SPOS Lab Manual

Assignment 1:

Problem Statement: Implement Pass-I of a two-pass assembler.

THEORY:

Assembler-

- 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.
- Language processing activity consists of two phases, Analysis phase and synthesis phase.
- Analysis of source program generates various data structures and intermediate code.
- Synthesis phase is concerned with construction of target language statements, which have the same meaning as source language statements. This consists of memory allocation and code generation.

Pass-1 Assembler-

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

Data Structures:

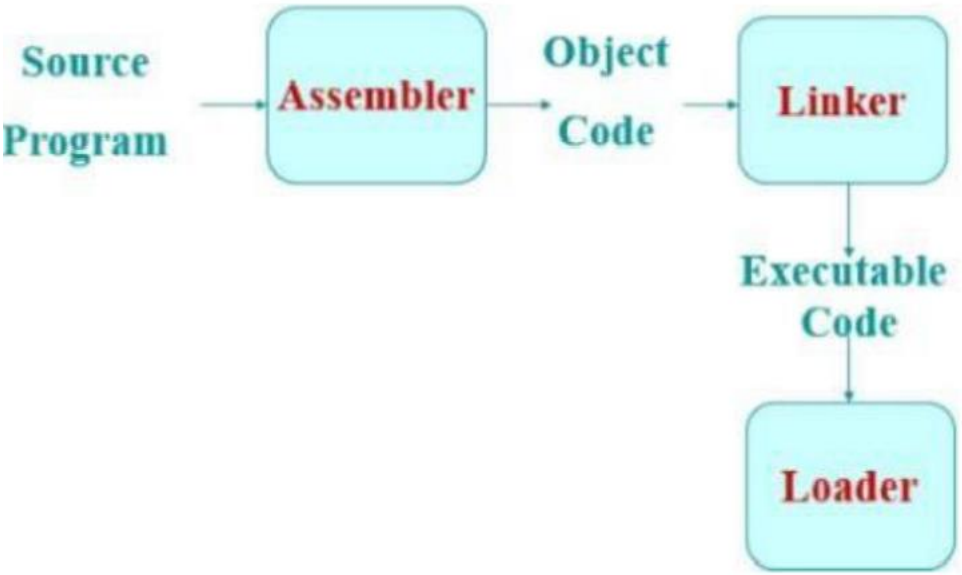
Source file containing assembly program.

MOT: A table of mnemonic op-codes and related information.

Symbol Table

Literal table

Pool table



Assignment 2

Problem Statement: Implement pass-II of a two-pass assembler.

Pass-2 Assembler-

Pass-2 of assembler generates machine code by converting symbolic machine-opcodes into their respective bit configuration (machine understandable form). It stores all machine-opcodes in MOT table (op-code table) with symbolic code, their length and their bit configuration. It will also process pseudo-ops and will store them in POT table (pseudo-op table).

- Generate object code by converting symbolic op-code into respective numeric op-code
- Generate data for literals and look for values of symbols

Various Data bases required by pass-2:

1. MOT table (machine opcode table)
2. POT table (pseudo-opcode table)
3. Base table (storing value of base register)
4. LC (location counter)

Assignment 3

Design suitable data structures and implement macro definition and macro expansion processing for a sample macro with positional and keyword parameters.

Macro are used to provide a program generation facility through macro expansion. Many programming language provide built in facilities for writing macros. E.g. Ada,C and C++. Higher version of processor family also provide such facility. “A macro is a unit of specification for program generation through expansion. Macro consist of name, a set of formal parameters and a body of code. “The use of macro name with a set of actual parameters is replaced by some code generated from its body, this is called macro expansion.”

```
procedure EXPAND
    begin
        EXPANDING  := TRUE
        get first line of macro definition (prototype) from DEFTAB
        set up arguments from macro invocation in ARGTAB
        write macro invocation to expanded file as a comment
        while not end of macro definition do
            begin
                GETLINE
                PROCESSLINE
            end {while}
        EXPANDING := FALSE
    end {EXPAND}
```

Assignment 4

Problem Statement: Implement the following using shell scripting Menu driven program for

a) Find the factorial of a no.

b) Find greatest of three numbers

c) Find a prime no

d) Find whether a number is palindrome

e) Find whether a string is palindrome

Theory-

A menu-driven shell script provides users more options/interactive interface. In a layman's term shell script is similar to the restaurant and you asked for a restaurant menu, so you can choose your favorite dish. Similarly, a menu-driven shell script serves the same purpose.

Using case esac statement to create a menu-driven shell script: -

A case statement is similar to a switch statement from another languages.

Case statement is an alternative to multilevel if-then-else statements.

Using case statement, we can avoid using multiple if-then-else and reduce script size.

Case is used to match several values against one value.

Shell Scripting is an open-source computer program designed to be run by the Unix/Linux shell. Shell Scripting is a program to write a

series of commands for the shell to execute. It can combine lengthy and repetitive sequences of commands into a single and simple script that can be stored and executed anytime which, reduces programming efforts.

A shell script is a list of commands in a computer program that is run by the Unix shell which is a command line interpreter. A shell script usually has comments that describe the steps. The different operations performed by shell scripts are program execution, file manipulation and text printing. A wrapper is also a kind of shell script that creates the program environment, runs the program etc.

Algorithm/ Flowchart:

1.Get a number

2. Use for loop or while loop to compute the factorial by using the below formula
3. fact(n) = n * n-1 * n-2 * .. 1

4. Display the result.

ASSIGNMENT 5

THEORY:

- Wait: When a process develops a child process, it's occasionally important for the parent process to wait till the child has completed it before continuing. This is exactly what the wait () system function accomplishes. Waiting causes the parent to wait for the child to alter its state. The status change could be due to the child process being terminated, stopped by a signal, or resumed by a signal.

- Zombie Process: A process which has finished the execution but still has entry in the process table to report to its parent process is known as a zombie process.

A child process always first becomes a zombie before being removed from the process table. The parent process reads the exit status of the child process which reaps off the child process entry from the process table.

- Orphan Process: A process whose parent process no more exists i.e. either finished or terminated without waiting for its child process to terminate is called an orphan process. In the following code, parent finishes execution and exits while the child process is still executing and is called an orphan process now

- Process: A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

System call **fork()** is used to create processes. It takes no arguments and returns a process ID. The purpose of **fork()** is to create a ***new*** process, which becomes the *child* process of the caller. After a new child process is created, ***both*** processes will execute the next instruction following the ***fork()*** system call. Therefore, we have to distinguish the parent from the child. This can be done by testing the returned value of **fork()**:

- If **fork()** returns a negative value, the creation of a child process was unsuccessful.
- **fork()** returns a zero to the newly created child process.
- **fork()** returns a positive value, the ***process ID*** of the child process, to the parent. The returned process ID is of type **pid_t** defined in **sys/types.h**. Normally, the process ID is an integer. Moreover, a process can use function **getpid()** to retrieve the process ID assigned to this process.

ASSIGNMENT 6

Thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler (typically as part of an operating system). The implementation of threads and processes differs from one operating system to another, but in most cases, a thread is a component of a process. Multiple threads can exist within the same process and share resources such as memory, while different processes do not share these
resources. Threads are created using pthread create(). Prototype of pthread create():
int pthread_create(pthread_t *thread_id,const pthread_attr_t *attributes, void *(*thread_function)(void *), void *arguments);
This function creates a new thread. pthread t is an opaque type which acts as a handle for the new thread.

Algorithm
Input: two matrices.
Output: Output matrix C.
procedure Matrix-Multiply(A, B)
1. if columns [A] != rows [B]
2.
then error “incompatible dimensions”
3. else
4.
for i = 1 to rows [A]
5.
for j = 1 to columns [B]
6.
C[i, j] =0
7.
for k = 1 to columns [A]
8.
C[i, j]=C[i, j]+A[i, k]*B[k, j]
9. return C

ASSIGNMENT 7

FCFS

Simplest CPU scheduling algorithm that schedules according to arrival times of processes. First come first serve scheduling algorithm states that the process that requests the CPU first is allocated the CPU first. It is implemented by using the FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue. FCFS is a non-preemptive scheduling algorithm.

Characteristics of FCFS:

- FCFS supports non-preemptive and preemptive CPU scheduling algorithm.
- Tasks are always executed on a First-come, First-serve concept.
- FCFS is easy to implement and use.
- This algorithm is not much efficient in performance, and the wait time is quite high.

The algorithm for the FCFS Scheduling program in C is as follows:

1. At first, the process of execution algorithm starts
2. Then, we declare the size of an array
3. Then we get the number of processes that need to insert into the program
4. Getting the value
5. Then the first process starts with its initial position and the other processes are in a waiting queue.
6. The total number of burst times is calculated.
7. The value is displayed
8. At last, the process stops.

SGF

In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.

However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.

Advantages of SJF

1. Maximum throughput
2. Minimum average waiting and turnaround time

Disadvantages of SJF

1. May suffer with the problem of starvation
2. It is not implementable because the exact Burst time for a process can't be known in advance.

Implementation of Algorithm-

- Practically, the algorithm cannot be implemented but theoretically it can be implemented.
- Among all the available processes, the process with smallest burst time has to be selected.
- Min heap is a suitable data structure where root element contains the process with least burst time.
- In min heap, each process will be added and deleted exactly once.
- Adding an element takes log(n) time and deleting an element takes log(n) time.
- Thus, for n processes, time complexity = n x 2log(n) = nlog(n)

RR

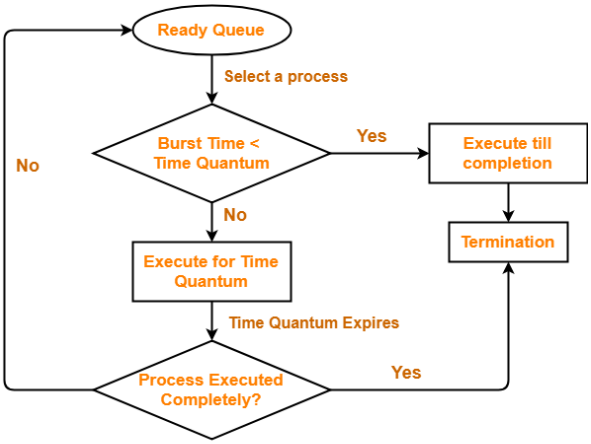
- CPU is assigned to the process on the basis of FCFS for a fixed amount of time.
- This fixed amount of time is called as **time quantum** or **time slice**.
- After the time quantum expires, the running process is preempted and sent to the ready queue.
- Then, the processor is assigned to the next arrived process.
- It is always preemptive in nature.

With decreasing value of time quantum,

- Number of context switch increases
- Response time decreases
- Chances of starvation decreases

Thus, smaller value of time quantum is better in terms of response time.

- With increasing value of time quantum, Round Robin Scheduling tends to become FCFS Scheduling.
- When time quantum tends to infinity, Round Robin Scheduling becomes FCFS Scheduling.



Round Robin Scheduling

ASSIGNMENT 8

Simulation of Banker's algorithm using 'C'/Python/Java language.

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Advantages

1. It contains various resources that meet the requirements of each process.
2. Each process should provide information to the operating system for upcoming resource requests, the number of resources, and how long the resources will be held.
3. It helps the operating system manage and control process requests for each type of resource in the computer system.
4. The algorithm has a Max resource attribute that represents indicates each process can hold the maximum number of resources in a system.

Disadvantages

1. It requires a fixed number of processes, and no additional processes can be started in the system while executing the process.
2. The algorithm does no longer allows the processes to exchange its maximum needs while processing its tasks.
3. Each process has to know and state their maximum resource requirement in advance for the system.
4. The number of resource requests can be granted in a finite time, but the time limit for allocating the resources is one year.

Safety Algorithm

Safety algorithm in OS is used to mainly check whether the system is in a safe state or not and it follows a sequence of following steps to check that:

- **STEP 1 ->** With reference to the above C++ code that we have implemented, we always need to use a finish array of size n (for each process) that stores value 1 if all the resources have been allocated to a process otherwise 0 as resultant value. finish[i] = 0; **//It contains 0 value initially for i = 1,2,3,4.....n**
- **STEP 2 ->** Next, we need to check which process hasn't been allocated all the resources yet with the condition provided that (Need)i <= available for that process. **Note: If such an index or the process doesn't exist and needs resources from the available array, go to step 4 directly.**

- **STEP 3 ->** If resources have been allocated to a process, we need to update the available and the finish array. available = available + (allocation)i **//getting new resource allocation** Also, finish[i] = 1 //for currently allocated process **Note: Go to step 2 for checking the status of resource availability for the next processes.**
- **STEP 4 ->** If finish[i] = 1 **//for all i: 1,2,3,4,....n then the system is in safe state otherwise not**

Resource Request Algorithm

The resource request algorithm checks the behavior of a system whenever a particular process makes a resource request in a system. It mainly checks whether resource requests can be safely granted or not within the system. **This algorithm takes the following actions upon receiving a resource request from a process 'Pi':**

Let (Request) be the request array for process 'Pi' where Request[j] = k means process 'Pi' needs k instances of resource type 'Rj'.

- **STEP 1 ->** If (Request)i <= (need)i (indicating that the requested resources are less than equal to the need matrix resources for that process) **Go to STEP 2 immediately or otherwise, an error will be generated because a process is requesting more resources than the maximum resources that can be allocated to that process.**
- **STEP 2 ->** If (Request)i <= available (indicating that the requested resources are within the range of available resources currently present in the system) **Go to STEP 3 immediately or otherwise, process 'Pi' will have to wait because the required number of resources are not available at present.**
- **STEP 3 ->** Assuming that the request has been granted and resources have been allocated to the desired process, next we need to modify the states: available = available – (Request)i (allocation)i = (allocation)i + (Request)i (need)i = (need)i – (Request)i

ASSIGNMENT 9

Implement the process synchronization with the structure of reader and writer process

Writer process:

1. Writer requests the entry to critical section.
2. If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.
3. It exits the critical section.

Reader process:

1. Reader requests the entry to critical section.
2. If allowed:
   - it increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the **wrt** semaphore to restrict the entry of writers if any reader is inside.
   - It then, signals mutex as any other reader is allowed to enter while others are already reading.
   - After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore “wrt” as now, writer can enter the critical section.
3. If not allowed, it keeps on waiting.

The readers-writers problem is a classical problem of process synchronization, it relates to a data set such as a file that is shared between more than one process at a time. Among these various processes, some are Readers - which can only read the data set; they do not perform any updates, some are Writers - can both read and write in the data sets. The readers-writers problem is used for managing synchronization among various reader and writer process so that there are no problems with the data sets, i.e. no inconsistency is generated.

ASSIGNMENT 10

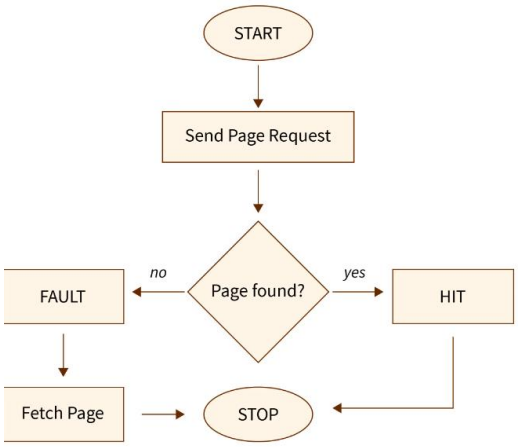
Implement the following page replacement algorithms ● FIFO ● LRU ● Optimal

FIFO

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

1. The OS maintains a list of all pages which are residing in the memory.
2. When a new page brings from the **secondary memory**.
3. The new page requests the **main memory**.
4. On a **page fault**, the head of the list i.e the oldest will be removed.
5. The new page will be added at the tail of the list.

Page Fault: A page fault occurs when a page requested by a program running in the CPU is not present in the main memory, but in the address page of that program. A page fault generally creates an alert for the OS.




```
# Declare 'S' as a set of current pages in the system.
# Declare a Queue name QPage which will be used to store the incoming pages.
# Queue will store in a FIFO manner
# page fault PF is Initially set to zero.
S= set();
QPage = Queue();
PF=0;
for k = 0 to length (N) do
        if length (S) < C then
                if P[k] not in S then
                        S.add(P[k]);
                        PF = PF + 1;
                        QPage.put(P[k]);
                end
        else
                if P[k] not in S then
                        val = QPage.queue[0];
                        QPage.get();
                        S.remove(val);
                        S.add(P[k]);
                        QPage.put(P[k]);
                        PF = PF + 1;
                end
        end
end
return PF;
```

Here,

- '**P**' is used to represent pages.
- '**N**' is the number of pages.

- '**C**' is the Capacity.

LRU

The primary purpose of any page replacement algorithm is to reduce the number of *page faults*. When a page replacement is required, the **LRU page replacement algorithm** replaces the least recently used page with a new page. This algorithm is based on the assumption that among all pages, the least recently used page will not be used for a long time. It is a popular and efficient page replacement technique.

PSEUDOCODE

1. Iterate through the referenced pages.
    - If the current page is already present in pages:
        1. Remove the current page from pages.
        2. Append the current page to the end of pages.
        3. Increment page hits.
    - Else:
        1. Increment page faults.
        2. If pages contains less pages than its capacity s:
            - Append current page into pages.
        3. Else:
            - Remove the first page from pages.
            - Append the current page at the end of pages.
2. Return the number of page hits and page faults.

Advantages of LRU Page Replacement Algorithm

Following are the advantages of the LRU page replacement algorithm:

- The page that has not been used for the longest time gets replaced.
- It gives lesser page faults than any other algorithm.
- The algorithm never suffers from the **Belady's anomaly** .
- The algorithm is capable of complete analysis.

OPTIMAL

"Optimal page replacement algorithm" is the most desirable page replacement algorithm that we use for replacing pages. This algorithm replaces the page whose demand in the future is least as compared to other pages from frames (secondary memory). The replacement occurs when the page fault appears. The purpose of this algorithm is to minimize the number of page faults. Also,

one of the most famous abnormalities in the paging technique is "Belady's Anomaly", which is least seen in this algorithm.

The **optimal page replacement algorithm** is used to reduce these page faults. It uses the principle that- "when a page is called by the system and it is not available in the frames, the frame which is not in demand for the longest future time is replaced by the new page".

Pseudocode:

```
1.predict() function

   Declare and initialize two variables result = -1, farthest = index
    Loop over i = 0 to i < frame.size()
        Loop over j = index to j < pageNumber-
            Check if frame[i] == page[j] then,
                if j > farthest
                    set farthest = j
            Set result = i
            break
        If j == pageNumber then,
           return i
    return (result == -1) ? 0 : result


2. boolSearch() function
   Loop over i = 0 to i < frame.size()
        if frame[i] == key then,
            return true
    return false


3. find() function
   Declare a vector called frame
    Set hits = 0
    Loop over i = 0 to i < pageNumber
    if search(page[i], frame) i.e. current page in found then,
        increment hits by 1
        continue
    if frame.size() < frameNumber then,
        frame.push_back(page[i])
    else
        Set j = predict(page, frame, pageNumber, i + 1)
        Set frame[j] = page[i]
        Print the number of hits
        Print the number of misses

4. main() function

    Declare and assign page array.
    Set pageNumber = sizeof(page) / sizeof(page[0])
    Set frameNumber
    find(page, pageNumber, frameNumber)
```

Advantages Of Optimal Page Replacement Algorithm:

Following are the advantages of the Optimal page replacement algorithm:

- Least page fault occurs as this algorithm replaces the page that is not going to be used for the longest time in the future.
- In this algorithm, Belady's Anomaly (the number of page faults increases when we increase the number of frames in secondary storage) does not occur because this algorithm uses a 'stack-based algorithm' for page replacement.
- Data structure used in this algorithm is easy to understand.
- The page that will not be in demand in the future time is replaced by a new page in the frames.