Лабораторная работа №2	M3136	2022
Моделирование схем в Verilog	Исаев Максим Викторович	

Цель работы.

Построение кэша и моделирование системы “процессор-кэш-память” на языке описания Verilog. Анализ и моделирование работы алгоритма умножения матриц на процессоре с L1 кэшем.

Инструментарий и требования к работе.

- Язык описания аппаратуры Verilog (Icarus Verilog version 10.1).
- Python 3.10.4 для аналитического решения.

Формулировка задачи из условия.

Описание.

В работе необходимо построить и смоделировать систему “Процессор-Кэш-Память”. Кэш работает по схеме look-trough write-back.

- **Look-through** означает, что либо запрос процессора к памяти проходит через кэш. В случае если в кэше нет нужных данных, то они сначала записываются в кэш из памяти, а затем кэш отдаёт их процессору.
- **Write-back** означает, что в случае записи данных процессором, они будут записаны в кэш в соответствующую кэш-линию. В случае кэш промаха, сначала в кэш запишется нужная линия, и на место её будут записаны нужные данные. Данные будут записаны в память только когда эта кэш-линию будет вытеснена.
- Кэш является **наборно-ассоциативным** – это означает, что каждой кэш-линии в памяти соответствует не определённая линия в кэше, а набор из линий (SET), т. е. каждая линия может быть записана в одну из линий

своего SET’а. Это позволяет хранить в кэше несколько любых линий параллельно. Это “несколько” определяется ассоциативностью кэша – количеством линий в одном SET’е.

- Политика вытеснения **LRU** (least recently used). При такой политике, когда весь SET занят, но необходимо записать в него ещё линию, то она будет записана на место той, что использовалась последний раз раньше всех остальных в SET’е.

Вариант.

Мне достался вариант 1. В нём заданы следующие параметры:

Ассоциативность	2 – **CACHE_WAY**
Размер тэга адреса	10 бит – **CACHE_TAG_SIZE**
Размер кэш-линии	16 байта – **CACHE_LINE_SIZE**
Кол-во кэш-линий	64 – **CACHE_LINE_COUNT**
Размер памяти	512 Кбайт – **MEM_SIZE**

Вычисление недостающих параметров системы.

$$CACHE\_SIZE = CACHE\_LINE\_COUNT * CACHE\_LINE\_SIZE = 1024 \text{ Байт}$$

$$CACHE\_SETS\_COUNT = \frac{CACHE\_LINE\_COUNT}{CACHE\_WAY} = 32$$

$$CACHE\_SET\_SIZE = log_2(CACHE\_SETS\_COUNT) = 5$$

$$ADDR\_SIZE = log_2(MEM\_SIZE) = 19$$

$$CACHE\_OFFSET\_SIZE = ADDR\_SIZE - CACHE\_TAG\_SIZE - CACHE\_SET\_SIZE = 4$$

Аналитическое решение задачи.

Для аналитического решения я написал программу на Python, которая повторяет предложенный алгоритм и симулирует запросы и состояния кэш-линий в кэше. Файл называется “analytic.py” и лежит в корневом каталоге

репозитория. Так же эта программа позволяет делать автоматической сравнение данных, полученных в симуляции и в модели. Подробнее об этом и о результатах в главе 9 “Сравнение полученных результатов”.

Моделирование заданной системы на Verilog.

Модель состоит из 3 элементов: CPU, cache и memory. Так же есть модуль testbench, однако он нужен только для коммутации остальных модулей вместе, сигнала reset и генерации clk.

CPU является инициатором всех взаимодействий (в том числе cache_dump и mem_dump). В нём выделены отдельные task для каждой команды, которые образуют своеобразный ассемблер, на котором написаны алгоритм и тесты. Все запуски производятся из главного блока initial.

CPU и cache соединены 3 шинами: A1 (15 бит), D1 (16 бит) и C1 (3 бита). Так как по шине A1 адрес передаётся за 2 такта, (TAG + SET | OFFSET) то размер шины вычислялся по формуле:

$$\max(CACHE\_TAG\_SIZE + CACHE\_SET\_SIZE, CACHE\_OFFSET\_SIZE)$$

Размер шина команд вычислялся так: log_2(колво команд)

MEM представляет оперативную память объёмом 512Кбайт. В этом модуле симулируются задержки, присущие настоящей памяти. Шина A2 имеет такой же размер, как и A1 (15 бит), D2 (16 бит), C2 (2 бита, формула аналогична C1).

Cache самый важный и сложный модуль. Он содержит в себе память для кэш-линий и вспомогательной информации, такой как tag, LRU, dirty и valid. Так же cache занимается подсчётом попаданий и промахов. По команде cache_dump эта информация выводится в консоль и в файл. Важно отметить, что на шине C1 команды C1_RESPONSE и C1_WRITE32 имеют одинаковые коды, но пишутся разными агентами. В коде эта команда называется C1_WRITE32_RESP.

Вся логика кэша и памяти прописана в блоках always, которые работают по сигналу clk. В каждом модуле есть task’и задержек (delay, read_bus_delay…) они нужны для синхронизации чтения и записи шин. По умолчанию шинами 1 (cpu – cache) владеет cpu, а шинами 2 (cache – mem) владеет cache. Передача владения происходит путём присваивания на шину сигнала z владеющей стороной. Это делает в случаях:

1. Процессор отдал кэшу команду C1_READ[8|16|32] или C1_WRITE[8|16|32] и ждёт от него C1_RESPONSE на шине C1.

2. Cache отдал памяти команду C2_READ/C2_WRITE и ждёт C2_RESPONSE для начала передачи информации.

Buffers.

Для каждой шины у каждого модуля есть буфер – переменная типа reg с суффиксом _buff. В эту переменную записывается значение, которые будет присвоено шине в следующий такт Verilog'а при помощи assign, которые располагаются внизу. Это необходимо, потому что внутри блоков always (где располагается логика) нельзя присваивать значения на wire.

Read and write.

Пересылка данных по шине D1 происходит за 1 или 2 такта. 1 такт требуется для 8 и 16 битных команд, для 32 битных же требуется 2 такта, т. к. иначе не хватает ширины шины.

По шине D2 передаются кэш линии целиком. В Cache и MEM одновременно крутятся 2 цикла, один из которых отправляет данные, а другой читает.

Race condition.

В связи с тем, что в реальном железе нельзя одновременно писать и читать с шины из-за физических задержек, то я добавил в протокол общения по шине условие, что запись происходит на falling edge/negedge, а чтение на rising edge/posedge. Это гарантирует что в момент чтения на шине находятся валидные данные и позволяет реализовать эту схему в железе. Этот протокол соблюдается при помощи методов задержек, о которых говорилось ранее.

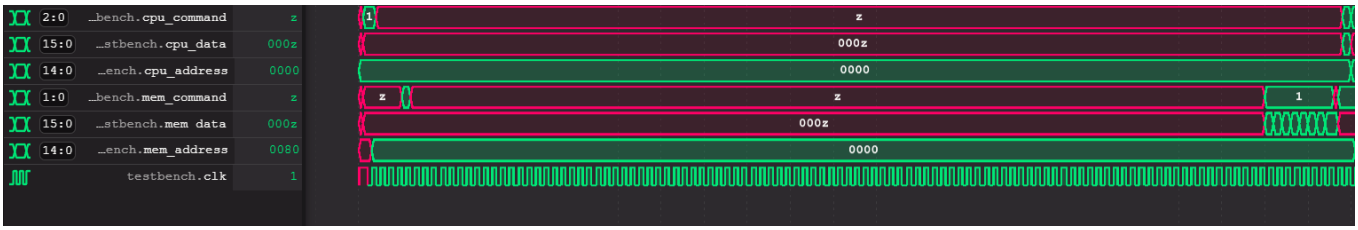
Tests.

Для тестирования работоспособности системы по ходу разработки я написал пару тестов. Они располагаются в cpu и вынесены в отдельные task'и: read_write_test, invalidate_test и eviction_test. Каждый из них воспроизводит какой-то сценарий работы. Все тесты построены по принципу: пишем, воспроизводим потенциально проблемный случай, читаем в надежде увидеть то, что записали. Тесты запускаются по окончании симуляции и дампинга кэша и памяти в файлы. Результат их работы выводится в консоль, чтобы можно было убедиться в работоспособности, увидев логи в README на GitHub.

Воспроизведение задачи на Verilog.

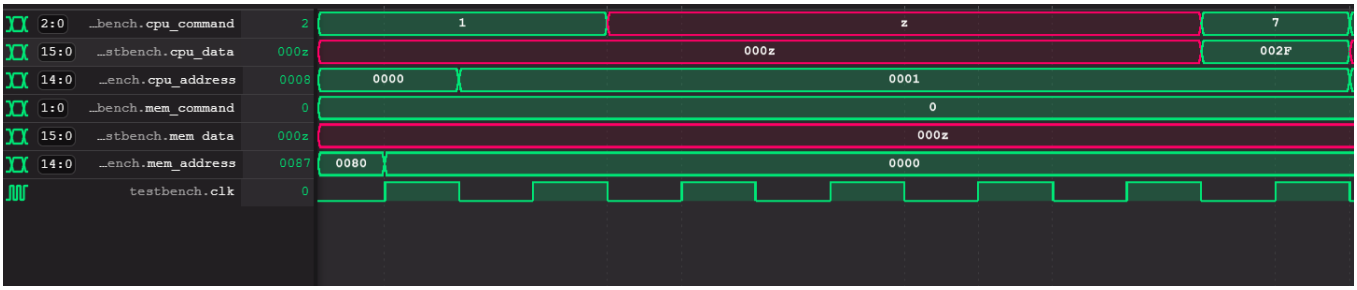
В модуле cpu в task “matrix_mull_sim” проводится симуляция заданного алгоритма. Это алгоритм умножения двух матриц. Там присутствуют команды READ8, READ16, WRITE32. Рассмотрим несколько запросов с временными диаграммами.

Пример кэш промаха.



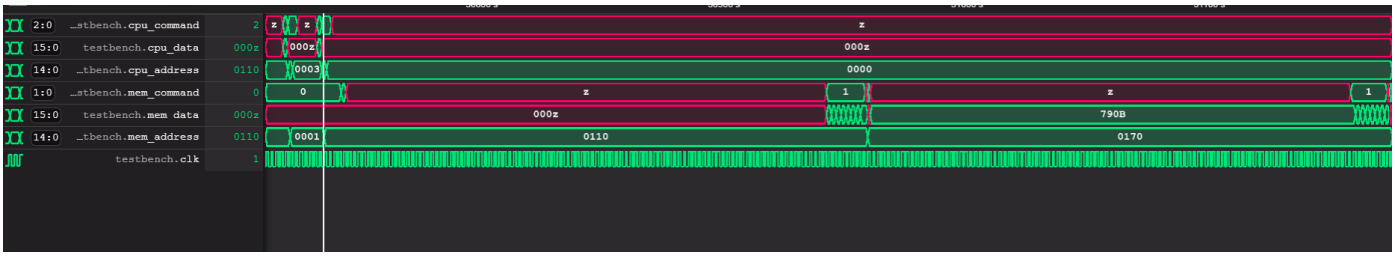
Поступает команда на чтение от cpu, cache читает адрес, фиксирует промах и через 4 такта отдаёт команду в memory. Через 200 тактов приходит ответ от памяти и начинается передача кэш-линии, которая занимает 8 тактов. В конце cache отдаёт ответ cpu и передаёт информацию за 1 такт.

Пример кэш попадания.



Как и в прошлом примере поступает команда на чтение от cpu, в этот раз запрашиваемая кэш-линия оказывается в памяти и через 6 тактов cache даёт ответ и передаёт нужные данные.

Пример кэш промаха с вытеснением.



От cpu поступает запрос, запрашиваемой кэш-линии не оказывает в кэше и cache отправляет запрос на чтение в память. Но места для этой линии в кэше нет, т. к. весь подходящий сет занят другими линиями, да ещё и линия, которую нужно вытеснить (least recently used), содержит изменённые данные (dirty). Поэтому cache начинает запись это кэш линии в memory. Суммарно мы имеем 4 на обращение к памяти + 200 на ответ от памяти + 8

на передачу новой линии + 200 на ожидания ответа от памяти для записи + 8 циклов на передачи старый dirty линии в память.

Сравнение полученных результатов.

Для сравнения результатов программа на Verilog записывает получившиеся в её симуляции значения запросов и кэш попаданий в файл “hit_stat.dump”. После чего вы можете запустить симуляцию на Python “analytic.py”, которая посчитает свой результат, после чего считает из файла результат Verilog’а и сравнит их. У меня получилось добиться полного совпадения значений запросов (249600) и кэш попаданий (228080). Чуть больше 91% запросов были кэш попаданиями.

Такой процент получился благодаря последовательному расположению данных в памяти. Так же данные разных массивов не конкурировали за одни cache SET’ы в силу не большого размера по сравнению с размером всей памяти (массив a: 2,048 байт, массив b: 3,840 байт, массив c: 15,360 байт) примерно 4 процента от всей памяти. Если бы, например массив a и c постоянно соперничали за кэш то было бы значительно больше промахов, да и чаще было бы вытеснение dirty данных, что в 2 раза дольше.

Кол-во тактов получилось порядка 5–7 миллионов. Это $2m$ суммирований во внешнем цикле, $2mnk$ суммирований во внутреннем, mnk умножений ($5mnk$ циклов), 228080 кэш попаданий по 7 циклов, и 21520 промахов по ~200-400 циклов. Симуляция подсчитала 5099575 циклов.

Запуск

Если запустить эти команды последовательно из корневой директории проекта

- iverilog -g2012 -o testbench.out testbench.sv
- vvp testbench.out
- python analytic.py

то будет запущена симуляция алгоритма на Verilog, автоматические тесты на Verilog, симуляция на Python, сравнение результатов. В случае необходимости тесты можно отключить, удалив строчки read_write_test, invalidate_test и eviction_test. из cpu initial. (225–229). Вся информация о запросах во время тестов выводится в консоль.

Вывод программы на Verilog	Вывод программы на Python
Filling the MEM... MEM filled Simulation started Simulation finished Cache dumped successful. Check cache.dump HIT statistic: Requests: 249600 Hits: 228080 HIT RATE: 0.913782 HIT stat dumped to hit_stat.dump. Total clock cycles: 5099575 Mem dumped successful. Check mem.dump	Data validation passed Requests: 249600 Hits: 228080 HIT RATE: 0.913782

Листинг кода.

Testbench.

```
`include "cache.sv"
`include "cpu.sv"
`include "mem.sv"

module testbench ();
    parameter BUS_SIZE = 16;
    parameter MEM_ADDR_SIZE = 10 + 9;   // log2(MEM_SIZE)
    parameter CACHE_OFFSET_SIZE = $clog2(CACHE_LINE_SIZE);
    parameter CACHE_LINE_SIZE = 16;
    wire [MEM_ADDR_SIZE-CACHE_OFFSET_SIZE-1:0] cpu_address;
    wire [BUS_SIZE-1:0] cpu_data;
    wire [3-1:0] cpu_command;
    wire [MEM_ADDR_SIZE-CACHE_OFFSET_SIZE-1:0] mem_address;
    wire [BUS_SIZE-1:0] mem_data;
    wire [2-1:0] mem_command;
    wire cache_dump;
    wire mem_dump;

    reg clk;
    reg reset;


    cpu #(MEM_ADDR_SIZE, BUS_SIZE, CACHE_OFFSET_SIZE)
    cpu (
        .clk(clk),
        .cache_dump(cache_dump),
        .mem_dump(mem_dump),
        .address(cpu_address),
        .data(cpu_data),
        .command(cpu_command)
        );

    cache #(BUS_SIZE, MEM_ADDR_SIZE, CACHE_OFFSET_SIZE, CACHE_LINE_SIZE)
```

```
    cache (
        .clk(clk),
        .reset(reset),
        .dump(cache_dump),

        .cpu_address(cpu_address),
        .cpu_data(cpu_data),
        .cpu_command(cpu_command),

        .mem_address(mem_address),
        .mem_data(mem_data),
        .mem_command(mem_command)
        );

    mem #(MEM_ADDR_SIZE, BUS_SIZE, CACHE_OFFSET_SIZE, CACHE_LINE_SIZE)
    mem(
        .clk(clk),
        .reset(reset),
        .dump(mem_dump),
        .address(mem_address),
        .data(mem_data),
        .command(mem_command)
        );

    int cyclesCount = 1;
    initial begin
        #2 forever begin
            #1 if (clk) begin
                ++cyclesCount;
            end
            if (mem_dump) begin
                $display("Total clock cycles: %0d", cyclesCount);
                repeat(2) begin
                    @(posedge clk);
                end
            end
        end
    end

    initial begin
        reset = 0;
        #1 reset = 1;
        #1 reset = 0;
        clk = 1'd0;

        forever begin
            #1 clk = ~clk;
        end
    end

endmodule
```

CPU.

```
module cpu #(
    parameter MEM_ADDR_SIZE = 10 + 9,
    parameter BUS_SIZE = 16,
    parameter CACHE_OFFSET_SIZE = 4
    ) (

    input clk,
    output cache_dump,
    output mem_dump,
    output [MEM_ADDR_SIZE-CACHE_OFFSET_SIZE-1:0] address,
    inout [BUS_SIZE-1:0] data,
    inout [3-1:0] command
);

    localparam  C1_NOP          = 3'd0,
                C1_READ8        = 3'd1,
                C1_READ16       = 3'd2,
                C1_READ32       = 3'd3,
                C1_INV_LINE     = 3'd4,
                C1_WRITE8       = 3'd5,
                C1_WRITE16      = 3'd6,
                C1_WRITE32_RESP = 3'd7;

    reg [MEM_ADDR_SIZE-1:0] cpu_address_buff;
    reg [MEM_ADDR_SIZE-CACHE_OFFSET_SIZE-1:0] address_bus_buff;
    reg [BUS_SIZE*2-1:0] data_to_write;
    reg [BUS_SIZE-1:0] data_buff;
    reg [3-1:0] cpu_command_buff;
    reg [BUS_SIZE-1:0] recieved_data;
    reg [BUS_SIZE*2-1:0] local_storage;

    reg cache_dump_buff;
    reg mem_dump_buff;

    task delay;
        begin
            @(negedge clk);
        end
    endtask

    task read_bus_delay;
        begin
            @(posedge clk);
        end
    endtask

    task send_address;
        address_bus_buff = cpu_address_buff[MEM_ADDR_SIZE-1:CACHE_OFFSET_SIZE];
        delay;
        address_bus_buff = cpu_address_buff[CACHE_OFFSET_SIZE-1:0];
        delay;
    endtask
```

```
task wait_for_resp;
    while (command !== C1_WRITE32_RESP) begin
        read_bus_delay;
    end
endtask

task READ8;
    cpu_command_buff = C1_READ8;
    READ;
    local_storage = recieved_data[8-1:0];
    delay;
endtask
task READ16;
    cpu_command_buff = C1_READ16;
    READ;
    local_storage = recieved_data;
    delay;
endtask
task READ32;
    cpu_command_buff = C1_READ32;
    READ;
    local_storage[BUS_SIZE-1:0] = recieved_data;
    read_bus_delay;
    recieved_data = data;
    local_storage[BUS_SIZE*2-1:BUS_SIZE] = recieved_data;
    delay;
endtask
task READ;
    send_address;
    cpu_command_buff = 'z;
    wait_for_resp;
    recieved_data = data;
endtask

task WRITE8;
    cpu_command_buff = C1_WRITE8;
    WRITE;
    delay;
endtask
task WRITE16;
    cpu_command_buff = C1_WRITE16;
    WRITE;
    delay;
endtask
task WRITE32;
    cpu_command_buff = C1_WRITE32_RESP;

    data_buff = data_to_write[BUS_SIZE-1:0];
    address_bus_buff = cpu_address_buff[MEM_ADDR_SIZE-1:CACHE_OFFSET_SIZE];
    delay;

    data_buff = data_to_write[BUS_SIZE*2-1:BUS_SIZE];
    address_bus_buff = cpu_address_buff[CACHE_OFFSET_SIZE-1:0];
```

```
    delay;

    cpu_command_buff = 'z;
    wait_for_resp;
    data_buff = 'z;
    delay;
endtask
task WRITE;
    data_buff = data_to_write;
    send_address;
    cpu_command_buff = 'z;
    wait_for_resp;
    data_buff = 'z;
endtask

task INV;
    cpu_command_buff = C1_INV_LINE;
    send_address;
    cpu_command_buff = 'z;
    wait_for_resp;
    delay;
endtask

initial begin
    address_bus_buff = 'z;
    cpu_command_buff = 'z;
    recieved_data = 0;
    data_buff = 'z;
    cache_dump_buff = 0;
    mem_dump_buff = 0;
end

task dump_cache;
    cache_dump_buff = 1;
    delay;
    cache_dump_buff = 0;
endtask

task dump_mem;
    mem_dump_buff = 1;
    delay;
    mem_dump_buff = 0;
endtask

task dump_all;
    delay;
    dump_cache;
    dump_mem;
endtask

parameter M = 19'd64;
parameter N = 19'd60;
parameter K = 19'd32;
```

```
parameter aStart = 19'd0;
parameter aIntSize = 19'd1;
parameter aSize = M*K*aIntSize;

parameter bStart = aStart + aSize;
parameter bIntSize = 19'd2;
parameter bSize = K*N*bIntSize;

parameter cStart = bStart + bSize;
parameter cIntSize = 19'd4;
parameter cSize = M*N*cIntSize;

int pa;
int pb;
int pc;
int s;
int j;
int k;
int prev_val;

task matrix_mull_sim;
    $display("Simulation started");
    pa = aStart;
    pc = cStart;
    for (int i=0; i<M; ++i) begin
        for (j=0; j<N; ++j) begin
            pb = bStart;
            s = 0;
            for (k=0; k<K; ++k) begin
                // a
                cpu_address_buff = pa + k*aIntSize;
                READ8;
                prev_val = local_storage[7:0];

                //b
                cpu_address_buff = pb + j*bIntSize;
                READ16;
                s += local_storage[15:0] * prev_val;
                repeat(6) begin
                    delay;
                end

                pb += N*bIntSize;
                delay;
            end
            // c
            cpu_address_buff = pc + j*cIntSize;
            data_to_write = s;
            WRITE32;
        end
        pa += K*aIntSize;
        delay;
```

```
        pc += N*cIntSize;
        delay;
    end
    $display("Simulation finished");
endtask

// Place for test calls
initial begin
    delay;
    matrix_mull_sim;

    dump_all;

    read_write_test;

    invalidate_test;

    eviction_test;

    $finish();
end

assign address = address_bus_buff;
assign data = data_buff;
assign command = cpu_command_buff;
assign cache_dump = cache_dump_buff;
assign mem_dump = mem_dump_buff;

task pass;
    $display("%c[5;32mPASS%c[0m",27,27);
endtask

task fail;
    $display("%c[1;31mFAIL%c[0m",27,27);
endtask

reg [19-1:0] test_addr;
reg [32-1:0] test_data;
task read_write_test;
    $display("\n##############################################");
    $display("##############  READ/WRITE TEST  ##############");
    $display("##############################################\n");

    $display("@@@@@@@@@@@@@@@");
    $display("@@@  8 bit @@@");
    $display("@@@@@@@@@@@@@@@");
    test_addr = 19'b0000000000_01110_0000;
    test_data = 8'b1111_0000;

    cpu_address_buff = test_addr;
    $display("read from %b", cpu_address_buff);
    READ8;
    $display("data      %b", local_storage[8-1:0]);
```

```
$display("----------------");

cpu_address_buff = test_addr;
data_to_write = test_data;
$display("write to  %b", cpu_address_buff);
$display("data      %b", data_to_write[8-1:0]);
WRITE8;
$display("----------------");

cpu_address_buff = test_addr;
$display("read from %b", cpu_address_buff);
READ8;
$display("data      %b", local_storage[8-1:0]);
$display("----------------");

$display("\nREAD/WRITE 8 bit");
if (test_data == local_storage[8-1:0]) begin
    pass;
end else begin
    fail;
    $display("expected  %b", test_data[8-1:0]);
    $display("actual    %b", local_storage[8-1:0]);
end

$display("\n@@@@@@@@@@@@@@@");
$display("@@@ 16 bit @@@");
$display("@@@@@@@@@@@@@@@");
test_addr = 19'b0000000000_01110_0010;
test_data = 16'b1111_1111_0000_0000;

cpu_address_buff = test_addr;
$display("read from %b", cpu_address_buff);
READ16;
$display("data      %b", local_storage[16-1:0]);
$display("----------------");

cpu_address_buff = test_addr;
data_to_write = test_data;
$display("write to  %b", cpu_address_buff);
$display("data      %b", data_to_write[16-1:0]);
WRITE16;
$display("----------------");

cpu_address_buff = test_addr;
$display("read from %b", cpu_address_buff);
READ16;
$display("data      %b", local_storage[16-1:0]);
$display("----------------");

$display("\nREAD/WRITE 16 bit");
if (test_data == local_storage[16-1:0]) begin
    pass;
end else begin
```

```
        fail;
        $display("expected  %b", test_data[16-1:0]);
        $display("actual    %b", local_storage[16-1:0]);
    end

    $display("\n@@@@@@@@@@@@@@@");
    $display("@@@ 32 bit @@@");
    $display("@@@@@@@@@@@@@@@");
    test_addr = 19'b0000000000_01110_0000;
    test_data = 32'b0101_0101_0101_0101_0101_0101_0101_0101;


    $display("-----------------");
    cpu_address_buff = test_addr;
    $display("read from %b", cpu_address_buff);
    READ32;
    $display("data      %b", local_storage[32-1:0]);
    $display("-----------------");

    cpu_address_buff = test_addr;
    data_to_write = test_data;
    $display("write to  %b", cpu_address_buff);
    $display("data      %b", data_to_write[32-1:0]);
    WRITE32;
    $display("-----------------");

    cpu_address_buff = test_addr;
    $display("read from %b", cpu_address_buff);
    READ32;
    $display("data      %b", local_storage[32-1:0]);
    $display("-----------------");

    $display("\nREAD/WRITE 32 bit");
    if (test_data == local_storage[32-1:0]) begin
        pass;
    end else begin
        fail;
        $display("expected  %b", test_data[32-1:0]);
        $display("actual    %b", local_storage[32-1:0]);
    end
    $display("");
endtask

reg [19-1:0] evtest_addr[3-1:0];
reg [32-1:0] evtest_data[3-1:0];
task eviction_test;
    $display("\n###########################################");
    $display("#############  EVICTION TEST  #############");
    $display("###########################################\n");

    $display("~~~~~~~~~~~~~~");
    $display("~~~ FIRST ~~~");
    $display("~~~~~~~~~~~~~~");
```

```
evtest_addr[0] = 19'b0000000000_01110_0000;
evtest_data[0] = 32'b1111_0000_1111_0000_0000_1111_0000_1111;

$display("----------------");
cpu_address_buff = evtest_addr[0];
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage);
$display("----------------");

cpu_address_buff = evtest_addr[0];
data_to_write = evtest_data[0];
$display("write to  %b", cpu_address_buff);
$display("data      %b", data_to_write);
WRITE32;
$display("----------------");

cpu_address_buff = evtest_addr[0];
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage);
$display("----------------\n");

$display("~~~~~~~~~~~~~~~");
$display("~~~ SECOND ~~~");
$display("~~~~~~~~~~~~~~~");
evtest_addr[1] = 19'b0000000001_01110_0000;
evtest_data[1] = 32'b1111_1111_1111_1111_0000_1111_1111_0000;

$display("----------------");
cpu_address_buff = evtest_addr[1];
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage);
$display("----------------");

cpu_address_buff = evtest_addr[1];
data_to_write = evtest_data[1];
$display("write to  %b", cpu_address_buff);
$display("data      %b", data_to_write);
WRITE32;
$display("----------------");

cpu_address_buff = evtest_addr[1];
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage);
$display("----------------\n");

$display("~~~~~~~~~~~~~~~");
$display("~~~ THIRD ~~~");
$display("~~~~~~~~~~~~~~~");
evtest_addr[2] = 19'b0000000010_01110_0000;
```

```
evtest_data[2] = 32'b1111_1111_0001_1001_1001_1000_1111_1111;

$display("----------------");
cpu_address_buff = evtest_addr[2];
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage);
$display("----------------");

cpu_address_buff = evtest_addr[2];
data_to_write = evtest_data[2];
$display("write to  %b", cpu_address_buff);
$display("data      %b", data_to_write);
WRITE32;
$display("----------------");

cpu_address_buff = evtest_addr[2];
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage);
$display("----------------\n");

$display("~~~~~~~~~~~~~~~~~~");
$display("~~~ FIRST AGAIN ~~~");
$display("~~~~~~~~~~~~~~~~~~");
$display("----------------");
cpu_address_buff = evtest_addr[0];
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage);
$display("----------------\n");

$display("EVICTION FIRST");
if (evtest_data[0] == local_storage[32-1:0]) begin
    pass;
end else begin
    fail;
    $display("expected  %b", evtest_data[0][32-1:0]);
    $display("actual    %b", local_storage[32-1:0]);
end

$display("\n~~~~~~~~~~~~~~~~~~");
$display("~~~ SECOND AGAIN ~~~");
$display("~~~~~~~~~~~~~~~~~~");
$display("----------------");
cpu_address_buff = evtest_addr[1];
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage);
$display("----------------\n");

$display("EVICTION SECOND");
if (evtest_data[1] == local_storage[32-1:0]) begin
```

```
        pass;
    end else begin
        fail;
        $display("expected  %b", evtest_data[1][32-1:0]);
        $display("actual    %b", local_storage[32-1:0]);
    end

    $display("\n~~~~~~~~~~~~~~~~~~~");
    $display("~~~ THIRD AGAIN ~~~");
    $display("~~~~~~~~~~~~~~~~~~~");
    $display("----------------");
    cpu_address_buff = evtest_addr[2];
    $display("read from %b", cpu_address_buff);
    READ32;
    $display("data      %b", local_storage);
    $display("----------------\n");

    $display("EVICTION THIRD");
    if (evtest_data[2] == local_storage[32-1:0]) begin
        pass;
    end else begin
        fail;
        $display("expected  %b", evtest_data[2][32-1:0]);
        $display("actual    %b", local_storage[32-1:0]);
    end
    $display("");
endtask

task invalidate_test;
    $display("\n##############################################");
    $display("#############  INVALIDATE TEST  #############");
    $display("##############################################\n");
    test_addr = 19'b0000000000_10001_0000;
    test_data = 32'b0101_0101_0101_0101_0101_0101_0101_0101;

    $display("----------------");
    cpu_address_buff = test_addr;
    $display("read from %b", cpu_address_buff);
    READ32;
    $display("data      %b", local_storage);
    $display("----------------");

    cpu_address_buff = test_addr;
    data_to_write = test_data;
    $display("write to  %b", cpu_address_buff);
    $display("data      %b", data_to_write);
    WRITE32;
    $display("----------------");

    cpu_address_buff = test_addr;
    $display("read from %b", cpu_address_buff);
    READ32;
    $display("data      %b", local_storage);
```

```
        $display("----------------");

        cpu_address_buff = test_addr;
        $display("inv_line  %b", cpu_address_buff);
        INV;
        $display("----------------");

        cpu_address_buff = test_addr;
        $display("read from %b", cpu_address_buff);
        READ32;
        $display("data      %b", local_storage);
        $display("----------------");

        $display("\nINV");
        if (test_data[8-1:0] == local_storage[8-1:0]) begin
            pass;
        end else begin
            fail;
            $display("expected  %b", test_data[32-1:0]);
            $display("actual    %b", local_storage[32-1:0]);
        end
    endtask

endmodule
```

Cache.

```
module cache#(
    parameter BUS_SIZE = 16 ,
    parameter MEM_ADDR_SIZE = 10 + 9,
    parameter CACHE_OFFSET_SIZE = 4,
    parameter CACHE_LINE_SIZE   = 16
    ) (
    input clk,
    input reset,
    input dump,
    input [MEM_ADDR_SIZE-CACHE_OFFSET_SIZE-1:0] cpu_address,
    inout [BUS_SIZE-1:0] cpu_data,
    inout [3-1:0] cpu_command,

    output [MEM_ADDR_SIZE-CACHE_OFFSET_SIZE-1:0] mem_address,
    inout [BUS_SIZE-1:0] mem_data,
    inout [2-1:0] mem_command
);

    parameter CACHE_WAY         = 2;
    parameter CACHE_LINE_COUNT  = 64;

    parameter CACHE_SET_SIZE    = $clog2(CACHE_SETS_COUNT);
    parameter CACHE_TAG_SIZE    = 10;
    parameter CACHE_SIZE        = CACHE_LINE_COUNT * CACHE_LINE_SIZE;
    parameter CACHE_SETS_COUNT  = CACHE_LINE_COUNT/CACHE_WAY;
```

```
    localparam  C1_NOP          = 3'd0,
                C1_READ8        = 3'd1,
                C1_READ16       = 3'd2,
                C1_READ32       = 3'd3,
                C1_INV_LINE     = 3'd4,
                C1_WRITE8       = 3'd5,
                C1_WRITE16      = 3'd6,
                C1_WRITE32_RESP = 3'd7;

    localparam  C2_NOP          = 2'd0,
                C2_RESPONSE     = 2'd1,
                C2_READ         = 2'd2,
                C2_WRITE        = 2'd3;

    // STORAGE
    reg valid_array [CACHE_SETS_COUNT-1:0][CACHE_WAY-1:0];
    reg dirty_array [CACHE_SETS_COUNT-1:0][CACHE_WAY-1:0];
    reg LRU_array [CACHE_SETS_COUNT-1:0];
    reg [CACHE_TAG_SIZE-1:0] tag_array [CACHE_SETS_COUNT-1:0][CACHE_WAY-1:0];
    reg [CACHE_LINE_SIZE*8-1:0] data_array [CACHE_SETS_COUNT-1:0][CACHE_WAY-1:0];
// stores lines

    reg [CACHE_TAG_SIZE-1:0] cpu_tag_buff;
    reg [CACHE_SET_SIZE-1:0] cpu_set_buff;
    reg [CACHE_OFFSET_SIZE-1:0] cpu_offset_buff;
    reg [BUS_SIZE-1:0] cpu_data_bus_buff;
    reg [BUS_SIZE*2-1:0] cpu_data_to_write;
    reg [3-1:0] cpu_command_buff;

    reg [MEM_ADDR_SIZE-CACHE_OFFSET_SIZE-1:0] mem_address_buff;
    reg [CACHE_LINE_SIZE*8-1:0] mem_line_buff;
    reg [BUS_SIZE-1:0] mem_data_buff;
    reg [2-1:0] mem_command_buff;

    // Analytic
    real req;
    real hit;

    // Tasks
    task delay;
        begin
            @(negedge clk);
        end
    endtask

    task read_bus_delay;
        begin
            @(posedge clk);
        end
    endtask

    task hit_resp_delay;
```

```
        repeat(4) begin
            delay;
        end
    endtask

    task miss_req_delay;
        repeat(3) begin
            delay;
        end
    endtask

    task wait_for_resp;
        while (mem_command !== C2_RESPONSE) begin
            read_bus_delay;
        end
    endtask

    task evict_if_dirty;
        if (dirty_array[cpu_set_buff][index_in_set] == 1) begin
            mem_address_buff = {tag_array[cpu_set_buff][index_in_set],
cpu_set_buff};
            write_to_MM;
        end
    endtask

    task replace_from_MM;
        // command
        mem_command_buff = C2_READ;
        delay;
        mem_command_buff = 'z;
        wait_for_resp;

        // data
        for (int i=0; i<CACHE_LINE_SIZE/2; i=i+1) begin
            mem_line_buff[BUS_SIZE*i +: BUS_SIZE] = mem_data;
            read_bus_delay;
        end

        // restore
        mem_command_buff = C2_NOP;

        if (valid_array[cpu_set_buff][0] == 0) begin
            index_in_set = 0;
            store;
        end else if (valid_array[cpu_set_buff][1] == 0) begin
            index_in_set = 1;
            store;
        end else begin
            // evict if no empty space
            index_in_set = LRU_array[cpu_set_buff];
            evict_if_dirty;

            index_in_set = LRU_array[cpu_set_buff];
```

```
            store;
        end
    endtask

    task write_to_MM;
        // command
        mem_command_buff = C2_WRITE;
        delay;
        mem_command_buff = 'z;
        mem_data_buff = data_array[cpu_set_buff][index_in_set][0 +: BUS_SIZE];
        wait_for_resp;

        // data
        delay;
        for (int i=1; i<CACHE_LINE_SIZE/2; i=i+1) begin
            mem_data_buff = data_array[cpu_set_buff][index_in_set][BUS_SIZE*i +:
BUS_SIZE];
            delay;
        end
        // restore
        mem_command_buff = C2_NOP;
        mem_data_buff = 'z;
    endtask

    reg index_in_set;
    task store;
        data_array[cpu_set_buff][index_in_set] = mem_line_buff;
        tag_array[cpu_set_buff][index_in_set] = cpu_tag_buff;
        valid_array[cpu_set_buff][index_in_set] = 1;
        dirty_array[cpu_set_buff][index_in_set] = 0;
        LRU_array[cpu_set_buff] = ~index_in_set;
    endtask

    task read_cpu_address;
        mem_address_buff = cpu_address;
        cpu_tag_buff = cpu_address[CACHE_TAG_SIZE+CACHE_SET_SIZE-
1:CACHE_SET_SIZE];
        cpu_set_buff = cpu_address[CACHE_SET_SIZE-1:0];

        delay;

        cpu_offset_buff = cpu_address[3:0];
    endtask

    reg [3-1:0] cur_cpu_command;
    task read_from_storage;
        delay;
        if (cur_cpu_command == C1_READ8) begin
            cpu_data_bus_buff =
data_array[cpu_set_buff][index_in_set][cpu_offset_buff*8 +: 8];
        end else if (cur_cpu_command == C1_READ16 || cur_cpu_command == C1_READ32)
begin
```

```
            cpu_data_bus_buff =
data_array[cpu_set_buff][index_in_set][cpu_offset_buff*8 +: 16];
        end
        LRU_array[cpu_set_buff] = ~index_in_set;
    endtask

    task write_to_storage;
        delay;
        if (cur_cpu_command == C1_WRITE8) begin
            data_array[cpu_set_buff][index_in_set][cpu_offset_buff*8 +: 8] =
cpu_data_to_write;
        end else if (cur_cpu_command == C1_WRITE16) begin
            data_array[cpu_set_buff][index_in_set][cpu_offset_buff*8 +: 16] =
cpu_data_to_write;
        end else if (cur_cpu_command == C1_WRITE32_RESP) begin
            data_array[cpu_set_buff][index_in_set][cpu_offset_buff*8 +: 32] =
cpu_data_to_write;
        end
        dirty_array[cpu_set_buff][index_in_set] = 1;
        LRU_array[cpu_set_buff] = ~index_in_set;
    endtask

    int hit_stat_file;
    task dump_hit_stat;
        $display("\nHIT statistic:");
        $display("req: %d\nhits: %d\nrate: %f", req, hit, hit/req);
        hit_stat_file = $fopen("hit_stat.dump", "w");
        if (hit_stat_file) begin
            $fdisplay(hit_stat_file, "%d\n%d\n", req, hit);
            $display("HIT stat dumped to hit_stat.dump.\n");
        end else begin
            $display("Error while hit stat dump");
        end
        $fclose(hit_stat_file);
    endtask

    int dump_f;
    task dump_to_file;
        dump_f = $fopen("cache.dump", "w");
        if (dump_f) begin
            $fdisplay(dump_f,"$$$$$$ CACHE DUMP $$$$$$");
            for (int i=0; i<CACHE_SETS_COUNT; i=i+1) begin
                $fdisplay(dump_f,"== SET 0x%0H\t==", i);

                $fdisplay(dump_f,"WAY %0d\nvalid: %b", 0, valid_array[i][0]);
                if (valid_array[i][0]) begin
                    $fdisplay(dump_f,"dirty: %b", dirty_array[i][0]);
                    $fdisplay(dump_f,"tag:   0x%0H", tag_array[i][0]);
                    $fdisplay(dump_f,"data:  0x%h", data_array[i][0]);
                end

                $fdisplay(dump_f,"WAY %0d\nvalid: %b", 1, valid_array[i][1]);
                if (valid_array[i][1]) begin
```

```
                    $fdisplay(dump_f,"dirty: %b", dirty_array[i][1]);
                    $fdisplay(dump_f,"tag:   0x%0H", tag_array[i][1]);
                    $fdisplay(dump_f,"data:  0x%h", data_array[i][1]);
                end
                $fdisplay(dump_f,"");
            end

            $display("Cache dumped successful. Check cache.dump");
        end else begin
            $display("Error while cache dump");
        end
        $fclose(dump_f);
    endtask


    always @(posedge clk or posedge reset) begin
        if (reset) begin
            for (int i=0; i<CACHE_SETS_COUNT; i=i+1) begin
                valid_array[i][0] =  0;
                dirty_array[i][0] =  0;
                tag_array[i][0]   = 'z;
                data_array[i][0]  = 'z;
                valid_array[i][1] =  0;
                dirty_array[i][1] =  0;
                tag_array[i][1]   = 'z;
                data_array[i][1]  = 'z;
            end
            mem_line_buff     =  0;
            cpu_data_bus_buff = 'z;
            cpu_command_buff  = 'z;
            cur_cpu_command   =  0;
            mem_command_buff  = 'z;
            mem_data_buff     = 'z;
        end else if (dump) begin
            dump_to_file;
            dump_hit_stat;
        end else if (cpu_command == C1_READ8 || cpu_command == C1_READ16 ||
cpu_command == C1_READ32) begin
            req = req + 1;

            cur_cpu_command = cpu_command;
            read_cpu_address;

            if (valid_array[cpu_set_buff][0] == 1 && tag_array[cpu_set_buff][0] ==
cpu_tag_buff) begin
                hit_resp_delay;
                hit = hit + 1;
                index_in_set = 0;
                read_from_storage;
            end else if (valid_array[cpu_set_buff][1] == 1 &&
tag_array[cpu_set_buff][1] == cpu_tag_buff) begin
                hit_resp_delay;
                hit = hit + 1;
```

```
                index_in_set = 1;
                read_from_storage;
            end else begin
                miss_req_delay;
                replace_from_MM;
                read_from_storage;
            end

            cpu_command_buff = C1_WRITE32_RESP;

            if (cur_cpu_command == C1_READ32) begin
                cpu_offset_buff += 2;
                read_from_storage;
            end
            delay;
            cpu_command_buff = 'z;
            cpu_data_bus_buff = 'z;
            cur_cpu_command = 'z;

        end else if (cpu_command == C1_WRITE8 || cpu_command == C1_WRITE16 ||
cpu_command == C1_WRITE32_RESP) begin
            req = req + 1;

            cur_cpu_command = cpu_command;
            cpu_data_to_write[0 +: BUS_SIZE] = cpu_data;

            read_cpu_address;

            if (cpu_command == C1_WRITE32_RESP) begin
                cpu_data_to_write[BUS_SIZE +: BUS_SIZE] = cpu_data;
            end

            if (valid_array[cpu_set_buff][0] == 1 && tag_array[cpu_set_buff][0] ==
cpu_tag_buff) begin
                hit_resp_delay;
                hit = hit + 1;
                index_in_set = 0;
                write_to_storage;
            end else if (valid_array[cpu_set_buff][1] == 1 &&
tag_array[cpu_set_buff][1] == cpu_tag_buff) begin
                hit_resp_delay;
                hit = hit + 1;
                index_in_set = 1;
                write_to_storage;
            end else begin
                miss_req_delay;
                replace_from_MM;
                write_to_storage;
                delay;
            end

            cpu_command_buff = C1_WRITE32_RESP;
            delay;
```

```
            cpu_command_buff = 'z;
            cur_cpu_command  = 'z;

        end else if (cpu_command == C1_INV_LINE) begin
            read_cpu_address;
            if (tag_array[cpu_set_buff][0] == cpu_tag_buff) begin
                index_in_set = 0;
                evict_if_dirty;
                valid_array[cpu_set_buff][0] = 0;
            end else if (tag_array[cpu_set_buff][1] == cpu_tag_buff) begin
                index_in_set = 0;
                evict_if_dirty;
                valid_array[cpu_set_buff][1] = 0;
            end

            cpu_command_buff = C1_WRITE32_RESP;
            delay;
            cpu_command_buff = 'z;
            cur_cpu_command  = 'z;

        end
    end

    assign mem_address = mem_address_buff;
    assign mem_data = mem_data_buff;
    assign mem_command = mem_command_buff;
    assign cpu_data = cpu_data_bus_buff;
    assign cpu_command = cpu_command_buff;

endmodule
```

Mem.

```
module mem #(
    parameter MEM_ADDR_SIZE = 10 + 9,
    parameter BUS_SIZE = 16 ,
    parameter CACHE_OFFSET_SIZE = 4,
    parameter CACHE_LINE_SIZE = 16
    ) (
    input clk,
    input reset,
    input dump,
    input [MEM_ADDR_SIZE-CACHE_OFFSET_SIZE-1:0] address,
    inout [BUS_SIZE-1:0] data,
    inout [2-1:0] command
);

    // 512KB = 2^9 * 2^10 = 2^19 = 2^15 lines * 2^4 bits in each line (16 8-bit
words)
    parameter MEM_SIZE = 1 << (MEM_ADDR_SIZE-CACHE_OFFSET_SIZE); // 2^15 cache
lines
    parameter RESPONSE_TIME = 100;

    localparam  C2_NOP      = 3'd0,
```

```
                C2_RESPONSE = 3'd1,
                C2_READ     = 3'd2,
                C2_WRITE    = 3'd3;

    reg [CACHE_LINE_SIZE*8-1:0] storage [MEM_SIZE-1:0]; // 2^15 16-byte lines
    reg [BUS_SIZE-1:0] data_buff; // single bus
    reg [2-1:0] command_buff;

    task delay;
        begin
            @(negedge clk);
        end
    endtask

    task read_bus_delay;
        begin
            @(posedge clk);
        end
    endtask

    task wait_and_response;
        repeat(RESPONSE_TIME) begin
            delay;
        end
        command_buff = C2_RESPONSE;
    endtask

    int dump_f;
    task dump_to_console;
        dump_f = $fopen("mem.dump", "w");
        if (dump_f) begin

            $fdisplay(dump_f, "$$$$$$ MEM DUMP $$$$$$");
            $fdisplay(dump_f, "TAG     SET     DATA");
            for (int i = 0; i < MEM_SIZE; i=i+1) begin
                $fdisplay(dump_f, "0x%0H \t0x%0H \t0x%h\n", (i >> 5), i%32,
storage[i]);
            end

            $display("Mem dumped successful. Check mem.dump");
        end else begin
            $display("Error while mem dump");
        end
        $fclose(dump_f);
    endtask


    integer SEED = 225526;
    int j;
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            $display("Filling the MEM...");
            for (int i = 0; i < MEM_SIZE; i=i+1) begin
```

```
                for (j=0; j<CACHE_LINE_SIZE; ++j) begin
                    storage[i][8*j +: 8] = $random(SEED)>>16;
                end
            end
            $display("MEM filled");
            data_buff = 'z;
            command_buff = 'z;
        end else if (dump) begin
            dump_to_console;
        end else begin
            if (command == C2_READ) begin
                command_buff = 'z;
                wait_and_response;

                // READ
                for (int i=0; i<CACHE_LINE_SIZE/2; i=i+1) begin
                    data_buff = storage[address][BUS_SIZE*i +: BUS_SIZE];
                    delay;
                end
                command_buff = 'z;
            end else if (command == C2_WRITE) begin
                data_buff = 'z;
                command_buff = 'z;
                wait_and_response;

                // WRITE
                for (int i=0; i<CACHE_LINE_SIZE/2; i=i+1) begin
                    read_bus_delay;
                    storage[address][BUS_SIZE*i +: BUS_SIZE] = data;
                end
                command_buff = 'z;
            end else begin
                data_buff = 'z;
            end
        end
    end

    assign data = data_buff;
    assign command = command_buff;
endmodule
```

Аналитическое решение на Python.

```
import numpy as np
import math

M = 64
N = 60
K = 32

CACHE_OFFSET_SIZE = 4
CACHE_SET_COUNT = 32
```

```
CACHE_SET_SIZE = int(math.log2(CACHE_SET_COUNT))  # 5
CACHE_LINE_SIZE = 16

aStart = 0
aIntSize = 1
aSize = M*K*aIntSize

bStart = aStart + aSize
bIntSize = 2
bSize = K*N*bIntSize

cStart = bStart + bSize
cIntSize = 4
cSize = M*N*cIntSize


class Cache:

    def __init__(self):
        self.tag_array = np.zeros((CACHE_SET_COUNT, 2), dtype=int)
        self.valid_array = np.zeros((CACHE_SET_COUNT, 2), dtype=bool)
        self.lru_array = np.zeros(CACHE_SET_COUNT, dtype=bool)
        self.reqCount = 0
        self.hitCount = 0

    def req(self, addr: int):
        setNum = Cache.getSet(addr)
        tag = Cache.getTag(addr)
        return self.checkHit(setNum, tag)

    def checkHit(self, setNum: int, tag: int):
        self.reqCount += 1
        for i in range(2):
            if (self.valid_array[setNum, i] and self.tag_array[setNum, i] == tag):
                self.hit(setNum, i)
                return True
        else:
            self.miss(setNum, tag)
            return False

    def hit(self, setNum: int, i: int):
        self.hitCount += 1
        self.lru_array[setNum] = (i == 0)

    def miss(self, setNum: int, tag: int):
        lru_index = int(self.lru_array[setNum])
        self.tag_array[setNum, lru_index] = tag
        self.valid_array[setNum, lru_index] = True
        self.lru_array[setNum] = not self.lru_array[setNum]

    @ staticmethod
    def getTag(address: int) -> int:
        return (address >> (CACHE_OFFSET_SIZE + CACHE_SET_SIZE))
```

```
    @ staticmethod
    def getSet(address: int) -> int:
        return (address >> CACHE_OFFSET_SIZE) % CACHE_SET_COUNT


def simulate(cache):
    pa = aStart
    pc = cStart
    for i in range(M):
        for j in range(N):
            pb = bStart
            for k in range(K):
                cache.req(pa + k*aIntSize)  # a
                cache.req(pb + j*bIntSize)  # b
                pb += N*bIntSize
            cache.req(pc + j*cIntSize)  # c
        pa += K*aIntSize
        pc += N*cIntSize


def validate(reqCount: int, hitCount: int):
    with open("hit_stat.dump", "r") as hit_stat:
        givenReqCount = int(hit_stat.readline())
        givenHitCount = int(hit_stat.readline())

        if (givenReqCount != reqCount or givenHitCount != hitCount):
            raise Exception(
                f'''{cl.OKBLUE}Results missmatch{cl.ENDC}
{cl.OKGREEN}Analytic{cl.ENDC}:   reqs {reqCount}, hits {hitCount}, rate
{round(hitCount/reqCount, 6)}
{cl.WARNING}Simulation{cl.ENDC}: reqs {givenReqCount}, hits {givenHitCount}, rate
{round(givenHitCount/givenReqCount, 6)}\n''')


def main():
    cache = Cache()
    simulate(cache)

    try:
        validate(cache.reqCount, cache.hitCount)
    except Exception as e:
        print("\nData validation" + cl.FAIL + " failed" + cl.ENDC)
        print(e)
    else:
        print("\nData validation" + cl.OKGREEN + " passed" + cl.ENDC)
        print(
            f'{cl.HEADER}{cl.BOLD}Requests{cl.ENDC}:
{cache.reqCount}\n{cl.HEADER}{cl.BOLD}Hits{cl.ENDC}:     {cache.hitCount}')
        print(f'{cl.UNDERLINE}{cl.OKBLUE}HIT RATE{cl.ENDC} is
{cl.WARNING}{round(cache.hitCount/cache.reqCount, 6)}{cl.ENDC}\n')
```

```
if __name__ == '__main__':
    class cl:
        HEADER = '\033[95m'
        OKBLUE = '\033[94m'
        OKCYAN = '\033[96m'
        OKGREEN = '\033[92m'
        WARNING = '\033[93m'
        FAIL = '\033[91m'
        ENDC = '\033[0m'
        BOLD = '\033[1m'
        UNDERLINE = '\033[4m'
    main
```