# 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)