Design Programming Project Assignment 1
CSIS 3810

Design and develop a High-level program in JAVA to simulate the operation of the Shark Machine (see description below). Use a 1024-element array to simulate primary memory, where each word is 32-bits. Use variables to represent the registers (ACC, PSIAR, ..). Each micro-operation is to be simulated by a statement in the language of choice. For example, the first micro code instruction of the LOAD operation, add one to the PSIAR register and store it in the ACC register, would look like: acc = psiar + 1; in Java.

Create an END instruction that dumps the contents of all registers and memory, then prints an "End of Job" message. Additionally, you will create a system clock that will be the basis of timer used to provide timer runout interrupts and add a YLD (yield) instruction that provides an interrupt to the CPU for the current job to give up the CPU.

Next you must design and implement a rudimentary operating system SharkOS that provides multi-tasking capability. You will implement process entities (jobs) and a program loader that will allow your machine/OS to run several Shark machine language programs to be loaded at startup.

To accomplish the multi-tasking capability, you will define a Process Control Block (PCB) data structure that will be created for each job. These PCBs will be linked to a job queue that will function in a round-robin fashion.

You will define the notion of an interrupt handler that will process a time quantum for each running job and also to handle the YLD instruction by providing the handler function to context switch the current job (yield the CPU) and interact with the short-term scheduler to choose the next process to run. On each job switch, you will provide the option to print some state information such as which job will be loaded and the current state of the job queue. Your version of SharkOS will then run six or more

Shark machine language programs simultaneously using round-robin scheduling. These programs will test the ability of your simulation to handle multi-tasking in a round-robin scheduling discipline. Each job will initially be placed in an arrival queue that will serve as the workload generator for the system. The format of each job will be the SharkOS machine program name (the text file that holds the program data and statements to load in the system) and the arrival time of the job (jobs may arrive at various times ranging from time zero to some end time). For example: sharkosprog1.txt 0; sharkosprog2.txt 6; sharkosprog3.txt 89.

You must develop the following Shark machine language programs to be run on your Shark machine/SharkOS:

1) Write a program in the machine language of the Shark machine that will total the numbers stored in locations 100 to 109 and place the result in location 200.

2) Write a program in the machine language of the Shark that will decrement the value stored in location 201 (must be at least 10) by one until the result is zero. Store the result in location 202.

3) Write a program in the machine language of the Shark that will increment the value stored in location 301 by two until the value has been increased by 20. Store the result in location 302.

You will be responsible for submitting to Assignments a ZIP file that contains a design document, the source code of your machine simulation (this code must be appropriately commented & readable), an executable version of your machine simulation, and the output generated from your Shark machine code program.

The code must be Virus free and please indicate the target platform your simulation was written for, if applicable.

MICROPROGRAMMING/MACHINE DISCRIPTION

The following is a description of a machine called Shark that contains the following:

1024 32-bit words of memory.

Each Instruction consists of a 16-bit opcode and a 16-bit operand (storage address).

An ALU for performing mathematical operations.

Registers

ACC Accumulator; A 32-bit register involved in all arithmetic operations. One of the operands in each arithmetic operation must be in the Accumulator; the other must be in primary storage.

PSIAR Primary Storage Instruction Address Register; This 16-bit register points to the location in primary storage of the next machine language instruction to be executed.

SAR Storage Address Register; This 16-bit register is involved in all references to primary storage. It holds the address of the location in primary storage being read from or written to.

SDR Storage Data Register; This 32-bit register is also involved in all references to primary storage. It holds the data being written to or receives the data being read from primary storage at the location specified in the SAR.

TMPR Temporary Register; This 32-bit register is used to extract the address portion (rightmost 16-bits) of the machine instruction in the SDR so that it may be placed in the SAR. (No SDR to SAR transfer.)

CSIAR Control Storage Instruction Address Register; This register points to the location of the next micro-instruction (in control storage) to be executed.

IR Instruction Register; This register contains the current instruction being executed.

MIR Micro-instruction Register; This register contains the current micro-instruction being executed.

Register Transfers (REG is ACC, PSIAR, or TMPR):

SDR < REG
REG < SDR
SAR < REG

Primary Storage Operations:

READ Data from primary storage location named in the SAR is placed in the SDR.

WRITE Data in the SDR is placed in primary storage location named in the SAR.

Sequencing operations:

CSIAR < CSIAR + 1
CSIAR < decoded SDR
CSIAR < constant
SKIP < (add 2 to CSIAR if ACC=0; else add 1)

Operations involving the accumulator:

ACC < ACC + REG
ACC < ACC - REG
ACC < REG
REG < ACC
ACC < REG + 1

Instruction fetch:

(00) SAR < PSIAR
(01) READ
(02) IR < SDR
(03) CSIAR < decoded IR (OP CODE)
(04) SDR < decoded IR (Operand)

ADD (Opcode 10):

(10) TMPR < ACC
(11) ACC < PSIAR + 1
(12) PSIAR < ACC
(13) ACC < TMPR
(14) TMPR < SDR
(15) SAR < TMPR
(16) READ
(17) TMPR < SDR
(18) ACC < ACC + TMPR
(19) CSIAR < 0

SUB (Opcode 20):

(20) TMPR < ACC
(21) ACC < PSIAR + 1
(22) PSIAR < ACC
(23) ACC < TMPR
(24) TMPR < SDR
(25) SAR < TMPR
(26) READ
(27) TMPR < SDR
(28) ACC < ACC - TMPR
(29) CSIAR < 0

LOAD (LDA, Opcode 30):

(30) ACC < PSIAR + 1
(31) PSIAR < ACC
(32) TMPR < SDR
(33) SAR < TMPR
(34) READ
(35) ACC < SDR
(36) CSIAR < 0

STORE (Name STR, Opcode 40):

(40) TMPR < ACC
(41) ACC < PSIAR + 1
(42) PSIAR < ACC
(43) ACC < TMPR
(44) TMPR < SDR
(45) SAR < TMPR
(46) SDR < ACC
(47) WRITE
(48) CSIAR < 0

BRANCH (Name BRH, Opcode 50):

(50) PSIAR < SDR
(51) CSIAR < 0

COND BRANCH (Name CBR, Opcode 60):

(60) SKIP
(61) CSIAR = 64
(62) PSIAR = SDR
(63) CSIAR = 0
(64) TMPR = ACC
(65) ACC = PSIAR + 1
(66) PSIAR = ACC
(67) ACC = TMPR
(68) CSIAR = 0

Shark Machine Programming Language Description

Addition

Usage: ADD <address>

Where <address> holds the value to add to the accumulator.

Subtraction

Usage: SUB <address>

Where <address> holds the value to subtract from the

accumulator.

Load

Usage: LDA <address>

Where <address> holds the value to load in to the accumulator.

Store

Usage: STR <address>

Where <address> is the storage location for the contents of the accumulator.

Branch

Usage: BRH <address>

Where <address> is the target of the absolute branch.

Conditional Branch

Usage: CBR <address>

Where <address> is the target of an absolute branch if the accumulator is zero.