];
    int processNumber = 0;
    int counter = 3;
    start = counter;
    int counter1 = 3;
    while ((sCurrentLine = process.readLine()) != null) {
        String a[] = sCurrentLine.split(" ");
        tracker[processNumber] = start +3;

        while (counter1 < a.length + counter) {
            mainMemory[counter1] = Integer.parseInt(a[counter1 - counter], 16);
            counter1++;
        }

        end = counter1;
        tracker[processNumber + 1] = end;
        counter = counter1;
        start = counter;
        processNumber = processNumber + 2;
    }


    //The following lines add each job to the job queue and run round robin
    RoundRobin rr = new RoundRobin();
    Job job0 = new Job(tracker[0], tracker[1]);
    rr.add(job0);
    System.out.println("Adding Job 0: " + job0);

    Job job1 = new Job(tracker[2], tracker[3]);
    rr.add(job1);
    System.out.println("Adding Job 1: " + job1);
```

```
            Job job2 = new Job(tracker[4], tracker[5]);
            rr.add(job2);
            System.out.println("Adding Job 2: " + job2);

            Job job3 = new Job(tracker[6], tracker[7]);
            rr.add(job3);
            System.out.println("Adding Job 3: " + job3);

            Job job4 = new Job(tracker[8], tracker[9]);
            rr.add(job4);
            System.out.println("Adding Job 4: " + job4);
            System.out.println(rr);
            rr.run(quantum, tracker); // Most important line!
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (process != null) process.close();
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }

    //Execute method where all the opcodes are processed
    static int[] execute(int start, int end, int state[], int quantum, int processNum)
{

        //Sets the state for each register. This state is used later in the process
control block
        int ACC = state[0];
        int SAR = state[1];
        int SDR = state[2];
        int PSIAR = state[3];
        int TMPR = state[4];
        int HALT = 0;
        {


            for (PSIAR = start; PSIAR < end; PSIAR++) {
               if(quantum == 0) {
                  //System.out.println("quantum: " +quantum); // Remove comment to see
quantum iterations
                  break;
               }
               else {
                  //System.out.println("quantum: " +quantum); // Remove comment to see
quantum iterations
                  quantum--;
               }
                //Each instruction is decoded and divided into opcode and operand
                int high = mainMemory[PSIAR] >> 16;
                int low = mainMemory[PSIAR] & 0xFFFF;

                //Gets opcode from instruction
                int opcode = high;

                //Gets address for instruction
                SAR = low;
```

```
if (opcode == 10) {
    //Instruction is ADD
    ACC = ACC + Integer.valueOf(mainMemory[SAR]);
    TMPR = ACC;
    ACC = PSIAR;
    PSIAR = ACC;
    ACC = TMPR;
    SDR = Integer.valueOf(mainMemory[SAR + 1]);
    TMPR = SDR;
} else if (opcode == 20) {
    //Instruction is SUB
    ACC -= Integer.valueOf(mainMemory[SAR]);
    TMPR = ACC;
    ACC = PSIAR;
    PSIAR = ACC;
    ACC = TMPR;
    SDR = Integer.valueOf(mainMemory[SAR + 1]);
    TMPR = SDR;
} else if (opcode == 30) {
    //Instruction is LOAD
    ACC = Integer.valueOf(mainMemory[SAR]);
    TMPR = ACC;
    ACC = PSIAR;
    PSIAR = ACC;
    ACC = TMPR;
    SDR = Integer.valueOf(mainMemory[SAR]);
    ACC = SDR;
} else if (opcode == 70) {
    //Instruction is LOAD IMMEDIATE.
    ACC = ACC + Integer.valueOf(SAR);
    TMPR = ACC;
    ACC = PSIAR + 1;
    PSIAR = ACC;
    ACC = TMPR;
    SDR = Integer.valueOf(mainMemory[SAR + 1]);
    ACC = SDR;
} else if (opcode == 40) {
    //Instruction is STORE
    TMPR = ACC;
    ACC = PSIAR;
    PSIAR = ACC;
    ACC = TMPR;
    SDR = ACC;
    mainMemory[SAR] = SDR;//WRITE
} else if (opcode == 50) {
    //Instruction is BRANCH
    SDR = Integer.valueOf(mainMemory[SAR]);
    PSIAR = SDR;
} else if (opcode == 5) {
    //Instruction is a HALT
    System.out.println("***End of Process " + processNum  + " ***");
    HALT = 1;
    System.out.println("ACC: " + ACC);
    System.out.println("TMPR: " + TMPR);
    System.out.println("SAR: " + SAR);
    System.out.println("SDR: " + SDR);
    System.out.println("PSIAR: " + PSIAR + "\n\n");
} else if (opcode == 60) {
    //Instruction is CONDITIONAL BRANCH
    if (ACC == 0) {
       ;
```

```
                    } else {
                        PSIAR = SAR - 1;
                    }
                }
            }
        }
        return new int[]{
                ACC,
                SAR,
                SDR,
                PSIAR,
                TMPR,
                HALT
        };
    }
}
```

```
package com.example.java;

import java.util.LinkedList;
import java.util.Queue;

import static com.example.java.Main.execute;

public class RoundRobin {

   //Job queue for all jobs/processes
    private Queue<Job> scheduler = new LinkedList<Job>();

    @Override
    public String toString() {
       String output = "*************JOB QUEUE**************\n";
       int x = 0;
       for (Job element : scheduler) {
            output += String.format("* Process: %-1d | %-19s *\n",
x,element.toString());
            x++;
          }
       output += "************************************";
        return output;
    }

   //Add method for adding all jobs/processes to scheduler
    public void add(Job job) {
        this.scheduler.add(job);
    }

   //Run method to run each job/process for a given quantum number.
    public void run(int quantum, int tracker[]) {

      //Array y holds the state of each register so round robin can pick up in the same
location on the second round
       int y[] = {0, 0, 0, 0, 0, 0};
      //Two-dimensional array pcb holds the status array y for each process. pcb[x][5]
!= 1 means that the process has not encountered a HALT instruction yet
       int[] pcb[] = {y,y,y,y,y};
       while( (pcb[0][5] != 1) || (pcb[1][5] != 1) || (pcb[2][5] != 1) || (pcb[3][5] !=
1) || (pcb[4][5] != 1) ) {
```

```
          int processNum = 0;
           for (int x = 0; x < 10; x+=2) {
              if(pcb[processNum][5] == 1)
                 ;
              else if( (tracker[x] != tracker[x+1] && pcb[processNum][3] == tracker[x])
|| pcb[processNum][3] == 0)
                 pcb[processNum] = execute(tracker[x], tracker[x + 1],
pcb[processNum],quantum,processNum);
              else if(tracker[x] != tracker[x+1] && pcb[processNum][3] != tracker[x] &&
pcb[processNum][3] !=0)
                 pcb[processNum] = execute(pcb[processNum][3], tracker[x + 1],
pcb[processNum],quantum,processNum);
               if(pcb[processNum][5] == 1) {
                  tracker[x] = 0;
                  tracker[x+1] = 0;
               }
               if(processNum == this.scheduler.size() - 1 && ((pcb[0][5] != 1) ||
(pcb[1][5] != 1) || (pcb[2][5] != 1) || (pcb[3][5] != 1) || (pcb[4][5] != 1)))
                  System.out.println("Done with Process: " + processNum + "\nContinuing
with process: " + 0 );
               else if(processNum == this.scheduler.size() - 1 && (pcb[0][5] != 1) ||
(pcb[1][5] != 1) || (pcb[2][5] != 1) || (pcb[3][5] != 1) || (pcb[4][5] != 1))
                  System.out.println("Done with Process: " + processNum + "\nContinuing
with process: " + (processNum+1) );
               else if( !((pcb[0][5] != 1) || (pcb[1][5] != 1) || (pcb[2][5] != 1) ||
(pcb[3][5] != 1) || (pcb[4][5] != 1)) )
                  System.out.println("Done with ALL Processes!");

               processNum++;

           }
       }
    }
}
```

```
package com.example.java;

public class Job {

    private int start;
    private int end;
    public boolean isFinished;

    public Job(int x, int y){
        this.start = x;
        this.end = y;
    }

    @Override
    public String toString(){
        return "Start: " + this.start + ", " + "End: " + end;
    }
}
```

SIMMAC Programs:

```
0 1 26 001E00FA 000A0190 000A0191 000A0192 000A0193 000A0194 000A0195 000A0196 000A0197
000A0198 000A0199 000A019A 000A019B 000A019C 000A019D 000A019E 000A019F 000A01A0 000A01A1
000A01A2 000A01A3 0028017C 00050000

1 2 26 001E00FA 000A0195 000A0196 000A0197 000A0198 000A0199 000A019A 000A019B 000A019C
000A019D 000A019E 000A019F 000A01A0 000A01A1 000A01A2 000A01A3 000A01A4 000A01A5 000A01A6
000A01A7 000A01A8 0028017D 00050000

2 3 26 001E00FA 000A019D 000A019E 000A019F 000A01A0 000A01A1 000A01A2 000A01A3 000A01A4
000A01A5 000A01A6 000A01A7 000A01A8 000A01A9 000A01AA 000A01AB 000A01AC 000A01AD 000A01AE
000A01AF 000A01B0 0028017E 00050000

3 4 8 001E00C9 0014019A 002800C9 003C0054 0028017F 001E017F 002800CA 00050000

4 5 10 001E012D 000A0192 0028012D 001E012F 00140130 0028012F 003C005F 001E012D 0028012F
00050000
```

Sample Output:

```
Welcome to SIMMAC!
Please, enter the quantum number:
20
Quantum: 20

Adding Job 0: Start: 6, End: 29
Adding Job 1: Start: 32, End: 55
Adding Job 2: Start: 58, End: 81
Adding Job 3: Start: 84, End: 92
Adding Job 4: Start: 95, End: 105
**********JOB QUEUE**********
* Process: 0 | Start: 6, End: 29   *
* Process: 1 | Start: 32, End: 55  *
* Process: 2 | Start: 58, End: 81  *
* Process: 3 | Start: 84, End: 92  *
* Process: 4 | Start: 95, End: 105 *
**********************************
Done with Process: 0
Continuing with process: 1
Done with Process: 1
Continuing with process: 2
Done with Process: 2
Continuing with process: 3
```

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

***End of Process 0 ***

ACC: 138

TMPR: 138

SAR: 0

SDR: 138

PSIAR: 28

Done with Process: 0

Continuing with process: 1

***End of Process 1 ***

ACC: 203

TMPR: 203

SAR: 0

SDR: 203

PSIAR: 54

Done with Process: 1

Continuing with process: 2

***End of Process 2 ***

ACC: 287

TMPR: 287

SAR: 0

SDR: 287

PSIAR: 80

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

***End of Process 3 ***

ACC: 0

TMPR: 0

SAR: 0

SDR: 0

PSIAR: 91

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

```
Continuing with process: 0
   Done with Process: 0
Continuing with process: 1
   Done with Process: 1
Continuing with process: 2
   Done with Process: 2
Continuing with process: 3
   Done with Process: 3
Continuing with process: 4
   Done with Process: 4
Continuing with process: 0
   Done with Process: 0
Continuing with process: 1
   Done with Process: 1
Continuing with process: 2
   Done with Process: 2
Continuing with process: 3
   Done with Process: 3
Continuing with process: 4
   Done with Process: 4
Continuing with process: 0
   Done with Process: 0
Continuing with process: 1
   Done with Process: 1
Continuing with process: 2
```

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

Continuing with process: 0

Done with Process: 0

Continuing with process: 1

Done with Process: 1

Continuing with process: 2

Done with Process: 2

Continuing with process: 3

Done with Process: 3

Continuing with process: 4

Done with Process: 4

```
Continuing with process: 0
   Done with Process: 0
Continuing with process: 1
   Done with Process: 1
Continuing with process: 2
   Done with Process: 2
Continuing with process: 3
   Done with Process: 3
Continuing with process: 4
***End of Process 4 ***
ACC: 220
TMPR: 220
SAR: 0
SDR: 220
PSIAR: 104


Done with ALL Processes!


Process finished with exit code 0
```