Assignment 3
Implementation of the XM23p's Instructions to access Data Memory
Design Document

Prepared for: Dr. Larry Hughes

Abdulla Sadoun B00900541

Table of Contents

Problem Introduction

Statement of Purpose

The purpose of this assignment is to design the implementation of a program written in C that would emulate the XM23p's execution of the instructions in charge of accessing data in Dmem, as the XM23p is a load-store machine. I will primarily focus on the implementation of the LD, LDR, ST, and STR instructions from the XM23p's ISA. The descriptions and format for the above mentioned instructions are highlighted below in the following figure.

0	1	0	1	1	0	PRPO	DEC	INC	W/B	S	S	S	D	D	D	LD	DST = mem[SRC plus addressing]
0	1	0	1	1	1	PRPO	DEC	INC	W/B	S	S	S	D	D	D	ST	mem[DST plus addressing] = SRC
1	0	OFF	OFF	OFF	OFF	OFF	OFF	OFF	W/B	S	S	S	D	D	D	LDR	DST = mem[SRC + sign-extended 7-bit offset]
1	1	OFF	OFF	OFF	OFF	OFF	OFF	OFF	W/B	S	S	S	D	D	D	STR	mem[DST + sign-extended 7-bit offset] = SRC

Objectives

The objective of this assignment is to design, implement and test the LD, ST, LDR and STR instructions within the emulator. Their functionality will be implemented according to the description mentioned above. As XM23p utilizes a harvard architecture unlike its previous XM23 predecessor, this means it will be using DMEM to get its data.

In this assignment I will create the design for the data memory instructions as an extension to the loader, debugger and IMEM instructions designed, implemented and tested in the previous labs and assignments.

The initial loader part section of the code will be disregarded and ignored for this assignment.

Since XM23p is an updated XM23, it must be noted that the new feature added where the processor fetches the next instruction while executing the current instruction simultaneously is still in place and will be put into consideration when creating the design for this Assignment and set of DMEM instructions. This approach to fetching, decoding and executing significantly lowers the amount of clock cycles that the processor has to go through to run code/programs.

Software Design

Data Dictionary

s-record = 's' + type + length of record + Address + Data

Type = [0|1|2|9]
Length of record = Byte Pair
Address = 2[Byte Pair]2
Address = 0000-ffff
Data = 1[Byte Pair]30
Byte Pair = character + character
Character = [0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F]

Registers[RegisterNo][BitNo] = [General Purpose Registers|Special Purpose Registers],[Bit Number]
General Purpose Register = [R0|R1|R2|R3|R4]
Special Purpose Registers = [PC|SP|LR]
R0 = [‘0’| ‘000’]
R1 = [‘1’| ‘001’]
R2 = [‘2’| ‘010’]
R3 = [‘3’| ‘011’]
R4 = [‘4’| ‘100’]
LR = 5
SP = 6
PC = 7

Instruction = Opcode + Operand
Opcode = 4{bit}13
Bit = [0|1]

Operand = [RC|WB|Source|Destination|Byte]
RC = [Register|Constant]
Register = 0
Constant = 1

WB = [Word|byte]
Word = 2{byte}2
Byte = 8{bit}8

Source = [R0-R4] *in bits* = 000-100
Destination = [R0-R4] *int bits* = 000-100

DMEM Instructions = [LD|ST|LDR|STR]
LD = LD_Opcode_bits + PRPO_bit + DEC_bit + INC_bit + WB_bit + SRC + DST
LD_Opcode_bits = “0”+“1”+“0”+”1”+”1”+”0”
ST = ST_Opcode_bits + PRPO_bit + DEC_bit + INC_bit + WB_bit + SRC + DST
ST_Opcode_bits = “0”+“1”+“0”+”1”+”1”+”1”
PRPO_bit = [0|1] # indicates whether pre or post increment/decrement action

DEC_bit = [0|1] # 0=no decrement pre/post 1=decrement pre/post
INC_bit = [0|1] # 0=no increment pre/post 1=increment pre/post
WB_bit = [0|1] # indicates whether data used is a word or a byte
SRC = [R0-R4] *in bits* = 000-100
DST = [R0-R4] *int bits* = 000-100

LDR = LDR_Opcode_bits + OFF_bits +WB_bit + SRC + DST
LDR_Opcode_bits = “1”+“0”
STR = STR_Opcode_bits + OFF_bits + WB_bit + SRC + DST
STR_Opcode_bits = “1”+“1”
OFF_bits = 7{bit}7
Bit = [0|1]
WB_bit = [0|1] # indicates whether data used is a word or a byte
SRC = [R0-R4] *in bits* = 000-100 # indicates number of source register
DST = [R0-R4] *int bits* = 000-100 # indicates number of destination register

Pseudo Code:

Header file:

- Include standard libraries

Global Variables:
- Char array: IMEM[MEMORY_SIZE]
- Char array: DMEM[MEMORY_SIZE]
- Char program counter (int): PC
- Define global int starting address (int): Start_Address
  # This is set by the Send2IMEM function in the loader when loading S-records to IMEM.

Registers:
- Define Registers as 2D array: Char RegistersBinary[8][16]
- Define Registers values as an array: Char RegisterValue[8]
  # Where 8 is the number of registers and 16 is the amount of bits.

Internal instruction numbering system enum:
- ENUM for instructions: {BL, BEQBZ, BNEBNZ, BCBHS, BNCBLO, BN, BGE, BLT, BRA, ADD, ADDC, SUB, SUBC, DADD, CMP, XOR, AND, OR, BIT, BIC, BIS, MOV, SWAP, SRA, RRC, SWPB, SXT, SETPRI, SVC, SETCC, CLRCC, CEX, LD, ST, MOVL, MOVLZ, MOVLS, MOVH, LDR, STR, Error}

Time Counter:
- Int timecount <- 0

Implementation: fetch.c (same as A2)

- Include Header file
FUNCTION Process Instruction:
    WHILE (TRUE)
    Call function fetch

    Call function decode

    Call function execute (decode value/internal execute number)
    Call function fetch
        IF instruction is not equal to 0000 and PC is not equal to breakpoint)
            BREAK
        End IF
    END WHILE
END FUNCTION

FUNCTION Fetch:
    IMAR = IMEM[PC]
END FUNCTION

Implementation: decode.c (same as A2)

- Include Header file

DEFINE LTCASE 0x7
DEFINE STCASE 0x6

FUNCTION Decode:
    ENUM {BLCase, BEQtoBRA, ADDtoST, MOVLtoMOVH, LLDR, LSTR}
    SET opcode = (IMARValue >> 13) & 0x07 # get first 3 bits

    SWITCH (opcode)
      CASE BLCase:
        RETURN BL
      CASE BEQtoBRA:
        RETURN betweenBEQandBRA(IMARValue)
      CASE ADDtoST:
        RETURN betweenADDandST(IMARValue)
      CASE MOVLtoMOVH:
        RETURN betweenMOVLandMOVH(IMARValue)
      CASE LLDR:

```
            RETURN LDR
        CASE LSTR:
            RETURN STR
        DEFAULT:
            PRINT "Error - instruction not yet implemented"
            RETURN Error
    END SWITCH
END FUNCTION


FUNCTION betweenADDandST (IMARValue):

    SET opcode = (IMARValue >> 10) & 0x07 # layer 2 opcode
    SET prpobuff = (IMARValue >> 9) & 0x01
    SET decbuff = (IMARValue >> 8) & 0x01
    SET incbuff = (IMARValue >> 7) & 0x01
    SET wbbuff = (IMARValue >> 6) & 0x01
    SET srcbuff = (IMARValue >> 3) & 0x07
    SET dstbuff = IMARValue & 0x07

    IF opcode equals LTCASE OR opcode equals STCASE
        SET opcode = (IMARValue >> 10) & 0x01 #  layer 3 opcode
        IF opcode equals SUB_LD
            RETURN LD
        ELSE
            RETURN ST
        END IF

    ELSE - CONTINUE PROCESSING ADD to CEX..

END FUNCTION
```

Other functions are irrelevant to this assignment

Implementation: execute.c (changes in LD-ST & LDR-STR)

- Include Header file

```
FUNCTION execute(int instruction number)
    SWITCH (instruction number)
        CASE BL:
            # execution for BL
                BREAK
        CASE “INSTRUCTION”:
```

```
    # execution for “INSTRUCTION”
        BREAK
CASE LD:
    PRINT "LD: PRPO:", prpobuff, " DEC:", decbuff, " INC:", incbuff, " WB:", wbbuff, "
    SRC:", srcbuff, " DST:", dstbuff
    IF prpobuff equals PRE
            IF incbuff equals SET
                    IF wbbuff equals WORD SET
                            EA = RegistersValue[srcbuff] + 2
                            RegistersValue[dstbuff] = DMEM[EA]
                    ELSE
                            SET EA = RegistersValue[srcbuff] + 1
                            RegistersValue[dstbuff] = DMEM[EA]
                    END IF
            IF decbuff equals SET
                    IF wbbuff equals WORD SET
                            EA = RegistersValue[srcbuff] - 2
                            RegistersValue[dstbuff] = DMEM[EA]
                    ELSE
                            SET EA = RegistersValue[srcbuff] - 1
                            RegistersValue[dstbuff] = DMEM[EA]
                    END IF
            END IF
    ELSE IF incbuff equals SET
            IF wbbuff equals WORD
                    SET EA = RegistersValue[srcbuff]
                    RegistersValue[dstbuff] = DMEM[EA]
                    EA = RegistersValue[srcbuff] + 2
            ELSE SET EA = RegistersValue[srcbuff]
                    RegistersValue[dstbuff] = DMEM[EA]
                    EA = RegistersValue[srcbuff] + 1
            END IF
    IF decbuff equals SET
            IF wbbuff equals WORD
                    SET EA = RegistersValue[srcbuff]
                    RegistersValue[dstbuff] = DMEM[EA]
                    EA = RegistersValue[srcbuff] - 2
            ELSE
                    SET EA = RegistersValue[srcbuff]
                    RegistersValue[dstbuff] = DMEM[EA]
                    EA = RegistersValue[srcbuff] - 1
            END IF
    END IF
```

```
END IF
        BREAK

    CASE ST:
        PRINT "ST: PRPO:", prpobuff, " DEC:", decbuff, " INC:", incbuff, " WB:", wbbuff, "
        SRC:", srcbuff, " DST:", dstbuff
        IF prpobuff equals PRE
                IF incbuff equals SET
                        IF wbbuff equals WORD SET
                                EA = RegistersValue[srcbuff] + 2
                                RegistersValue[dstbuff] = DMEM[EA]
                        ELSE
                                SET EA = RegistersValue[srcbuff] + 1
                                RegistersValue[dstbuff] = DMEM[EA]
                        END IF
                IF decbuff equals SET
                        IF wbbuff equals WORD SET
                                EA = RegistersValue[srcbuff] - 2
                                RegistersValue[dstbuff] = DMEM[EA]
                        ELSE
                                SET EA = RegistersValue[srcbuff] - 1
                                RegistersValue[dstbuff] = DMEM[EA]
                        END IF
                END IF
        ELSE IF incbuff equals SET
                IF wbbuff equals WORD
                        SET EA = RegistersValue[srcbuff]
                        RegistersValue[dstbuff] = DMEM[EA]
                        EA = RegistersValue[srcbuff] + 2
                ELSE SET EA = RegistersValue[srcbuff]
                        RegistersValue[dstbuff] = DMEM[EA]
                        EA = RegistersValue[srcbuff] + 1
                END IF
        IF decbuff equals SET
                IF wbbuff equals WORD
                        SET EA = RegistersValue[srcbuff]
                        RegistersValue[dstbuff] = DMEM[EA]
                        EA = RegistersValue[srcbuff] - 2
                ELSE
                        SET EA = RegistersValue[srcbuff]
                        RegistersValue[dstbuff] = DMEM[EA]
                        EA = RegistersValue[srcbuff] - 1
                END IF
```

```
            END IF
        END IF
            BREAK

        CASE LDR:
            Print:"LDR: WB:{wbbuff} SRC:{srcbuff} DST:{dstbuff}"
            IF (offbuff >> msb) & 0x01 equals 1
                offbuff = offbuff OR ((0xFFFF) << msb)
            END IF

            SET EA =RegistersValue[dstbuff] + offbuff
            BREAK

        CASE STR:
            Print:"STR: WB:{wbbuff} SRC:{srcbuff} DST:{dstbuff}"
            IF (offbuff >> msb) & 0x01 equals 1
                offbuff = offbuff OR ((0xFFFF) << msb)
            END IF

            SET EA =RegistersValue[dstbuff] + offbuff
            BREAK

        CASE Error:
           PRINT "Error - instruction not yet implemented"
            BREAK
        DEFAULT:
           PRINT "Error - instruction not yet implemented"
            BREAK
        RETURN
    END SWITCH
END FUNCTION
```

Main: (same as A2)

How to use/run the software:

To run the program, since it is written entirely in C, any machine with a gcc/gnu compiler can be used in any machine.
First ensure you have all the files in one directory or folder to be able to run this program, the files are the loader.h file, loader.c and main.c files. Navigate to that directory using the terminal using “cd <directory>”. Once in the correct directory use “gcc -o main main.c debugger.c decode.c execute.c fetch.c loader.c xm23p.c” to compile the program and create the “.o” file

named loader, now run loader using the following command “./emulator”. You should see the command window pop up and you would be able to use the menu to perform different functions. To enter the debugger mode and test the new functions, select the “d” option.

Implementation:

Included in the submission zip.
Changes mainly made in fetch.c, decode.c, and execute.c

Testing

Test 1: Testing LD instruction

Purpose/Objective: The purpose of this test is to check if the LD function is working properly as it should (loading the value from DMEM with the source register containing the address into the destination register).
Test Configuration: I have started the emulator loading the .xme file for the LDST1 program that I have created to test the LD and ST instructions and their functionality (included in Extra content and notes section below). The program starts defining a word in Data Memory with the value 0xFFFF, it then loads it from memory in r1.

```
Enter filename: LDST1.xme
LDST1.asm was loaded succefully
==========MENU==========
l - Load file
m - Print memory
f - Fetch (BETA)
d - Debug (BETA)
q - Quit
Enter choice: m
select Memory I=IMEM D=DMEM B=both
b
Select IMEM range: 0150 0250
Select DMEM range: 0000 0100
IMEM:
0150: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0170: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01f0: 00 00 00 00 00 00 00 00 60 80 78 00 58 01 40 00
0210: 5C 08 00 9B 00 00 00 00 00 00 00 00 00 00 00 00
0230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
DMEM:
0000: 00 00 00 00 00 00 00 00 FF FF EC 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
==========MENU==========
```

Initial IMEM and DMEM before executing the code

Expected Results: The program should work as expected loading the value from DMEM into r1.

Actual Results: The program ran as expected, getting the value from DMEM and saving it into r1 and then using r1 to store the data in a different location.

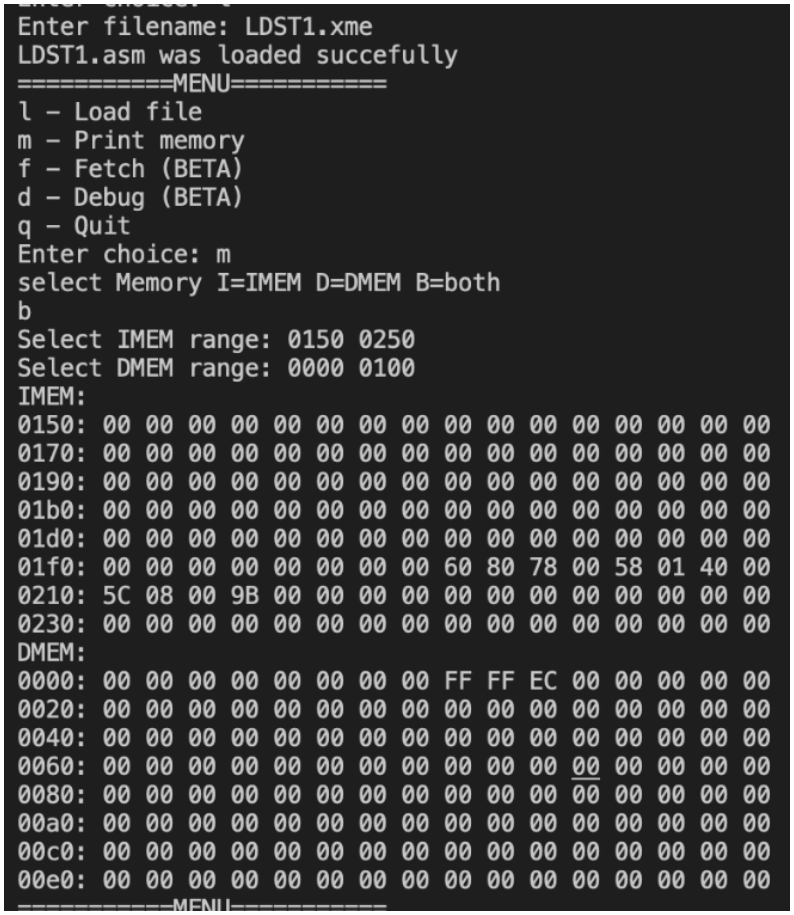
```
Breakpoint reached or the end of exec.
==========MENU==========
l - Load file
m - Print memory
f - Fetch (BETA)
d - Debug (BETA)
q - Quit
Enter choice: m
select Memory I=IMEM D=DMEM B=both
b
Select IMEM range: 0000 0100
Select DMEM range: 0000 0100
IMEM:
0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
DMEM:
0000: 00 00 00 00 00 00 00 00 FF FF EC 00 00 00 00 00
0020: FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
==========MENU==========
```

Pass/Fail: Pass

Test 2: Testing ST instruction

Purpose/Objective: The purpose of this test is to check if the ST function is working properly as it should (storing the value into DMEM with the source register containing the address, from the destination register).

Test Configuration: I have continued running the previous program (LDST1.xme included in Extra content and notes section below) using the emulator. The program starts defining a word in Data Memory with the value 0xFFFF, it then loads it from memory in r1 using the memory address in r0. It then changed the value of r0 by adding it to itself and uses this new address to store the value of r1 again into the new data memory address in r0.



Initial IMEM and DMEM before executing the code

Expected Results: The program should work as expected storing the word (0xFFFF) into the new memory location in DMEM.

Actual Results: The program ran as expected, getting the value from DMEM and saving it into r1 and then using r1 to store the data in a different location.

```
Breakpoint reached or the end of exec.
==========MENU==========
l - Load file
m - Print memory
f - Fetch (BETA)
d - Debug (BETA)
q - Quit
Enter choice: m
select Memory I=IMEM D=DMEM B=both
b
Select IMEM range: 0000 0100
Select DMEM range: 0000 0100
IMEM:
0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
DMEM:
0000: 00 00 00 00 00 00 00 FF FF EC 00 00 00 00 00
0020: FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
==========MENU==========
```

Pass/Fail: Pass

Test 3: Testing LDR instruction

Purpose/Objective: The purpose of this test is to check if the LDR function is working properly as it should (loading the value from DMEM with the source register containing the address + the offset from the address into the destination register).

Test Configuration: For this test I have edited the file I have created for the first 2 tests (LDST1.asm) now it uses LDR and STR instead and does not need the add function to create the offset, it just offsets using the instruction and loads from a location or stores in a different location. The content of this file (LDRSTR.lis & LDRSTR.xme) are attached below and in the submission zip.

```
Enter choice: l
Enter filename: LDRSTR.xme
LDRSTR.asm@ was loaded succefully
===========MENU===========
l - Load file
m - Print memory
f - Fetch (BETA)
d - Debug (BETA)
q - Quit
Enter choice: m
select Memory I=IMEM D=DMEM B=both
b
Select IMEM range: 0 0
Select DMEM range: 0 100
IMEM:
DMEM:
0000: 00 00 00 00 00 00 00 00 FF FF EC 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
===========MENU===========
```

Expected Results: In the output we should see the values in DMEM repeated in another location in memory due to the effect of the LDR and STR instructions.
Actual Results: The program executed as expected getting the value from memory into R1.

```
Breakpoint reached or the end of exec.
==========MENU==========
l - Load file
m - Print memory
f - Fetch (BETA)
d - Debug (BETA)
q - Quit
Enter choice: m
select Memory I=IMEM D=DMEM B=both
b
Select IMEM range: 0 0
Select DMEM range: 0 100
IMEM:
DMEM:
0000: 0F FF F0 00 00 00 00 00 FF FF EC 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
==========MENU==========
```

Pass/Fail: Pass

Test 4: Testing STR instruction

Purpose/Objective: The purpose of this test is to check if the STR function is working properly as it should (storing the value into DMEM with the source register containing the address + the offset from the address, from the destination register).
Test Configuration: For this test I have continued executing the LDRSTR.xme program to see if the STR function works by placing the content of r1 in a different address in DMEM.

```
Enter choice: l
Enter filename: LDRSTR.xme
LDRSTR.asm@ was loaded succefully
===========MENU===========
l - Load file
m - Print memory
f - Fetch (BETA)
d - Debug (BETA)
q - Quit
Enter choice: m
select Memory I=IMEM D=DMEM B=both
b
Select IMEM range: 0 0
Select DMEM range: 0 100
IMEM:
DMEM:
0000: 00 00 00 00 00 00 00 00 FF FF EC 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
===========MENU===========
```

Expected Results: In the output we should see the values in DMEM repeated in another location in memory due to the effect of the LDR and STR instructions.
Actual Results: The program executed as expected storing the content of R1 into a different address in memory due to the offset introduced.

```
Breakpoint reached or the end of exec.
===========MENU===========
l - Load file
m - Print memory
f - Fetch (BETA)
d - Debug (BETA)
q - Quit
Enter choice: m
select Memory I=IMEM D=DMEM B=both
b
Select IMEM range: 0 0
Select DMEM range: 0 100
IMEM:
DMEM:
0000: 0F FF F0 00 00 00 00 00 FF FF EC 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
===========MENU===========
```

Pass/Fail: Pass

Extra Content and Notes

LDST1.lis (file used)

```
X-Makina Assembler - Version XM-23P Single Pass+ Assembler - Release 24.04.17
Input file name: LDST1.asm
Time of assembly: Thu 11 Jul 2024 21:32:37
 1                              ;
 2                              ; Example of direct addressing
 3                              ; Load and store
 4                              ;
 5                              ; V1 <- V1 + 2
 6                              ;
 7                                      data
 8                                      org     #10
 9      0010    FFFF    V1      word    #FFFF
10                              ;
11                                      code
12                                      org     #200
13                              LDST1
14                              ;
15                              ; Get address of V1 into R0
16                              ;
17      0200    6080                    movl    V1,R0           ; R0 = #??00
18      0202    7800                    movh    V1,R0           ; R0 = #0010
19                              ;
20                              ; R1 <- mem[R0]
21                              ;
22                              ; R0 contains the effective address
23                              ; mem[R0] is rvalue (a value or expression)
24                              ; R1 is lvalue (where the result is to be stored)
25                              ;
26                              ; LD from-memory (rvalue),to-register (lvalue)
27                              ;
28      0204    5801                    LD      R0,R1           ; R1 <- #FFFF
29                              ;
30      0206    4000                    ADD     R0,R0           ; R0 <- R0 + R0
31                              ;
32                              ; mem[R0] <- R1
33                              ;
34                              ; R1 is rvalue
35                              ; R0 is the effective address
36                              ; mem[R0] is lvalue
37                              ;
38      0208    5C08                    ST      R1,R0
39                              ;
40                              ; Put breakpoint on next address
41                              ;
42                              BrkPtHere
43                              ;
44                                      end LDST1

Successful completion of assembly - 1P

** Symbol table **

Constants (Equates)
```

Name	Type	Value	Decimal	

Labels (Code)

Name	Type	Value	Decimal	
BrkPtHere	REL	020A	522	PRI
LDST1	REL	0200	512	PRI

Labels (Data)

Name	Type	Value	Decimal	
V1	REL	0010	16	PRI

Registers

Name	Type	Value	Decimal	
R7	REG	0007	7	PRI
R6	REG	0006	6	PRI
R5	REG	0005	5	PRI
R4	REG	0004	4	PRI
R3	REG	0003	3	PRI
R2	REG	0002	2	PRI
R1	REG	0001	1	PRI
R0	REG	0000	0	PRI

.XME file: \\Mac\Home\Desktop\Computer Architecture\Assembler\LDST1.xme

LDST1.xme (file used)

```
S00C00004C445354312E61736D1C
S2050010FFFFEC
S10D020080600078015800400085C9B
S9030200FA
```

LDRSTR1.lis (file used)

```
X-Makina Assembler - Version XM-23P Single Pass+ Assembler - Release 24.04.17
Input file name: LDRSTR.asm
Time of assembly: Fri 12 Jul 2024 00:57:37
 1
 2                                  data
 3                                  org       #10
 4        0010      FFFF    V1      word      #FFFF
 5                          ;
 6                                  code
 7                                  org       #200
 8                          LDST1
 9                          ;
10                          ; Get address of V1 into R0
11                          ;
12                          ;       movl      V1,R0           ; R0 = #??00
13                          ;       movh      V1,R0           ; R0 = #0010
14                          ;
15                          ; R1 <- mem[R0]
16                          ;
17                          ; R0 contains the effective address
18                          ; mem[R0] is rvalue (a value or expression)
19                          ; R1 is lvalue (where the result is to be stored)
20                          ;
21                          ; LD from-memory (rvalue),to-register (lvalue)
```

```
22                              ;
23                              ;       LD      R0,R1           ; R1 <- #FFFF
24      0200    8B01            LDR R0,#10,R1
25                              ;
26                              ;       ADD     R0,R0           ; R0 <- R0 + R0
27                              ;
28                              ; mem[R0] <- R1
29                              ;
30                              ; R1 is rvalue
31                              ; R0 is the effective address
32                              ; mem[R0] is lvalue
33                              ;
34      0202    C088            STR     R1,R0,#1
35                              ;
36                              ; Put breakpoint on next address
37                              ;
38                              BrkPtHere
39                              ;
40                                      end LDST1

Successful completion of assembly - 1P

** Symbol table **

Constants (Equates)
Name                            Type    Value   Decimal

Labels (Code)
Name                            Type    Value   Decimal
BrkPtHere                       REL     0204    516     PRI
LDST1                           REL     0200    512     PRI

Labels (Data)
Name                            Type    Value   Decimal
V1                              REL     0010    16      PRI

Registers
Name                            Type    Value   Decimal
R7                              REG     0007    7       PRI
R6                              REG     0006    6       PRI
R5                              REG     0005    5       PRI
R4                              REG     0004    4       PRI
R3                              REG     0003    3       PRI
R2                              REG     0002    2       PRI
R1                              REG     0001    1       PRI
R0                              REG     0000    0       PRI

.XME file: \\Mac\Home\Desktop\Computer Architecture\Assembler\LDRSTR.xme
```

LDRSTR1.xme (file used)

```
S00D00004C44525354522E61736DA8
S2050010FFFFEC
S1070200018888C025
S9030200FA
```