Dalhousie University
Department of Electrical and Computer Engineering

ECED 3403 – Computer Architecture

Assignment 2: Implementation of the XM23p Pipelined Register Instructions

Objectives

With the completion of the implementation and testing of XM23p's basic debugger (S-Record loader and memory display), we are now ready to begin the implementation of XM23p's pipeline and register operations.

XM23's von-Neumann architecture has three stages (Fetch, Decode, and Execute) that operate sequentially taking four clock cycles for non-data memory access instructions:

Clock	Instruction	Stages	Comments
N+0	`ADD R1,R2`	F0	Write PC to memory; Increment PC
N+1		F1	Read instruction from memory location
N+2		D0	Inst ← ADD; SRC ← R1; DST ← R2
N+3		E0	R2 ← R1 + R2; Update PSW

In XM23p's pipeline implementation, the processor still has three stages (Fetch, Decode, and Execute) but they overlap:

Clock	Instruction	Stages			Comments
		Fetch	Decode	Execute	
N+0	`ADD R1,R2`	F0	D0		
N+1		F1		E0	Previous instruction is completed.
N+2	`MOV R2,R0`	F0	D0		
N+3		F1		E0	R2 ← R1 + R2; Update PSW

One of the benefits of this approach is that the non-data memory access instructions complete every two clock cycles rather than four.

For more details on the XM23p design, please see the *XM-23p – The XM-23 Pipeline Processor Design Document* available in several locations on the course website.

Requirements

In this assignment, you are to implement a three-stage pipeline for XM23p as described in the *XM-23p – The XM-23 Pipeline Processor Design Document* for XM23p's eighteen register instructions (ADD through SXT):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Mnemonic	Description
0	1	0	0	0	0	0	0	R/C	W/B	S/C	S/C	S/C	D	D	D	ADD	Add: DST = DST + SRC/CON
0	1	0	0	0	0	0	1	R/C	W/B	S/C	S/C	S/C	D	D	D	ADDC	Add: DST = DST + (SRC/CON + Carry)
0	1	0	0	0	0	1	0	R/C	W/B	S/C	S/C	S/C	D	D	D	SUB	Subtract: DST = DST + (¬SRC/CON + 1)
0	1	0	0	0	0	1	1	R/C	W/B	S/C	S/C	S/C	D	D	D	SUBC	Subtract: DST = DST + (¬SRC/CON + Carry)
0	1	0	0	0	1	0	0	R/C	W/B	S/C	S/C	S/C	D	D	D	DADD	Decimal add: DST = DST + (SRC/CON + Carry)
0	1	0	0	0	1	0	1	R/C	W/B	S/C	S/C	S/C	D	D	D	CMP	Compare: DST – SRC/CON
0	1	0	0	0	1	1	0	R/C	W/B	S/C	S/C	S/C	D	D	D	XOR	Exclusive OR: DST = DST ⊕ SRC/CON
0	1	0	0	0	1	1	1	R/C	W/B	S/C	S/C	S/C	D	D	D	AND	AND: DST = DST & SRC/CON
0	1	0	0	1	0	0	0	R/C	W/B	S/C	S/C	S/C	D	D	D	OR	OR: DST = DST | SRC/CON
0	1	0	0	1	0	0	1	R/C	W/B	S/C	S/C	S/C	D	D	D	BIT	Bit test: DST & (1 << SCR/CON)
0	1	0	0	1	0	1	0	R/C	W/B	S/C	S/C	S/C	D	D	D	BIC	Bit clear: DST = DST & ~(1 << SRC/CON)
0	1	0	0	1	0	1	1	R/C	W/B	S/C	S/C	S/C	D	D	D	BIS	Bit set: DST = DST | (1 << SRC/CON)
0	1	0	0	1	1	0	0	0	W/B	S	S	S	D	D	D	MOV	DST = SRC
0	1	0	0	1	1	0	0	1	0	S	S	S	D	D	D	SWAP	Swap SRC and DST
0	1	0	0	1	1	0	1	0	W/B	0	0	0	D	D	D	SRA	Shift DDD right (1 bit) arithmetic
0	1	0	0	1	1	0	1	0	W/B	0	0	1	D	D	D	RRC	Rotate DDD right (1 bit) through carry
0	1	0	0	1	1	0	1	0	0	0	1	1	D	D	D	SWPB	Swap bytes in DDD
0	1	0	0	1	1	0	1	0	0	1	0	0	D	D	D	SXT	Sign-extend byte to word in DDD

And its four register initialization instructions (`MOVL` to `MOVH`):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Mnemonic	Description
0	1	1	0	0	B	B	B	B	B	B	B	B	D	D	D	MOVL	DST.Low byte = BBBBBBBB; DST.High byte unchanged
0	1	1	0	1	B	B	B	B	B	B	B	B	D	D	D	MOVLZ	DST.Low byte = BBBBBBBB; DST.High byte = 00000000
0	1	1	1	0	B	B	B	B	B	B	B	B	D	D	D	MOVLS	DST.Low byte = BBBBBBBB; DST.High byte = 11111111
0	1	1	1	1	B	B	B	B	B	B	B	B	D	D	D	MOVH	DST.Low byte unchanged; DST.High byte = BBBBBBBB

Stages

Stages F0 and D0 are to operate in parallel during a clock cycle, and F1 and E0 are to operate in parallel during the next clock cycle.

The start of any clock cycle is to start two of the stages (F0 and D0 or F1 and E0). At the end of the clock cycle, the output of F0 is to be the input for F1, the output for D0 is to be the input for E0, and the output of F1 is to be the input to the next D0.

A CPU clock will be required.

Fetch

Fetch is to be divided into two stages. The first stage (F0) is to copy the PC to the IMAR, signal the ICTRL to read, and then increment the PC by 2. In the second stage (F1), when IMEM returns the word, the word is to be supplied to the decode Instruction Register (IR).

Decode

Decode is to be a single stage. It is to take a copy of the IR and extract the opcode and operand bits. This information is to be structured so it can be used in the execute stage.

Remember, the decode function will eventually decode all instructions.

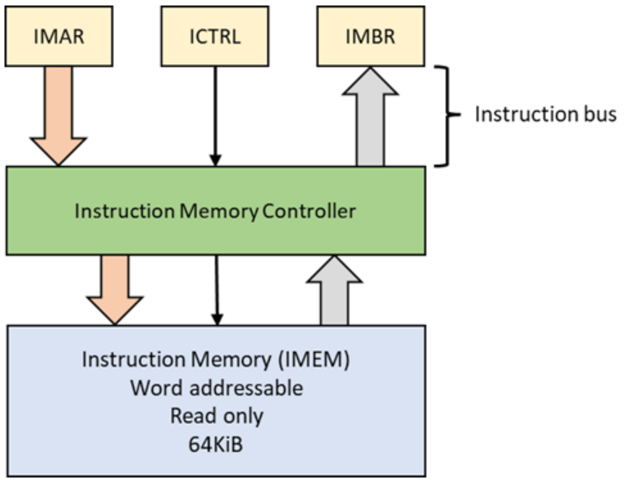
Execute

The execute stage is to take the decoded opcode and operands to perform the operation. The registers and constants it accesses are to be in XM23p's Register File. Special code should not be necessary since the machine has a highly orthogonal ISA and each instruction works any of the registers, including the PC and SP.

All instructions except MOVL to MOVH can affect one or more of the program status word (PSW) bits. For example, ANDing a register with zero should set the Zero bit in the PSW, whereas ADD can change any of the four PSW bits (oVerflow, Negative, Zero, and Carry). The PSW is to be implemented; it will be used in subsequent assignments.

Memory access

Your implementation will only access the machine's IMEM to fetch, decode, and execute the instructions. For this assignment only, it will not be necessary to access the machine's data memory (DMEM) because there are no memory access instructions (LD, ST, LDR, and STR) used in this assignment:



The IMEM will be accessed through the IMAR (Instruction Memory Address Register), ICTRL (Instruction Control register), and the IMBR (Instruction Memory Buffer Register).

The IMEM controller returns 16-bit words. The program counter (PC) holds a 16-bit *word* address that *always* starts on an even byte.

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 (0.5), a design section (3), and a data dictionary (1.5).

Total points: 5.

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

Total points: 11.

Testing: In addition to any test software supplied as part of the assignment, you will be responsible for developing sufficient distinct tests demonstrating that the software operates according to the design specification. 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: 4.

All three sections are to be submitted electronically.

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

Important Dates

Available: 20 May 2024

Design document due: 27 May 2024 at midnight, Atlantic.

Software and testing due: 25 June 2024 at midnight, Atlantic.

Demonstration: 25 and 26 June 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 work is to be implemented in C. 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 15% 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 or has used AI will result in an immediate dismissal from this course.