CISC 640 Nova Southeastern University

OS Problem Set

Introduction

The following machine description will provide the basis for this assignment. You will create a virtual machine/operating system for the machine described below that will accept programs in the target machine language. The details for this assignment are presented below following the machine description.

MICROPROGRAMMING/MACHINE DISCRIPTION

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

512 32-bit words of memory (memory is word addressable).

Each Instruction consists of a 16-bit opcode and a 16-bit operand.

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) SDR = decoded IR (Operand)
(04) CSIAR = decoded IR (OP CODE)

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) TMPR = ACC
(31) ACC = PSIAR + 1
(32) PSIAR = ACC
(33) ACC = TMPR
(34) TMPR = SDR
(35) SAR = TMPR
(36) READ
(37) ACC = SDR
(38) 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
(65) PSIAR = ACC
(66) ACC = TMPR
(67) CSIAR = 0

LOAD IMMEDIATE (LDI, Opcode 70):

(70) ACC = PSIAR + 1
(71) PSIAR = ACC
(72) ACC = SDR
(73) CSIAR = 0

SIMMAC 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.

Load Immediate

Usage: LDI number

Where number is 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.

Project Description

Design and implement a program to simulate the operation of the SIMMAC based on the descriptions above.

Add a HALT instruction that dumps the contents of all registers and memory and then prints an “End of Job” message.

The SIMMAC machine does not provide a mechanism for interrupts and does not provide a means for selecting the next job to run. Your project must address multi-tasking in a single queue of jobs without the access to machine interrupts. You will implement process entities (jobs) that will allow your machine to run several SIMMAC machine-language programs. In order to do this you will define a Process Control Block (PCB) data structure that will be created for each job in your system.

You will design and implement the ability for each running process to participate in a system level job switch. Thus the processes may execute instructions but may not execute entirely in batch mode (IE. The system may not allow any process to run in its entirety without switching at least once to another job). Thus your design and implementation must provide a mechanism for processes to surrender the CPU to another process and retain the ability to resume processing from where the process surrendered the CPU at some point in the future. This must all be accomplished without an interrupt clock or handler. Additionally, when a process surrenders the CPU you must design and implement a mechanism for the system to begin/resume executing a waiting process from the process pool.

Your version of SIMMAC will then run multiple SIMMAC machine language programs simultaneously. These programs will test the ability of your SIMMAC to handle multi-tasking and scheduling. You must design your system such that all SIMMAC machine programs are loaded from text files.

The SIMMAC must be designed to take command line arguments or to prompt for input files in addition to the previously specified items. By this mechanism, all SIMMAC language programs will be loaded. Since there is the LDI command, all data can be loaded into SIMMAC memory using SIMMAC programming statements.

You must develop SIMMAC language programs to be run on your SIMMAC machine. There must be at least three different programs that exercise the system and are of significant length to demonstrate the multitasking/scheduling ability of your system. You must clearly document your programs so that it is clear as to the logic and intent of each SIMMAC language program.

Each line of any SIMMAC program must have the following format:

Opcode Operand

You will be responsible for turning in the system design document, source code of your version of SIMMAC (this code must be appropriately commented & readable), an executable version of your SIMMAC, and the output generated from your SIMMAC programs running on your version of SIMMAC. These items are to be placed in ZIP format and submitted to the OS Problem Set Assignment in Canvas Assignments.