COMP5201 Assignment 3 Fall 2020 Prof. D.K. Probst

Output:

f: Fetched instruction: l|124.  
d: Set opc to 'l'  
d: Set D\_Out1 to 105.  
d: Set D\_Out2 to 4.  
d: Set dreg to f2.  
x: X\_Out = 105 + 4 = 109  
m: Set M\_Out to 19  
w: Set f2 to 19

f: Fetched instruction: l|348.  
d: Set opc to 'l'  
d: Set D\_Out1 to 203.  
d: Set D\_Out2 to 8.  
d: Set dreg to f4.  
x: X\_Out = 203 + 8 = 211  
m: Set M\_Out to 43  
w: Set f4 to 43

f: Fetched instruction: m|246.  
d: Set opc to 'm'  
d: Set D\_Out1 to 19.  
d: Set D\_Out2 to 43.  
d: Set dreg to f6.  
x: X\_Out = 19 \* 43 = 817  
m: No operation  
w: Set f6 to 817

f: Fetched instruction: a|468.  
d: Set opc to 'a'  
d: Set D\_Out1 to 43.  
d: Set D\_Out2 to 817.  
d: Set dreg to f8.  
x: X\_Out = 43 + 817 = 860  
m: No operation  
w: Set f8 to 860

f: Fetched instruction: b|550.  
d: Set opc to 'b'  
d: Set D\_Out1 to 301.  
d: Set D\_Out2 to 301.  
d: 301 == 301: equal, branch not taken  
x: No operation  
m: No operation  
w: No operation

f: Fetched instruction: s|368.  
d: Set opc to 's'  
d: Set D\_Out1 to 203.  
d: Set D\_Out2 to 8.  
d: Set sval to f6's value: 817.  
x: X\_Out = 203 + 8 = 211  
m: Store in memory: Mem[211] 'f6' that is equal to 817  
w: No operation

f: Fetched instruction: s|584.  
d: Set opc to 's'  
d: Set D\_Out1 to 301.  
d: Set D\_Out2 to 4.  
d: Set sval to f8's value: 860.  
x: X\_Out = 301 + 4 = 305  
m: Store in memory: Mem[305] 'f8' that is equal to 860  
w: No operation

f: Fetched instruction: b|790.  
d: Set opc to 'b'  
d: Set D\_Out1 to 148.  
d: Set D\_Out2 to 156.  
d: 148 != 156: not equal, branch taken  
d: Terminate program  
x: No operation  
m: No operation  
w: No operation

Code:

/\*

// this is the

l.d f2,4(r1) // program to be

l.d f4,8(r3) // simulated;

mul.d f6,f2,f4 // no pipelining

add.d f8,f4,f6

bne r5,r5,target

s.d f6,8(r3)

s.d f8,4(r5)

bne r7,r9,target

l.d f2,8(r7)

s.d f2,8(r7)

target:

\*/

#include <iostream> // for std::cout

int main()

{

// all submissions must contain this prelude, or equivalent

// register file

int Reg[10];

Reg[1] = 105;

Reg[3] = 203;

Reg[5] = 301; // r-register values

Reg[7] = 148;

Reg[9] = 156;

// main memory

int Mem[500];

char XMem[] = "llmabssbls"; // ten opcodes, and

Mem[0] = 124;

Mem[1] = 348;

Mem[2] = 246; // their ten argument

Mem[3] = 468;

Mem[4] = 550;

Mem[5] = 368; // lists; object code

Mem[6] = 584;

Mem[7] = 790;

Mem[8] = 728; // for ten instructions

Mem[9] = 728;

Mem[109] = 19;

Mem[156] = 25; // assorted data memory

Mem[211] = 43; // values

bool branch; // for bne results

std::cout << "\n"; // blank line

// loop over instructions; pretend next instruction is at PC + 1

for (int PC = 0; PC < 10; PC++)

{

// f-box FETCH

char head = XMem[PC]; // these are shared variables opcode

int tail = Mem[PC]; // that the d-box will use values

std::cout << "f: Fetched instruction: " << head << "|" << tail << ".\n";

// d-box DECODE

char opc = head; // decode instruction

int arg3 = tail % 10; // into its opcode and

tail = tail / 10; // its three arguments

int arg2 = tail % 10; // inst = {opc,arg1,...}

tail = tail / 10;

int arg1 = tail;

std::cout << "d: Set opc to '" << opc << "'\n";

// 'opc' is the next datapath "shared variable" to be initialized;

// 'arg1', 'arg2', 'arg3' are d-box local variables

// all code, or equivalent, above this line is mandatory

int D\_Out1, D\_Out2, dreg, sval; // more shared variables

switch (opc)

{ // You may imitate this style.

case 'a':

D\_Out1 = Reg[arg1]; // localize reg. operand and latch

std::cout << "d: Set D\_Out1 to " << D\_Out1 << ".\n";

D\_Out2 = Reg[arg2]; // localize reg. operand and latch

std::cout << "d: Set D\_Out2 to " << D\_Out2 << ".\n";

dreg = arg3; // latch dest. register designator

std::cout << "d: Set dreg to f" << dreg << ".\n";

break;

case 'b':

D\_Out1 = Reg[arg1]; // localize reg. operand and latch

std::cout << "d: Set D\_Out1 to " << D\_Out1 << ".\n";

D\_Out2 = Reg[arg2]; // localize reg. operand and latch

std::cout << "d: Set D\_Out2 to " << D\_Out2 << ".\n";

dreg = arg3; // latch dest. register designator

if (D\_Out1 != D\_Out2)

{

std::cout << "d: " << D\_Out1 << " != " << D\_Out2 << ": not equal, branch taken\n";

std::cout << "d: Terminate program\n";

PC += 2; // If they are not equal we want to target (skip the next two instructs)

}

else if (D\_Out1 == D\_Out2)

{

std::cout << "d: " << D\_Out1 << " == " << D\_Out2 << ": equal, branch not taken\n";

}

break;

case 'm':

D\_Out1 = Reg[arg1]; // localize reg. operand and latch

std::cout << "d: Set D\_Out1 to " << D\_Out1 << ".\n";

D\_Out2 = Reg[arg2]; // localize reg. operand and latch

std::cout << "d: Set D\_Out2 to " << D\_Out2 << ".\n";

dreg = arg3; // latch dest. register designator

std::cout << "d: Set dreg to f" << dreg << ".\n";

break;

case 'l':

D\_Out1 = Reg[arg1]; // localize reg. operand and latch

std::cout << "d: Set D\_Out1 to " << D\_Out1 << ".\n";

D\_Out2 = arg3; // localize reg. operand and latch

std::cout << "d: Set D\_Out2 to " << D\_Out2 << ".\n";

dreg = arg2; // latch dest. register designator

std::cout << "d: Set dreg to f" << dreg << ".\n";

break;

case 's':

D\_Out1 = Reg[arg1]; // data from this register is to be stored in D\_Out2+dreg

std::cout << "d: Set D\_Out1 to " << D\_Out1 << ".\n";

D\_Out2 = arg3; // byte-offset to add to D\_Out1 to have the address to store arg2

std::cout << "d: Set D\_Out2 to " << D\_Out2 << ".\n";

dreg = arg2; // byte-offset to add to D\_Out1 to have the address to store D\_Out2

std::cout << "d: Set sval to f" << dreg << "'s value: " << Reg[dreg] << ".\n";

break;

}

// x-box EXECUTE

int X\_Out;

switch (opc)

{

case 'a':

X\_Out = D\_Out1 + D\_Out2;

std::cout << "x: X\_Out = " << D\_Out1 << " + " << D\_Out2 << " = " << X\_Out << "\n";

break;

case 'b':

std::cout << "x: No operation\n";

break;

case 'l':

X\_Out = D\_Out1 + D\_Out2;

std::cout <<"x: X\_Out = "<<D\_Out1<< " + " <<D\_Out2<<" = "<< D\_Out1 + D\_Out2 << "\n";

break;

case 'm':

X\_Out = D\_Out1 \* D\_Out2;

std::cout << "x: X\_Out = " << D\_Out1 << " \* " << D\_Out2 << " = " << X\_Out << "\n";

break;

case 's':

X\_Out = D\_Out1 + D\_Out2;

std::cout << "x: X\_Out = " << D\_Out1 << " + " << D\_Out2 << " = " << X\_Out << "\n"; // address of memory to store in

break;

}

// m-box MEMORY

int M\_Out;

switch (opc) {

case 'l': //load

M\_Out = Mem[X\_Out];

std::cout << "m: Set M\_Out to " << M\_Out << "\n";

break;

case 's': //store

Mem[X\_Out] = Reg[dreg];

std::cout << "m: Store in memory: Mem[" << X\_Out << "] 'f" << dreg << "' that is equal to " << Reg[dreg] << "\n";

break;

// case else: not supposed to do anything

case 'a':

std::cout << "m: No operation\n";

break;

case 'b':

std::cout << "m: No operation\n";

break;

case 'm':

std::cout << "m: No operation\n";

break;

}

// w-box WRITE-BACK

switch (opc) {

case 'a':

Reg[dreg] = X\_Out;

std::cout << "w: Set f" << dreg << " to " << X\_Out << "\n";

break;

case 'l':

Reg[dreg] = Mem[X\_Out];

std::cout << "w: Set f" << dreg << " to " << Mem[X\_Out] << "\n";

break;

case 'm':

Reg[dreg] = X\_Out;

std::cout << "w: Set f" << dreg << " to " << X\_Out << "\n";

break;

case 's':

std::cout << "w: No operation\n";

break;

case 'b':

std::cout << "w: No operation\n";

break;

}

std::cout << "\n"; // blank line

}

}