CS203A Course Project: A Tomasulo Algorithm Simulation Sketch

Hand out date: Nov 4, 2004

Due date: Dec 14-15, 2004

By

Kan Liu, Piyush Ranjan Satapathy & Ricky J Sethi

Project Specification Given By Course Instructor, Dr Jun Yang

In this project, you will need to implement the Tomasulo algorithm for an out-of-order execution pipeline architecture. It is preferred that you write this project in C or C++. You should find one or two partner(s) to form a group of two or three. Please document your source code as you develop it. On the due day, you will demonstrate your project to me on a linux machine such as eon.cs.ucr.edu.

- **Pipeline Configuration:**

You will implement the following 4 stages: IF, Issue, Execute, and Writeback. The tasks performed in each stage were explained in class. You can refer to the detailed actions taken for bookkeeping from the textbook (Figure 3.5 on page 193). Those will help you greatly in coding the dynamic scheduler. You will implement only the FP pipeline. We will consider the following FUs:

1. FP adder which can perform both FP add and FP sub; pipelined
2. FP multiplier which performs FP multiplication; pipelined
3. FP divider which performs FP division; not pipelined
4. load unit; pipelined
5. store unit; pipelined
6. 32 general-purpose integer registers and 32 general-purpose floating-point registers; infinite number of ports for each register file.

The hardware configuration should be fully parameterizable, i.e., the number of reservation stations for each FU, and the number of execution cycles for each FU should all be the inputs to your simulator. You will need to simulate a "memory" which could be as simple as a data array. I'll leave the implementation details to you as long as you have your own way of initializing the memory and printing it out finally. During the demo, I will let you initialize your own memory at the addresses and values I defined. Your simulator should be able to read and write to your own memory correctly.

- **Instructions**

  Data Transfer Instructions

ld Fa, offset(Ra)	load a single precision floating point value to Fa
sd Fa, offset(Ra)	Store a single precision floating point value to memory

  Control Transfer Instructions

beq Rs, Rt, offset	if Rs==Rt then branch to PC+offset
bne Rs, Rt, offset	if Rs<>Rt then branch to PC+offset

  ALU Instructions

add Rd, Rs, Rt	Rd = Rs+Rt
add.d Fd, Fs, Ft	Fd = Fs+Ft
addi Rt, Rs, immediate	Rt=Rs+immediate
sub Rd, Rs, Rt	Rd = Rs-Rt
sub.d Fd, Fs, Ft	Fd = Fs-Ft
mult.d Fd, Fs, Ft	Fd = Fs*Ft, assuming that Fd is enough to hold the result
div.d Fd, Fs, Ft	Fd = Fs/Ft

Assume that each integer instruction takes only one cycle to finish (just as what we discussed in class) and integer instructions do not contend resources with FP instructions.

- **Inputs**

The input to your simulator is a text file containing assembly instructions specified in the above format. Read the input file and proceed your simulation with reading the file line by line as if you are fetching binary instructions from an instruction memory. The ID stage is also simplified to parsing the assembly instruction as opposed to decoding the binary.

Other inputs are: FU execution cycles as well as their allocated reservation station numbers. I will leave it to you on how to define the input format.

- **Outputs**

The output from your program should contain messages about the final instruction status table including the cycle information in each stage. It is possible that during the demo, I ask you to print out the status of the reservation status table and register result status table at any cycle.

The program execution results should be reflected in the integer register file, floating point register file, and the memory. You should provide clear output information of those.

- **Tips:**
    1. Write your own test code. Come up with the solutions by going through the algorithm cycle by cycle. Before you start coding, you should first understand fully how the algorithm works. Developing your own test cases helps a lot.
    2. Start from small sample code, one instruction at a time. If a single instruction doesn't work, neither does a sequence of code.
    3. Vary the input parameters during debugging. For example, change the number of reservation numbers and the FU execution latencies, and see how your program reacts.
    4. Write a powerful print_stat() that can generate pretty on-screen display of the reservation station status, instruction status, and the register result status on every cycle. This can be of great help in debugging.
    5. Also design a clear output display for registers and memory contents. This can help improve my impression on your code quality.

6. Divide the work among the group members. When you can distribute the load in a team efficiently, you can improve your time management and the quality of your project.
7. Make plans ahead of time. Conquer the project step by step. For example, instruction parsing, input interface, and output display can all be done first and separately. Leave at least one week for final testing and debugging. Don't wait till last minute.

Source Code:

```
/*****************************************************************
 *  File name: TOMASULO.h
 *       This is a class declaration to simulate the Tomasulo algorithm
 *         for an out-of-order execution pipeline architecture.
 *       Authors:
 *                       Kan Liu
 *                       Piyush Ranjan Satapathy
 *                       Ricky J. Sethi
 *       CS203A The University of California, Riverside
 *       11/15/2004
 *****************************************************************/

#include <deque>
#include <string>
using namespace std;

#ifndef  __TOMASULO_H__
#define __TOMASULO_H__

#define NUM_IN_GPR 32    //define the number of general-purpose integer registers.
#define NUM_FP_GPR 32    //define the number of general-purpose floating-point
registers.
#define SIZE_MEM 32      //define the size of memory(words).
#define NUM_FUS 5        //the number of Function Units.
#define CYCS_IN 1        //the number of cycles to finish an integer instruction.
#define EMPTY -1         //for convenient,define EMPTY to be -1
#define ZERO 0.0         // For memory comparison


/*Implement 11 instructions where 5 of them are integer instructions, 6 of them are
floating-point instructions.*/
enum Instrs {LD,SD,BEQ,BNE,ADD,ADD_D,ADDI,SUB,SUB_D,MULT_D,DIV_D};
enum FUs {READY=-1,ADDER=0,MULTIPLIER,DIVIDER,LOAD,STORE};
//Implement 5 Function Units.Add "READY" for convenience.
enum Res_States {CAL_AD,FIN_AD,NOT_EX,STARTED_EX,COMPLETED_EX};
//To show the current state of a reservation station.
```

```
//the data structure to represent the instruction status.
struct Instruction_Sta {
	Instrs Op;
	int rd;
	int rs;
	int rt;
	int in_fetch;
	int issue;
	int exe_start;
	int exe_compl;
	int wri_back;
	//int res_id;	//the index of the reservation station for this instruction.
};//end Instruction_Sta

//the data structure represent the type and fu index and res index of a reservation station.
struct Res_Sta_Id {
	FUs type;	//one of ADDER,MULTIPLIER,DIVIDER,LOAD,STORE
	int fu_in;
	int index;	//0~
};//end Res_Sta_Id

//the data structure to represent reservation stations.
struct Reservation_Sta {
	bool Busy;
	Instrs Op;
	float Vj;
	float Vk;
	struct Res_Sta_Id Qj;
	struct Res_Sta_Id Qk;
	int A;

	int des;
	Res_States curr_sta;
	int Exe;
	struct Instruction_Sta* curr_ins;
};//end Reservation_Sta

//the data structure to represent an instruction.
struct Instruction {	//please refer enum Instrs
{LD,SD,BEQ,BNE,ADD,ADD_D,ADDI,SUB,SUB_D,MULT_D,DIV_D};
	Instrs Op;
	int label;
	int rd;
	int rs;
	int rt;
```

```
};//end Instruction

//the data structure to represent the index of a reservation stations
struct Res_Fu_Index{
	int in_fu;//the index of the fus
	int in_res;//the index of res in a fu
};


//class declaration.
class TOMASULO {

private:
	int In_Gpr[NUM_IN_GPR];	//the array of general-purpose integer registers.
	float Fp_Gpr[NUM_FP_GPR];	//the array of general-purpose floating-point
registers.
	float Mem[SIZE_MEM];	//the array to represent the memory.
	int PC;			//the PC register.
	int cycle;

	string input_filename;
	int Num_FU[NUM_FUS];				//the array holding the number of
the 5 kinds of Function units
	int *Num_Res_Stas[NUM_FUS];		//the array holding the number of
reservation stations for each FU.
	int *Exe_Cycs[NUM_FUS];			//the array holding the number of
execution cycles for each FU.

	/* we have five kinds of FUs, each has several FUs, each FU has several Res
stas.*/
	struct Reservation_Sta ** Res_Stas[NUM_FUS];  //the three-dimentional
reservation stations.
	//bool * FU_Stas[NUM_FUS];					//indicate the
status of fus.
	bool mem_busy;// true means there is a ins using the memory port
	bool *div_busy;//true means there is a div ins using the fu
	struct Res_Sta_Id Register_Stas[NUM_FP_GPR]; //the array for register status.
	struct Instruction_Sta * Intr_reg;		//the instruction waiting for issuing

	deque <struct Instruction> inst;		// RJS: deque of instructions
	deque <struct Instruction_Sta> Intrs_Stas;	//the queue containing instruction
status.
	deque <struct Res_Fu_Index> Load_q;	//load queue
	deque <struct Res_Fu_Index> Store_q;	//store queue

public:
```

```
        TOMASULO();          //constructor.
        ~TOMASULO();         //destructor.

        int init_in_gpr();    //Initialize general-purpose integer registers.
        int init_fp_gpr();    //Initialize general-purpose floating-point registers.
        int init_mem();       //Initialize memory;

        bool init_from_file(const char *);      // RJS: Initialize integer registers, fp
registers, and memory from file

        void print_stat();                  //RJS: Display res sta stat, inst stat, and reg result
stat on every cycle
        void print_final_stats();             // RJS: Print the summary
        bool parse_input_assembly(const char*); // RJS: Parse the input file

        int simu_run();                     //simulate the algorithm and run on input assembly
instructions.

        void print_stat_local();
        void print_mem();                    //Display the content of memory.
        void print_in_gpr();                 //Display the content of integer registers.
        void print_fp_gpr();                 //Display the content of floating point registers.
        void print_ins(ostream & out);         //Display the instructions in the queue "inst".

        void check_regsiter(struct Instruction i);  //make sure that the register no. is in
valid range.
        bool check_Issue();  //check whether an instruction is waiting for issuing,issue an
instruction when it is possible
        void Issue_book(int ty,int fu,int in);           //bookkeeping for issue
        void Issue_arith(FUs ty,int fu,int in);          //bookkeeping for FP operation issue
        bool fp_arith_issue(FUs ty);
        bool load_issue();
        bool store_issue();
        bool check_start_exe(int ty,int fu);       //check whether the reservation station can
start execution.
        bool check_compl_exe(int ty,int fu,int in);       //check whether the reservation
station finishs execution.
        bool write_result(int ty,int fu,int in);          //write result
        bool Instr_Fetch(int p);                 //instruction fetch.
        bool cal_address(int ty,int fu,int in);                    //calculate address for load
and store
        bool memory_Access();                                      //memory
access(load or store),every cycle,at most one ins stared

};//end class TOMASULO
```

```
#endif

/*****************************************************************
 *  File name: TOMASULO.cpp, v0.2.3
 *        This is a class to simulate the Tomasulo algorithm for an
 *        out-of-order execution pipeline architecture.
 *        Authors:
 *                        Kan Liu
 *                        Piyush Ranjan Satapathy
 *                        Ricky J. Sethi
 *        CS203A The University of California, Riverside
 *        11/15/2004
 *
 *        TODO:
 *             - Add # Cycles in Memory, # of FUs to init
 *
 *****************************************************************/

#include "TOMASULO.h"
#include <iostream>  // Need to do tons of I/O
#include <string>    // Make sure we use strings
#include <fstream>   // Get the filestream stuff (in and out)
#include <cstdio>    // For stderr, etc.
#include <map>       // For the maps
#include <vector>    // For the vectors
using namespace std;


//=================================================================
======
// Some Constant and Global Variables
//=================================================================
======
const int DEBUG         = 0;
const int MAX_SIZE      = 256;
const int MAX_BUFFER    = 65536;

const char * DELIM      = ", \t()RF";
const char * DELIM2     = ", \t";
const char LABEL_END    = ':';
const char PAREN        = '(';
const char REGF         = 'F';
const char REGI         = 'R';
const int FP_REG_START  = 32;

const int WORDSIZE      = 4;
```

```
const string HTMLFILE    = "output.html";



//=================================================================
======
// Prototypes:
//=================================================================
======
void dbg(string msg);

inline string fixByteOffset(string imm) {
  int foo = atoi(imm.c_str());
  if ((foo % WORDSIZE) != 0) {
    cerr << "ERROR:  Byte Offset for " << imm << " is not on " << WORDSIZE << endl;
    exit(-1);
  } else {
    foo = foo / WORDSIZE;
  }
  char bar[MAX_SIZE];
  sprintf(bar, "%d", foo);
  return bar;
}

inline string isEmpty(int foo) {
  char buffer[50]; sprintf(buffer, "%d", foo); string bar = buffer;
  return (foo == EMPTY) ? "" : bar;
}

inline string isEmpty(float foo) {
  char buffer[50]; sprintf(buffer, "%f", foo); string bar = buffer;
  return (foo == EMPTY) ? "" : bar;
}

inline string isEmptyStart(int foo) {
  char buffer[50]; sprintf(buffer, "%d - ", foo); string bar = buffer;
  return (foo == EMPTY) ? "" : bar;
}

inline string isEmptyMem(float foo) {
  char buffer[50]; sprintf(buffer, "%f", foo); string bar = buffer;
  return (foo == ZERO) ? "" : bar;
}
```

```
//=============================================================================
// Functions:
//=============================================================================

/**
 *
 * FUNCTION:      dbg(msg)
 * DESCRIPTION:   Local debug fxn
 * RECEIVE:       Uses the following EXPLICIT parameters:
 *                - msg: string, message to print out
 *              Uses the following IMPLICIT parameters:
 *                - ostream & cout
 * RETURN:        EXPLICITLY returns:
 *                - nothing
 *              IMPLICITLY returns:
 *                - none
 * OUTPUT:        Prints out the msg (if active only)
 * PRECONDITION:  msg is a valid string
 * POSTCONDITION: msg was printed
 *
 ****************************************************************/

void dbg(string msg) {
  //cout << msg << endl;
  //msg = "";
  return;
}


/**
 *
 * FUNCTION:      parse_input_assembly()
 * DESCRIPTION:   Parses the input assembly file and stores instructions in the deque,
inst
 * RECEIVE:       Uses the following EXPLICIT parameters:
 *                - filename: const char *, input assembly filename
 *              Uses the following IMPLICIT parameters:
 *                - none
 * RETURN:        EXPLICITLY returns:
 *                - bool, status
 *              IMPLICITLY returns:
 *                - nothing
 * OUTPUT:        Debugging statements
```

```
 * PRECONDITION:  filename is a valid string and file exists and contains assembly
instructions with format like "sd Fa, offset(Ra)"
 * POSTCONDITION: instructions from file are in the inst deque
 *
 ***************************************************************/

bool TOMASULO::parse_input_assembly(const char * filename) {
  input_filename = filename;

  // Let's open the input file first:
  ifstream infile;
  infile.open(filename,ios::in);
  if (infile.fail()) {
    cerr << "Couldn't open file: " << filename << endl;
    exit(-1);
  }

  // Tokenize each line of assembly code into space or tab delimited tokens.
  // Check for branch/loop labels (starts with L1:), offset in offset(R3), etc.
  // Dump each instruction structure into the inst deque
  char buf[MAX_BUFFER];
  while (infile.getline(buf, MAX_BUFFER)) {
    char * word;
    vector<char *> words;
          Instrs opcode;
    string token, label, op, rd, rs, rt;

    int i_rs=EMPTY,i_rt=EMPTY,i_rd=EMPTY,i_label=EMPTY;

    word = strtok(buf, DELIM);
          if(word == NULL)//blank line.
                    continue;
    while (word) {
      words.push_back(word);
      word = strtok(NULL, DELIM);
    }


    // Get the first assembly token:
    token = words[0];

    // Is first entry a label?
    if (token[token.size()-1] == LABEL_END) {
      // TODO: Should label's value be a word address instead of byte address (divide by
4)?
      label = token.substr(0,token.size()-1);        // Get the label (sans the i_label_END)
```

```
  words.erase(words.begin(), words.begin()+1);   // Hose the label
  i_label = atoi(label.c_str());                 // Get label ready for export...
}

// Set the operator:
op = words[0];

// Parse the op:
if (op == "sd" || op == "ld") {
  rd = words[1];                          // Destination
  rt = words[2];                          // Immediate
  rs = words[3];                          // Source
        if(op == "ld")
                opcode=LD;
        else
                opcode=SD;

  //rt = fixByteOffset(rt);

} else if (op == "beq" || op == "bne") {
  rs = words[1];                          // Destination
  rt = words[2];                          // Source
  rd = words[3];                          // Immediate

  if(op == "beq")
       opcode=BEQ;
  else
       opcode=BNE;

  rd = fixByteOffset(rd);


} else if (op == "addi") {
  rd = words[1];                          // Destination
  rs = words[2];                          // Source
  rt = words[3];                          // Immediate
        opcode=ADDI;

  //rt = fixByteOffset(rt);

} else {
  rd = words[1];                          // Destination
  rs = words[2];                          // Source 1
  rt = words[3];                          // Source 2
        if(op == "add")
                opcode=ADD;
```

```
            else if(op == "add.d")
                    opcode=ADD_D;
            else if(op == "sub")
                    opcode=SUB;
            else if(op == "sub.d")
                    opcode=SUB_D;
            else if(op == "mult.d")
                    opcode=MULT_D;
            else if(op == "div.d")
                    opcode=DIV_D;
            else{
                    cout<<"Not a valid operation code:"<<op<<endl;
                    continue;
            }
    }
    if (DEBUG) {cout<<op<<"\t"<<rd<<"\t"<<rs<<"\t"<<rt<<endl;}
    // Now, let's hose the register prefixes:
    i_rs = atoi(rs.c_str());
          i_rt = atoi(rt.c_str());
          i_rd = atoi(rd.c_str());

          if (DEBUG) {cout << "Value of rd is: " << rd << " and value of i_rd is: " << i_rd
<< endl;}

    // Add to the inst deque:
    struct Instruction i;
    i.Op=opcode;
    i.label = i_label;
    i.rd = i_rd;
    i.rs = i_rs;
    i.rt = i_rt;
          check_regsiter(i);
    inst.push_back(i);

  }  // End while Loop
  infile.close();

  return true;
}



/**
 *
 * FUNCTION:      init_from_file(const char *)
 * DESCRIPTION:   Parses the initialization input file
```

```
 * RECEIVE:       Uses the following EXPLICIT parameters:
 *                - const char * filename
 *              Uses the following IMPLICIT parameters:
 *                - INITFILE
 *                - In_Gpr[NUM_IN_GPR]
 *                - Fp_Gpr[NUM_FP_GPR]
 *                - Mem[SIZE_MEM]
 * RETURN:        EXPLICITLY returns:
 *                - bool, status
 *              IMPLICITLY returns:
 *                - nothing
 * OUTPUT:        Debugging statements
 * PRECONDITION:  INITFILE is valid and is accessible and contains correct
initialization strings (comma separated)
 * POSTCONDITION: The int reg, fp reg, and memory is initialized to specified values
 *
 ****************************************************************/

bool TOMASULO::init_from_file(const char * filename) {
  // Let's open the input file first:
  ifstream infile;
  infile.open(filename,ios::in);
  if (infile.fail()) {
    cerr << "Couldn't open file: " << filename << endl;
    //;exit(-1);
    return false;
  }



  // Vectors for FUs
  vector<int> vAddRS;
  vector<int> vAddEC;
  vector<int> vMultRS;
  vector<int> vMultEC;
  vector<int> vDivRS;
  vector<int> vDivEC;
  vector<int> vLoadRS;
  vector<int> vLoadEC;
  vector<int> vStoreRS;
  vector<int> vStoreEC;



  // Tokenize each line of init file into space or comma delimited tokens.
  char buf[MAX_BUFFER];
```

```
  while (infile.getline(buf, MAX_BUFFER)) {
    // For the tokenizing:
    char * word;
    vector<char *> words;
    string token;

    word = strtok(buf, DELIM2);

    if (word == NULL)    // Skip blank lines
      continue;
    if (word[0] == '#')  // Skip comment lines
      continue;

    while (word) {       // Store the rest
      words.push_back(word);
      word = strtok(NULL, DELIM2);
    }


    // Get the line's first token:
    token = words[0];

    if (token.at(0) == 'I') {
      // Initialize Integer registers
      int limit = (words.size()+1 < NUM_IN_GPR) ? words.size()+1 : NUM_IN_GPR;
      for (int i=1; i<limit; i++) {
           In_Gpr[i-1] = atoi(words[i]);
      }
    } else if (token.at(0) == 'F') {
      // Initialize FP registers
      int limit = (words.size()+1 < NUM_FP_GPR) ? words.size()+1 : NUM_FP_GPR;
      for (int i=1; i<limit; i++) {
           Fp_Gpr[i-1] = atof(words[i]);
      }
    }
           /*else if (token.at(0) == 'S') {
      // Initialize Reservation Stations
      int limit = (words.size()+1 < NUM_FUS) ? words.size()+1 : NUM_FUS+1;
      for (int i=1; i<limit; i++) {
           Num_Res_Stas[i-1] = atoi(words[i]);
           if (DEBUG) {cout << "Res Stats 2nd #: " << i << "=" << Num_Res_Stas[i-1] <<
endl;}
      }
    } else if (token.at(0) == 'E') {
      // Initialize Execution Cycles
      int limit = (words.size()+1 < NUM_FUS) ? words.size()+1 : NUM_FUS+1;
```

```
    for (int i=1; i<limit; i++) {
        Exe_Cycs[i-1] = atoi(words[i]);
        if (DEBUG) {cout << "Exe Cycles #: " << i << "=" << Exe_Cycs[i-1] << endl;}
    }
  }*/

  // Get the FUs:
  else if (token.at(0) == 'A') {
    // Get Adder FU
    vAddRS.push_back(atoi(words[1]));
    vAddEC.push_back(atoi(words[2]));
  } else if (token.at(0) == 'M') {
    // Get Multiplier FU
    vMultRS.push_back(atoi(words[1]));
    vMultEC.push_back(atoi(words[2]));
  } else if (token.at(0) == 'D') {
    // Get Divider FU
    vDivRS.push_back(atoi(words[1]));
    vDivEC.push_back(atoi(words[2]));
  } else if (token.at(0) == 'L') {
    // Get Loader FU
    vLoadRS.push_back(atoi(words[1]));
    vLoadEC.push_back(atoi(words[2]));
  } else if (token.at(0) == 'S') {
    // Get Storer FU
    vStoreRS.push_back(atoi(words[1]));
    vStoreEC.push_back(atoi(words[2]));
  } else {
    // Initialize Memory array
    int limit = (words.size()+1 < SIZE_MEM) ? words.size()+1 : SIZE_MEM;
    for (int i=1; i<limit; i++) {
        Mem[i-1] = atof(words[i]);
    }
  }


}  // End while Loop
infile.close();



// DEBUG:
if (DEBUG) {
    for (int x=0; x<vAddRS.size(); x++) {
        cout << "Adder" << x+1 << ": " << vAddRS[x] << ", " << vAddEC[x] << endl;
    }
```

```
    for (int x=0; x<vMultRS.size(); x++) {
        cout << "Multiplier" << x+1 << ": " << vMultRS[x] << ", " << vMultEC[x] <<
endl;
    }
 }


 // Setup the sizes:
 vector<int> vSizes;
 vSizes.push_back(vAddRS.size());
 vSizes.push_back(vMultRS.size());
 vSizes.push_back(vDivRS.size());
 vSizes.push_back(vLoadRS.size());
 vSizes.push_back(vStoreRS.size());


 int i,j,k;

        for(i=0; i<NUM_FP_GPR; i++){//initialize Register Status
                Register_Stas[i].type=READY;
                Register_Stas[i].fu_in=EMPTY;
                Register_Stas[i].index=EMPTY;
        }


        /* i can be adder ~store
        j vary from 0 ~ num-1 fus
        k vary from 0~ res_num-1 for each FU*/
        for(i=0; i<NUM_FUS; i++){//initialize Reservation Stations.
          //;Num_FU[i]=2;//initialize number of fus (a,m,d,l,s)
          Num_FU[i]=vSizes[i];//initialize number of fus (a,m,d,l,s)

                Res_Stas[i]=new struct Reservation_Sta*[Num_FU[i]];//[Num_FU[i]];//5
kind of fus
                Num_Res_Stas[i]=new int[Num_FU[i]];//5 kind of fus
                Exe_Cycs[i]=new int[Num_FU[i]];//5 kind of fus

                for(j=0; j < Num_FU[i]; j++){
                  //;Num_Res_Stas[i][j]=2;
                  //;Exe_Cycs[i][j]=4;
                  switch(i) {
                  case 0:
                    Num_Res_Stas[i][j]=vAddRS[j];
                    Exe_Cycs[i][j]=vAddEC[j];
                    break;
                  case 1:
```

```
                    Num_Res_Stas[i][j]=vMultRS[j];
                    Exe_Cycs[i][j]=vMultEC[j];
                    break;
                  case 2:
                    Num_Res_Stas[i][j]=vDivRS[j];
                    Exe_Cycs[i][j]=vDivEC[j];
                    break;
                  case 3:
                    Num_Res_Stas[i][j]=vLoadRS[j];
                    Exe_Cycs[i][j]=vLoadEC[j];
                    break;
                  case 4:
                    Num_Res_Stas[i][j]=vStoreRS[j];
                    Exe_Cycs[i][j]=vStoreEC[j];
                    break;
                  }
                        Res_Stas[i][j]=new struct
Reservation_Sta[Num_Res_Stas[i][j]];//5 kind of fus
                        for(k=0; k < Num_Res_Stas[i][j];k++){
                                Res_Stas[i][j][k].Busy=false;
                                Res_Stas[i][j][k].Qj.type=READY;   // RJS: added proper
intialization
                                Res_Stas[i][j][k].Qj.fu_in=EMPTY;
                                Res_Stas[i][j][k].Qj.index=EMPTY;
                                Res_Stas[i][j][k].Vj=EMPTY;
                                Res_Stas[i][j][k].Qk.type=READY;
                                Res_Stas[i][j][k].Qk.fu_in=EMPTY;
                                Res_Stas[i][j][k].Qk.index=EMPTY;
                                Res_Stas[i][j][k].Vk=EMPTY;
                        }
                }
        }


  return true;
}

/*constructor
 */
TOMASULO::TOMASULO(){
  // Initialize cycle var:
        cycle = 0;
        //the array of general-purpose integer registers.
    init_in_gpr();
        //Initialize general-purpose floating-point registers.
        init_fp_gpr();
```

```
        //Initialize memory;
        init_mem();
        PC=0;                   //the PC register.
        cycle=0;

        for(int i=0;i<NUM_FUS;i++){
                Num_Res_Stas[i]=NULL;
                Exe_Cycs[i]=NULL;
                Res_Stas[i]=NULL;
        }
        mem_busy=false;// true means there is a ins using the memory port
        div_busy=NULL;//true means there is a div ins using the fu

}//end of constructor.

//destructor
TOMASULO::~TOMASULO(){
        for(int i=0; i<NUM_FUS; i++){
                if(NULL != Res_Stas[i]){
                        for(int j=0; j<Num_FU[i]; j++){
                                if(NULL !=Res_Stas[i][j])
                                        delete[] Res_Stas[i][j];
                        }
                        delete Res_Stas[i];
                }
                if(NULL !=Num_Res_Stas[i])
                        delete[] Num_Res_Stas[i];
                if(NULL !=Exe_Cycs[i])
                        delete[] Exe_Cycs[i];
        }
        if(div_busy != NULL)
                delete[] div_busy;
        Intrs_Stas.~deque();
}//end of destructor.

/**
 *
 * FUNCTION:      print_stat()
 * DESCRIPTION:   Displays the reservation station status, instruction status, and the
register result status for ONE cycle
 * RECEIVE:       Uses the following EXPLICIT parameters:
 *                - none
 *               Uses the following IMPLICIT parameters:
 *                - Intrs_Stas deque (Instruction Status)
 *                - Res_Stas array (Reservation Stations)
 *                - Register_Stas array (Register Status)
```

```
 *                - cycle integer
 * RETURN:        EXPLICITLY returns:
 *                - nothing
 *               IMPLICITLY returns:
 *                - nothing
 * OUTPUT:        HTML file containing the output
 * PRECONDITION:  The deques exist and we can write a file here
 * POSTCONDITION: The HTML file is created with the tables
 *
 ************************************************************************/

void TOMASULO::print_stat() {
  // Declare our locals:
  struct Instruction_Sta is;                    // Use with Intrs_Stas
  deque <struct Instruction_Sta> i = Intrs_Stas;      // Copy Intrs_Stas deque
  vector<string> vResStations;                  // Map integers to reservations tations
  vResStations.push_back("ADDER");
  vResStations.push_back("MULTIPLIER");
  vResStations.push_back("DIVIDER");
  vResStations.push_back("LOAD");
  vResStations.push_back("STORE");


  vector<string> vInstructions;                 // Map integers to reservations tations
  vInstructions.push_back("LD");
  vInstructions.push_back("SD");
  vInstructions.push_back("BEQ");
  vInstructions.push_back("BNE");
  vInstructions.push_back("ADD");
  vInstructions.push_back("ADD_D");
  vInstructions.push_back("ADDI");
  vInstructions.push_back("SUB");
  vInstructions.push_back("SUB_D");
  vInstructions.push_back("MULT_D");
  vInstructions.push_back("DIV_D");


  // Let's open the output file first:
  ofstream outfile;

  // Start output (check for 2 since odd incrementing in simu_run() -- added additional
print_stat() to compensate for it):
  if (cycle == 1) {
    outfile.open(HTMLFILE.c_str(),ios::out);
    if (outfile.fail()) {
      cerr << "Couldn't open file: " << HTMLFILE << endl;
```

```
    exit(-1);
   }

   outfile << "<html>\n<title>Tomasulo Simulation Results</title>" << endl;
   outfile << "<head><link href='http://www.cs.ucr.edu/~rsethi/includes/stylesheet-
default.css' type='text/css' rel='stylesheet' /></head>\n<body>" << endl;
   outfile << "<h1><b>Tomasulo Simulation Results</b></h1><hr />" << endl;
   outfile << "<h3>The Input Assembly Code (Original and Translation)</h3>" << endl;
   outfile << "<pre><code>" << endl;
   print_ins(outfile); outfile << "</code></pre><hr />" << endl;
  } else {
   outfile.open(HTMLFILE.c_str(),ios::out | ios::app);
   if (outfile.fail()) {
    cerr << "Couldn't open file: " << HTMLFILE << endl;
    exit(-1);
   }
  }

  // ----------------------------------------------
  // Print out the instruction status
  // ----------------------------------------------
  outfile << "<h3>Instruction Status for Cycle " << cycle << "</h3>" << endl;
  outfile << "<table cols='7' border='1' cellspacing='2' cellpadding='2'>" << endl;
  outfile <<
"<tr><th>Instruction</th><th>i</th><th>j</th><th>k</th><th>Issue</th><th>Execute<
/th><th>Write-Back</th></tr>" << endl;
  while (!i.empty()) {
   is = i.front();
   i.pop_front();

   outfile << "<tr><td>" << vInstructions[int(is.Op)]  << "</td><td>" << isEmpty(is.rd)
<< "</td><td>" << isEmpty(is.rs) << "</td><td>" << isEmpty(is.rt) << "</td><td>" <<
isEmpty(is.issue) << "</td><td>" << isEmptyStart(is.exe_start) <<
isEmpty(is.exe_compl) << "</td><td>" << isEmpty(is.wri_back) << "</td></tr>" <<
endl;
  }
  outfile <<"</table>" << endl;


  // ----------------------------------------------
  // Print out the reservation stations
  // ----------------------------------------------
  outfile << "<h3>Reservation Stations</h3>" << endl;
  outfile << "<table cols='7' border='1' cellspacing='2' cellpadding='2'>" << endl;
```

```
  outfile <<
"<tr><th>Name</th><th>Busy</th><th>Op</th><th>Vj</th><th>Vk</th><th>Qj</th>
<th>Qk</th></tr>" << endl;
  //; This only got the FIRST element of each row!  Skipped the other columns!
  //;for (int x=0; x<NUM_FUS; x++)
  //;outfile << "<tr><td>" << Res_Stas[x]->Op  << "</td><td>" << Res_Stas[x]->Busy
<< "</td><td>" << Res_Stas[x]->Op << "</td><td>" << Res_Stas[x]->Vj <<
"</td><td>" << Res_Stas[x]->Vk << "</td><td>" << Res_Stas[x]->Qj.type <<
Res_Stas[x]->Qj.index << "</td><td>" << Res_Stas[x]->Qk.type << Res_Stas[x]-
>Qk.index << "</td></tr>" << endl;


  for(int k=0; k<NUM_FUS; k++){
    for(int j=0; j<Num_FU[k]; j++){
      for(int n=0; n<Num_Res_Stas[k][j]; n++){
          string qj, qk, busy;
          qj = (Res_Stas[k][j][n].Qj.type == FUs(EMPTY)) ? "" :
vResStations[int(Res_Stas[k][j][n].Qj.type)];
          qk = (Res_Stas[k][j][n].Qk.type == FUs(EMPTY)) ? "" :
vResStations[int(Res_Stas[k][j][n].Qk.type)];
          busy = (Res_Stas[k][j][n].Busy == 0) ? "No" : "Yes";

          if (busy == "Yes") {
            outfile << "<tr><td>" << vResStations[k] << j  << n << "</td><td>" << busy
<< "</td><td>" << Res_Stas[k][j][n].Op << "</td><td>" <<
isEmpty(Res_Stas[k][j][n].Vj) << "</td><td>" << isEmpty(Res_Stas[k][j][n].Vk) <<
"</td><td>" << qj << isEmpty(Res_Stas[k][j][n].Qj.fu_in) <<
isEmpty(Res_Stas[k][j][n].Qj.index) << "</td><td>" << qk <<
isEmpty(Res_Stas[k][j][n].Qk.fu_in) << isEmpty(Res_Stas[k][j][n].Qk.index) <<
"</td></tr>" << endl;
          } else {
            outfile << "<tr><td>" << vResStations[k] << j  << n << "</td><td>" << busy
<< "</td><td></td><td></td><td></td><td></td><td></td></tr>" << endl;
          }
      }
    }
  }

  outfile <<"</table>" << endl;


  // ---------------------------------------------
  // Print out the register status
  // ---------------------------------------------
  int cols=NUM_FP_GPR;
  outfile << "<h3>Register Result Status</h3>" << endl;
```

```
  outfile << "<table cols='" << cols << "' border='1' cellspacing='2' cellpadding='2'>" <<
endl;
  outfile << "<tr>" << endl;
  for (int x=0; x<cols; x++) {
    outfile << "<th>" << REGF << x << "</th>";
    //;if (x%cols == 0) {
    //;  outfile << "</tr>\n<tr>" << endl;
    //;}
  }
  outfile << "</tr>" << endl;
  outfile << "<tr>" << endl;
  for (int x=0; x<cols; x++) {
    string reg;
    reg = (Register_Stas[x].type == FUs(EMPTY)) ? "" :
vResStations[int(Register_Stas[x].type)];
    //;outfile << "<td>" << Register_Stas[x].type << Register_Stas[x].index << "</td>";
    outfile << "<td>" << reg << isEmpty(Register_Stas[x].fu_in) <<
isEmpty(Register_Stas[x].index) << "</td>";
    //;if (x%cols == 0) {
    //;  outfile << "</tr>\n<tr>" << endl;
    //;}
  }
  outfile << "</tr>" << endl;
  outfile <<"</table>" << endl;



  // End the output
  outfile <<"<br /><hr />" << endl;
  outfile.close();
  return;
}



/**
 *
 * FUNCTION:      print_final_stats()
 * DESCRIPTION:   Displays memory, integer, and fp registers and closes HTMLFILE
 * RECEIVE:       Uses the following EXPLICIT parameters:
 *                - none
 *               Uses the following IMPLICIT parameters:
 *                - Mem[SIZE_MEM] (Memory array)
 *                - In_Gpr[NUM_IN_GPR] (Integer registers)
 *                - Fp_Gpr[NUM_FP_GPR] (FP registers)
 * RETURN:        EXPLICITLY returns:
```

```
 *                - nothing
 *              IMPLICITLY returns:
 *                - nothing
 * OUTPUT:        HTML file containing the output
 * PRECONDITION:  The memory and register arrays exist and are properly intialized
 * POSTCONDITION: The HTML file is created with the tables
 *
 ***************************************************************************/

void TOMASULO::print_final_stats() {
  int cols = 0;

  // Let's open the output file first:
  ofstream outfile;
  outfile.open(HTMLFILE.c_str(),ios::out | ios::app);
  if (outfile.fail()) {
    cerr << "Couldn't open file: " << HTMLFILE << endl;
    exit(-1);
  }

  // ------------------------------------------
  // Print out the memory contents
  // ------------------------------------------
  cols=SIZE_MEM;
  outfile << "<h1>The Final States</h1>" << endl;
  outfile <<"<hr />" << endl;

  outfile << "<h3>The Memory Contents</h3>" << endl;
  outfile << "<table cols='" << cols << "' border='1' cellspacing='2' cellpadding='2'>" <<
endl;
  outfile << "<tr>" << endl;
  for (int x=0; x<cols; x++) {
    outfile << "<th>M" << x << "</th>";
  }
  outfile << "</tr>" << endl;
  outfile << "<tr>" << endl;
  for (int x=0; x<cols; x++) {
    outfile << "<td>" << isEmptyMem(Mem[x]) << "</td>";
  }
  outfile << "</tr>" << endl;
  outfile <<"</table>" << endl;


  // ------------------------------------------
  // Print out the Integer Registers
  // ------------------------------------------
```

```
  cols=NUM_IN_GPR;
  outfile << "<h3>The Integer Registers</h3>" << endl;
  outfile << "<table cols='" << cols << "' border='1' cellspacing='2' cellpadding='2'>" <<
endl;
  outfile << "<tr>" << endl;
  for (int x=0; x<cols; x++) {
    outfile << "<th>R" << x << "</th>";
  }
  outfile << "</tr>" << endl;
  outfile << "<tr>" << endl;
  for (int x=0; x<cols; x++) {
    outfile << "<td>" << isEmptyMem(float(In_Gpr[x])) << "</td>";
  }
  outfile << "</tr>" << endl;
  outfile <<"</table>" << endl;


  // -------------------------------------------
  // Print out the FP Registers
  // -------------------------------------------
  cols=NUM_FP_GPR;
  outfile << "<h3>The FP Registers</h3>" << endl;
  outfile << "<table cols='" << cols << "' border='1' cellspacing='2' cellpadding='2'>" <<
endl;
  outfile << "<tr>" << endl;
  for (int x=0; x<cols; x++) {
    outfile << "<th>F" << x << "</th>";
  }
  outfile << "</tr>" << endl;
  outfile << "<tr>" << endl;
  for (int x=0; x<cols; x++) {
    outfile << "<td>" << isEmptyMem(Fp_Gpr[x]) << "</td>";
  }
  outfile << "</tr>" << endl;
  outfile <<"</table>" << endl;



  // End the output
  outfile <<"<br /><hr />" << endl;
  outfile << "</body></html>" << endl;
  outfile.close();
  return;
}
```

```
/**
 *
 * FUNCTION:      check_regsiter()
 * DESCRIPTION:   Check whether an instruction use an invalid register.
 */
void TOMASULO::check_regsiter(struct Instruction i){
        if(i.Op == LD || i.Op == SD){//rt---immediate
                if(i.rd<0 || i.rd>=NUM_FP_GPR || i.rs<0 || i.rs>=NUM_IN_GPR){
                        cout<<"invalid register number."<<endl;
                        exit(0);
                }
        }
        else if(i.Op == BEQ || i.Op == BNE){//rd---immediate
                if(i.rt<0 || i.rt>=NUM_IN_GPR || i.rs<0 || i.rs>=NUM_IN_GPR){
                        cout<<"invalid register number."<<endl;
                        exit(0);
                }
        }
        else if(i.Op == ADD || i.Op == SUB || i.Op == ADDI){
                if(i.rd<0 || i.rd>=NUM_IN_GPR || i.rs<0 || i.rs>=NUM_IN_GPR){
                        cout<<"invalid register number."<<endl;
                        exit(0);
                }
        }
        else{
                if(i.rd<0 || i.rd>=NUM_FP_GPR || i.rt<0 || i.rt>=NUM_FP_GPR || i.rs<0 ||
i.rs>=NUM_FP_GPR){
                        cout<<"invalid register number."<<endl;
                        exit(0);
                }
        }
}

/**
 *
 * FUNCTION:      print_ins()
 * DESCRIPTION:   print the instructions in the queue "inst".
 */
void TOMASULO::print_ins(ostream & out){
  // Print out original file:
  // Let's open the input file first:
  ifstream infile;
```

```
  infile.open(input_filename.c_str(),ios::in);
  if (infile.fail()) {
    cerr << "Couldn't open file: " << input_filename << endl;
    exit(-1);
  }
  char buf[MAX_BUFFER];
  while (infile.getline(buf, MAX_BUFFER)) {
    out << buf << endl;
  }
  infile.close();


  int num=inst.size();
  for(int i=0; i<num; i++){

out<<inst[i].Op<<"\t"<<inst[i].label<<"\t"<<inst[i].rd<<"\t"<<inst[i].rs<<"\t"<<inst[i].rt
<<endl;
  }
}


/**
 *
 * FUNCTION:      init_in_gpr()
 * DESCRIPTION:   initialize the integer registers.
 */
int TOMASULO::init_in_gpr(){//32 integers
        //0~7, 8~15, 16~23, 24~31
        int a[NUM_IN_GPR]={0,24,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0};
        for(int i=0; i<NUM_IN_GPR; i++){
                In_Gpr[i]=a[i];
        }
        return 0;
}

/**
 *
 * FUNCTION:      init_fp_gpr()
 * DESCRIPTION:   initialize the general-purpose floating-point registers.
 */
int TOMASULO::init_fp_gpr(){//32 fps
        float a[NUM_FP_GPR]={0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
```

```
	0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
	0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
	for(int i=0; i<NUM_FP_GPR; i++){
		Fp_Gpr[i]=a[i];
	}
	return 0;
}

/**
 *
 * FUNCTION:	init_mem()
 * DESCRIPTION:	initialize memory.
 */
int TOMASULO::init_mem(){
	float a[SIZE_MEM]={0, 0, 0, 0, 0, 0, 0, 0,
	5, 0, 0, 0, 0, 0, 0, 0.0,
	5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
	5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
	for(int i=0; i<SIZE_MEM; i++){
		Mem[i]=a[i];
	}
	return 0;
}

/**
 *
 * FUNCTION:	Issue_book()
 * DESCRIPTION:	bookkeeping for issue.
 */
void TOMASULO::Issue_book(int ty,int fu,int in){
	Res_Stas[ty][fu][in].Exe=cycle+1;//for next stage
	Res_Stas[ty][fu][in].curr_ins=Intr_reg;
	Intr_reg->issue=cycle;//issue time
	Res_Stas[ty][fu][in].Op=Intr_reg->Op;
	Res_Stas[ty][fu][in].Busy=true;
}


/**
 *
 * FUNCTION:	Issue_arith()
 * DESCRIPTION:	bookkeeping for FP operation issue.
 */
void TOMASULO::Issue_arith(FUs ty,int fu,int in){
	Res_Stas[ty][fu][in].des=Intr_reg->rd;
	if(Register_Stas[Intr_reg->rs].type != READY){//Qj wait
```

```
                Res_Stas[ty][fu][in].Qj.type=Register_Stas[Intr_reg->rs].type;
                Res_Stas[ty][fu][in].Qj.fu_in=Register_Stas[Intr_reg->rs].fu_in;
                Res_Stas[ty][fu][in].Qj.index=Register_Stas[Intr_reg->rs].index;
        }
        else{//Qj ready
                Res_Stas[ty][fu][in].Qj.type=READY;
                Res_Stas[ty][fu][in].Vj=Fp_Gpr[Intr_reg->rs];
        }
        if(Register_Stas[Intr_reg->rt].type != READY){//Qk wait
                Res_Stas[ty][fu][in].Qk.type=Register_Stas[Intr_reg->rt].type;
                Res_Stas[ty][fu][in].Qk.fu_in=Register_Stas[Intr_reg->rt].fu_in;
                Res_Stas[ty][fu][in].Qk.index=Register_Stas[Intr_reg->rt].index;
        }
        else{//Qk ready
                Res_Stas[ty][fu][in].Qk.type=READY;
                Res_Stas[ty][fu][in].Vk=Fp_Gpr[Intr_reg->rt];
        }
        Register_Stas[Intr_reg->rd].type=ty;//register status update
        Register_Stas[Intr_reg->rd].fu_in=fu;
        Register_Stas[Intr_reg->rd].index=in;

        if(Exe_Cycs[ty][fu] > 0){//for next execution
                Res_Stas[ty][fu][in].curr_sta=NOT_EX;
        }
        else{//possible exe cycles=0
                Res_Stas[ty][fu][in].curr_sta=COMPLETED_EX;
        }
}

bool TOMASULO::load_issue(){
        int i,j;
        for(i=0; i<Num_FU[LOAD]; i++){
                for(j=0; j<Num_Res_Stas[LOAD][i]; j++){
                        if(! Res_Stas[LOAD][i][j].Busy){
                                //Issue_Load
                                Issue_book(LOAD,i,j);
                                Res_Stas[LOAD][i][j].curr_sta=CAL_AD;

                                Res_Stas[LOAD][i][j].A=In_Gpr[Intr_reg->rs]+Intr_reg-
>rt;
                                Res_Stas[LOAD][i][j].des=Intr_reg->rd;
                                Res_Stas[LOAD][i][j].Qj.type=READY;
                                Res_Stas[LOAD][i][j].Qk.type=READY;
                                Register_Stas[Intr_reg->rd].type=LOAD;
                                Register_Stas[Intr_reg->rd].fu_in=i;
                                Register_Stas[Intr_reg->rd].index=j;
```

```
                                ++PC;
                                return true;
                        }
                }
        }
        return false;
}

bool TOMASULO::store_issue(){
        int i,j;
        for(i=0; i<Num_FU[STORE]; i++){
                for(j=0; j<Num_Res_Stas[STORE][i]; j++){
                        if(! Res_Stas[STORE][i][j].Busy){
                                //Issue_store
                                Issue_book(STORE,i,j);
                                Res_Stas[STORE][i][j].curr_sta=CAL_AD;

                                Res_Stas[STORE][i][j].A=In_Gpr[Intr_reg->rs]+Intr_reg-
>rt;
                                Res_Stas[STORE][i][j].Qj.type=READY;
                                Res_Stas[STORE][i][j].Vj=In_Gpr[Intr_reg->rs];
                                if(Register_Stas[Intr_reg->rd].type != READY){//Vk
holding the value to be stored

        Res_Stas[STORE][i][j].Qk.type=Register_Stas[Intr_reg->rd].type;

        Res_Stas[STORE][i][j].Qk.fu_in=Register_Stas[Intr_reg->rd].fu_in;

        Res_Stas[STORE][i][j].Qk.index=Register_Stas[Intr_reg->rd].index;
                                }
                                else{
                                        Res_Stas[STORE][i][j].Qk.type=READY;
                                        Res_Stas[STORE][i][j].Vk=Fp_Gpr[Intr_reg->rt];
                                }
                                ++PC;
                                return true;
                        }
                }
        }
        return false;
}

bool TOMASULO::fp_arith_issue(FUs ty){
        int i,j;
        for(i=0; i<Num_FU[ty]; i++){
                for(j=0; j<Num_Res_Stas[ty][i];j++){
```

```
                        if(! Res_Stas[ty][i][j].Busy){
                                //Issue_adder
                                Issue_book(ty,i,j);
                                Issue_arith(ty,i,j);
                                ++PC;
                                return true;
                        }
                }
        }
        return false;
}

/**
 *
 * FUNCTION:      check_Issue()
 * DESCRIPTION:   Return ture if there is an instruction waiting for issuing.
 *                                Issue an instruction when it is possible.
 */
bool TOMASULO::check_Issue(){
        bool Iss=false;//have issed sth.
        if(EMPTY == PC)
                return false;//nothing for issue;

        switch(Intr_reg->Op){
        case LD:
                //find a free resevation station
                Iss=load_issue();

                break;
        case SD:
                //find a free reservation station
                Iss=store_issue();
                break;
        case ADD_D:
        case SUB_D:
                Iss=fp_arith_issue(ADDER);
                break;
        case MULT_D:
                Iss=fp_arith_issue(MULTIPLIER);
                break;
        case DIV_D:
                Iss=fp_arith_issue(DIVIDER);
                break;
        case BEQ://nothing
        case BNE:
                Intr_reg->issue=cycle;
```

```
                Iss=true;
                break;
        case ADD:
        case ADDI:
        case SUB:
                ++PC;
                Intr_reg->issue=cycle;
                Iss=true;
                break;
        default:
                break;
        }
        if(Iss){//issue a instruction then get another one.
                if(!Instr_Fetch(PC))
                        PC=EMPTY;
        }
        return true;
}

/*cal_address for load and store*/
bool TOMASULO::cal_address(int ty,int fu,int in){
        if(Res_Stas[ty][fu][in].Exe > cycle){
                return false;
        }
        Res_Stas[ty][fu][in].Exe=cycle+1;//for next stage
        int addr =Res_Stas[ty][fu][in].A;
        if ((addr % WORDSIZE) != 0) {
                cerr << "ERROR:  Byte Offset for ADDRESS " << addr << " is not on "
<< WORDSIZE << endl;
                exit(-1);
        } else {//from byte offset to word offset
                addr = addr / WORDSIZE;
        }

        if(addr<0 || addr>=SIZE_MEM){//memory checking
                        cout<<"invalid memory access."<<endl;
                        exit(0);
                }
        else{
                Res_Stas[ty][fu][in].A=addr;
        }
        struct Res_Fu_Index temp;
        temp.in_fu=fu;
        temp.in_res=in;
        if(Exe_Cycs[ty][fu] > 0){
                Res_Stas[ty][fu][in].curr_sta=FIN_AD;
```

```
                if(ty == LOAD)
                        Load_q.push_back(temp);//put the load station to load queue
                if(ty == STORE)
                        Store_q.push_back(temp);//put the store station to load queue
        }
        else{//possible exe cycles=0
                Res_Stas[ty][fu][in].curr_sta=COMPLETED_EX;
        }
        return true;
}

/* memory access for load and store*/
bool TOMASULO::memory_Access(){
        int f,r;
        int fs,rs;
        if(mem_busy)
                return false;
        if(Load_q.empty() && Store_q.empty())
                return false;

        else if(Load_q.empty() && (!Store_q.empty())){
                f=(Store_q[0]).in_fu;
                r=(Store_q[0]).in_res;
                if(Res_Stas[STORE][f][r].Qk.type == READY){//check whether the
value is ready.
                        Res_Stas[STORE][f][r].curr_sta=NOT_EX;
                        Store_q.pop_front();
                }
        }
        else if((!Load_q.empty()) && Store_q.empty()){
                f=(Load_q[0]).in_fu;
                r=(Load_q[0]).in_res;//load always ready
                Res_Stas[LOAD][f][r].curr_sta=NOT_EX;
                Load_q.pop_front();
        }
        else{//both of them have sth waited.
                f=(Load_q[0]).in_fu;
                r=(Load_q[0]).in_res;
                fs=(Store_q[0]).in_fu;
                rs=(Store_q[0]).in_res;
                if(Res_Stas[LOAD][f][r].A == Res_Stas[STORE][fs][rs].A){//same
address
                        if((Res_Stas[LOAD][f][r].curr_ins->issue) <
(Res_Stas[STORE][fs][rs].curr_ins->issue)){//check the program order
                                Res_Stas[LOAD][f][r].curr_sta=NOT_EX;//load first
                                Load_q.pop_front();
```

```
                        }
                        else{//store first
                                if(Res_Stas[STORE][fs][rs].Qk.type == READY){//check
whether the value is ready.
                                        Res_Stas[STORE][fs][rs].curr_sta=NOT_EX;
                                        Store_q.pop_front();
                                }
                        }
                }
                else{//Not same address, Maybe load first will help others.
                        Res_Stas[LOAD][f][r].curr_sta=NOT_EX;
                        Load_q.pop_front();
                }
        }
        return true;
}

/**
 *
 * FUNCTION:      Instr_Fetch()
 * DESCRIPTION:   Fetch instruction from the queue "inst".
 *                              If no instruction can be got, return false and let
PC=EMPTY.
 */
bool TOMASULO::Instr_Fetch(int p){
        struct Instruction_Sta ins_temp;
        if(p<0 || p>=inst.size()){
                PC=EMPTY;
                return false;
        }
        ins_temp.Op=inst[p].Op;//get the instruction
        ins_temp.rd=inst[p].rd;
        ins_temp.rs=inst[p].rs;
        ins_temp.rt=inst[p].rt;
        ins_temp.in_fetch=cycle;
        ins_temp.exe_compl=EMPTY;
        ins_temp.exe_start=EMPTY;
        ins_temp.issue=EMPTY;
        ins_temp.wri_back=EMPTY;

        switch(ins_temp.Op){
        case BEQ:
                if(In_Gpr[ins_temp.rs] == In_Gpr[ins_temp.rt]){
                        PC=PC+ins_temp.rd;
                }
                else
```

```
                        PC++;
                break;
        case BNE:
                if(In_Gpr[ins_temp.rs] != In_Gpr[ins_temp.rt]){
                        PC=PC+ins_temp.rd;
                }
                else
                        PC++;
                break;
        case ADD:
                In_Gpr[ins_temp.rd]=In_Gpr[ins_temp.rs]+In_Gpr[ins_temp.rt];//need to
check for 0~31.
                break;
        case ADDI:
                In_Gpr[ins_temp.rd]=In_Gpr[ins_temp.rs]+ins_temp.rt;
                break;
        case SUB:
                In_Gpr[ins_temp.rd]=In_Gpr[ins_temp.rs]-In_Gpr[ins_temp.rt];
                break;

        case LD:
        case SD:
        case ADD_D:
        case SUB_D:
        case MULT_D:
        case DIV_D:
                break;
        default:
                break;
        }

        //put it into the instruction status queue.
        Intrs_Stas.push_back(ins_temp);
        //let it be the current instruction waiting for issue
        Intr_reg=&(Intrs_Stas.back());

        return true;
}

/**
 *
 * FUNCTION:      check_start_exe()
 * DESCRIPTION:   check whether the reservation station can start execution.
 */
bool TOMASULO::check_start_exe(int ty,int fu){
        int in;
```

```
	if(ty == DIVIDER){//not pipelined
		if(div_busy[fu])
			return false;
	}
	/*for every fu, only one can start at each cycle, for load and store, we use memory
access*/
	for(in=0; in<Num_Res_Stas[ty][fu]; in++){
	  if((NOT_EX == Res_Stas[ty][fu][in].curr_sta) &&(Res_Stas[ty][fu][in].Exe <=
cycle) && (Res_Stas[ty][fu][in].Qj.type== READY) &&
(Res_Stas[ty][fu][in].Qk.type== READY)){
		Res_Stas[ty][fu][in].curr_ins->exe_start=cycle;
		if(Exe_Cycs[ty][fu] > 1){
			Res_Stas[ty][fu][in].curr_sta=STARTED_EX;
			Res_Stas[ty][fu][in].Exe=cycle+Exe_Cycs[ty][fu]-1;
			if(ty==LOAD || ty== STORE)//memeory port release
				mem_busy=true;
			if(ty==DIVIDER)//divider release
				div_busy[fu]=true;
		}
		else{//maybe the execution only need 1 cycle
			Res_Stas[ty][fu][in].curr_sta=COMPLETED_EX;
			Res_Stas[ty][fu][in].Exe=cycle+1;
			Res_Stas[ty][fu][in].curr_ins->exe_compl=cycle;
			if(ty==LOAD || ty== STORE)//memeory port release
				mem_busy=false;
			if(ty==DIVIDER)//divider release
				div_busy[fu]=false;
		}
		return true;
	  }
	}
	return false;
}

/**
 *
 * FUNCTION:      check_compl_exe()
 * DESCRIPTION:   check whether the reservation station completes execution.
 */
bool TOMASULO::check_compl_exe(int ty,int fu,int in){
	if(Res_Stas[ty][fu][in].Exe == cycle){
		Res_Stas[ty][fu][in].curr_sta=COMPLETED_EX;
		Res_Stas[ty][fu][in].Exe=cycle+1;
		Res_Stas[ty][fu][in].curr_ins->exe_compl=cycle;
		if(ty==LOAD || ty== STORE)//memeory port release
				mem_busy=false;
```

```
                        if(ty==DIVIDER)//divider release
                                div_busy[fu]=false;
                return true;
        }
        return false;
}

/**
 *
 * FUNCTION:      write_result()
 * DESCRIPTION:   write result to register, memory and CDB.
 */
bool TOMASULO::write_result(int ty,int fu,int in){
        if(Res_Stas[ty][fu][in].Exe != cycle){
                return false;
        }
        if(ty == STORE){
                Mem[Res_Stas[ty][fu][in].A]=Res_Stas[ty][fu][in].Vk;
        }
        else{
                float result=0.0;
                switch(Res_Stas[ty][fu][in].Op){
                case LD:
                        result=Mem[Res_Stas[ty][fu][in].A];
                        break;
                case ADD_D:
                        result=Res_Stas[ty][fu][in].Vj+Res_Stas[ty][fu][in].Vk;
                        break;
                case SUB_D:
                        result=Res_Stas[ty][fu][in].Vj-Res_Stas[ty][fu][in].Vk;
                        break;
                case MULT_D:
                        result=Res_Stas[ty][fu][in].Vj*Res_Stas[ty][fu][in].Vk;
                        break;
                case DIV_D:
                        result=Res_Stas[ty][fu][in].Vj/Res_Stas[ty][fu][in].Vk;
                        break;
                default:
                        cout<<"Something wrong at write_result"<<endl;
                        break;
                }
                int q_des=Res_Stas[ty][fu][in].des;
                if((q_des != EMPTY) && (Register_Stas[q_des].type == ty) &&
(Register_Stas[q_des].fu_in == fu)&& (Register_Stas[q_des].index == in)){
                        Fp_Gpr[q_des]=result;
                        Register_Stas[q_des].type=READY;
```

```
                        Register_Stas[q_des].fu_in=EMPTY;
                        Register_Stas[q_des].index=EMPTY;   // RJS: needed to reset this,
too
                }
                for(int i=0; i<NUM_FUS; i++){
                        for(int j=0; j<Num_FU[i]; j++){
                                for(int k=0; k<Num_Res_Stas[i][j]; k++){
                                  if((Res_Stas[i][j][k].Busy) && (Res_Stas[i][j][k].Qj.type
== ty) && (Res_Stas[i][j][k].Qj.fu_in == fu)&& (Res_Stas[i][j][k].Qj.index == in)){
                                        Res_Stas[i][j][k].Vj=result;
                                        Res_Stas[i][j][k].Qj.type=READY;
                                        Res_Stas[i][j][k].Qj.fu_in=EMPTY;
                                        Res_Stas[i][j][k].Qj.index=EMPTY;  // RJS: needed
to reset this, too
                                        //Res_Stas[i][j][k].Exe=cycle+1;//maybe the res sta
can start executing at next cycle.
                                  }
                                  if((Res_Stas[i][j][k].Busy) && (Res_Stas[i][j][k].Qk.type
== ty) && (Res_Stas[i][j][k].Qk.fu_in == fu)&& (Res_Stas[i][j][k].Qk.index == in)){
                                        Res_Stas[i][j][k].Vk=result;
                                        Res_Stas[i][j][k].Qk.type=READY;
                                        Res_Stas[i][j][k].Qk.fu_in=EMPTY;
                                        Res_Stas[i][j][k].Qk.index=EMPTY;  // RJS:
needed to reset this, too
                                        //Res_Stas[i][j][k].Exe=cycle+1;//maybe the res sta
can start executing at next cycle.
                                  }
                                }
                        }
                }
        }
        Res_Stas[ty][fu][in].curr_ins->wri_back=cycle;
        Res_Stas[ty][fu][in].Busy=false;
        return true;
}


/**
 *
 * FUNCTION:      simu_run()
 * DESCRIPTION:   simulate the algorithm and run on input assembly instructions.
 */
int TOMASULO::simu_run(){

        int i,j,k;
```

```
	mem_busy=false;// true means there is a ins using the memory port
	div_busy=new bool[Num_FU[DIVIDER]];
	for(i=0;i<Num_FU[DIVIDER]; i++)
		div_busy[i]=false;
	cycle=0;//cycle =1, fetch the first instruction.
	PC=0;
	Instr_Fetch(PC);
	print_stat();
	//cycle add 1
	cycle++;

	//more instructions
	bool more_instru=true;
	//still have a busy resvertion station
	bool not_finish=true;

	while((more_instru || not_finish)){//Loop every cycle. &&cycle<1200
	  //;print_stat();
	  //;print_stat_local();
		more_instru=false;
		not_finish=false;
		if(check_Issue()){
			more_instru=true;
		}
		if(memory_Access()){
			more_instru=true;
		}
		/*else
			cout<<"no issue."<<endl;*/
		for(i=0; i<NUM_FUS; i++){
			for(j=0; j<Num_FU[i]; j++){
				check_start_exe(i,j);// can start a ins for each fu.
			  for(k=0; k < Num_Res_Stas[i][j]; k++){
				if(!Res_Stas[i][j][k].Busy)//free, donot need to do anything
					continue;
				not_finish=true;//some is not free.

				//use Exe in a res-station to make sure all of these can only
be started at next cycle.
				if(CAL_AD == Res_Stas[i][j][k].curr_sta){//calculate
address?
					cal_address(i,j,k);
					cout<<"Now the cycle is "<<cycle<<endl;
				}
				else if(STARTED_EX ==
Res_Stas[i][j][k].curr_sta){//have completed?
```

```
                                        check_compl_exe(i,j,k);
                                }
                                else if(COMPLETED_EX ==
Res_Stas[i][j][k].curr_sta){//write result?
                                        write_result(i,j,k);
                                }
                                else if(FIN_AD !=Res_Stas[i][j][k].curr_sta &&
NOT_EX!=Res_Stas[i][j][k].curr_sta){
                                        cout<<"There is something wrong at
simu_run."<<endl;
                                }
                        }
                    }
                }
                print_stat();
                cycle++;
        }
        return 0;
}

/*Display the reservation station status, instruction status,
        and the register result status on every cycle.*/
void TOMASULO::print_stat_local(){
        int num=Intrs_Stas.size();
        for(int i=0; i<num; i++){

        cout<<Intrs_Stas[i].Op<<"\t"<<Intrs_Stas[i].rd<<"\t"<<Intrs_Stas[i].rs<<"\t"<<In
trs_Stas[i].rt<<"\t";
                //cout<<Intrs_Stas[i].in_fetch<<"\t";
                cout<<Intrs_Stas[i].issue<<"\t";
                cout<<Intrs_Stas[i].exe_start<<"\t";
                cout<<Intrs_Stas[i].exe_compl<<"\t";
                cout<<Intrs_Stas[i].wri_back<<endl;
        }
        for(int k=0; k<NUM_FUS; k++){
                        for(int j=0; j<Num_FU[k]; j++){
                                for(int n=0; n<Num_Res_Stas[k][j]; n++){
                                cout<<k<<" "<<j<<" "<<n<<" ";
                                cout<<Res_Stas[k][j][n].Qj.type<<"
"<<Res_Stas[k][j][n].Qk.type<<" ";
                                cout<<Res_Stas[k][j][n].Vj<<"
"<<Res_Stas[k][j][n].Vk<<endl;
                                }
                        }
        }
}
```

```
//Display the content of memory.
void TOMASULO::print_mem(){
	cout<<"the following are contents of some memory."<<endl;
	for(int i=0; i<10; i++){
		cout<<Mem[i]<<" ";
	}
	cout<<endl;
}

//Display the content of integer registers.
void TOMASULO::print_in_gpr(){
}

//Display the content of floating point registers.
void TOMASULO::print_fp_gpr(){
	cout<<"the following are contents of some fp registers."<<endl;
	for(int i=0; i<10; i++){
		cout<<Fp_Gpr[i]<<" ";
	}
	cout<<endl;
}

/*****************************************************************
 *  Filename: main.cpp
 *	Main test driver
 *	Syntax:  tomasulo.exe <input.asm> <init.txt>
 *	Authors:
 *			Kan Liu
 *			Piyush Ranjan Satapathy
 *			Ricky J. Sethi
 *	Project for CS203A The University of California,Riverside
 *	Date: 11/15/2004
 *****************************************************************/

#include "TOMASULO.h"
#include <string>
#include <iostream>
using namespace std;


//=================================================================
======
// Some Constant and Global Variables
//=================================================================
======
```

```
string inputfile        = "input.txt";
string initfile         = "init.txt";



//=========================================================================
======
// Main:
//=========================================================================
======
int main(int argc, char ** argv) {

  if(argc == 2) {
    inputfile = argv[1];
  } else if(argc == 3) {
    inputfile = argv[1];
    initfile = argv[2];
  }

  cout << "Using the assembly filename:  " << inputfile
       << " and initialization file: " << initfile << endl;

  TOMASULO * simulator = new TOMASULO();

  // Setup:
  simulator->parse_input_assembly(inputfile.c_str());
  simulator->init_from_file(initfile.c_str());
  simulator->print_ins(cout);

  // Run:
  simulator->simu_run();
  simulator->print_final_stats();

  // Debugging:
  //simulator->print_mem();
  //simulator->print_fp_gpr();
  //simulator->print_stat_local();

  return 0;
}

#########################################################################
#                                                                       #
# Assignment:  Tomasulo simulator for CS203a                            #
#                                                                       #
#########################################################################
#
# Makefile for tomasulo
```

```
#
######################################################################
#
# Authors:
#       Kan Liu
#       Piyush Ranjan Satapathy
#       Ricky J. Sethi
#
######################################################################


PROG                = tomasulo
OUTPUT_PREFIX             = /tmp/
PROG_OUTPUT         = $(OUTPUT_PREFIX)tomasulo.exe

CC                        = g++
OFLAGS                    = -o
CFLAGS                    = -c

#
# Primary make target
#
all: $(PROG)

tomasulo: main.o TOMASULO.o
      $(CC) $(OFLAGS) $(PROG_OUTPUT) main.o TOMASULO.o

main.o : main.cpp
      $(CC) $(CFLAGS) main.cpp

TOMASULO.o : TOMASULO.cpp
      $(CC) $(CFLAGS) TOMASULO.cpp

clean:
      rm ./*.o ./*~

tarball:
      tar czf tomasulo.tgz Makefile TOMASULO.h TOMASULO.cpp main.cpp
init*.txt *.asm
```

```
####################################
# Testcase 1
####################################
div.d F4, F2, F1
add.d F6, F4, F2
add.d F6, F2, F1
Input
------
# Integer:
I,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
# Floating Point:
F,0.0, 20.0, 10.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
# Main Memory Cache:
```

```
C,0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0.0,0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,5, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0


# Reservation Stations (ADDER,MULTIPLIER,DIVIDER,LOAD,STORE)
#S,1,3,1,3,3
# Execution Cycles (ADDER,MULTIPLIER,DIVIDER,LOAD,STORE)
#E,4,4,40,2,2


# Adders: Res Stations, Ex Cycles
A,1,4
A,1,4

# Multipliers: Res Stations, Ex Cycles
M,3,4

# Dividers: Res Stations, Ex Cycles
D,1,40

# Loaders: Res Stations, Ex Cycles
L,3,2

# Storers: Res Stations, Ex Cycles
S,3,2
######################################
#Testcase 2
######################################
add.d F4, F2, F1
sd F4, 0(R1)
addi R1, R1, 0
ld F2, 0(R1)
Input
------
# Integer:
I,0,20,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
# Floating Point:
F,0.0, 5.4, 4.6, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
# Main Memory Cache:
C,0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0.0,0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,5, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0


# Reservation Stations (ADDER,MULTIPLIER,DIVIDER,LOAD,STORE)
#S,3,3,1,3,3
```

Execution Cycles (ADDER,MULTIPLIER,DIVIDER,LOAD,STORE)
#E,3,4,40,1,1

Adders: Res Stations, Ex Cycles
A,3,3

Multipliers: Res Stations, Ex Cycles
M,3,4

Dividers: Res Stations, Ex Cycles
D,1,40

Loaders: Res Stations, Ex Cycles
L,3,1,5

Storers: Res Stations, Ex Cycles
S,3,1,5

```
########################################
# Testcase 3
########################################
      ld R3,0(R0)
          ld F2,28(R0)
100:     ld F0,0(R1)
       mult.d F4,F0,F2
       sd F4,0(R1)
       addi R1,R1,-8
       bne R1,R3,-16
Input
------
# Integer:
I,0,24,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
# Floating Point:
F,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
# Main Memory Cache:
C,0, 0, 0, 0, 0, 0, 0, 7,5, 0, 0, 0, 0, 0, 0, 0.0,5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,5, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0


# Reservation Stations (ADDER,MULTIPLIER,DIVIDER,LOAD,STORE)
#S,3,3,3,3,3
# Execution Cycles (ADDER,MULTIPLIER,DIVIDER,LOAD,STORE)
#E,2,4,10,2,2
```

```
# Adders: Res Stations, Ex Cycles
A,3,2
A,3,2

# Multipliers: Res Stations, Ex Cycles
M,3,4
M,3,4
M,3,4

# Dividers: Res Stations, Ex Cycles
D,3,10

# Loaders: Res Stations, Ex Cycles
L,3,2

# Storers: Res Stations, Ex Cycles
S,3,2
#####################################
# Testcase 4
#####################################
      ld F4,0(R2)
      ld F6,20(R2)
       ld F2,28(R3)
       mult.d F0,F2,F4
       sub.d F8,F6,F2
       div.d F10,F0,F6
       add.d F6,F8,F2
Input
------
# Integer:
I,0,0,20,28,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
# Floating Point:
F,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
# Main Memory Cache:
C,0, 0, 0, 0, 0, 11, 0, 0,0, 0, 13, 0, 0, 0, 14, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0


# Reservation Stations (ADDER,MULTIPLIER,DIVIDER,LOAD,STORE)
#S,3,3,3,3,3
# Execution Cycles (ADDER,MULTIPLIER,DIVIDER,LOAD,STORE)
#E,2,4,10,2,2


# Adders: Res Stations, Ex Cycles
```

A,3,2
A,3,2

Multipliers: Res Stations, Ex Cycles
M,3,4
M,3,4
M,3,4

Dividers: Res Stations, Ex Cycles
D,2,10

Loaders: Res Stations, Ex Cycles
L,2,2

Storers: Res Stations, Ex Cycles
S,2,2
##
Testcase 5
##
```
        add R1,R2,R3
        sub R1,R1,R3
        beq R1,R2,-4
        addi R3,R3,1
        beq R1,R3,-12
```
Input

Integer:
I,0,0,5,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
Floating Point:
F,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
Main Memory Cache:
C,0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0

Reservation Stations (ADDER,MULTIPLIER,DIVIDER,LOAD,STORE)
#S,2,2,2,3,3
Execution Cycles (ADDER,MULTIPLIER,DIVIDER,LOAD,STORE)
#E,3,4,10,2,2

Adders: Res Stations, Ex Cycles
A,3,2
A,3,2

```
# Multipliers: Res Stations, Ex Cycles
M,3,4
M,3,4
M,3,4

# Dividers: Res Stations, Ex Cycles
D,3,10

# Loaders: Res Stations, Ex Cycles
L,3,2

# Storers: Res Stations, Ex Cycles
S,3,2
###########################################
# Testcase 6
###########################################
Loop:   add R3,R2,R1
        sub R2,R2,R1
        bne R2,R1,-8
        ld F1,0(R1)
        ld F2,1(R2)
        mult.d F3,F1,F2
        add.d F4,F1,F2
        sd F3,3(R1)
        sd F4,4(R1)
Input
------
# Integer:
I,0,1,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
# Floating Point:
F,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
# Main Memory Cache:
C,0, 5, 6, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0.0,0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0

# Reservation Stations (ADDER,MULTIPLIER,DIVIDER,LOAD,STORE)
#S,1,1,1,2,2
# Execution Cycles (ADDER,MULTIPLIER,DIVIDER,LOAD,STORE)
#E,3,7,10,2,2

# Adders: Res Stations, Ex Cycles
A,3,2
A,3,2

# Multipliers: Res Stations, Ex Cycles
```

M,3,4
M,3,4
M,3,4

Dividers: Res Stations, Ex Cycles
D,3,10

Loaders: Res Stations, Ex Cycles
L,3,2

Storers: Res Stations, Ex Cycles
S,3,2