XSM
eXperimental String Machine
Version 1.0

Dr. K. Muralikrishnan
kmurali@nitc.ac.in
NIT Calicut

September 3, 2012

Contents

1 Introduction

1.1 Brief Machine Description

The machine simulator is known as Experimental String Machine (XSM). It is an interrupt driven uniprocessor machine. The machine handles data as 16 byte strings. A string is a sequence of characters terminated by '\0'. The length of a string is atmost 15 characters. The machine interprets a single character also as a string.

1.2 Components of the Machine

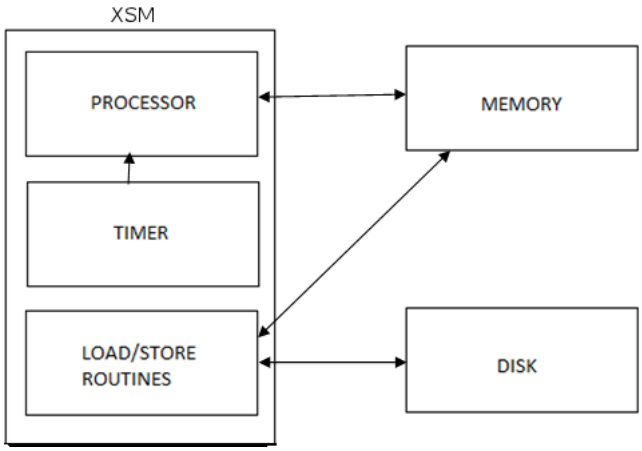

Figure 1: Components of the Machine

- Disk : It is a non-volatile storage that stores user programs (executables) and data files.
- Memory : It is a volatile storage that stores the programs to be run on the machine as well as the operating system that manages the various programs.
- Processor : It is the main computational unit that is used to execute the instructions.
- Timer : It is a device that interrupts the processor after a pre-defined specific time interval.

- Load/Store : It is a macro that performs the functionalities of DMA (Direct Memory Access) controller.

2 Registers

2.1 Introduction

The XSM architecture maintains 24 string registers.

2.2 Register Set

There are 16 General Purpose Registers (GPR), R0 - R15, of which R0 - R7 are Program Registers and R8 - R15 are Kernel Registers. There are 4 temporary registers T0 - T3 which are reserved for code translation. These registers cannot be used by the system programmer. In addition to these 20 registers there are 4 special purpose registers BP, IP, SP and PID which are used as Base Pointer, Instruction Pointer, Stack Pointer and Process Identifier respectively.

Name	Register
Program Register	R0-R7
Kernel Register	R8-R15
Temporary Registers	T0-T3
Base Pointer	BP
Instruction Pointer	IP
Stack Pointer	SP
Process Identifier	PID

3 Memory

3.1 Introduction

- The basic unit of memory in XSM is a 16 byte string.
- The machine memory can be thought of as a linear sequence of strings.
- A collection of 256 contiguous strings is known as a page.
- The total size of the memory is 64 pages or 16384 (256×64) strings.

Page no	Contents	Word addr
0	ROM code	0 – 255
1	OS Startup code	256 – 511
2	Static Page Tables	512 – 575
	Memory Free List	576 – 639
	Global File Table	640 – 735
	Ready Queue	736 – 751
	Unallocated	752 – 767
3	Process Table	767 – 1023
4	File Allocation Table	1024 – 1535
5		
6	Disk Free List	1536 – 2047
7		
8 – 55	⋮ User Programs ⋮	⋮ 1792 – 13823 ⋮
56	INT 0	13824 – 14079
57	INT 1	14080 – 14335
⋮	⋮	⋮
63	INT 7	15872 – 16127

Figure 2: Main Memory Block Diagram

- Each string in the memory is identified by the *string address* in the range 0 to 16383(256×64 - 1). Similarly, each page in the memory is identified by the *page number* in the range 0 to 63.

3.2 Page Table

Before explaining the page table, we explain two well known terms:

- Logical address : It is the CPU generated address of the data.
- Physical address : It is the exact location of the data in the main memory.

The page table contains information relating to the actual location in the memory, i.e., the physical address, of the data specified by the logical address. Each entry of a page table contains the page number in the memory where the data specified by the logical address resides.

3.3 Address Translation

It is the process of obtaining the physical address from the logical address. It is done by the machine in the following way.

1. The logical address generated by the CPU is divided by the page size (256) to get the logical page number.
2. The remainder got after performing the above division gives the offset within that page.
3. The logical page number is then used to index the page table to get the corresponding physical page number in the memory.
4. The offset got in step 2 is then used to refer to the word in the physical page containing the data.

3.4 Memory Free List

- The free list of the memory consists of 64 entries. Each entry is of size one word.
- The total size of the free list is thus 64 words (64 (= no. of entries) x 1 (= size of one entry) = 64 words).
- It is present in the second 64 words of page 2 of the memory.

- Each entry of the free list contains a value of either 0 or 1 indicating whether the corresponding page in the memory is free or not respectively.

4 File System

4.1 Introduction

Block : It is the basic unit of storage in the disk.
The disk can be thought of as consisting of a linear sequence of 512 blocks. The size of each block isequal to that of a page in the memory (256 words).

4.2 Disk Structure

The basic structure of the disk is shown in figure below.

0	1–8	9–10	11–12	13–511
OS Startup code	OS Code	Free List	FAT	Data Blocks

Figure 3: Structure of the disk

4.3 Addressing

Block number : Any particular block in the disk is addressed by the corresponding number in the sequence 0 to 511 known as the block number.

4.4 Disk Free List

- The Free List of the disk consists of 512 entries. Each entry is of size one word.
- The total size of the free list is thus 2 blocks or 512 words (512(= no. of entries) x 1(= size of one entry) = 512 words).
- It is present in blocks 9 and 10 of the disk.
- Each entry of the free list contains a value of either 0 or 1 indicating whether the corresponding block in the disk is free or not respectively.

4.5 File

A file is a collection of data identified by a name. Every file in the disk has a Basic Block and several Data Blocks. They are defined as follows:

- Data Blocks : These blocks contain the actual data of a file.
- Basic Block : It consists of information about the data of a file.Composed of Block List and Header.

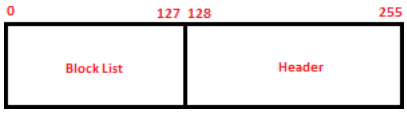

Figure 4: Structure of a Basic Block of a file

  - Block List : It is similar to an index in a book which tells which chapter starts from which page.The block list consists of 128 entries each of size one word.The value contained in an entry of the block list gives the block number of the corresponding data block in the disk.
  - Header : The header contains the header information relating to the file. Currently this is unused, but at a later stage can be used to store information such as file modification date/time, author of the file etc.

4.5.1 File Types

There are two types of files in the XSM architecture. They are:

1. Data files : These files contain data or information that is used by the programs. They can occupy a maximum of 129 blocks (1 basic block + 0 - 128 data blocks).
2. Executable files : These contain programs that the user wishes to run on the machine. They occupy 3 blocks (1 basic block + 2 data blocks) of the disk.

4.5.2 Executable File Format

It consists of the Code section which contains the actual code to be run on the machine. It can span upto a maximum of 2 blocks.

0	1–8	9–10	11–12	13–511
OS Startup code	OS Code	Free List	FAT	Data Blocks

Figure 5: Structure of a FAT entry

4.6 File Allocation Table (FAT)

File allocation table (FAT) is a table that has an entry for each file present in thedisk.

- FAT of the filesystem consists of 32 entries. Thus there can be a maximum of 32 files.
- Each entry is of size 16 words.
- Total size of the FAT is thus 512 words (32 (= number of entries) x 16(= size of one entry) = 512 words).
- It is a disk data structure and occupies block numbers 11 and 12 of the disk.

4.6.1 Structure of a FAT entry

1. File Name : It is an identification of a file. It can be of maximum 15 characters (and thus requires 1 word). Typical file names are student.txt, calc.sim.
2. File size : It indicates the number of words occupied by a file. It varies from 0 words to (128 x 256) words (depending upon the number of data blocks it has). It occupies one word in the FAT entry.
3. Block number of basic block : It contains the block number where the basic block of a file resides in the disk. It occupies one word in the FAT entry.

5 Process

5.1 Introduction

Process : Any program written by the user is run as a process by the kernel.

- The XSM architecture supports a maximum of 12 processes to be run at a time.
- Each process is allowed to occupy a maximum of 4 pages of the memory.

5.2 Process Structure

Any process in the memory has the following structure.

- Code Area : These are pages of the memory that contain the actual code to be run on the machine. It occupies 2 pages of the memory.
- Stack : This is the user stack used in program execution. It is used to pass arguments during function calls, storing activation record of a function etc. It occupies 2 pages of the memory and grows in the direction of increasing word address.

5.3 Registers Associated with a Process

- Every process is allotted a unique integer identifier in the range 0 to 15, known as the PID (Process Identifier) which is stored in the PID register. This register can be used as an operand in any instruction only when executing in the kernel mode.
- The word address of the currently executing instruction is stored in the IP (Instruction Pointer) register. This register can be used as an operand in any instruction only when executing in the kernel mode.
- The base address of the user stack is stored in the BP (Base Pointer) register.
- The address of the stack top is stored in the SP (Stack Pointer) register. Each process has its own set of values for the various registers.

5.4 Data Structures Associated with a Process

The following are the various data structures associated with a process. They are explained in the following subsections.

0	1	2	3	4–11	12–15
PID	BP	SP	IP	R0 – R7	Local File Table

Figure 6: Structure of Process Control Block

5.4.1 Process Control Block (PCB)

It contains data pertaining to the current state of the process. Note that the size of each PCB (Process Control Block) is 16 words.

5.4.2 The Page Table

The page table stores the exact location in the memory of the data related to a process.

- Each process has 4 entries in the page table.
  - zeroth entry corresponds to the first page of code area.
  - first entry corresponds to the second page of code area.
  - third and fourth entry corresponds to the stack.
- Each entry contains the page number where the data specified by the logical address resides in the memory.

5.4.3 Page Tables

The page tables of the 12 processes are stored in the first 48 words of page 2 of the memory.

- The size of each page table is 4 words (4(= no. of entries) x 1(= size of an entry)= 4 words).
- There are a total of 12 processes, thus accounting for the 48 words(12 X 4 words).
- The page tables are indexed by multiplying the PID of a process by the size of a page table to get the starting word address of the page table of that process.

5.4.4 Process Table

- The process table contains the PCB of each of the 12 processes (Each entry occupies 16 words).
- The page 3 of the memory contains the process table.
- There are a total of 12 processes, thus accounting for the 192 words (12 16 words).
- The process table is indexed by multiplying the PID of a process by the size of a PCB to get the starting word address of the PCB of that process.

6 Instructions

6.1 Introduction

The instructions provided by the XSM architecture can be classified into privileged and unprivileged instructions.

6.2 Classification

6.2.1 Unprivileged Instructions

1. MOV
    - Immediate Addressing:
      Syntax : MOV Ri, NUM/STRING
    - Register Addressing:
      Syntax : MOV Ri, Rj
      Source register - Ri, Destination register - Rj
    - Register Indirect Addressing:
      Syntax : MOV Ri, [Rj]
      Copy contents of memory location pointed by Rj to Ri.
      Syntax : MOV [Ri], Rj
      Contents of Rj are copied to the location whose address is in Ri.
    - Direct Addressing:
      Syntax : MOV [LOC], Rj
      Contents of Rj are transferred to the address LOC.
      Syntax : MOV Rj, [LOC]
      Contents of the memory location LOC are transferred to Rj.

2. Arithmetic Instructions

   - ADD, SUB, MUL, DIV and MOD.
     General Syntax : OP Ri, Rj
     The result of Ri op Rj is stored in Ri. If Ri and/or Rj are not integers, these instructions does no operation.
   - INR
     Syntax : INR Rj
     Increments the value of register Rj by one.
   - DCR
     Syntax : DCR Rj Decrements the value of register Rj by one.
     Here Ri, Rj may be any registers except SP, BP and IP.

3. Logical Instructions

   - LT
     Syntax : LT Ri, Rj
     Stores 1 in Ri if the value stored in Ri is less than that in Rj. Ri is set to 0 otherwise.
   - GT
     Syntax : GT Ri, Rj
     Stores 1 in Ri if the value stored in Ri is greater than that in Rj. Ri set to 0 otherwise.
   - EQ
     Syntax : EQ Ri, Rj
     Stores 1 in Ri if the value stored in Ri is equal to that in Rj. Set to 0 otherwise.Here content of Ri and Rj can be strings also.
   - NE
     Syntax : NE Ri, Rj
     Stores 1 in Ri if the value stored in Ri is not equal to that in Rj. Set to 0 otherwise.Here content of Ri and Rj can be strings also.
   - GE
     Syntax : GE Ri, Rj
     Stores 1 in Ri if the value stored in Ri is greater than or equal to that in Rj. Set to 0 otherwise.
   - LE
     Syntax : LE Ri, Rj
     Stores 1 in Ri if the value stored in Ri is less than or equal to

that in Rj. Set to 0 otherwise.
Here Ri, Rj may be any registers except SP, BP and IP.

4. Branching Instructions
   Branching is achieved by changing the value of the IP to the address of a specified LABEL.

   - JZ
     Syntax : JZ Ri, LABEL
     Jumps to LABEL if the contents of Ri is zero.
   - JNZ
     Syntax : JNZ Ri, LABEL
     Jumps to LABEL if the contents of Ri is not zero.
   - JMP
     Syntax : JMP LABEL
     Unconditional Jump to instruction specified at LABEL
     Here Ri can be any register except SP, BP and IP.

5. Stack Instructions

   - PUSH
     Syntax : PUSH Ri
     Increment SP by 1 and copy contents of Ri to the location pointed to by SP.
   - POP
     Syntax : POP Ri
     Copy contents of the location pointed to by SP into Ri and decrement SP by 1.
     For both these instructions Ri may be any register except IP.

6. Subroutine Instructions
   The CALL instruction copies the address of the next instruction to be fetched (IP + 1) on to the stack, and transfers control to the label specified. The RET instruction restores the IP value stored in the stack and continues execution fetching the next instruction pointed to by IP. The subroutine instructions provide a neat mechanism for procedure evocations.

   - CALL
     Syntax : CALL LABEL
     Increment SP by 1, transfers IP+1 to location pointed to by SP and jumps to LABEL

- RET
  Syntax : RET
  Sets IP to the value pointed to by SP and decrements SP.

7. Input/Output Instructions

   - IN
     Syntax : IN Ri
     Transfers the contents of the standard input to Ri.
   - OUT
     Syntax : OUT Ri
     Transfers the contents of Ri to the standard output.
     Ri can be any register except IP, BP and SP.

8. START
   IP will be initialised to this instruction automatically when a program is taken for execution.

9. END
   Syntax : END
   This instruction is put at the end of a user program.

10. INT
    Syntax : INT no
    This instruction generates an interrupt to the kernel with no as a parameter.

6.2.2 Privileged Instructions

There are four privileged instructions. They are:

1. IRET
   Syntax : IRET
   This instruction pops the return address from the user stack of the process which invoked the interrupt and assigns it to the IP register. IRET does the same action as RET, but it tells the processor that the interrupt handler has finished. With the execution of the IRET instruction, interrupts are enabled.

2. LOAD
   Syntax : LOAD pg_no block_no
   This instruction loads the block specified by the block_no, from the

disk, to the page specified by the pg_no, in the memory. block_no and page_no can be numbers or registers containing numbers.

3. STORE
   Syntax : STORE block_no pg_no
   This instruction stores the page specified by the pg_no, from the memory, to the block specified by the block_no, in the disk. block_no and page_no can be numbers or registers containing numbers.

4. HALT
   Syntax : HALT
   This instruction causes the simulator to halt immediately.

6.3 Processor Modes

The XSM architecture is interrupt driven and uses a single processor. There are two modes of operation, the user mode and the kernel mode.

- User mode : All unprivileged instructions can be executed in this mode.
- Kernel mode : Both privileged and unprivileged instructions can be executed in this mode. The processor comes to know about the mode in which the system is running by looking at the value in the IP register.

7 Interrupts

7.1 Introduction

Interrupts are mechanisms by which the user code interrupts the execution of the processor and passes control to the kernel to accomplish low level functionalities like disk access, arithmetic exception handling etc.
Interrupt Service Routine(ISR) : The kernel provides routines to accomplish the functionality for which an interrupt has been generated. These routines are known as Interrupt Service Routines.
Note: Every ISR should end with an IRET instruction.

7.2 The INT instruction

The instruction used to generate an interrupt is INT.
Syntax : INT n

The INT instruction passes control to the Interrupt Service Routine (ISR) for this interrupt located at the physical address computed using the value n. Address computation is done as follows. The physical address of the ISR corresponding to interrupt number n is given by: Physical Address = (56 + n) Page Size Note that the interrupts are disabled once this instruction is executed, since we do not allow interrupts to occur in kernel mode.

7.3 Types of Interrupts

There are 8 interrupts (numbered from 0 to 7) supported by the SSM architecture. The interrupts 0 and 7 are hardware interrupts and the remaining interrupts (1 to 6) are software interrupts.
Details of hardware interrupts are as follows.

- INT 0 : This is the timer interrupt which interrupts the processor forcing a context switch. It contains the code for the scheduler of the operating system, which schedules the CPU time among the various active processes. Note that this interrupt cannot be called from the user/kernel mode.
- INT 7 : It is generated when the following exceptions1 occur:
    1. Illegal memory access : occurs when any address generated by the process does not lie in the range [0, 959].
    2. Arithmetic exception : occurs when divisor is 0.
    3. Illegal instruction : occurs when an attempt is made to execute an instruction not belonging to the instruction set and also when the operands to the instruction is not legal. Eg: MOV 4 R0, MOV IP 4 when executed in user mode. These instructions are considered illegal.
    4. Stack overflow and stack underflow : Stack overflow occurs when the value in the SP register exceeds 959 and stack underflow occurs when the value falls below 768.
- INT 1, INT 2 : These interrupts are used for the various file system calls.
- INT 3, INT 4 : These interrupts are used for the various process system calls.

- INT 5, INT 6 : These interrupts have been kept reserved for future use.The interrupts 1, 2, 3 and 4 are unprivileged and can be called from user mode.