Dalhousie University
Department of Electrical and Computer Engineering

ECED 3403 – Computer Architecture

Assignment 4: XM23p Transfer of Control (Branching)

Objectives

Most algorithms use three basic coding structures: sequential operations, conditionals, and repetition. Our emulator currently supports sequential operations only, including load and store instructions. In this assignment, we will design, implement, and test the XM23p emulator instructions to support changing the flow of control using branching.

Overview

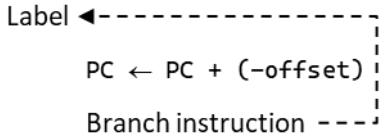
Branching instructions allow a program to change its flow-of-control. XM23p's branching instructions are:

Mnemonic	**Instruction**	**PSW bits to inspect**			
		V	**N**	**Z**	**C**
BL	Branch with Link	-	-	-	-
BEQ/BZ	Branch if equal or zero	-	-	= 1	-
BNE/BNZ	Branch if not equal or not zero	-	-	= 0	-
BC/BHS	Branch if carry/higher or same (unsigned)	-	-	-	= 1
BNC/BLO	Branch if no carry/lower (unsigned)	-	-	-	= 0
BN	Branch if negative	-	= 1	-	-
BGE	Branch if greater or equal (signed)	N ⊕ !V		-	-
BLT	Branch if less (signed)	N ⊕ V		-	-
BRA	Branch Always	-	-	-	-

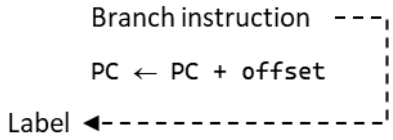
Branching with Link and Branch Always are unconditional. When executed, control passes to the address of the label. The remaining branch instructions are conditional, meaning that one or more of the PSW bits must be inspected.

Branch with Link is a subroutine call. The return address is saved in the Link Register before control passes to the label. The subroutine can return to the caller by storing LR in the PC.

The location can be either a backward or a forward relative to the PC:



Backward reference



Forward reference

The distance between the instruction and the label are determined by the offset stored in the instruction:

- The offset is either 13 bits long (Branch with Link) or 10 bits long (all other branching instructions):

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	13-bit offset												

BL has a 13-bit offset

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	10-bit offset									

All other branching instructions have a 10-bit offset

- The MSBit of either offset can indicate either a positive direction, a forward reference (MSBit = 0), or a negative direction, a backward reference (MSBit = 1).
- The LSBit of an offset can be either clear (0) or set (1). This raises a question, since instructions must always fall on even-byte addresses, how can the offset be odd?

  The offset is an encoding of the distance. It is the distance (in bytes) from the instruction's PC to the label. Since the distance is always even (why?), the LSBit of the distance will always be zero. The assembler shifts the distance to the right by 1 and stores this as the offset.

  To get the decoded distance, XM23 shifts the offset left by 1, making the offset even, thereby effectively doubling the range.

- The effective address, the address of the next instruction to execute, must be determined. The steps required are:

  a) Extract offset (either 10- or 13-bits).
  b) Shift left to get the even value.
  c) Sign-extend the offset to 16-bits.
  d) Add the decoded offset (the distance) to the program counter. The program counter should contain the address of the instruction after the branch (why?); the assembler takes this into account when calculating the offset.

Example

In the following example, the contents of a block of data (an array) are summed. The value is to be stored in register 1.

The data is a block of values in data memory:

```
      DATA
      org   #100
DVal  word  $10
      word  $5
      word  $64
      word  $-1          ; End-of-list indicator
```

The steps required are:

1. ASSIGN zero TO sum
2. READ a value FROM data array
3. IF value < 0 THEN GOTO 6
4. ADD value TO sum
5. GOTO 2
6. STOP

In step 3, the check for value being less than 0 can be achieved using the `CMP` instruction and a `BN` (branch if negative or less than zero) to a label outside the loop. Repeating the instructions takes place in step 5, with the `GOTO 2`, or an unconditional branch, `BRA`, to a label at the start of the loop.

The XM23 instructions are:

```
      CODE
      org   #200
SumMain
; R0 <- address of DVal
      movl  DVal,R0
      movh  DVal,R0
;
      movlz #0,R1       ; Sum <- 0
;
Loop  ld    R0+,R2      ; R2 <- mem[R0]
      cmp   #0,R2
      bn    Done        ; If negative, exit
      add   R2,R1       ; Add value to sum
      bra   Loop        ; Repeat for next value
; End of loop - Result in R1
Done
      bra   Done
;
      end   SumMain
```

Control (or Branch) hazards

We have seen examples of data hazards on XM23p when changing the value of the program counter. A generalized data hazard can occur on many pipeline processors using any register. XM23p is designed so this does not occur.

A control (or branch) hazard refers to transfer-of-control instructions specifically.[1] One or more instructions following the branch could be executed if the branch is taken. We saw some solutions when working with non-branching instructions that modify the PC, there are others, both in hardware and software, that will be discussed in class.

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.

[1] Most pipeline examples describe five-stage pipelines (in XM23p, we are using a three-stage pipeline). The stages in a five-stage pipeline consist of: Instruction Fetch, IF; Instruction Decode, ID; Execution, EX; Memory access, MEM; and Write Back (a register value to the register file), WB. All stages take the same time, meaning that the slowest stage determines the speed of the other stages. In XM23p, the MEM stage is the second step of the execute stage, E1, and the register writeback occurs during the first or second step of the execute stage.

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: 7 July 2024

Design document due: 12 July 2024 at 11:59pm, Atlantic.

Software and testing due: 17 July 2024 at 11:59pm, Atlantic.

Demonstration: 17 and 18 July 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 10% 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.