Assignment 4
XM23p Transfer of Control (Branching) 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 an extension to the emulator program written in C, that would emulate the XM23p's execution of the instructions in charge of changin the transfer of control or branching. These functions serve a cruicial purpose as they are necessary to implement conditionals, comparison and looping which are vital concepts when it comes to computer programming. The descriptions and format for these branching instructions yet to be implemented are highlighted below in the following figure.

0	0	0	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	BL	Branch with Link
0	0	1	0	0	0	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	BEQ/BZ	Branch if equal or zero
0	0	1	0	0	1	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	BNE/BNZ	Branch if not equal or not zero
0	0	1	0	1	0	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	BC/BHS	Branch if carry/higher or same (unsigned)
0	0	1	0	1	1	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	BNC/BLO	Branch if no carry/lower (unsigned)
0	0	1	1	0	0	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	BN	Branch if negative
0	0	1	1	0	1	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	BGE	Branch if greater or equal (signed)
0	0	1	1	1	0	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	BLT	Branch if less (signed)
0	0	1	1	1	1	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	BRA	Branch Always

As well as this table which highlights Program status register (PSW) changes for each instruction:

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	-	-	-	-

Objectives

The objective of this assignment is to design, implement and test the BL-BRA instructions within the emulator. Their functionality will be implemented according to the description mentioned above.

In this assignment I will create the design for the branching instructions as an extension to the loader, debugger, IMEM and DMEM 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.

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

BRANCHING INSTRUCTIONS = [BL|BEQ/BZ|BNE/BNZ|BC/BHS|BNC/BLO|BN|BGE|BLT|BRA]

BL = BL_Opcode + OFF_bits
BL_Opcode = “0” + “0” + “0”
OFF_bits = 13{bit}13
Bit = [0|1]

BEQ/BZ = BEQ/BZ_Opcode + OFF_bits
BEQ/BZ_Opcode = “0” + “0” + “1” + “0” + “0” + “0”

OFF_bits = 10{bit}10
Bit = [0|1]

BEQ/BZ = BEQ/BZ_Opcode + OFF_bits
BEQ/BZ_Opcode = “0” + “0” + “1” + “0” + “0” + “1”
OFF_bits = 10{bit}10
Bit = [0|1]

BC/BHS = BC/BHS_Opcode + OFF_bits
BC/BHS_Opcode = “0” + “0” + “1” + “0” + “1” + “0”
OFF_bits = 10{bit}10
Bit = [0|1]

BNC/BLO = BNC/BLO_Opcode + OFF_bits
BNC/BLO_Opcode = “0” + “0” + “1” + “0” + “1” + “1”
OFF_bits = 10{bit}10
Bit = [0|1]

BN = BN_Opcode + OFF_bits
BN_Opcode = “0” + “0” + “1” + “1” + “0” + “0”
OFF_bits = 10{bit}10
Bit = [0|1]

BGE = BGE_Opcode + OFF_bits
BGE_Opcode = “0” + “0” + “1” + “1” + “0” + “1”
OFF_bits = 10{bit}10
Bit = [0|1]

BLT = BLT_Opcode + OFF_bits
BLT_Opcode = “0” + “0” + “1” + “1” + “1” + “0”
OFF_bits = 10{bit}10
Bit = [0|1]

BRA = BRA_Opcode + OFF_bits
BRA_Opcode = “0” + “0” + “1” + “1” + “1” + “1”
OFF_bits = 10{bit}10
Bit = [0|1]

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 betweenBEQandBRA (IMARValue):

        ENUM {LBEQBZ, LBNEBNZ, LBCBHS, LBNCBLO, LBN, LBGE, LBLT, LBRA}  # Local
L2 opcode cases

        #GET OFFSET USING & and >>

        SWITCH (opcode):  // Opcode cases

           CASE LBEQBZ:

              RETURN BEQBZ
```

```
            CASE LBNEBNZ:
                RETURN BNEBNZ
            CASE LBCBHS:
                RETURN BCBHS
            CASE LBNCBLO:
                RETURN BNCBLO
            CASE LBN:
                RETURN BN
            CASE LBGE:
                RETURN BGE
            CASE LBLT:
                RETURN BLT
            CASE LBRA:
                RETURN BRA
            DEFAULT:
                PRINT "instruction not yet implemented"
                RETURN Error
        END SWITCH
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 “INSTRUCTION”:
      # execution for “INSTRUCTION”
BREAK
CASE BL:
         PRINT "BL:"
         SET RegistersValue[PC] = RegistersValue[8] // Increment the PC by offset
BREAK

CASE BEQBZ:  // Branch if equal
      PRINT "BEQ/BZ:"
      IF PSW.z equals TRUE
         SET RegistersValue[PC] = RegistersValue[PC] + offsetbuff  // Increment the PC
by offset
      END IF
BREAK

CASE BNEBNZ:  // Branch if not equal
      PRINT "BNE/BNZ:"
      IF PSW.z equals FALSE
         SET RegistersValue[PC] = RegistersValue[PC] + offsetbuff  // Increment the PC
by offset
      END IF
BREAK

CASE BCBHS:  // Branch if carry
      PRINT "BC/BHS:"
      IF PSW.c equals TRUE
         SET RegistersValue[PC] = RegistersValue[PC] + offsetbuff  // Increment the PC
          by offset
      END IF
BREAK

CASE BNCBLO:  // Branch if not carry
      PRINT "BNV/BLO:"
      IF PSW.c equals FALSE
         SET RegistersValue[PC] = RegistersValue[PC] + offsetbuff  // Increment the PC
by offset
      END IF
BREAK

CASE BN:  // Branch if negative
      PRINT "BN:"
```

```
            IF PSW.n equals TRUE
                SET RegistersValue[PC] = RegistersValue[PC] + offsetbuff  // Increment the PC
        by offset
            END IF
        BREAK

        CASE BGE:  // Branch if greater than or equal
            PRINT "BGE:"
            IF PSW.n equals PSW.v
                SET RegistersValue[PC] = RegistersValue[PC] + offsetbuff  // Increment the PC
        by offset
            END IF
        BREAK

        CASE BLT:  // Branch if less than
            PRINT "BLT:"
            IF PSW.n not equals PSW.v
                SET RegistersValue[PC] = RegistersValue[PC] + offsetbuff  // Increment the PC
        by offset
            END IF
        BREAK

        CASE BRA:
            PRINT "BRA:"
            SET RegistersValue[PC] = RegistersValue[PC] + offsetbuff  // Increment the PC by
        offset
        BREAK
    END SWITCH
END FUNCTION
```

Main: (same as A2)