# Project 4

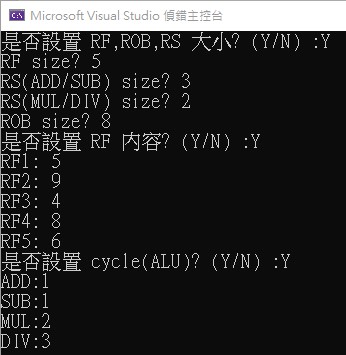
目的

模擬processor使用Tomasulo algorithm的行為，使用到RF、RAT(儲存ROB的位子)、RS(分為加減以及乘除)、RS Buffer(代表要執行的instruction，儲存RS的位子)

輸入/輸出

輸入：

1. input.txt，為input data
2. 相關的設定

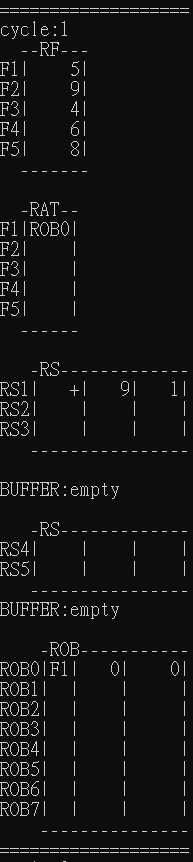


若沒有設定，則為預設。

輸出：

1. cycle
2. RF
3. RAT
4. RS and Buffer
5. ROB (REG, value, done)

只會印出有改變的cycle



架構介紹

class

class Text

{

public:

//型態，用於儲存未處理的instruction

using text\_type = vector<string>;

//型態，用於儲存處理後的instruction

using inst\_type = vector<string>;

//將str放進text中

void push(string str);

//得到一個instruction，從text\_after\_deal中

inst\_type get(int); //get a instruction

//得到一個instruction，從text\_after\_deal中，const型態

inst\_type get(int) const;

//得到text

const text\_type& getText();

//得到text\_after\_deal的大小

size\_t size(); //text size

//得到text\_after\_deal的大小，const 型態

size\_t size() const;

//清除所有內容

void clear();

private:

//將一個instruction做分割，放進text\_after\_deal

void deal(string&);

//儲存未處理的instruction

text\_type text;

//儲存處理後的instruction

vector<inst\_type> text\_after\_deal;

};

class Test

{

public:

struct rob;

//放置RS

struct rs

{

char opprand = ' ';

string val1;

string val2;

rob\* rob = nullptr;

bool isEmpty = true;

};

  //型態，放置ROB

struct rob

{

int REG = 0; //index in RF

int val = 0; //value

bool done = false;

};

//型態，RS Buffer的型態

struct rsbuffer

{

rs\* which\_rs;

int cycle = 0;

};

//型態，內容為RF中的值

using rf\_type = vector<int>;

//型態，內容儲存ROB中的位址

using rat\_type = vector<rob\*>;

//型態，儲存RS的資訊

using rs\_type = vector<rs>;

//型態，儲存RS位址及cycle(用於模擬ALU)

using rs\_buffer = rsbuffer;

//型態，儲存ROB資訊

using rob\_type = vector<rob>;

//constructor，對應text到input(指令碼)

Test(const Text&);

//設置RF, RS, ROB 大小

void setSize(int, int, int, int);

//設置ALU的執行cycle

void setCycle(int, int, int, int); //add, sub, mul, div

//設置RF的初始內容

void setRF();

//執行

void run();

private:

//執行1個cycle，若程式碼結束會回傳false，包括例外狀況

bool step(); //do 1 cycle

//印出所有資訊

void printAll();

//印出RF

void printRF();

//印出RAT

void printRAT();

//印出RS

void printRS();

//印出ROB

void printROB();

//const 指標，指向constructor傳入的參數(input)

const Text \*text;

//text的index，用於辨認issue到哪裡了

int index\_text = 0;

  //RF

rf\_type RF; //register file

//RAT

rat\_type RAT; //

//RS，ADD and SUB

rs\_type RS\_1;

//RS，MUL and DIV

rs\_type RS\_2;

//RS Buffer

vector<rs\_buffer> RS\_Buffer;

  //ROB

rob\_type ROB;

//rob 需要 commit 的位置

int rob\_commit;

//放置issue的位置

int rob\_issue;

//目前執行到的cycle

int cycle = 0;

//issue:

//若有改變任何值，則回傳true

//1. 分辨該instruction要放到哪個RS

//2. 若RS滿了，結束

//3. 若沒有滿，放置opprand

//4. 檢查RAT中val1要讀取的地方有沒有在ROB內，有的話打上ROB的標記

//5. 檢查val2，與4.一樣，但是有可能val2是數值，直接放入

//6. 在ROB[rob\_issue]內的REG放入instruction的rd，++rob\_issue

//7. 將放入的RS中打上ROB的標記

bool issue();

//dispatch:

//若有改變任何值，回傳true

//檢查所有RS(ADD/SUB及MUL/DIV)，若值已經準備好了，且沒有任何在RS

//Buffer中執行的instruction，放入buffer中

bool dispatch();

//execute:

//由於是模擬，所以每一次將有內容的RS Buffer 的cycle加1，回傳false

bool execute();

//broadcast:

//1. 每一次都檢查RS Buffer中的cycle是否到了執行結束的時間

//2. 若已經到了，把值寫進ROB，且done設為true

//3. 清空此instruction使用的RS

//4. 若有例外出現，丟出false到step中，會使step回傳false，程式結束

bool broadcast();

//write result:

//在這個project中，每個cycle會將所有已經做完的commit掉

//這個函式就是執行commit

//1. 迴圈檢查，直到還沒執行完的位置，或是沒有放任何issue過的地方

//2. 若可以commit，將val寫回RF以及任何有使用此ROB的RS中

//3. 若RAT中有使用此ROB的地方，刪除

//4. 刪除此位置(ROB)

bool writeResult();

  //得到 ALU 的cycle ，c 為 +-\*/

int getCycle(char c);

  // ALU cycle

int addcycle;

int subcycle;

int mulcycle;

int divcycle;

  //ALU行為及回傳值

int add(int, int);

int sub(int, int);

int mul(int, int);

int div(int, int);

};

執行過程

main function 會呼叫Test::run()，run()是一個無限迴圈(呼叫step())或是有發生例外狀況，無論如何，以step的回傳值判斷。

step()會以writeResult()->broadcast()->execute()->dispatch()->issue()的順序執行，以這種順序是因為如果按照順序的話，只要issue完基本上就可以馬上做dispatch，execute也可以馬上做，會無法有pipeline的行為，因此倒轉順序可以消除這種狀況。

exception的handle放在broadcast，因為broadcast每一次要檢查是否execute結束了才能進行broadcast，而執行結束才能知道有沒有發生例外。

在step()的最後會印出所有資訊(在有任何數值發生改變時)，判定有沒有值發生改變的方法，在於每個function都會回傳是否有值發生改變，只要其中一個有就印出。