Dalhousie University
Department of Electrical and Computer Engineering

ECED 3403 – Computer Architecture

Assignment 1: The XM23 emulator

1 Objectives

XMC, the XM Corporation, has a new processor, XM23, which is an extension of the original XM processor with a revised Instruction Set Architecture (or ISA) and new instructions. The processor release date is mid-Fall 2023.

However, XMC is facing a serious problem. They need to test software on XM23 but the hardware is not ready. The decision has been taken to develop a software emulator for XM23 to examine how the new machine will operate under certain test conditions.

A C compiler has been promised, although that it not expected to be completed until later in the year. Fortunately, an XM23 assembler is available that can produce XM23 executables that can be loaded into the emulator.

The XM23 assembler takes XM23 assembly-language programs and produces S-Records for either the emulator or machine.

You have been asked to write the emulator because of your experience in software design and C-programming, and understand the importance of system testing.

2 The Emulator

Details XM23's ISA can be found in the XM23 Instruction Set Architecture document available from the course website.

The following parts of XM23's ISA are to be emulated: all CPU registers (R0-R7, R4/BP, R5/LR, R6/SP, and R7/PC), a subset of instructions (see below), the program status word (PSW), addressing modes, encoded constants, 64 KiB of byte-addressable primary memory, and CPU execution time.

Exceptions (interrupts, faults, and traps), exception-related instructions (`SETPRI`, SVC, `SETCC`, and `CLRCC`) and XM23's external devices (clock, keyboard, and console) are not to be implemented in this assignment. They will be addressed in future assignments.

The emulator is to be written in C.

2.1 Instruction Set Architecture

2.1.1 Central Processing Unit (CPU)

XM23 has a 16-bit CPU. It is responsible for fetching, decoding, and executing instructions. The internals of the CPU are to be emulated, including the memory address register, the memory data register, the control register, the instruction register, instruction decoding, and instruction execution.

2.1.2 Registers

XM23 has eight programmer-accessible registers: five are available for arithmetic and logical operations (R0 through R4) and the remaining three are specialized: R5 or LR (link register), R6 or SP (stack pointer),

and R7 or PC (program counter), respectively. Each register is 16-bits long and can be accessed as either a word or a byte.

In addition to its eight registers, XM23 has eight built-in constants, which can be treated like a value from a register in all arithmetic and logical instructions requiring a source register.

The registers and constants are to be stored in the CPU. They will need to be emulated as well.

2.1.3 Instructions

XM23 is a RISC with 41 instructions. Each instruction is 16-bits long and fits into a word. Instructions are fetched, decoded, and executed in a single instruction cycle. All instructions except SETPRI, SVC, SETCC, CLRCC, and CEX are to be emulated. The instructions to be emulated are listed in Table 1.

2.1.4 Instruction cycles

The instruction cycle must be emulated: instructions are *fetched* from memory, *decoded*, and *executed*. The emulator is to recognize each instruction and perform its function. Any status changes associated with the instruction as it is executing should be indicated in the PSW's status bits.

A system clock should be maintained keeping track of the number of instruction-cycles a program has used during its execution. The CPU clock is independent of the external clock.

Execution times are summarized in the following table:

Activity	**CPU clock cycles**
Accessing memory	3
Accessing CPU registers and constants	0
Fetch cycle	1
Decode cycle	1
Execute cycle	1

2.1.5 Bus and primary memory

The bus function should take four arguments: the contents of the memory address register, the address of the memory buffer register (to allow data to be returned), the value of the control register (read or write), and an indication whether a byte (1) or a word (0) is to be accessed (by rights, this should be part of the control register); for example:

```
/* To read: */
mar = effective_address;  /* Address to be read */
bus(mar, &mbr, READ, byte_word);
/* mbr has contents of location specified by 'address' */

/* To write: */
mar = effective_address; /* Address to be written */
mbr = data;
bus(mar, &mbr, WRITE, byte_word);
```

Primary memory will require special code (i.e., soft "circuitry") that supports the CPU reading and writing the memory reserved for devices (also referred to as *port memory* or *device ports*). XM23's 64 KiB of memory and the bus (software shown above) are to be emulated.

3 Support software

In addition to the XM23 emulator, you will need to write a simple debugger (in C) which should be part of the emulator. Some useful debugging tools are suggested here.

3.1.1.1 Loader

It will be necessary to include a loader as part of the XM23 debugger. The loader will take files containing S-records (see the X-Makina Assembler User's Guide) and load the contents of the file into the memory locations specified. The program counter must be initialized to the value in the S9 record. If the S9 record is omitted, the PC address should default to zero.

3.1.1.2 Running a program

In addition to the loader, it will be necessary to begin program execution at a specific memory location. The address can be supplied in an S9 record (see Loader, above) and execution explicitly started by the user. If it is possible to access the PC (register R7), the user could also change the starting address by writing to the PC and then starting the program.

3.1.1.3 Register display

The contents of the eight registers should be accessible to the user. At a minimum, it should be possible to display their contents. You might also consider letting the user change register values for testing purposes.

3.1.1.4 Memory display

Displaying the contents of a range of memory locations can be useful to verify, for example, that a load was successful or that a memory location was updated correctly.

3.1.1.5 Breakpoints

To facilitate debugging, supporting one or more breakpoints can be useful to let the user stop program execution at specific locations.

3.1.1.6 Aborting programs

To catch run-away programs, the user should be able to stop them using control -C. The software to handle control-C signal catcher will be provided and explained.

4 Marking

The assignment will be marked using the following marking scheme:

Design: The design description must include a brief introduction as to the problem being solved (1), a design section (5), and a data dictionary (2).

Total points: 8.

Software: A fully commented, indented, magic-numberless, tidy piece of software that meets the requirements described above and follows the design description.

Total points: 12.

Testing: In addition to any test software supplied as part of the assignment, you will be responsible for developing a minimum of five (5) distinct tests demonstrating that the software operates according

to the design description. Some of the tests should exercise the software. The tests must include the name of the test, its purpose or objective, the test configuration, and the test results.

Total points: 5.

All three sections are to be submitted electronically. The software and testing must be submitted on paper.

The emulator must be demonstrated before the software and testing will be marked.

5 Important Dates

Available: 5 May 2023

Design document due: 21 May 2023 at midnight, Atlantic.

Software and testing due: 11 June 2023 at midnight, Atlantic.

Demonstration: 13 or 14 June 2023. Your implementation must be demonstrated before the software and testing will be marked.

6 Miscellaneous

This assignment is to be completed individually.

If you are having *any* difficulty with this assignment or the course, *please* contact Dr. Hughes or Manav as soon as possible.

This assignment is worth 18% of your overall course grade.

This is a non-trivial assignment. It should be started as soon as it is made available.

Assignments that are found to be copies of work done by other students from either this year or pervious years will result in an immediate dismissal from this course.

Table 1: Required instructions

Mnemonic	Instruction	Mnemonic	Instruction
BL	Branch with Link	BIT	Bit test: DST & (1 << SCR/CON)
BEQ/BZ	Branch if equal or zero	BIC	Bit clear: DST ← DST & ~(1 << SRC/CON)
BNE/BNZ	Branch if not equal or not zero	BIS	Bit set: DST ← DST | (1 << SRC/CON)
BC/BHS	Branch if carry/higher or same (unsigned)	MOV	DST ← SRC
BNC/BLO	Branch if no carry/lower (unsigned)	SWAP	Swap SRC and DST
BN	Branch if negative	SRA	Shift DDD right (1 bit) arithmetic
BGE	Branch if greater or equal (signed)	RRC	Rotate DDD right (1 bit) through carry
BLT	Branch if less (signed)	COMP	Ones' complement of DDD
BRA	Branch Always	SWPB	Swap bytes in DDD
ADD	Add: DST ← DST + SRC/CON	SXT	Sign extend byte to word in DDD
ADDC	Add: DST ← DST + (SRC/CON + Carry)	LD	DST ← mem[SRC plus addressing]
SUB	Subtract: DST ← DST + (¬SRC/CON + 1)	ST	mem[DST plus addressing] ← SRC
SUBC	Subtract: DST ← DST + (¬SRC/CON + Carry)	MOVL	DST.Low byte ← BBBBBBBB; DST.High byte unchanged
DADD	Decimal add: DST ← DST + (SRC/CON + Carry)	MOVLZ	DST.Low byte ← BBBBBBBB; DST.High byte ← 00000000
CMP	Compare: DST – SRC/CON	MOVLS	DST.Low byte ← BBBBBBBB; DST.High byte ← 11111111
XOR	Exclusive OR: DST ← DST ⊕ SRC/CON	MOVH	DST.Low byte unchanged; DST.High byte ← BBBBBBBB
AND	AND: DST ← DST & SRC/CON	LDR	DST ← mem[SRC + sign-extended 7-bit offset]
OR	OR: DST ← DST | SRC/CON	STR	mem[DST + sign-extended 7-bit offset] ← SRC