Dalhousie University
Department of Electrical and Computer Engineering

ECED 3403 – Computer Architecture

Assignment 1: The XM23p Loader and Memories

Objectives

XMC, the XM Corporation, is facing a problem. XM23p is an extension of the original XM processor. It has the same Instruction Set Architecture (or ISA) and supports the same devices using a "traditional" von Neumann architecture. XM23p differs from the original in that it is a pipeline processor, with a Harvard architecture utilizing two separate memories, one for instructions and the other for data.

Before head office can decide whether to proceed with the XM23p project, it has been decided that an emulator must be written in software to examine how the pipelined XM23 ISA will operate under certain test conditions.

In addition to the emulation of the new computer, support software can be required if the computer has an entirely new architecture. This software can include new compilers, assemblers, linkers, and even operating systems.

The software available for XM23p consists of an assembler that takes XM23 assembly-language programs and produces S-Records for the yet-to-be-written XM23p emulator.

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.

The first part of the work is to design, implement, and test a loader, a Harvard architecture consisting of 64 KiB of code memory and 64 KiB of data memory and parts of a debugger.

Requirements

In this assignment, the loader is to be a standalone program loads instructions and data into XM23p's memories. In the next part, the loader will be incorporated as part of the debugger in the XM23p's emulator.

XM23p's Memories

Figure 1 shows XM23p's two memories: a 64 KiB word-addressable instruction memory (IMEM) and a 64 KiB byte-addressable data memory (DMEM).

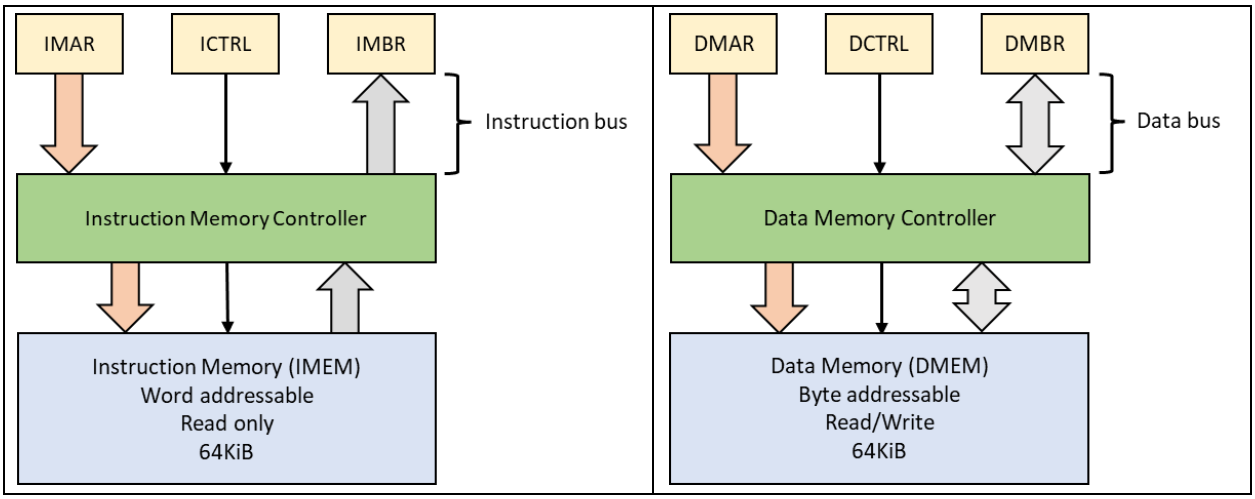

Figure 1: XM23p's Instruction Memory (left) and Data Memory (Right)

The memories are accessed through their respective memory registers:

xMAR. Instruction or Data Memory Address Register. The register takes a 16-bit address that is supplied to its memory controller. It is write-only.

xCTRL. Instruction or Data Control register. The CTRL register informs the memory controller how the contents of the address are to be accessed. Instruction memory can only be read as 16-bit words. Data memory can be read or written as a byte or a word.

xMBR. Instruction or Data Memory Buffer Register. The IMBR is read-only and contains the 16-bit instruction of the memory location specified in the IMAR. The DMBR is read/write and can contain an eight- or sixteen-bit data value to be read from, or written to, the address specified in the DMAR.

The three registers are connected to their respective Memory Controllers through a bus. The basis structures of the buses are the same; however, the data flow between the Instruction Memory Controller and the IMDR is unidirectional, whereas the flow is bidirectional between the Data Memory Controller and the DMBR.

Some common points to note regarding memory:

- Word addresses *always* fall on an even byte boundary. A word address can be converted to a byte address by shifting it to the left by 1.
- Byte addresses can fall on either an even or odd byte boundary. A byte address can be converted to an even word address by shifting it to the right by 1.
- All memory sizes are expressed in terms of bytes. This is true for both Instruction memory (accessed as 16-bit words) or Data memory (accessed as 8-bit bytes or 16-bit words).
- A KiB or kilo-bit byte refers to 1024 bytes as opposed to 1000 bytes (KB). In the XM series, both Instruction and Data memory are 64 KiB in length.

The Loader and S-Records

The loader is a software module that loads executable files into a machine's memory for subsequent execution.

The XM23p loader is to load XM23p executable files in the emulator's IMEM and DMEM. XM23p executables (denoted by having an extension of .XME) are produced by the XM23p assembler. The executable file consists of one or more S-records: (the records can be in any order, although S0 and S9 are usually first and last, respectively):

S0: The header record containing the name of the .ASM file from which the executable was obtained.

S1: The S1 record contains bytes to be written to the emulator's IMEM.

S2: The S2 record contains bytes to be written to the emulator's DMEM. This is XM23p's version of the S2 record. It has the same format as the S1 record.[1]

S9: This record contains the starting address of the executable file in the machine's IMEM.

A detailed description of the S-Record structure and how to interpret it can be found in the *XM23p Assembler User's Guide*.

Displaying memory contents

In this assignment, the loader is to be a standalone program containing the IMEM and DMEM. The implementation is to show how data can be loaded into the correct memory. This will also require the ability to display the contents of memory. In the existing XM23 emulator, memory is displayed as the byte value of the location and the ASCII equivalent. The display routine (shown in Figure 2) only displays XM23's single memory shared by both instructions and data (the XM23p's memory display would need to distinguish between instruction and data memory).

```
Option: m 800 820
Enter lower and upper bound
0800: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0810: 00 00 00 00 00 00 00 00 00 00 23 00 00 00 00 00 ..........#.....
Option: m 1000 1016
Enter lower and upper bound
1000: 08 6a d1 6a 02 60 42 78 c2 5c 41 45 02 20 c8 40 .j.j.`Bx.\AE. .@
1010: fb 3f 51 69 fe 3f 00 00 00 00 00 00 00 00 00 00 .?Qi.?..........
```

Figure 2: Example of displaying a program in XM23's memory
'm' denotes display memory from a lower bound (0x1000) to an upper bound (0x1016)

The output shown in Figure 2 came from the following XM23 single-memory executable.

```
S0100000417272617949 6E69742E61736DED
S104080000F3
S104081A23B6
S1191000086AD16A02604278C25C41450220C840FB3F5169FE3F0E
S9031000EC
```

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 (2), and a data dictionary (1).

[1] *Note*: The original S2 record specification stipulates that its address field is a "3-byte [24-bit] address". For the purposes of this course, we are opting to ignore the specification and define it as a two-byte (16-bit) address.

Total points: 4.

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

Total points: 6.

Testing: In addition to any test software supplied as part of the assignment, you will be responsible for developing a minimum of four (4) 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: 2.

All three sections are to be submitted electronically.

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

Important Dates

Available: 9 May 2024

Design document due: 15 May 2024 at midnight, Atlantic.

Software and testing due: 23 May 2024 at midnight, Atlantic.

Demonstration: 23 May 2024. Your implementation must be demonstrated before the software and testing will be marked.

Late submissions will be penalized 1 point every 24 hours.

Miscellaneous

This assignment is to be completed individually.

The loader can write to both the IMEM and DMEM address spaces. Since the S1 and S2 records are stored as bytes, it can be simpler to write the record contents as bytes.

The work is to be implemented in C. As always, it is recommended that you use Visual Studio for this assignment.

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

This assignment is worth 5% 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 previous years, or has used AI will result in an immediate dismissal from this course.