Assignment 2
Implementation of the XM23p's Pipelined Register Instructions Design, Implementation and Testing 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 pipelined register architecture. This will later be a part of the XM23p's emulator along with the loader.

Objectives

The program's primary objective is to take go through code completing the three stages for each instruction received; Fetch refers to receiving or getting the instruction or asm directive, Decoding it refers to the process of finding out what the opcode and operand mean, which directive they stand for or refer to, and which constants/registers are involved? The final stage is execute which would execute the instruction that has been fetched and decoded.

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

Since XM23p is an updated XM23, a new feature has been added where the processor fetches the next instruction while executing the current instruction simultaneously. This significantly lowers the amount of clock cycles that the processor has to go through to run code.

Software Design/Implementation

Data Dictionary

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*
Destination = [R0-R4] *int bits*

Implementation

For this Assignment, the implementation can be found in the zipped file labeled A2 submitted with this document, it should contain the following files: main.c debugger.c decode.c execute.c fetch.c loader.c xm23p.c and xm23p.h.
Alongside these program files the .lis and .xme files for the test cases will be found within the zipped folder.

How to Run

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. 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 “./loader”. You should see the command window pop up and you would be able to use the menu to perform different functions.

Testing

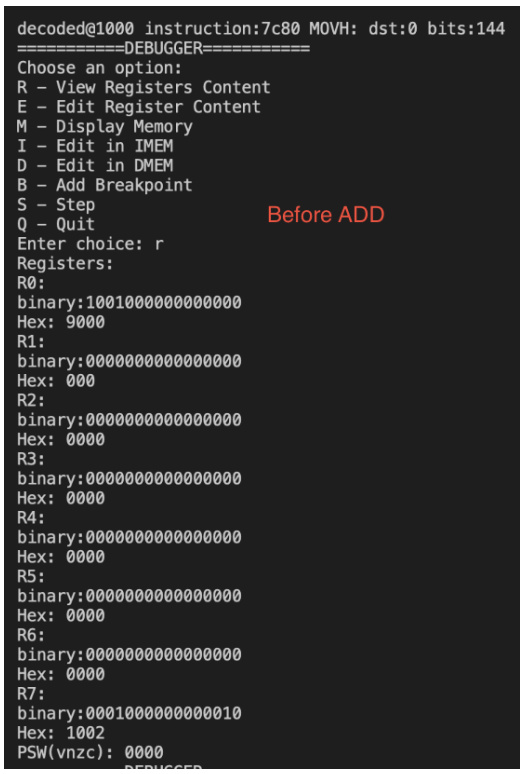
Test 1: Testing Arithmetic functions (ADD)

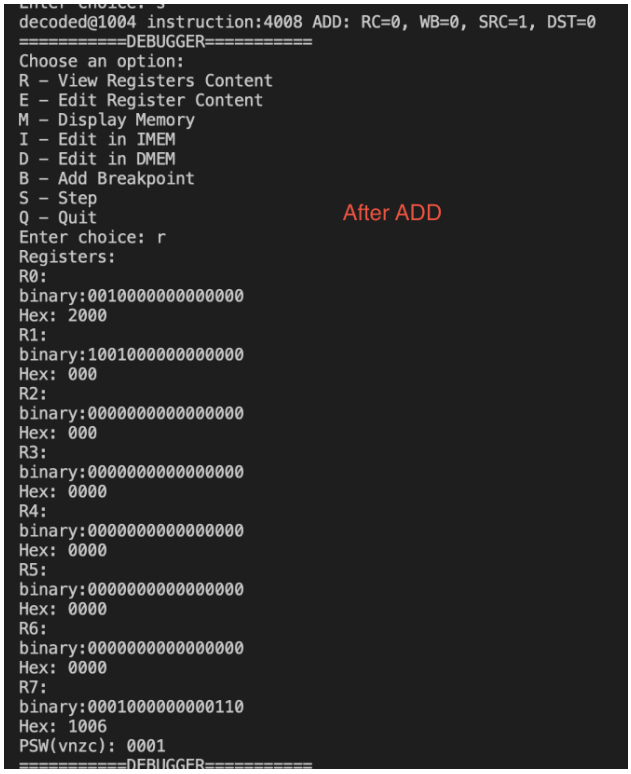
Purpose/Objective: To test the arithmetic function ADD, to ensure that it is working correctly. This test will also test the fetch, decode, execute and PSW functions.

Test Configuration: I will use the ADD function to add two numbers together using the following code:
I will use the following code from ADD-ADDC.asm

```
org #1000
MAIN
movh #9000,R0
movh #9000,R1
add R1,R0 ;carry and overflow flag using word addition
addc $0,R0 ;adding 0 to a register with carry of 1, clearing flags
movl $1,R0
movl $-1,R1
add.b R1,R0 ;producing a zero flag using byte addition
add.b R1,R0 ;producing a negative flag
DONE
bra DONE ; loop
end MAIN
```

Expected Results: I expect the code to run properly and produce the correct flags for the ADD function. I expect the carry flag to be set after the first addition.
Actual Results: The program performed as expected setting the carry flag after ADD was executed.





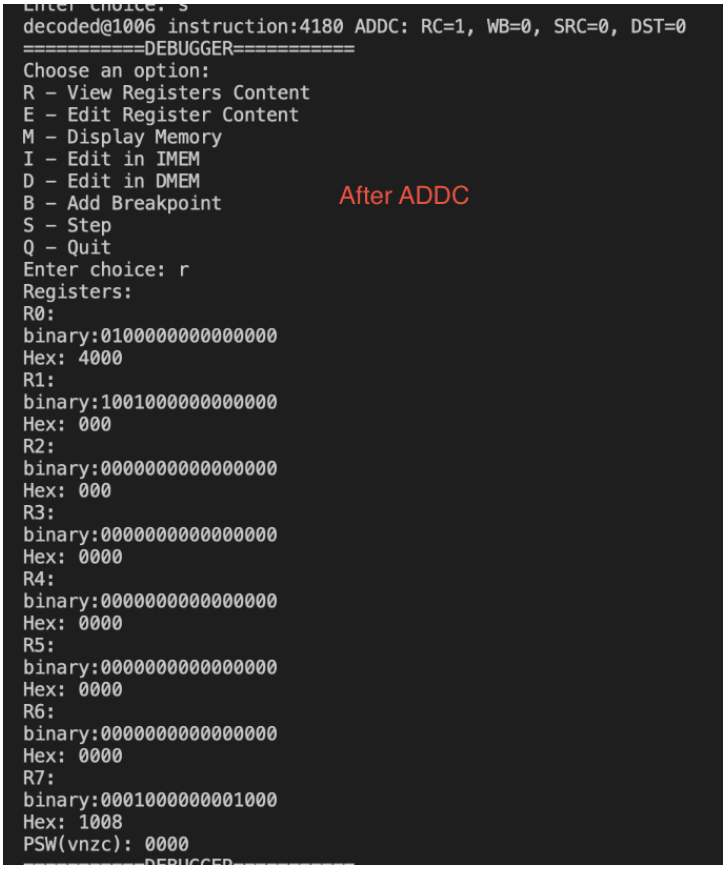
Pass/Fail: Pass

Test 2: Testing Arithmetic Functions with carry (ADDC)

Purpose/Objective: To test the arithmetic function ADDC, to ensure that it is working correctly. this test will also test the fetch, decode, execute and PSW functions.
Test Configuration: I will use the ADDC function to add two numbers together using the same code and I will use debug mode to step through and check the change in registers and PSW flags
I will use the following code from ADD-ADDC.asm

```
org #1000
MAIN
movh #9000,R0
movh #9000,R1
add R1,R0 ;carry and overflow flag using word addition
addc $0,R0 ;adding 0 to a register with carry of 1, clearing flags
movl $1,R0
movl $-1,R1
add.b R1,R0 ;producing a zero flag using byte addition
add.b R1,R0 ;producing a negative flag
DONE
bra DONE ; loop
end MAIN
```

Expected Results: I expect the code to run properly and produce the correct flags for the ADDC function. I expect the carry flag to be set after the first addition and cleared after the second addition.
Actual Results: The program performed as expected, the carry flag was infact cleared after ADDC was executed.



Pass/Fail: Pass

Test 3: Testing Arithmetic functions in bytes (ADD.B)

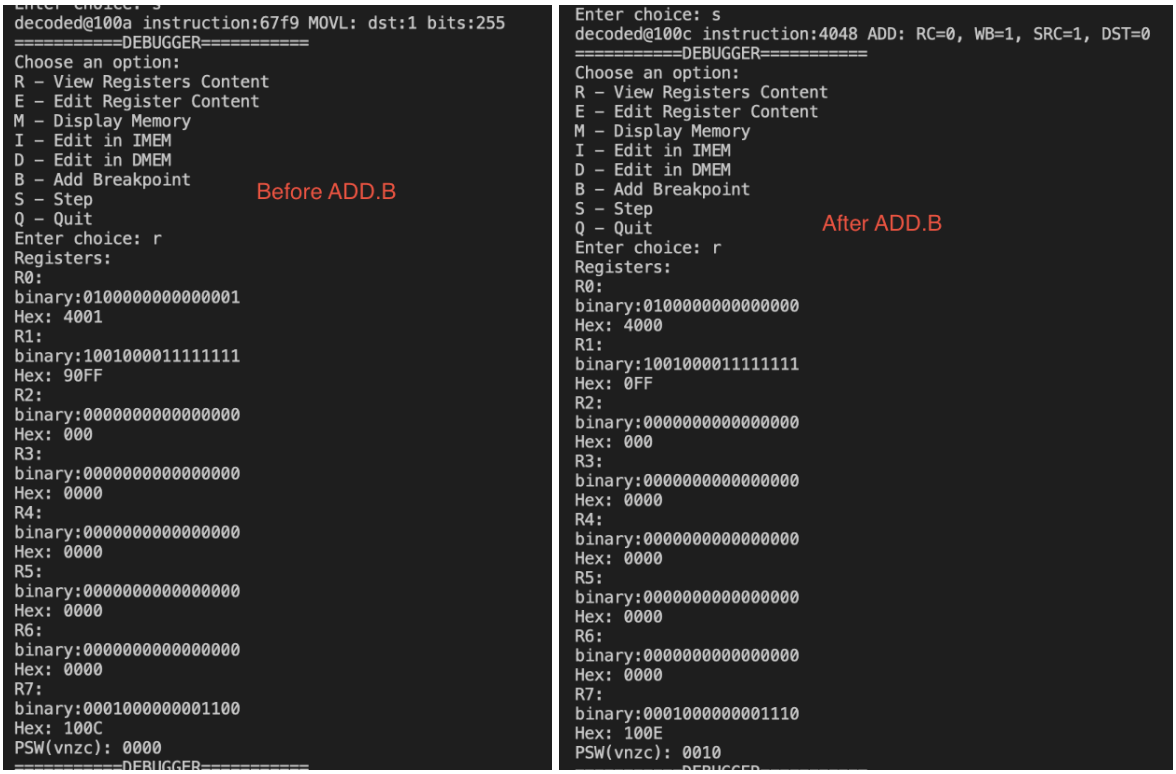
Purpose/Objective: To test the arithmetic function ADD.B, to ensure that functions that only alter the lower byte and not the whole word in the register function properly
Test Configuration: I will use the ADD.B function to add two numbers together using the same code and I will use debug mode to step through and check the change in registers and PSW flags. I will use the following code from ADD-ADDC.asm

```
org #1000
MAIN
movh #9000,R0
movh #9000,R1
add R1,R0 ;carry and overflow flag using word addition
addc $0,R0 ;adding 0 to a register with carry of 1, clearing flags
movl $1,R0
movl $-1,R1
add.b R1,R0 ;producing a zero flag using byte addition
add.b R1,R0 ;producing a negative flag
DONE
```

bra DONE ; loop
end MAIN

Expected Results: I expect the code to run properly and produce the correct flags for the ADD.B function.

Actual Results: The program performed as expected setting the correct flags and changing the register values accordingly.



Pass/Fail: Pass

Test 4: Testing Arithmetic function (AND)

Purpose/Objective: To test the arithmetic function AND, to ensure that it is working correctly. This test will also test the fetch, decode, execute and PSW functions.

Test Configuration: I will use the AND function to AND(&) two numbers together using the following code and I will use debug mode to step through and check the change in registers and PSW flags

AND.asm
org #1000
MAIN
MOVH #FFFF,R0
MOVL #FF,R0
MOVL #08,R1
AND R0,R1 ;produce no flags using byte and register source

```
MOVL #F0,R2
AND.B R0,R2 ;produce negative flag using word and register source
AND $0,R0 ;produce zero flag using word and constant source
DONE
bra DONE ; loop
end MAIN
```

Expected Results: I expect the code to run properly and produce the correct flags for the AND function. I expect the zero flag to be set after the third AND operation and the negative flag to be set after the second AND operation.

Actual Results: The program performed as expected producing the following change in the registers:

```
decoded@1004 instruction:6041 MOVL: dst:1 bits:8
==========DEBUGGER==========
Choose an option:
R - View Registers Content
E - Edit Register Content
M - Display Memory
I - Edit in IMEM
D - Edit in DMEM
B - Add Breakpoint
S - Step
Q - Quit
Enter choice: r
Registers:
R0:
binary:1111111111111111
Hex: FFFF
R1:
binary:0000000000001000
Hex: 0008
R2:
binary:0000000000000000
Hex: 000
R3:
binary:0000000000000000
Hex: 0000
R4:
binary:0000000000000000
Hex: 0000
R5:
binary:0000000000000000
Hex: 0000
R6:
binary:0000000000000000
Hex: 0000
R7:
binary:0001000000000110
Hex: 1006
PSW(vnzc): 0000
```

Before AND

```
Enter choice: s
decoded@1006 instruction:4701 AND: RC=0, WB=0, SRC=0, DST=1
==========DEBUGGER==========
Choose an option:
R - View Registers Content
E - Edit Register Content
M - Display Memory
I - Edit in IMEM
D - Edit in DMEM
B - Add Breakpoint
S - Step
Q - Quit
Enter choice: r
Registers:
R0:
binary:1111111111111111
Hex: FFFF
R1:
binary:0000000000001000
Hex: 0008
R2:
binary:0000000000000000
Hex: 000
R3:
binary:0000000000000000
Hex: 0000
R4:
binary:0000000000000000
Hex: 0000
R5:
binary:0000000000000000
Hex: 0000
R6:
binary:0000000000000000
Hex: 0000
R7:
binary:0001000000001000
Hex: 1008
PSW(vnzc): 0000
```

After AND

Pass/Fail: Pass

Test 5: Testing MOVL & MOVH

Purpose/Objective: To test the MOVL and MOVH functions, to ensure that they are working correctly. this test will also test the fetch, decode, execute and PSW functions.

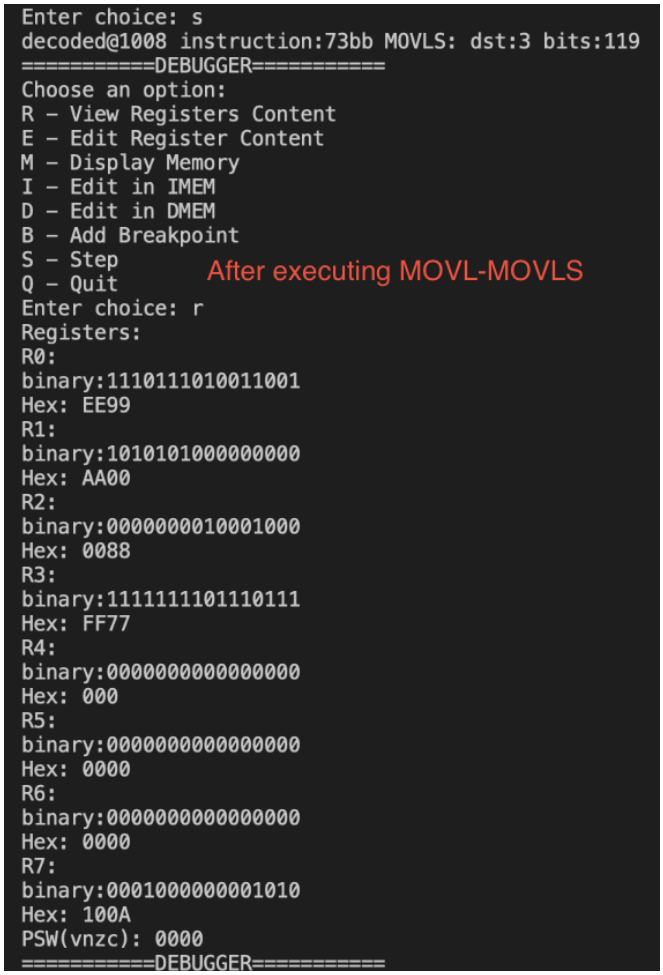
Test Configuration: I will use the MOVL and MOVH functions to move a value into a register using the following code and I will use debug mode to step through and check the change in registers and PSW flags

MOVL-MOVH.asm

```
org #1000
MAIN
movh #EE00,R0
movl #99,R0 ;set low bytes
movh #AA00,R1 ;set high bytes
movlz #88,R2 ;set low bytes without changing high bytes
movls #77,R3 ;set low bytes and clear high bytes
DONE
bra DONE ; loop
end MAIN
```

Expected Results: I expect the code to run properly for the MOVL and MOVH functions. I expect the low bytes to be set after the first MOVL and the high bytes to be set after the first MOVH. I expect the low bytes to be set after the first MOVL and the high bytes to be cleared after the first MOVLS.

Actual Results: The actual result was as expected and the software did not have any problems executing.



Pass/Fail: Pass

Test 6: Testing Register Functions (CMP)

Purpose/Objective: The purpose of this test is to test the CMP instruction, I have chosen this instruction because it is very common and used alot in programming for comparison which is a fundamental part of programming.

Test Configuration: I wil