Assignment Pass 2

Sakif Fahmid Zaman
B00756635
Dalhousie University
ECED 3403

August 7, 2019

Contents

1 Main **1**

2 CPU **2**

3 Globals **4**

4 Instructions **6**

5 Loader **10**
This is just the work I have done. It does not compile.

1 Main

header

```
1  #ifndef _MAIN_H_
2  #define _MAIN_H_
3
4  #include "Globals.h"
5  #include "cpu.h"
6  #include "debugger.h"
7  #include <iostream>
8  // #include "devices.h"
9  // #include "memory.h"
10
11 #define NUMDEVS 8
12
13 struct Emulator {
14   Debugger debugger;
15   CPU cpu; //emulates a specific cpu (that specify, registers, associated
         instructions, memory model etc)
16             //Device device[NUMDEVS]; //to be implemented, if time permits
                 :-((
17 };
18
19 #endif //_MAIN_H_
```

source

```
#include "main.h"

// short int counter;
// short int PC;

// for(;;){
//     opCode = Memory[PC++];
//     counter -= Cycles[opCode];

//     switch(opCode){
//         case opCode1:
//         case opCode2:
//         ......
//     }

//     if(counter <= 0){
//         /* check for interrupt */
//         /* cyclic tasks */

//         counter += InterruptedPeriod;
//         if(ExitRequired) break;
//     }
// }

int main() {
}
```

2 CPU

header

```
#ifndef _CPU_H_
#define _CPU_H_
#include "Globals.h"
#include "debugger.h"
#include "instructions.h"
#include "main.h"
#include "stdio.h"

struct CPU {
  bool isRunning;    //cpu status, either running or sleeping
  bool exitRequired; // check is exit is rquired - cntrl+break or other
};

//functions for cpu-cycle
void go(Emulator &emulator);
void fetch();              //fetch(Emulator & emulator);
void decode_and_execute(); //decode(Emulator & emulator);
void execute(Emulator &emulator);
void check(Emulator &emulator); //other than regular cpu task,like,
    interrupt handle, check for exit etc

void bus(unsigned short mar, unsigned short &mbr, ACTION rw, SIZE wb); //
    emulating bus behavior
```

```
22
23 #endif //_CPU_H_
```

source

```
1  #include "cpu.h"
2
3  void dev_mem_access(unsigned short mar, unsigned short &mbr, ACTION rw,
       SIZE wb) {
4    //TODO
5  }
6
7  /* Emulates the bus and the memory access
8      - mar:  memory address
9      - mbr: ref to data to read/write
10     - rw: READ|WRITE
11     - wb: WORD|BYTE
12  */
13 void bus(unsigned short mar, unsigned short &mbr, ACTION rw, SIZE wb) {
14   if (rw == READ) {
15     mbr = (wb == WORD) ? memory.word_mem[mar >> 1] : memory.byte_mem[mar];
16   } else { //rw == WRITE
17     if (wb == WORD) {
18       memory.word_mem[mar >> 1] = mbr;
19     } else { //BYTE
20       memory.byte_mem[mar] = (unsigned char)(mbr & 0xFF);
21     }
22     // zero latency for system memomory - not increment required for
          defult case(sys mem)
23   }
24   if (mar < DEVMEM) { //this part is not implemented - a dummy function is
          placed instead TODO: if time permits
25     dev_mem_access(mar, mbr, rw, wb);
26   }
27
28   if (mar < VECTORBASE) {
29     sysclk += 3;
30   }
31 }
32
33 /*  runs fetch-decode-execute-check cycle
34     The emulator contains all required vars to carryout the tasks
35  */
36 void go(Emulator &emulator) {
37   debug(emulator);
38   emulator.cpu.isRunning = true;
39   emulator.cpu.exitRequired = false;
40   for (;;) {
41     fetch();
42     decode_and_execute(); //(emulator);
43     //execute(emulator);
44     check(emulator); //
45     if (emulator.cpu.exitRequired) {
46       break;
47     }
```

```
48    }
49
50    return;
51  }
52
53  /* Get (fetch) data from memory
54     - Obtain data from PC and updated accordingly,
55     - NOTE: 'bus' is used
56  */
57  void fetch() {
58    MAR = PC;
59  #ifdef DEBUG
60    printf("\nFetching from location 0x%x", MAR);
61  #endif //DEBUG
62    //read from PC and store in IR - use bus
63    bus(MAR, MBR, READ, WORD);
64    IR = MBR;
65    //increment PC
66    PC += INSTSIZE;
67    SET_BIT(PC, 0, 0); //PC should be even!
68
69    return;
70  }
71
72  /* Deassembly opcode and operand from Instruction in the IR
73      - no input or output
74   */
75  void decode_and_execute() { //(Emulator & emulator){
76    decode_execute();
77  }
78  void execute(Emulator &emulator);
79  void check(Emulator &emulator); //other than regular cpu task,like,
       interrupt handle, check for exit etc
```

3 Globals

header

```
1  #ifndef _GLOBALS_H_
2  #define _GLOBALS_H_
3
4  #include <bitset>
5
6  //control enumerations
7  enum SIZE { WORD = 0,
8              BYTE };
9  enum ACTION { READ = 0,
10               WRITE };
11
12 #define INSTSIZE 2
13 //#define SET_BIT(var, pos, val) ((var) |= (val << pos)) //set bit at 'pos
       ' in 'var' to 'val' [0|1]
14
15 /*Memory */
```

```
16 #define BYTEMAXMEM (1 << 16) /* 2^16 bytes */
17 #define WORDMAXMEM (1 << 15) /* 2^15 words */
18 #define VECTORBASE 0xFFC0    /*base address for vectors */
19 #define DEVMEM 0x0010
20
21 union mem_ex {
22   unsigned char byte_mem[BYTEMAXMEM];
23   unsigned short word_mem[WORDMAXMEM];
24 };
25
26 extern union mem_ex memory;
27 /* Registers */
28 #define REGCNT 8 //number of registers - in cols
29 #define ROWS 2   //number of rows, first row (0-index)for different
      registers, 2nd row for corresponding constants
30
31 extern unsigned reg_file[ROWS][REGCNT]; //available to programmer
32
33 //define alias names for REGs
34 #define LR reg_file[0][4]  //Link Register
35 #define SP reg_file[0][5]  //Stack Pointer
36 #define PSW reg_file[0][6] //Program Status Word
37 #define PC reg_file[0][7]  //Program Counter
38
39 /* PSW related */
40 extern unsigned carry[2][2][2];
41 extern unsigned overflow[2][2][2];
42
43 // not accessible by programmer
44 extern unsigned short MAR;
45 extern unsigned short MBR;
46 extern unsigned short IR;
47 extern unsigned short TMP;
48
49 extern short int sysclk; //system clock counter
50
51 // check if bit "pos" is set in "var"
52 #define CHECK_BIT(var, pos) (((var) & (1 << pos)) >> pos)
53 // set bit "pos" in "var". "val" = [0|1]
54 #define SET_BIT(var, pos, val) ((var) |= (val << pos))
55
56 // sign extension masks
57 #define SEXT_BL 0xE000 // 1110.0000.0000.0000 -> sign extend from bit 13
58 #define SEXT_BR 0xF800 // 1111.1000.0000.0000 -> sign extend from bit 10
59
60 // byte isolator
61 #define LO_BYTE 0x00FF
62 #define HI_BYTE 0xFF00
63 // /* set bit at 'pos' in 'var' to 'val' [0|1] */
64 // inline void SET_BIT(unsigned int &var, unsigned short pos, unsigned
      short val){
65 //     (var |= (val << pos));
66 // }
67
```

```
68 // /* check if the bit 'pos' is set in a 'var' */
69 // typedef std::bitset<sizeof(int)> IntBits;
70 // inline bool CHECK_BIT(unsigned int var, unsigned short pos){ return
       IntBits(var).test(pos);}
71
72 #endif //_GLOBALS_H_
```

source

```
1  #include "Globals.h"
2
3  #define DEBUG
4
5  union memory_ex memory;
6
7  short int sysclk = 0; //system clock counter
8  unsigned reg_file[ROWS][REGCNT] =
9      {
10         {0, 0, 0, 0, 0, 0, 0, 0},   //initial REG contents
11         {0, 1, 2, 4, 8, 32, 48, -1} //constants
12 };
13
14 unsigned carry[2][2][2] = {0, 0, 1, 0, 1, 0, 1, 1};
15 unsigned overflow[2][2][2] = {0, 1, 0, 0, 0, 0, 1, 0};
16
17 unsigned short MAR = 0;
18 unsigned short MBR = 0;
19 unsigned short IR = 0;
20 unsigned short TMP = 0;
```

4 Instructions

header

```
1  #ifndef _INSTRUCTIONS_H_
2  #define _INSTRUCTIONS_H_
3
4  #include "Globals.h"
5  #include "cpu.h"
6
7  #define OPMASK(x) (((x) >> 14) & 0x03) /* B15:B14 */
8  #define OPBIT13(x) ((x)&0x2000)        // B13 - REG inst + special (not
       LDR STR)
9  #define OPBIT12(x) ((x)&0x1000)        //B12 - LD, ST and special (SVC and
       CEX)
10
11 struct PSWbits {
12   unsigned _Carry : 1;
13   unsigned _Zero : 1;
14   unsigned _Negative : 1;
15   unsigned _SLP : 1;
16   unsigned _oVerflow : 1;
17   unsigned _cPriority : 3;
18   unsigned _notUsed : 5;
19   unsigned _pPririty : 3;
```

```
20 };
21
22 union PswOverlay {
23   short int sh;
24   PSWbits bits;
25 };
26
27 // LD & ST INSTRUCTIONS
28 struct LDSTbits {
29   unsigned short dst : 3;  // 000 - 111
30   unsigned short src : 3;  // 000 - 111
31   unsigned short wb : 2;   // 00 is byte, 01 is word
32   unsigned short inc : 1;  // error if both inc
33   unsigned short dec : 1;  // & dec are set
34   unsigned short prpo : 1; // 1 - PR | 0 - PO
35   unsigned short type : 3; // 000 - LD | 001 - ST
36   unsigned short cat : 2;  // error if not 10
37 };
38
39 union LDSTOverlay {
40   unsigned short inst;
41   LDSTbits bits;
42 };
43
44 // LDR & STR INSTRUCTIONS
45 struct LDRSTRbits {
46   unsigned short dst : 3; // 000 - 111
47   unsigned short src : 3; // 000 - 111
48   unsigned short wb : 1;  // 0 is byte, 1 is word
49   unsigned short off : 6;
50   unsigned short type : 1; // 0 - LDR | 1 - STR
51   unsigned short cat : 2;  // error if not 11
52 };
53
54 union LDRSTROverlay {
55   unsigned short inst;
56   LDRSTRbits bits;
57 };
58
59 // MOVL, MOVLZ, & MOVH INSTRUCTIONS
60 struct MOVbits {
61   unsigned short dst : 3;  // 000 - 111
62   unsigned short byte : 8; // 0000.0000 - 1111.1111
63   unsigned short type : 3; // 100 - MOVH | 011 - MOVLZ | 010 - MOVL | 001
       - ERROR
64   unsigned short cat : 2;  // error if not 2
65 };
66
67 union MOVOverlay {
68   unsigned short inst;
69   MOVbits bits;
70 };
71
72 // BL, BEQ/BZ, BNE/BNZ, ... , & BAL INSTRUCTIONS
```

```
73  struct BRAbits {
74    unsigned short off : 10;  // 00.0000.0000 - 11.1111.1111
75    unsigned short type : 3;  // 000 - 111
76    unsigned short other : 1; // 0 if BL, 1 otherwise
77    unsigned short cat : 2;   // error if not 00
78  };
79
80  union BRAOverlay {
81    unsigned short inst;
82    BRAbits bits;
83  };
84
85  // ADD, ADDC, SUB, ... , & SXT INSTRUCTIONS
86  struct ArithBits {
87    unsigned short dst : 3; // 000 - 111
88    unsigned short src : 3; // 000 - 111
89    unsigned short wb : 1;  // 0 is byte, 1 is word
90    unsigned short rc : 1;
91    unsigned short other : 1; // 0 if ADD-SWAP, 1 if SRA-SXT
92    unsigned short type : 5;  // 0.0000 - 1.0000
93    unsigned short cat : 2;   // error if not 01
94  };
95
96  union ArithOverlay {
97    unsigned short inst;
98    ArithBits bits;
99  };
100
101 void set_bits(ArithOverlay, unsigned short);
102
103 // DADD structures
104 struct DADDbits {
105   unsigned short nib0 : 4;
106   unsigned short nib1 : 4;
107   unsigned short nib2 : 4;
108   unsigned short nib3 : 4;
109 };
110
111 union DADDOverlay {
112   unsigned short word;
113   DADDbits bits;
114 };
115
116 void decode_execute(); // decode instruction at IR and call executioner
117
118 #endif //_INSTRUCTIONS_H_
```

source

```
1  #include "instructions.h"
2
3  void reg_const_ops() {
4  }
5
6  void ldr_str_ops() {
```

```
7    PswOverlay psw;
8    psw.sh = PSW;
9    LDRSTROverlay ldrstr;
10   ldrstr.inst = IR;
11   unsigned short EA;
12 #ifdef DEBUG
13   printf("\nExecuting LDRSTR type instruction");
14   printf("\nDST %d SRC %d OFF %d", ldrstr.dst, ldrstr.src, ldrstr.off);
15 #endif
16   TMP = ldrstr.bits.off;   //store offset bits to TMP Reg
17   if (CHECK_BIT(TMP, 6)) { //sign extension bit- Sec 6.1.2 XM2 ISA
18     TMP |= 0xFFE0;
19   }
20   if (ldrstr.bits.type == 1) {                //LDR
21     EA = reg_file[0][ldrstr.bits.src] + TMP; //effective address
22     MAR = EA;
23     MBR = reg_file[0][ldrstr.bits.dst];
24     if (ldrstr.bits.wb == 0) { //WORD
25       bus(MAR, MBR, READ, WORD);
26     } else { //BYTE
27       bus(MAR, MBR, READ, BYTE);
28     }
29
30   }
31
32   else if (ldrstr.bits.type == 0) {           //STR
33     EA = reg_file[0][ldrstr.bits.dst] + TMP; //effective address
34     MAR = EA;
35     MBR = reg_file[0][ldrstr.bits.src];
36     if (ldrstr.bits.wb == 0) { //WORD
37       bus(MAR, MBR, WRITE, WORD);
38     } else { //BYTE
39       bus(MAR, MBR, WRITE, BYTE);
40     }
41   }
42
43   return;
44 }
45 /* Decode instruction and call execute  */
46 void decode_execute() {
47   //check B15:B14
48   switch (OPMASK(IR)) {
49   case 0: //Branching
50     branch_ops();
51     break;
52   case 1:              //REG-CON ops, REG ops, LDST, and special cases
53     if (OPBIT13(IR)) { // MOV instructions
54       mov_ops();
55     } else if (OPBIT12(IR)) { // LD, ST, SVC, and CEX
56       ld_st_special();
57     } else { // all REG ops
58       reg_const_ops();
59     }
60     break;
```

```
61    case 2: //LDR access
62    case 3: //STR access
63      ldr_str_ops();
64      break;
65    }
66  }
```

5 Loader

```
1   #ifndef _LOADER_H_
2   #define _LOADER_H_
3   #include <iostream>
4   #include <string>
5
6   #include "cpu.h"
7
8   using namespace std;
9   /*<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>> */
10  enum SRec_Type {
11    S0,
12    S1,
13    S9
14  };
15
16  struct SRec {
17    SRec_Type _type;
18    string _count;    //char _count[2];
19    string _addr;     //char _addr[4];
20    string _data;     //should be max 28 byte
21    string _checksum; //char _checksum[2];
22  };
23
24  bool loader(Emulator &emulator, string fname); /* basically called by the
        debugger, and the debugger is a member of an emulator
25                                      and the emulator emulats a CPU :-)
26                                      loads from the input .xme file
27                                     */
28
29  void read_record(SRec &rec, string line); //
30
31  bool load_record(Emulator &emulator, SRec rec);
32
33  void str2int(string s, short int &value);
34
35  #endif //_LOADER_H_
```