ECED3403 – Assignment 2

Grace Yu

B00902046

May 30th, 2024

1. Design

1.1. Problem Introduction

This assignment aims to further develop the XM23p emulator that was begun in assignment 1 and lab 2. The main goal is to implement the XM23p pipeline. This pipeline consists of three stages: Fetch0, Fetch1, Decode0, and Execute0. A characteristic of XM23p is that it performs two stages per clock tick, unlike some other coding architecture. On even ticks, Fetch0 and Decode0 occur. On odd ticks, Fetch1 and Execute0 occur. The two Fetch stages together determine the current memory address and instruction bits. Decode then uses the instruction bits to find the opcode and its associated operands. This opcode and these operands will then be sent to the Execute stage, which uses this information to perform the instruction. The successfully completed assignment will successfully fetch, decode, and execute all specified 22 out of 40 instructions of an inputted program until it encounters an instruction with bits 0x0000.

1.2. Design Section

PSEUDOCODE:

```
FUNCTION pipeline:
   SET PC to S9 record value or 0
   SET CLOCK to 0
   SET ICNTRL to 0
   WHILE instructionbit is not 0
      IF CLOCK value is even THEN
         CALL F0
         CALL D0
      ELSE
         CALL F1
         CALL E0
      END IF
   END WHILE
   CLOCK++
   PRINT PC and 0000 to indicate end of program
END FUNCTION

FUNCTION fetch0
   SET instructionaddress to PC
   INCREMENT PC by 2
   SET ictrl to READ
   RETURN instructionaddress
END FUNCTION




FUNCTION fetch1
   SETNinstructionbit to CALL im_controller
RETURN instructionbit
END FUNCTION

FUNCTION im_controller
   IF ictrl is READ
         FETCH two byte long instructionbit from imem array
         SET ictrl to NOT READ
   END IF
RETURN imbr
END FUNCTION

FUNCTION decode
   IF instructionbit is between LDR and STR
      instruction is not part of A2
   ELSE IF instructionbit is between BL and BRA
      instruction is not part of A2
   ELSE IF instructionbit is between MOVL and MOVH
      SET arrayplace to instructionbit
      MASK arrayplace except for bits 12, 11
      SHIFT arrayplace to the right by 11
      SET insturctionmnem to MOVL + arrayplace
      SET bytevalue to CALL savebytevalue
   ELSE IF instructionbit is between LD and ST
      instruction is not part of A2
   ELSE IF instructionbit is between MOV and CLRCC
      IF instructionbit is between SETPRI to CLRCC
         instruction is not part of A2
      ELSE instruction is between MOV and SXT
         IF instructionbit is between MOV and SWAP
            SET arrayplace to instructionbit
            MASK arrayplace except for bits 7
            SHIFT arrayplace to the right by 7
            SET intructionmnem to MOV + arrayplace
            SET sourceconstant to CALL savesourceconstant
         ELSE instruction is between SRA and SXT
            IF instructionbit is between SRA and RRC
                SET arrayplace to instructionbit
               MASK arrayplace except for bits 5, 4, 3
               SHIFT arrayplace to the right by 3
               SET insturctionmnem to SRA + arrayplace
            ELSE instructionbit is between SWPB and SXT
               SET arrayplace to instructionbit
               MASK arrayplace except for bits 5
               SHIFT arrayplace to the right by 5
               SET insturctionmnem to SWPB + arrayplace
            END IF
         END IF
         SET wordbyte to CALL savewordbyte
      END IF
   ELSE instructionbit is between ADD and BIS
      SET arrayplace to instructionbit
      MASK arrayplace except for bits 11, 10, 9, 8
      SHIFT arrayplace to the right by 8
      SET insturctionmnem to ADD + arrayplace
      SET wordbyte to CALL savewordbyte
      SET sourceconstantcheck to CALL savesourceconstantcheck
      SET sourceconstant to CALL savesourceconstant
   END IF
   CALL printdecode
END FUNCTION

FUNCTION savesourceconstant
   MASK instructionbit except for bits 5, 4, 3
   SHIFT instructionbit by 3
   SET sourceconstant to constantarray[instructionbit]
   RETURN sourceconstant
END FUNCTION

FUNCTION savewordbyte
   MASK instructionbit except for bits 5, 4, 3
   SHIFT instructionbit by 3
   SET sourceconstant to constantarray[instructionbit]
   RETURN wordbyte
END FUNCTION

FUNCTION savebytevalue
   MASK instructionbit except for bits 10, 9, 8, 7, 6, 5, 4, 3
   SHIFT instructionbit by 3
   SET sourceconstant to constantarray[instructionbit]
   RETURN bytevalue
END FUNCTION

FUNCTION printdecode
```

```
    IF instruction is part of A2
       PRINT instructionaddress and mnemarray[instructionmnem]
    ELSE
       PRINT instructionaddress and instructionbit
    END IF

    IF instructionmnem is between ADD and BIS
       PRINT sourceconstantcheck
    END IF
    IF instructionmnem is between ADD and MOV, OR instructionmnem is between
    SRA and RRC
       print wordbyte
    END IF
    IF instructionmnem is between ADD and SWAP
       IF sourceconstantcheck is 0, OR instructionmnem is MOV, or
       instrutionmnem is SWAP
          PRINT source
       ELSE
          PRINT constant
       END IF
    PRINT destination
 END FUNCTION

 FUNCTION execute
    SWITCH instructionmnem
       CASE ADD:
       CASE ADDC:
       CASE DADD:
          IF instructionmnem is ADD
             SET carry to 0
          END IF
          IF sourceconstantcheck is source
             SET regarray[destination] to regarray[destination] +
                regarray[sourceconstant] + carry
          ELSE sourceconstantcheck is constant
             SET regarray[destination] to regarray[destination] +
                sourceconstant + carry
          END IF
          CALL psw_check
          BREAK
       CASE SUB:
       CASE SUBC:
          SET regarray[destination] to ~regarray[destination] & 0x8000
          IF instructionmnem is SUB
             SET carry to 0
          END IF
```

```
   IF sourceconstantcheck is source
      SET regarray[destination] = regarray[destination] +
         regarray[sourceconstant] + carry
   ELSE sourceconstantcheck is constant
      SET regarray[destination] = regarray[destination] +
         sourceconstant + carry
   END IF
   CALL psw_check
   BREAK
CASE CMP:
   IF sourceconstantcheck is source
      SET tempreg to regarray[destination] – regarray[source]
   ELSE sourceconstantcheck is constant
      SET tempreg to regarray[destination] – constant
   END IF
   IF tempreg is 0
      SET zeroflag to 1
   END IF
   BREAK
CASE XOR
   IF sourceconstantcheck is source
      SET regarray[destination] to regarray[destination] ^
         regarray[source]
   ELSE sourceconstantcheck is constant
      SET regarray[destination] to regarray[destination] ^ constant
      END IF
   IF regarray[destination] is 0
      SET zeroflag to 1
   END IF
   BREAK
CASE AND
   IF sourceconstantcheck is source
      SET regarray[destination] to regarray[destination] &
         regarray[source]
   ELSE sourceconstantcheck is constant
      SET regarray[destination] to regarray[destination] & constant
      END IF
   IF regarray[destination] is 0
      SET zeroflag to 1
   END IF
   BREAK
CASE OR
   IF sourceconstantcheck is source
      SET regarray[destination] ti regarray[destination] |
         regarray[source]
   ELSE sourceconstantcheck is constant
```

```
            SET regarray[destination] to regarray[destination] | constant
            END IF
        IF regarray[destination] is 0
            SET zeroflag to 1
        END IF
        BREAK
    CASE BIT
        IF sourceconstantcheck is source
            SET tempregister to 1 << regarray[source]
        ELSE sourceconstantcheck is constant
            SET tempregister to 1 << constant
        END IF
        tempregister = regarray[destination] & tempregister
        IF tempregister is 0
            SET zeroflag to 1
        END IF
        BREAK
    CASE BIC
        IF sourceconstantcheck is source
            SET tempregister to ~(1 << regarray[source])
        ELSE sourceconstantcheck is constant
            SET tempregister to ~(1 << constant)
        END IF
        tempregister = regarray[destination] & tempregister
        IF tempregister is 0
            SET zeroflag to 1
        END IF
        BREAK
    CASE BIS
        IF sourceconstantcheck is source
            SET tempregister to 1 << regarray[source]
        ELSE sourceconstantcheck is constant
            SET tempregister to 1 << constant
        END IF
        tempregister = regarray[destination] | tempregister
        IF tempregister is 0
            SET zeroflag to 1
        END IF
        BREAK
    CASE MOV
        SET regarray[destination] to regarray[source]
        BREAK
    CASE SWAP
        SET tempreg to regarray[source]
        SET regarray[source] to regarray[destination]
        SET regarray[destination] to tempreg
```

```
            BREAK
        CASE SRA
            SET regarray[destination] >> 1
            BREAK
        CASE RRC
            tempreg = regarray[destination] >> 1
            IF carryflag is 1
                tempreg | 0x8000
            END IF
            IF regarray[destination] & 0x0001 is 1
                SET carryflag to 1
            END IF
            SET regarray[destination] to tempreg
            BREAK
        CASE SWPB
            SET tempreg = regarray[destination] & 0x00FF
            SET regarray[destination] >> 8
            SET tempreg to tempreg << 8
            SET regarray[destination] to regarray[destination] | tempreg
            BREAK
        CASE SXT
            EXTEND byte to word
            BREAK
        CASE MOVL
            SET regarray[destination] to regarray[destination] & 0x7F00
            SET regarray[destination] to regarray[destination] & bytevalue
            BREAK
        CASE MOVLZ
            SET regarray[destination] to regarray[destination] & 0x0000
            SET regarray[destination] to regarray[destination] & bytevalue
            BREAK
        CASE MOVLS
            SET regarray[destination] to regarray[destination] | 0x7F00
            SET regarray[destination] to regarray[destination] & 0x7F00
            SET regarray[destination] to regarray[destination] & bytevalue
            BREAK
        CASE MOVH
            SET regarray[destination] to regarray[destination] & 0x00FF
            SET bytevalue to bytevalue << 8
            SET regarray[destination] to regarray[destination] + bytevalue
            BREAK
END FUNCTION

FUNCTION psw_check
    IF regarray[destination] is negative
        SET regarray[destination] to ~regarray[destination] + 1
```

```
      negativeflag = 1
   END IF
   IF regarray[destination] is greater than max
      SET regarray[destination] to 0x7FFF
      SET overflowflag to 1
   END IF
   IF regarray[destination] is 0
      SET zeroflag to 1
   END IF
   IF sourceconstantcheck is source
      SET tempreg to sourceconstantcheck >> 16
   ELSE
      SET tempreg to constant >> 16
   END IF
   regarray[destination] >> 16
   SET carryflag to carryarray[tempreg][regarray[destination]]
END FUNCTION
```

DATA DICTIONARY

```
regarray = r + [0-7]
r = [int | short] * how many bits are in the register *
r4 = basepointer
r5 = linkregister
r6 = stackpointer
r7 = programcounter
word = 16{bit}16
byte = 8{bit}8
bit = [1|0]

carryarray = [[[0 | 0] | [1 | 0]] | [[1 | 0] | [0 | 1]]]

psw = [negativeflag | overflowflag | carryflag | zeroflag] * array of flags *
negativeflag = status
overflowflag = status
carryflag = status
zeroflag = status
status = [1 | 0] * 1 for TRUE, 0 for FALSE *

constantarray = [ 0 | 2 | 4 | 8 | 16 | 32 | -1] * array of constants *

instructiontype = [BL | BEQBZ | BNEBNZ | … | MOVH | LDR | STR] * enum *
BL = 0
```

```
BEQBZ = 1
BNEBZ = 2
…
MOVH = 37
LDR = 38
STR = 39
```

```
mnemarray = [“BL” | “BEQBZ” | “BNEBZ” | … | “MOVH” | “LDR” | “STR”] * array *

reg_const_operands = sourceconstantcheck + wordbyte + sourceconstant +
destination
sourceconstantcheck = unsigned int
wordbyte = unsigned int
sourceconstant = unsigned int
destination = unsigned int
unsigned int = [0-4294967295]

movx_operands = bytevalue, destination
bytevalue = unsigned int
destination = unsigned int

instructionaddress = int * address *

programcounter = int * address *

ictrl = [READ | DONEREAD] * ready for address save from buffer? *
READ = 1
DONEREAD = 0

imbr = int * instruction memory buffer *

instructionmnem = 1[mnemarray]1

instructionbit = 16[bit]16

nota2 = [TRUE|FALSE]
TRUE = 1
FALSE = 0

arrayplace = int
int = 2147483647
```