Speculative Tomasulo simulator project report

Qingyang Li,qil77

Discussion peers: Shibo Xing, Aaron Wu

Design:

The simulator project is written in C++ and coding part is mainly consist with 3 files: *main.cpp*, *simulator.h* and *simulator.cpp*. The main function of the simulator is inside *simulator.h* and *.cpp* files.

main.cpp

Read instructions file and memory file, initialize the simulator object, pass the instr file and memory file to simulator, and execute the simulator. The config of the simulator is set through set_parameter function before starting the simulator. *read_memory* function and *read_instructions* tokenize the memory content and instructions and push to deque.

simulator.h and *simulator.cpp*

```
void initlize();                                    // start initialization
int init_mem();                                     // Initialize memory;
int init_reservationStation();                      // Initialize Reservation Sta;
int init_ROB();                                     // Initialize ROB
int init_register();                                // Initialize registers
void sim_start();                                   // start the simulator;
void set_parameter(int NF, int NW, int NR, int NB); // Set parameters
bool read_memory(const char *);                     // read memory content from file
bool read_instructions(const char *);               // read instruction from file
bool fetch();                                       // Fetch instructions from instruction list;
bool decode();                                      // Decode instructions from fetch queue
bool issue();                                       // issue instructions from decode queue to reservation stations
bool execute();
double getValue(ROB_status rob);
string register_rename(string reg, bool des); // perform register rename at decode stage and add renamed instruction int
void reset_address(string const &addr);       // push back address to freelist and set mapping table
```

The simulator are mainly operations by these four stages: fetch, decode, issue and execute. There are functions named as *read_memory*, *read_instructions, getValue*, *register_rename* and *reset_address* assist during these stages.

The details of these functions are commented in the source code.

fetch():

fetch stage, fetch instructions from instructions list that extract from the input file.

Return if program counter larger than the size of instruction.

Fetch NF number of instructions every cycle.

decode():

Decode stage, get instruction from fetch queue, do the register renaming and push back to decode queue. Also initialize the BTB

issue():

Issue stage, issue NW instructions from decode stage to *reservation_station* and *ROB*, if no available *reservation_station*, stall.

execute():

For instructions in issue status, if the operands have dependency with other instruction in ROB, wait for other ROB in commit or WB status. If ready, change the status to Execute add latency and calculate the value.

For instructions in execute status, if latency is larger than 0, continue to decrement the latency value. If LD/SD instruction, set memory to busy, or wait until the memory is not busy. If BNE instruction, check the BTB is taken or not, if taken, change the Program counter to the address that at the first instruction of that branch, if not taken, increment the program counter to next instruction after the branch, and flush out the instruction for branch is take out of the fetch queue, decode queue, ROB and reservation station. Else, set instruction to WB status, and increment the WB count, if equal to NB, stall other instructions. Add the value of instructions in WB to CDB

For instruction in WB status, set to commit status and wait for commitment.

For instruction in Commit status, if in the head of ROB, update the value to register or memory if FSD op, free the renamed register to the free list;

Sim_start():

Call the function in the order of *execute()*, *issue()*,*decode()*,*fetch().* Four stages are a cycle, stop the execution until ROB, fetch queue and decode queue are all empty.

The data structure in simulator:

```
deque<Instruction> instruction_list;         // store instruction read from inputfile
deque<Instruction> fetch_queue;              // store fetched insrtuctions
deque<Instruction> decode_queue;             // store decoded instructions
unordered_map<string, string> mapping_table; // key is the register before rename, value is after renamed
unordered_map<string, int> branch_address;   // store the address of branch instruction
unordered_map<string, Reservation_station_status> reservation_stations;
unordered_map<int, pair<int, int>> BTB;        // store the branch target address and its target address
deque<ROB_status> ROB;                         // store the ROB status
unordered_map<string, double> CDB;             // store the register value to CDB for forward or commit.
unordered_map<string, double> register_status; // register result status
deque<string> free_list;                       // Free list for avaliable physical registers
deque<string> free_free_list;                  // Free list for freeing next avaliable physical registers
unordered_map<int, double> memory_content;     // store memory values that read from inputfile
```

```
/*Implement 9 instructions*/
enum Instrs
{
    FLD,
    FSD,
    ADD,
    ADDI,
    FADD,
    FSUB,
    FMUL,
    FDIV,
    BNE
};
enum State
{
    Issued,
    Execute,
    Memory,
    WB,
    Commit
};
struct Instruction
{
    int address;
    Instrs Op;
    string rd;
    string rs;
    string rt;
    int imme;
};
```

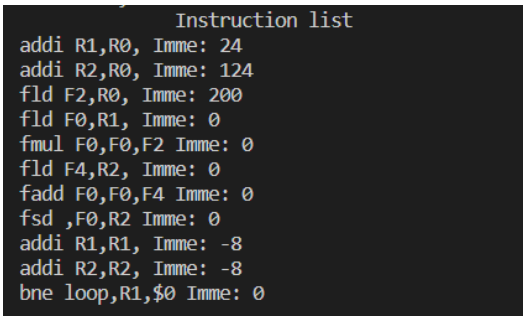
```
struct Reservation_station_status
{
    bool busy = false;
    Instrs Op;
    string Vj = "";
    string Vk = "";
    string Qj = "";
    string Qk = "";
    string dest;
    int a = -1;
};

struct ROB_status
{
    string unit;
    string name;
    bool busy = false;
    Instruction ins;
    State state = Issued;
    string dest;
    double value = __DBL_MIN__;
    int cycles = 0;
};
```

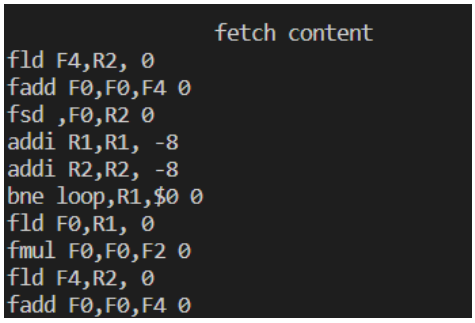
Output explain：

The program displays the following content results, some of display can be comment out in *display_data* function in simulator.cpp

Instruction list: instruction that extract from instruction files

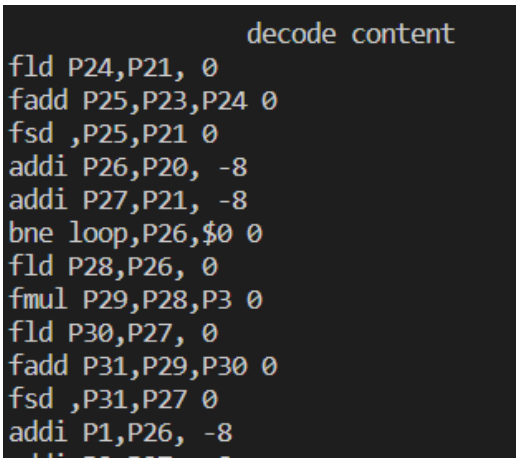
Fetch content: instruction that fetch from instruction list each cycle.



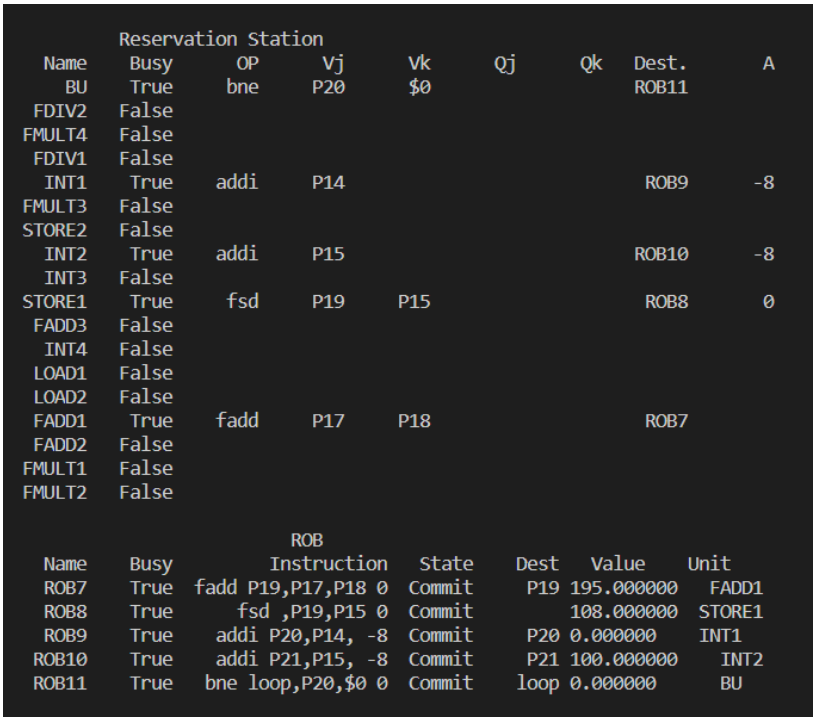


Decode content: renamed instruction in decode queue

Freelist :avaliable physical register for renaming purpose.



Reservation station and ROB



Reservation Station

Name	Busy	OP	Vj	Vk	Qj	Qk	Dest.	A
BU	True	bne	P20	$0			ROB11	
FDIV2	False							
FMULT4	False							
FDIV1	False							
INT1	True	addi	P14				ROB9	-8
FMULT3	False							
STORE2	False							
INT2	True	addi	P15				ROB10	-8
INT3	False							
STORE1	True	fsd	P19	P15			ROB8	0
FADD3	False							
INT4	False							
LOAD1	False							
LOAD2	False							
FADD1	True	fadd	P17	P18			ROB7	
FADD2	False							
FMULT1	False							
FMULT2	False							

ROB

Name	Busy	Instruction	State	Dest	Value	Unit
ROB7	True	fadd P19,P17,P18 0	Commit	P19	195.000000	FADD1
ROB8	True	fsd ,P19,P15 0	Commit		108.000000	STORE1
ROB9	True	addi P20,P14, -8	Commit	P20	0.000000	INT1
ROB10	True	addi P21,P15, -8	Commit	P21	100.000000	INT2
ROB11	True	bne loop,P20,$0 0	Commit	loop	0.000000	BU

CDB: value for forwarding and WB

Register: value for physical register.

Memory content: memory values.

CDB

ROB	value
P19,	195
P20,	0
P21,	100

Register

reg	value
F0(P16)	14
F4(P11)	60
F2(P3)	12
R2(P14)	8
R1(P13)	63
R0(P0)	0

Memory content

memory	value
200,	12
124,	128
116,	63
108,	27
100,	2
24,	10
16,	5
8,	14
0,	111

Comparative analysis:

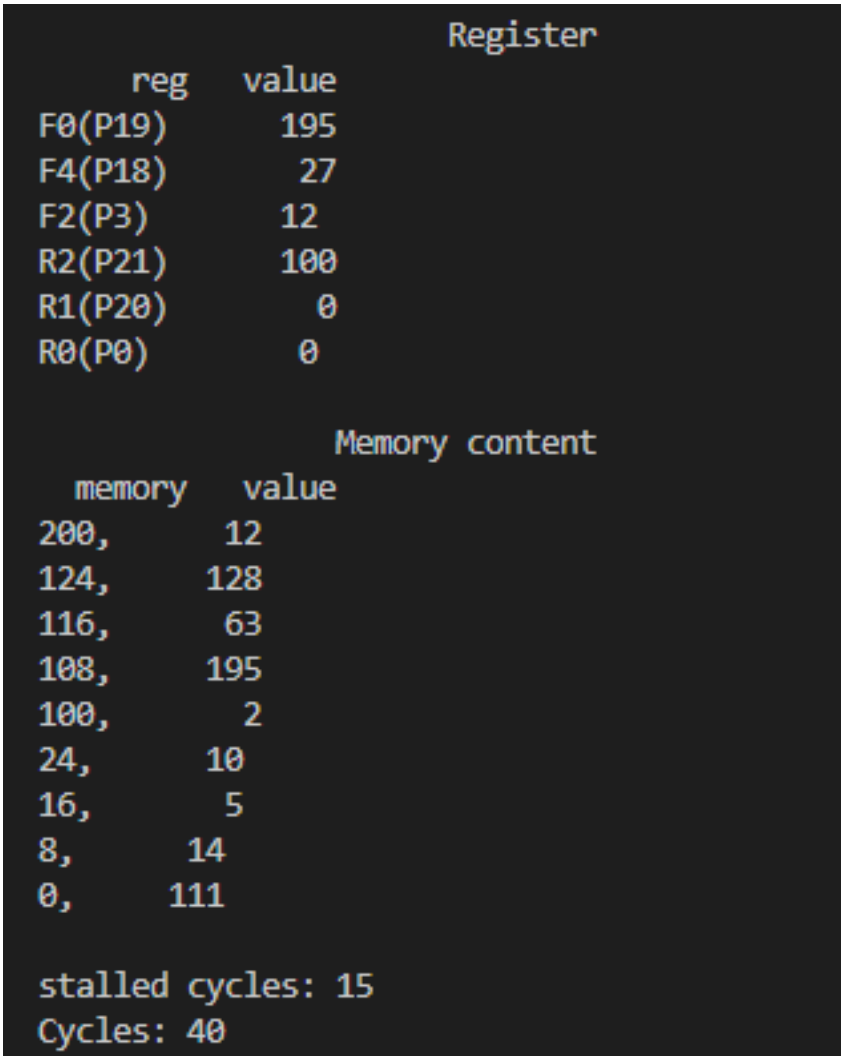
The following results is when the parameters are NF=4, NW=4, NR=16,NB=4

The cycles are 39.

```
                    Register
     reg   value
F0(P19)      195
F4(P18)       27
F2(P3)       12
R2(P21)      100
R1(P20)        0
R0(P0)         0

                Memory content
  memory   value
200,       12
124,      128
116,       63
108,      195
100,        2
24,       10
16,         5
8,       14
0,      111

stalled cycles: 21
Cycles: 39
```

1. The following result is setting the NW=NB=2

   Compare with default, the cycles increase due to the width of commit decrease, the simulator can only issue two instruction at most each time and takes more cycles.



Register

reg	value
F0(P19)	195
F4(P18)	27
F2(P3)	12
R2(P21)	100
R1(P20)	0
R0(P0)	0

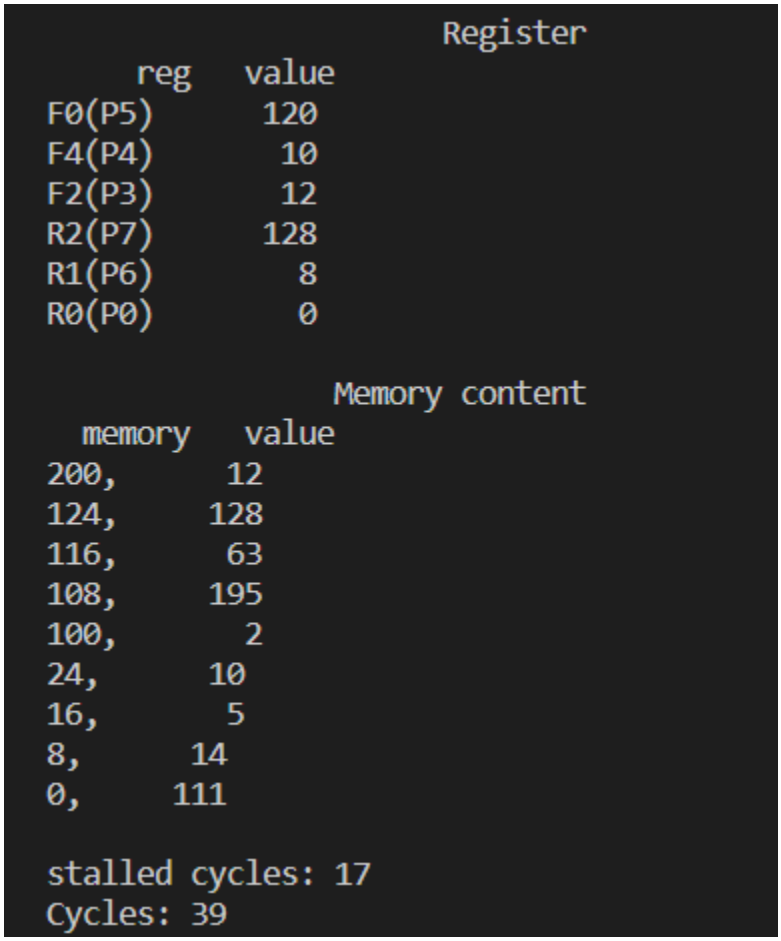
Memory content

memory	value
200,	12
124,	128
116,	63
108,	195
100,	2
24,	10
16,	5
8,	14
0,	111

stalled cycles: 15
Cycles: 40

2. The following is the result after setting NF = 2

   The total cycles are not changed compared to the default, but the stalled cycles decrease, reduce the number of fetch instructions prevent congestion for waring simulator to finish previous instructions.



```
                    Register
     reg   value
F0(P5)      120
F4(P4)       10
F2(P3)       12
R2(P7)      128
R1(P6)        8
R0(P0)        0

                Memory content
  memory   value
200,       12
124,      128
116,       63
108,      195
100,        2
24,       10
16,        5
8,       14
0,      111

stalled cycles: 17
Cycles: 39
```

3. When NR = 8 stalled cycles: 23, total cycles:39

   When NR = 32 stalled cycles:21, total cycles:39

   When NR=4 stalled cycles:37, total cycles:48

   When NR is greater than 8, the total cycles is unchanged, means ROB didn't full during the execution, so the cycles remain the same, however when change NR to 4, the stalled cycles increase a lot, means the ROB is congestion. More stalled is needed.