Branch: master ▾ Find file Copy path

ECE252 / Lessons / lesson20 / Datapath_Skeleton.cp

skyyHDx datapath

3f8f145 a day ago

1 contributor

Raw Blame History

383 lines (324 sloc) 7.6 KB

```
1  //**********************************
2  //DataPath   JDW
3  //
4  // Implements simple datapath
5  //
6  //**********************************
7
8  //********************
9  //
10 // ISA:
11 //
12 // 0: STOP
13 // 1: GOTO #n
14 // 2: LDI  Rd, #n
15 // 3: ADD  Rd, Rs
16 // 4: MULT Rd, Rs
17 //
18 // to be implemented later:
19 //
20 // 5: LOAD  Rd, M[Rs]
21 // 6: STORE M[Rd], Rs
22 //
23 //************************
24 #define NUM_REGISTERS 10
25 #define MAX_MEMORY 1000
26 #define MAX_VALUE 1000
27
28 #include <iostream>
29 using namespace std;
30
31 class DataPath {
32
33         private:
34
35         //member data - the register file, PC, RAM
36         int registers[NUM_REGISTERS];
37         int pc;
38         int ram[MAX_MEMORY];
39         //define inner variables for opcode, arg1, arg2
40         //int instruction;
41         int opcode;
42         int arg1;
43         int arg2;
44
45
46         //********PRIVATE FUNCTION THAT I FOUND HELPFUL
47         //updates the opcode, arg1, and arg2 based on the instruction
48         // located at the current program counter
49         // Note: constants here are hard-wired to be 3 digits
50         void instructionDecode() {
51                 int tmp;
52                 int count = 0;
53                 int digit[3];
```

```
54                tmp = ram[pc];
55                if(tmp != 0){
56                        while(tmp > 0){
57                                digit[count] = tmp % 10;
58                                tmp/=10;
59                                count++;
60                        }
61                        opcode = digit[2];
62                        arg1 = digit[1];
63                        arg2 = digit[0];
64                        /*
65                        if(opcode >6){
66                                opcode = 0;
67                                arg1 = 0;
68                                arg2 = 0;
69                        }*/
70
71                }
72                else{
73                        opcode = 0;
74                        arg1 = 0;
75                        arg2 = 0;
76                }
77
78
79
80                //cout << "OP: " << opcode << endl;
81                //cout << "Arg1: " << arg1 << endl;
82                //cout << "Arg2: " << arg2 << endl;
83        }
84        //*****************************
85
86        public:
87
88        DataPath() {
89                //initialize all values to zero
90        for(int i =0; i<NUM_REGISTERS; i++){
91                registers[i] = 0;
92        }
93        pc = 0;
94        for(int i =0; i<MAX_MEMORY; i++){
95                ram[i] = 0;
96        }
97
98        }
99        void execute(){
100               //int tmp;
101               switch(opcode){
102                       case 0:
103                               //cout << "HALT" << endl;
104                               //pc++;
105                               break;
106                       case 1: //done
107                               //cout << "GOTO" << endl;
108                               pc = registers[arg1];
109                               break;
110                       case 2: // done
111                               //cout << "LDI" << endl;
112                               registers[arg1] = arg2;
113                               pc++;
114                               break;
115                       case 3:
116                               //cout << "ADD" << endl;
117                               registers[arg1] = registers[arg1] + registers[arg2];
118                               if(registers[arg1] >= MAX_VALUE){
119                                       registers[arg1] = registers[arg1] % MAX_VALUE;
```

```
120                             }
121                             pc++;
122                             break;
123                     case 4:
124                             //cout << "MULT" << endl;
125                             registers[arg1] = registers[arg1] * registers[arg2];
126                             if(registers[arg1] >= MAX_VALUE){
127                                     registers[arg1] = registers[arg1] % MAX_VALUE;
128                             }
129                             pc++;
130                             break;
131                     case 5:
132                             //cout << "LOAD" << endl;
133                             //registers[arg1] = ram[arg2];
134                             registers[arg1] = ram[registers[arg2]];
135                             if(registers[arg1] >= MAX_VALUE){
136                                     registers[arg1] = registers[arg1] % MAX_VALUE;
137                             }
138                             pc++;
139                             break;
140                     case 6:
141                             //cout << "STORE" << endl;
142                             //ram[arg1] = registers[arg2];
143                             ram[registers[arg1]] = ram[arg2];
144                             if(ram[arg1] >= MAX_VALUE){
145                                     ram[arg1] = ram[arg1] % MAX_VALUE;
146                             }
147                             pc++;
148                             break;
149                     default:
150                             //cout << "ERROR" << endl;
151                             pc++;
152                             break;
153             }
154     }
155     // step() executes the current instruction and
156     // loads the program counter with the address of
157     // the next instruction.
158     // Changes state of register file and memory as appropriate
159     //
160     // If the program has halted on a previous instruction,
161     // step() has no effect.  If HALT is encountered, PC is
162     // not updated
163     void step() {
164             getMemory(pc);
165             instructionDecode();
166
167             // if the program has halted, do nothing
168             execute();
169
170     }
171
172
173     // Returns the current value of the program counter
174     int getPC() {
175             return pc;
176     }
177
178     // Returns the code for the current instruction
179     int getCurrentInstruction() { //need to make sure this doesnt loop over 1k
180             if(pc >= MAX_MEMORY){
181                     pc = pc % MAX_MEMORY;
182             }
183             return ram[pc];
184     }
185
```

```
186         // Returns the value stored at Rx
187         //  if x is not within range, x mod (number of registers)
188         //  is returned
189         int getRegister(int x) {
190                 if(x < NUM_REGISTERS){
191                         return registers[x];
192                 }
193                 else{
194                         return registers[x % NUM_REGISTERS];
195                 }
196         }
197
198         //returns the value stored at M[address]
199         // if address is not in range, address mod (memory size) is
200         // assumed
201         int getMemory( int address) {
202                 if(address < MAX_MEMORY){
203                         return ram[address];
204                 }
205                 else{
206                         return ram[address % MAX_MEMORY];
207                 }
208         }
209
210         // stores value at M[address]
211     // if address is not in range, address mod (memory size) is
212     // assumed
213     void loadMemory( int address, int data) {
214                         /*if(address < MAX_MEMORY){
215                                 ram[address] = data;
216                         }
217                         else{
218                                 ram[address % MAX_MEMORY] = data;
219                         }*/
220                         if(address >= MAX_MEMORY){
221                                 address = address % MAX_MEMORY;
222                         }
223                         if(data >= 700){
224                                 data = data % MAX_VALUE;
225                         }
226                         ram[address] = data;
227         }
228
229
230         // Returns true of the program has halted.
231         bool isHalted() {
232         if(pc !=0){
233                 if(opcode == 0){
234                         return true;
235                 }
236                 else{
237                         return false;
238                 }
239         }
240         else{
241                 return false;
242         }
243         }
244
245         void print() {
246
247                 cout << "System Status:\n";
248
249                 //print each register, 5 to a line.
250                 for (int i=0; i<NUM_REGISTERS; i++) {
251                         cout << "R" << i << ": " << getRegister(i) << "\t";
```

```
252                     if ((i+1)%5 == 0) cout << "\n";
253             }
254
255             //if we don't have a multiple of 5 registers, add endln
256             if ((NUM_REGISTERS%5) != 0) cout << "/n";
257
258             cout << "PC: " << getPC() << "\t";
259             cout << "M[PC]: " << getCurrentInstruction() << "\n\n";
260
261     }
262
263
264 };
265
266
267 int main() {
268
269     int numInstr, tmpInstr;
270     cout << "Datapath Simulator"  << endl;
271     cout << "Implements Stop->Mult" << endl << endl;
272
273     DataPath* cpu = new DataPath();
274     cpu->print();
275     cout << endl << endl;
276
277
278     cout << "Instruction Loading..."  << endl;
279
280     cout << "Enter in # of Instructions to Load: ";
281     cin >> numInstr;
282
283     cout << "Enter in 3-digit instructions, one per line: ";
284
285     for (int i=0; i<numInstr; i++) {
286             cin >> tmpInstr;
287             cpu->loadMemory(i, tmpInstr);
288     }
289
290     cout << "Initializing Datapath...." << endl;
291     cout << "Datapath initial state:" << endl;
292     cpu->print();
293     cout << endl << endl;
294
295     while (!cpu->isHalted()) {
296             //cout << "Halt status: " << cpu->isHalted() << endl;
297             cpu->step();
298             cout << "Issuing Step Command:" << endl;
299             cpu->print();
300     }
301
302     return 0;
303 }
304
305 /* OUTPUTS
306 * TEST 1 (Last 4)
307 Issuing Step Command:
308 System Status:
309 R0: 0   R1: 222 R2: 2   R3: 6   R4: 4
310 R5: 5   R6: 6   R7: 7   R8: 16  R9: 30
311 PC: 19  M[PC]: 999
312
313 Issuing Step Command:
314 System Status:
315 R0: 0   R1: 222 R2: 2   R3: 6   R4: 4
316 R5: 5   R6: 6   R7: 7   R8: 16  R9: 30
317 PC: 20  M[PC]: 274
```

```
318
319 Issuing Step Command:
320 System Status:
321 R0: 0   R1: 222 R2: 2   R3: 6   R4: 4
322 R5: 5   R6: 6   R7: 4   R8: 16  R9: 30
323 PC: 21  M[PC]: 0
324
325 Issuing Step Command:
326 System Status:
327 R0: 0   R1: 222 R2: 2   R3: 6   R4: 4
328 R5: 5   R6: 6   R7: 4   R8: 16  R9: 30
329 PC: 21  M[PC]: 0
330
331
332 *TEST 2 (last 4)
333 Issuing Step Command:
334 System Status:
335 R0: 0   R1: 1   R2: 0   R3: 0   R4: 0
336 R5: 0   R6: 0   R7: 7   R8: 9   R9: 18
337 PC: 18  M[PC]: 568
338
339 Issuing Step Command:
340 System Status:
341 R0: 0   R1: 1   R2: 0   R3: 0   R4: 0
342 R5: 0   R6: 395 R7: 7   R8: 9   R9: 18
343 PC: 19  M[PC]: 687
344
345 Issuing Step Command:
346 System Status:
347 R0: 0   R1: 1   R2: 0   R3: 0   R4: 0
348 R5: 0   R6: 395 R7: 7   R8: 9   R9: 18
349 PC: 20  M[PC]: 0
350
351 Issuing Step Command:
352 System Status:
353 R0: 0   R1: 1   R2: 0   R3: 0   R4: 0
354 R5: 0   R6: 395 R7: 7   R8: 9   R9: 18
355 PC: 20  M[PC]: 0
356
357 *TEST 3 (last 4)
358 Issuing Step Command:
359 System Status:
360 R0: 18  R1: 18  R2: 0   R3: 0   R4: 0
361 R5: 0   R6: 0   R7: 801 R8: 0   R9: 18
362 PC: 34  M[PC]: 568
363
364 Issuing Step Command:
365 System Status:
366 R0: 18  R1: 18  R2: 0   R3: 0   R4: 0
367 R5: 0   R6: 211 R7: 801 R8: 0   R9: 18
368 PC: 35  M[PC]: 687
369
370 Issuing Step Command:
371 System Status:
372 R0: 18  R1: 18  R2: 0   R3: 0   R4: 0
373 R5: 0   R6: 211 R7: 801 R8: 0   R9: 18
374 PC: 36  M[PC]: 0
375
376 Issuing Step Command:
377 System Status:
378 R0: 18  R1: 18  R2: 0   R3: 0   R4: 0
379 R5: 0   R6: 211 R7: 801 R8: 0   R9: 18
380 PC: 36  M[PC]: 0
381
382 */
```