Page No.
Date

2.

★ Problem Statement -
Design suitable data structures & implement Pass I & Pass II of a two-pass macro processor. The output of Pass-I (MNT, MDT & intermediate code file without any macro-definition) should be input for pass-II.

★ Prerequisites -
Basic concepts of assembler pass I & pass II / syntax analyzer.

★ Learning Objectives -
1) To understand data structure to be used in Pass I of an assembler.
2) To implement a two-pass assembler.

★ Theory Concepts -

* In the analysis synthesis model of a compiler, the front end of a compiler translates a source program into an independent intermediate code, then the back end of the compiler uses this intermediate code to generate the target code.

* The benefits of using machine independent code are -
⊙ Because of all the machine independent intermediate code, probability will be enhanced.

For example, suppose a compiler translates the source language to its target machine language without having the options for generating required, because obviously there are some modifications in the compiler itself according to the machine specification.

- If machine code is generated directly from source code, then for target machine code, we will have an optimised code generators but if we will have a machine independent intermediate code.
- It is easier to apply source code modification to the machine & we see the performance of source code by optimizing the intermediate code.

* The following are commonly used intermediate representation -

1. Postfix Notation
2. Syntax Tree
3. Three address code

* Conclusion -

Thus, we have studied the data structure & Implementation of Pass I & Pass II of a two-pass macro processor.

3.

* Problem Statement -
Write a program to simulate CPU scheduling algorithms - FCFS, SJF (Preemptive), Priority (Non-preemptive) & Round Robin (Preemptive)

* Prerequisites -
Basic concepts of scheduling, types of scheduling, algorithms

* Learning Objectives -
Understand the implementation of the scheduling algorithms.

* Theory

* Scheduler - It is an operating system module that selects the next job to be admitted into the system & next process, to run. There are 3 major types of scheduler as follows -

1. Short-term scheduler
2. Mid-term scheduler
3. Long-term scheduler

* Scheduling - It is the method specified by some means that is assigned to resources that complete the work, which may either be visual computation elements like thread, processes, data flows, etc. There are 2 types of scheduling algorithms to solve

any sort of operations -

1. Preemptive Scheduling
2. Non-preemptive scheduling.

* Scheduling Algorithms -

1. FCFS CPU Scheduling Algorithms -

For FCFS scheduling algorithms, ~~reed~~ read the number of processes/jobs in the system, their CPU burst times. The scheduling is performed on the basis of arrival time of the processes irrespective of their other parametres. Each process will be executed according to its arrival time. Calculate the waiting time & then turn around time of each of the processes accordingly

Process	Arrival Time	Execute Time	Service Time
P_0	0	5	0
P_1	1	3	5
P_2	2	8	8
P_3	3	6	16

P_0	P_1	P_2	P_3	
0	5	8	16	22

Process	Wait time = Service - Arrival time
P_0	$0 - 0 = 0$
P_1	$5 - 1 = 4$
P_2	$8 - 2 = 6$
P_3	$16 - 3 = 13$

$\therefore$ Avg. wait time $= \frac{0+4+6+13}{4} = 5.75.$

2. SJF CPU Scheduling Algorithm

For SJF Scheduling Algorithm, read the number of jobs/processes in the system, their CPU burst times, and arrange all the jobs in order with respect to their burst times. There may be two jobs in queue with the same execution time, & then FCFS approach is to be performed. Each process will be executed according to the length of its burst time. Then calculate

Process	Arrival Time	Execute Time	Service Time	Wait Time
P_0	0	5	3	$3-0=3$
P_1	1	3	0	$0-0=0$
P_2	2	8	16	$16-2=14$
P_3	3	6	8	$8-3$ 5

P_1 P_0 P_3 P_2

0 3 8 16 22

Avg. waiting time $= \frac{3+14+5}{4} = 5.5$

3. Round Robin CPU Scheduling Algorithm -

For Round Robin Scheduling Algorithm, read the number of jobs/processes in the system, their CPU burst times & the size of the time slice. Time slices are assigned to each process in equal portions & in circular order, handling all process execution. This allows every process to get an equal chance. Calculate the ~~wait~~ waiting time & turn around time of each of the processes accordingly.

P_0	P_1	P_2	P_3	P_0	P_2	P_3	P_2	
0	3	6	9	12	14	17	20	22

Process	Wait Time = Service - Arrival Time	
P_0	$(0-0) + (12-3)$	$= 9$
P_1	$3-1$	$= 2$
P_2	$(6-2) + (14-9) + (20-17)$	$= 12$
P_3	$(9-3) + (17-12)$	$= 11$

$$\text{Avg. wait time} = \frac{(9+2+12+11)}{4} = 8.5$$

4. Priority CPU Scheduling Algorithm -

For priority scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times & the priorities. Arrange all the jobs in order with respect to their priorities. There may be two jobs in queue with the same priority, & then FCFS approach is to be performed. Each process will be executed according to its priority. ~~There may be two jobs~~ Calculate the waiting time & turn around time of each of the processes accordingly.

Process	Arrival Time	Service Time	Execute Time	Priority
P_0	0	9	5	1
P_1	1	6	3	2
P_2	2	14	8	1
P_3	3	0	6	3

P_3	P_1	P_0	P_2	
0	6	9	14	22

Process	Wait Time = Service Time - Arrival Time
P_0	$9 - 0 = 9$
P_1	$6 - 1 = 5$
P_2	$14 - 2 = 12$
P_3	$0 - 0 = 0$

Avg. waiting time = $\frac{9+5+12+0}{4} = 6.5$

★ Conclusion -

Thus, we have understood CPU scheduling algorithms.

ADP

4.

★ Problem Statement -
Write a program to simulate page ~~translation~~ replacement algorithms - Least Recently Used (LRU), First In First Out (FIFO), Optimal Page Replacement.

★ Prerequisites -
Basic concepts && idea of page replacement.

★ Learning Objectives -
1. To understand page fault in paging
2. To implement various page replacement algorithms.

★ Theory -

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in

* Page Fault -
A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page fault happens. In case of a page fault, operating system might have to replace one

of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target of all algorithms is to reduce the number of page faults.

* Page Replacement Algorithms -

1) First In First Out (FIFO) -

→ This is the simplest page replacement algorithm. The operating system keeps track of all pages in the memory in a queue. Here, the oldest page is in the front of ~~a~~ queue. When a page needs to be replaced, page in front of the queue is selected for removal.

→ Eg - Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3 page frames. Find the number of page faults.

1	3	0	3	5	6	3
		0	0	0	0	3
	3	3	3	3	6	6
1	1	1	1	5	5	5
Miss	Miss	Miss	Hit	Miss	Miss	Miss

⇒ Total page faults = 6

→ Initially, all slots are empty, so when 1, 3, 0 come along, they are allocated to the empty slots → 3.
→ Faults - When 3 comes, its already in memory, so → 0 page faults.
→ Then 5 comes, it isn't available in memory so it replaces the oldest page slot, i.e. 3 → 1 page fault. Finally when 3 comes, it isn't available, so it replaces 0 → 1 page fault.

2) <u>Optimal Page Replacement</u> -
→ In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.
→ Eg - Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3 with 4-page frame. Find number of page faults.

7	0	1	2	0	3	0
			2	2	2	2
		1	1	1	1	1
	0	0	0	0	0	0
7	07	7	7	7	3	3
Miss	Miss	Miss	Miss	Hit	Miss	Hit

4	2	3	0	3	2	3
2	2	2	2	2	2	2
4	4	4	4	4	4	4
0	0	0	0	0	0	0
3	3	3	3	3	3	3
Miss	Hit	Hit	Hit	Hit	Hit	Hit.

Total page fault = 6

→ 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 faults. When 3 came, it will take the ~~plage~~ place of 7 as it is not used for the longest duration of time in the 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're already available in the memory.

3) Least Recently Used -

→ In this algorithm, page that will be replaced will be the one which is least recently used.

→ Eg - Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page ~~faults~~ frames. Find number of page faults.

7	0	1	2	0	3	0
			2	2	2	2
		1	1	1	1	1
	0	0	0	0	0	0
7	7	7	7	7	3	3
Miss	Miss	Miss	Miss	Hit	Miss	Hit

4	2	3	0	3	2	3
2	2	2	2	2	2	2
4	4	4	4	4	4	4
0	0	0	0	0	0	0
3	3	3	3	3	3	3
Miss	Hit	Hit	Hit	Hit	Hit	Hit

→ Initially, all slots are empty. So when 7, 0, 1, 2 are allocated to the empty slots → 4 page faults. 0 is already there → 0 page faults. When 3 comes, it takes place of 7 as it is least recently used → 1 page fault. 0 is already there, so → 0 page faults. 4 will take place of 1 → 1 page fault. Now, for further page reference strings → 0 page faults because they are already available in the memory.

1.

★ Problem Definition -
Design suitable data structures & implement pass I & II of a two-pass assembler for a pseudo machine in Java, using Object-oriented feature. Implementation should consist of a few instructions from each category & a few assembler directives.

★ Prerequisites -
Basic Concepts of assembler pass I & pass II (syntax analyser)

★ Learning Objectives -
→ To understand data structures to be used in pass I of an assembler.
→ To implement pass I of an assembler.

★ Theory Concepts -

* A language translator bridges an execution gap to machine language of computer system. An assembler is a language translator whose source language is assembly language. An assembler is a program that accepts as input an assembly language program & then converts it into machine language.

* Two - pass translation scheme -

In a 2-pass assembler, the first pass constructs an intermediate representation of the source program for use by the second pass. This representation consists of 2 main components - data structures like symbol tables, literal tables, & processed form of the source program, called as intermediate code. This intermediate code (IC) is represented by the syntax of variant I.

Forward reference of a program entity is a reference to the entity, which precedes its definition in the program. While processing a statement containing a forward reference, language processor does not possess all relevant information concerning referenced entity. This creates difficulty in synthesizing the equivalent target statements. This problem can be solved by postponing the generation of target code until more information concerning the entity is available. This reduces memory requirements of RPLP & simplifies its organization. This leads to multi-pass model of language processing.

* Data Structures of a Two-Pass Assembler -

1. Operation Code Table (OPTAB) -
   This is used for storing mnemonic, operation code, and class of instruction.

2. Data structures updated during translation -
These are also called translation time data structure. They are -
   a. Symbol Table (SYMTAB) -
   It contains entries such as symbol, its address & value
   b. Literal Table (LITTAB) -
   It contains entries such as literals & their values
   c. Pool Table (POOLTAB) -
   It contains literal number of the starting literal of each literal pool.

★ Design of a Two - Pass Assembler -

Tasks performed by the passes of a two-pass assembler ~~are~~ as follows -

Pass I -

Seperate the symbol, mnemonic opcode & operand fields.
Determine the storage required for every assembly language statement & update the location counter.
Build the symbol & literal table.
Construct the intermediate code for every assembly language statement.

Synthesize the target code by processing the ~~into~~ intermediate code ~~dra~~ generated during pass I.

INTERMEDIATE CODE REPRESENTATION

The intermediate code consists of a set of IC units, each IC unit consisting of the following fields. -

1. Address
2. Representation of the mnemonic opcode
3. Representation of operands.

where Statement class can be one of IS, DL & AD, which stand for imperative statement, declaration statement & assembleer directive respectively.

★ PASS I -

- Initialize location counter, entries of all tables as zero
- Read statements from input file one-by-one.
- While next statement is not END statement,

I. Tokenize or seperate out input statements as lable, mnemonic, ~~or~~ operand 1 & operand 2.

II. If a label is present, insert ~~lable~~ label into symbol table

III. If the statement is LTORG statement processes it by ~~mal~~ making its entry into literal table, pool table & allocate memory.

IV. If statements are start or origin, process location counter accordingly.
V. If an EQU statement, assign values to adding entry in symbol table.
VI. For declarative statement, update code, size & location counter.
VII. Generate the intermediate code
VIII. Pass the intermediate code to pass-2.

★ Conclusion -
Thus, we have studied visual programming & implemented dynamic link library application for arithmetic operation.