

COMP 4611 Programming Assignment:

MIPS Computer Simulator with Two Level Branch Prediction

Step I. Implement the MIPS Computer Simulator [50 marks]

1. Introduction

In this step, you are asked to implement a simple MIPS computer simulator using C++. The computer would read in a binary file which stores MIPS instructions, execute every instruction and finally get the result of the MIPS program. To make it simple, this simulator is supposed to support only the following MIPS instructions:

* R type: add, sub, and, or, sll, srl, slt

* I type: addi, lw, sw, beq, bne

* J type: j

2. Implementation

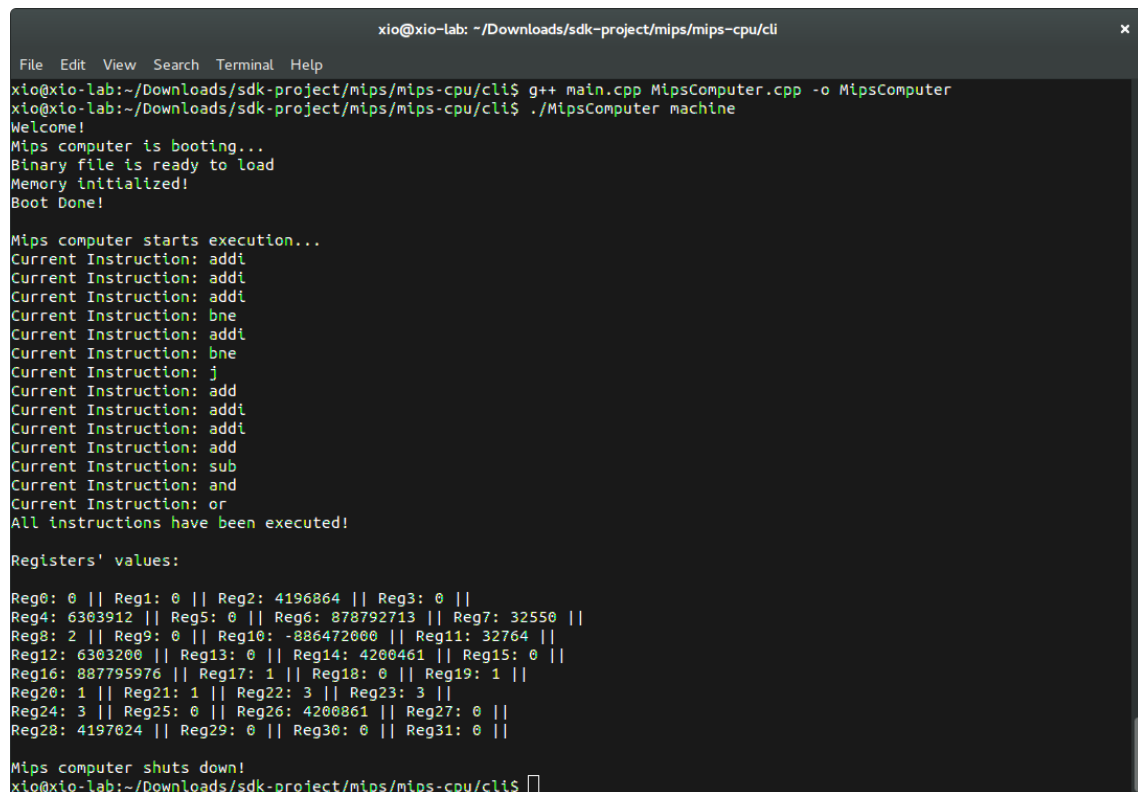
To make your assignment clearer, we provide you a programming template for this MIPS computer simulator. In this template, a class "MIPSCComputer" is declared as following:

```
class MIPSCComputer{
private:
    unsigned char Memory[MAXMEM];    //Memory, MAXMEM is a macro
    int Reg[32];                     //Registers
    unsigned int PC;                  //pointer PC
    GlobalBranchPredictor globalPredictor; //used in step II
    LocalBranchPredictor localPredictor;  //used in step II
public:
    MIPSCComputer();                 //constructor
    void boot(char* file);            //function to initialize the computer
                                      //and load MIPS instructions
    int run();                        //function to execute instructions
    void printRegisters();            //function to print registers' values;
    void printBranchPredictionResult(); //used in step II
    ~MIPSCComputer();                //destructor
};
```

Reg[0] is initialized to 0 in constructor. This step involves three files: main.cpp, MIPSCComputer.h, MIPSCComputer.cpp. A MIPSCComputer object is created in main(), and a binary instruction file name is passed to boot() for loading instructions into the computer's memory. After that, run() is called to run all the instructions, and this function is left for you to finish.

The run() function should work in this way: It executes instructions one by one; every time it fetches four bytes of data (one instruction) from memory, analyzes the MIPS instruction using bit operations, and executes the instruction; it stops execution when four bytes of zero are fetched from the memory. To execute an instruction, you should first decide its type (R, I or J), then simply use C++ operations to operate the registers, pointer PC and memory based on that instruction's functionality.

A binary file "machine1" is provided to you to test your codes. At the end of the binary file, there are four bytes of zero to indicate end of the MIPS program. You can call printRegisters() in main() to see the execution results. A running example is shown below:



```
xio@xio-lab: ~/Downloads/sdk-project/mips/mips-cpu/cli
File Edit View Search Terminal Help
xio@xio-lab:~/Downloads/sdk-project/mips/mips-cpu/cli$ g++ main.cpp MipsComputer.cpp -o MipsComputer
xio@xio-lab:~/Downloads/sdk-project/mips/mips-cpu/cli$ ./MipsComputer machine
Welcome!
Mips computer is booting...
Binary file is ready to load
Memory initialized!
Boot Done!

Mips computer starts execution...
Current Instruction: addi
Current Instruction: addi
Current Instruction: addi
Current Instruction: bne
Current Instruction: addi
Current Instruction: bne
Current Instruction: j
Current Instruction: add
Current Instruction: addi
Current Instruction: addi
Current Instruction: add
Current Instruction: sub
Current Instruction: and
Current Instruction: or
All instructions have been executed!

Registers' values:
Reg0: 0 || Reg1: 0 || Reg2: 4196864 || Reg3: 0 ||
Reg4: 6303912 || Reg5: 0 || Reg6: 878792713 || Reg7: 32550 ||
Reg8: 2 || Reg9: 0 || Reg10: -886472000 || Reg11: 32764 ||
Reg12: 6303200 || Reg13: 0 || Reg14: 4200461 || Reg15: 0 ||
Reg16: 887795976 || Reg17: 1 || Reg18: 0 || Reg19: 1 ||
Reg20: 1 || Reg21: 1 || Reg22: 3 || Reg23: 3 ||
Reg24: 3 || Reg25: 0 || Reg26: 4200861 || Reg27: 0 ||
Reg28: 4197024 || Reg29: 0 || Reg30: 0 || Reg31: 0 ||

Mips computer shuts down!
xio@xio-lab:~/Downloads/sdk-project/mips/mips-cpu/cli$
```

Figure 1. Running example

3. Bonus [extra 20 marks]

In the requirements above, MIPS operations are executed though C++ operations, and this is a naive simulation without considering the actual MIPS architecture. After you finish the task above, you can try this extra task by rewriting the run() function, simulating the five stages of MIPS pipeline. To make this less difficult, this simulated pipeline would not support parallelized execution of different stages, that is, next instruction will start only after the current instruction finishes all its five stages.

Under this requirement, you should follow the provided MIPS pipeline, adding essential units like adder, ALU and multiplexers, going through five stages to execute an instruction in the run()

function. To make the codes clear, for each unit, you should write an independent function to receive inputs and return outputs.

Submit an independent copy of the codes (create another folder) besides the normal version.

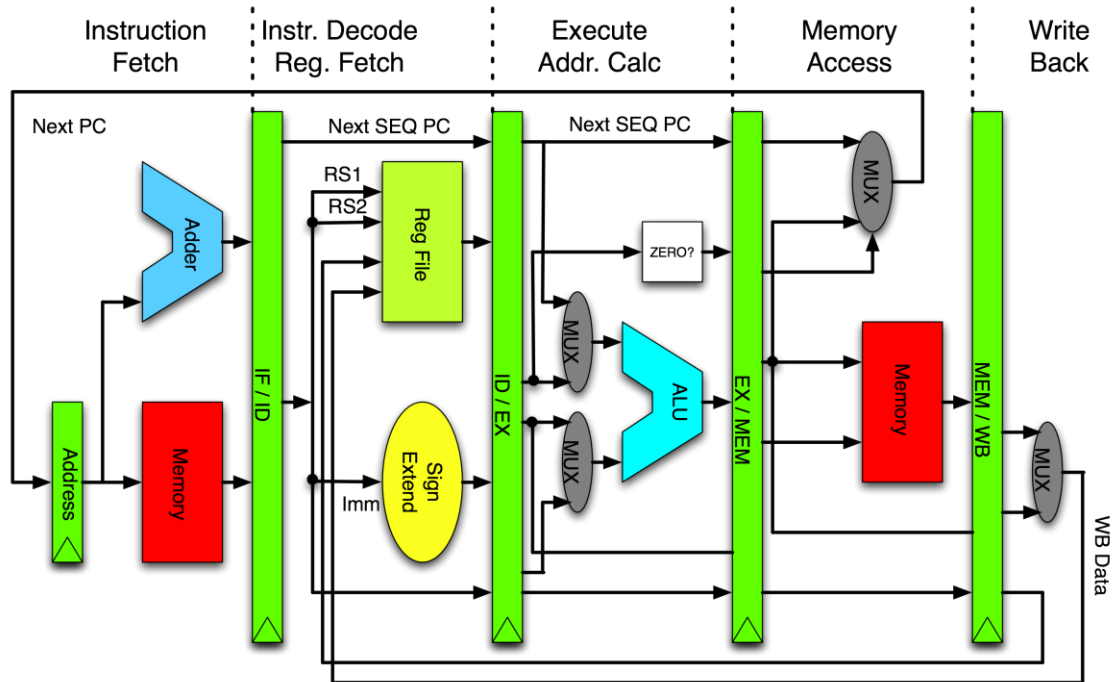


Figure 2. MIPS Pipeline

Step II. Implement Two Branch Predictors [50 marks]

1. Introduction

In this step, you are asked to implement two types of branch predictors that you learned about in class: global branch predictor and local branch predictor.

2. Requirements

Global Branch Predictor [25 marks]. The global predictor consists of the following components:

- A global history register that contains the outcome of the past 8 branches.
- A pattern history table that contains 256 entries.
- A 2-bit saturating counter for each entry in the pattern history table.

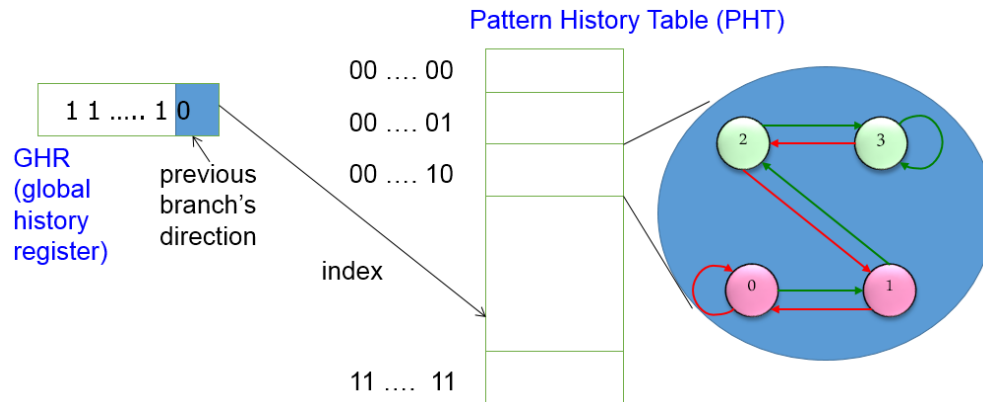


Figure 3. Global Branch Prediction

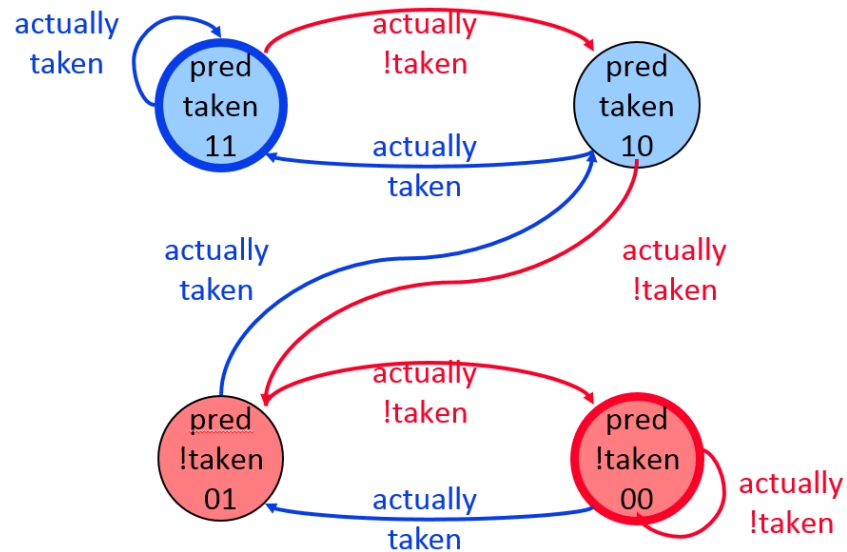


Figure 4. 2-bit Saturating Counter

The starting values for these components are all 0. To update the global history register and pattern history table entries, you should wait until the branch direction has been determined and update them based on the actual outcome of the branch.

Local Branch Predictor [25 marks]. The local predictor consists of the following components:

- 10 local history registers.
- An 8-bit branch history for each local history register based on the PC of the branch.
- A pattern history table containing 256 entries for each local history register, 10 tables in total.
- A 2-bit saturating counter for each entry in the pattern history tables.

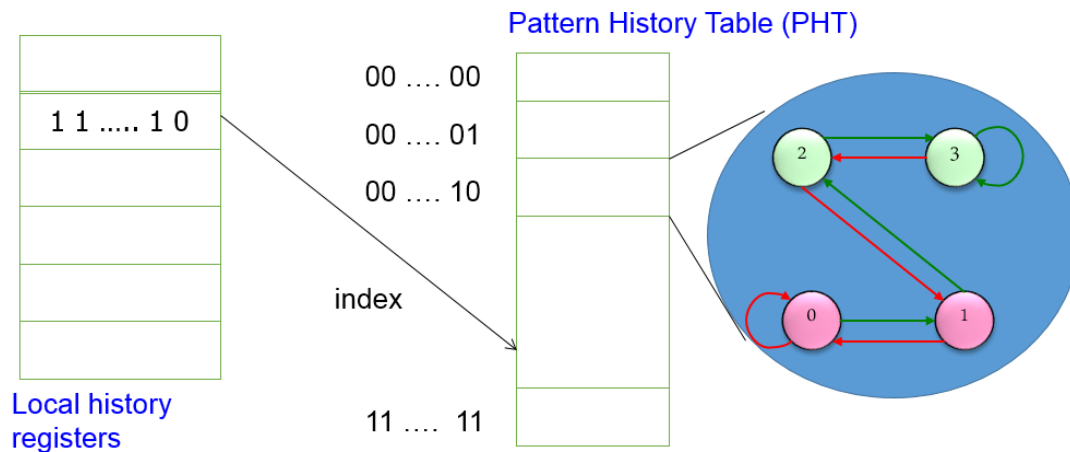


Figure 5. Local Branch Prediction

The starting values for these components are all 0. To update the local history registers and pattern history table entries, you should wait until the branch direction has been determined and update them based on the actual outcome of the branch.

3. Implementation

This step involves all 7 C++ source/header files, 1 data file "machine2". You need to modify only "GlobalBranchPredictor.cpp" and "LocalBranchPredictor.cpp" to implement the two predictors. The MIPS assemble codes for "machine2" is shown below:

```
INITIALIZE:
addi $s1, $zero, 100
addi $s2, $zero, 50
add $s3, $zero, $zero
add $t0, $zero, $zero
add $t1, $zero, $zero
add $t2, $zero, $zero
add $t3, $zero, $zero
add $t4, $zero, $zero
add $t5, $zero, $zero
add $t6, $zero, $zero
add $t7, $zero, $zero
LOOP:
beq $s3, $s2, 1
addi $t0, $t0, 1
ENDBRANCH0:
bne $s3, $s2, 1
addi $t1, $t1, 1
ENDBRANCH1:
blt $s3, $s2, 1
addi $t2, $t2, 1
ENDBRANCH2:
```

```
bgt $s3, $s2, 1
addi $t3, $t3, 1
ENDBRANCH3:
ble $s3, $s2, 1
addi $t4, $t4, 1
ENDBRANCH4:
bge $s3, $s2, 1
addi $t5, $t5, 1
ENDBRANCH5:
sll $s0, $s3, 30
srl $s0, $s0, 30
beq $s0, $zero, 1
addi $t6, $t6, 1
ENDBRANCH6:
bne $s0, $zero, 1
addi $t7, $t7, 1
ENDBRANCH7:
addi $s3, $s3, 1
blt $s3, $s1, -25
end:
```

Basically, the MIPS program contains a loop, which is executed for 100 times. Inside each loop there are some branch instructions, some of which may be pseudo instructions, but they are finally translated into sequence of normal MIPS instructions in “machine2”. In the MIPS simulator, predictors work when beq and bne instructions are met, but they just make branch prediction decisions and update themselves without actually influence the branch executions.

Grading Scheme

Your marks are mainly based on the running result. If the results are not correct, we will evaluate your codes, and you can get at most 60% of the full marks in this case.