Лабораторная работа №2	Группа: М3137	2022
Название работы: Моделирование схем в Verilog	ФИО: Кузнецов Сергей Павлович	

Цель работы: построение кэша и моделирование системы “процессор-кэш-память” на языке описания Verilog.

Инструментарий и требования к работе: использовался Icarus Verilog 10

Формулировка задачи из условия: построить, промоделировать и проанализировать модель процессор-кэш-память

Вариант – 2

кэш		
Размер кэша	1 Кб – **CACHE_SIZE**	Размер полезных данных
Размера кэш-линии	32 байта – **CACHE_LINE_SIZE**	Размер полезных данных
Кол-во бит под хранение индекса набора	4 бита – **CACHE_SET_SIZE**	
Количество кэш-линий	32 – **CACHE_LINE_COUNT**	
Ассоциативность	2 – **CACHE_WAY**	
Кол-во наборов кэш-линий	16 - **CACHE_SETS_COUNT**	
Размер тэга адреса	11 - **CACHE_TAG_SIZE**	

Размер смещения	5 – **CACHE_OFFSET_SIZE**	
Размера адреса	20 - **CACHE_ADDR_SIZE**	
Память		
Размер памяти	1 Мбайт – **MEM_SIZE**	

CACHE_LINE_COUNT = CACHE_SIZE / CACHE_LINE_SIZE

CACHE_SETS_COUNT = 2 ^ CACHE_SET_SIZE

CACHE_WAY= CACHE_LINE_COUNT / CACHE_SETS_COUNT

CACHE_ADDR_SIZE = $\log_2$(MEM_SIZE)

CACHE_OFFSET_SIZE = $\log_2$(CACHE_LINE_SIZE)

CACHE_TAG_SIZE = CACHE_ADDR_SIZE - CACHE_OFFSET_SIZE - CACHE_SET_SIZE

Размерность шин

Шина	Обозначение	Размерность
A1, A2	ADDR1_BUS_SIZE, ADDR2_BUS_SIZE	15 бит
D1, D2	DATA1_BUS_SIZE, DATA2_BUS_SIZE	16 бит
C1, C2	CTR1_BUS_SIZE, CTR2_BUS_SIZE	1. 3 бита, 2. 2 бита

ADDR1_BUS_SIZE = max(**CACHE_TAG_SIZE + CACHE_SET_SIZE, CACHE_OFFSET_SIZE**)

ADDR2_BUS_SIZE = max(**CACHE_TAG_SIZE + CACHE_SET_SIZE**)

8 команд CTR1_BUS_SIZE = 3 бит

3 команды CTR2_BUS_SIZE = 2 бит

Аналитическое решение

```
MEM_SIZE = 2 ** 20  # байт
CACHE_SIZE = 2 ** 10  # байт
CACHE_LINE_SIZE = 2 ** 5 # байт
CACHE_LINE_COUNT = 2 ** 5
CACHE_WAY = 2
CACHE_SETS_COUNT = 16
CACHE_TAG_SIZE = 11
CACHE_SET_SIZE = 4
CACHE_OFFSET_SIZE = 5
CACHE_ADDR_SIZE = 20


cache = []
Hits = 0
Miss = 0
all_cnt = 0
for i in range(CACHE_LINE_SIZE):
    cache.append("0" * (CACHE_TAG_SIZE + 3))


def read(adr):
    global Count
    bin_adr = bin(adr)[2:]
    bin_adr = (20 - len(bin_adr)) * "0" + bin_adr
    tag, _set = bin_adr[:CACHE_TAG_SIZE],
bin_adr[CACHE_TAG_SIZE:CACHE_TAG_SIZE + CACHE_SET_SIZE]
    ind = search(tag, _set)
    Count += 2


def write(adr):
    global Count, cache
    bin_adr = bin(adr)[2:]
    bin_adr = (20 - len(bin_adr)) * "0" + bin_adr
    tag, _set = bin_adr[:CACHE_TAG_SIZE],
bin_adr[CACHE_TAG_SIZE:CACHE_TAG_SIZE + CACHE_SET_SIZE]
    num = int(_set, 2) * CACHE_WAY
    ind = search(tag, _set)
    new = list(cache[num + ind])
    new[CACHE_TAG_SIZE + 1] = "1"
    cache[num + ind] = "".join(new)
    Count += 2


def search(tag, _set):
    global Count, cache, all_cnt, Hits, Miss
    num = int(_set, 2) * CACHE_WAY
    line1 = cache[num]
    line2 = cache[num + 1]
    index = 0
    all_cnt += 1
    if line1[:CACHE_TAG_SIZE] == tag and line1[CACHE_TAG_SIZE] == "1":
```

```
        index = 0
        Count += 6
        Hits += 1
    elif line2[:CACHE_TAG_SIZE] == tag and line2[CACHE_TAG_SIZE] == "1":
        index = 1
        Count += 6
        Hits += 1
    else:
        Miss += 1
        Count += 4
        if line1[-1] == "1":
            index = 1
        else:
            index = 0

        if cache[num + index][CACHE_TAG_SIZE] == "1" and cache[num +
index][CACHE_TAG_SIZE + 1] == "1":
            # write
            Count += 108
        # read
        Count += 108
        new = tag + "1" + "0" + "1"
        cache[num + index] = new
    if index == 0:
        new = list(cache[num + 1])
        new[-1] = "0"
        cache[num + 1] = "".join(new)
    else:
        new = list(cache[num])
        new[-1] = "0"
        cache[num] = "".join(new)
    return index


M = 64
N = 60
K = 32
Count = 0
a = 0
b = M * K
c = b + K * N * 2
pa = a
pc = c
for y in range(M):
    for x in range(N):
        pb = b
        s = 0
        Count += 2  # initial pb and s
        for k in range(K):
            #s += read(pa + K) * read(pb + x * 2)
            read(pa + K)
            read(pb + x * 2)
            pb += N * 2
            Count += 5
        write(pc + 4 * x)
        Count += 1
    pa += K
    pc += 4 * N
    Count += 3
Count += 1
print(Count)
```

```
print(all_cnt)
print(Hits / all_cnt)
print(Miss)
```

Итог

```
Tact - 7295789
Cnt - 249600
% - 0.8382532051282051
miss - 40372
```

Ошибка где-то в конкатенации строк, логика работы идентична с логикой в моделе verilog

Моделирование заданной системы на Verilog

Кэш – look-through. В такой системе процессор и память не связаны, процессор подает запрос в кэш, если ответ найден, то кэш его отправляет, иначе подает запрос в память. Write-back – мы пишем измененные данные в кэш, а когда нужно выкинуть эту линию, то записываем её в память. Недостаток write-back при reset кэша, измененные данные будут утеряны. Но так как у нас моделирование, такой проблемы не будет. Политика вымещения LRU – вытесняется последний измененный из блока. Передача данных происходит начиная с младших бит, little endian

Основные элементы системы кэш, процессор, память связаны между собой наборами проводов. Процессор подает команды в кэш:

- READ8/16/32 – прочитать 8/16/32 бит по адресу
- WRITE8/16/32 – записать 8/16/32 бит по адресу
- INVALIDATE_LINE – выполнить инвалидацию кэш-линии с адресом

```
task read_8(input reg[CACHE_ADDR_SIZE - 1:0] addr, output reg[7:0] data);
    _C1 = 1;
    _A1 = addr[CACHE_ADDR_SIZE - CACHE_OFFSET_SIZE - 1:CACHE_OFFSET_SIZE]; // tag+set
    wait_cnt(1);
    _A1 = addr[CACHE_OFFSET_SIZE - 1:0]; // offset
    _C1 = 'bz;
    wait(C1 === 7);
    data = D1[15:8];
    wait_cnt(1);
endtask
```

Рисунок 1 – метод чтения из памяти 8 бит

Функция передает кэшу по каналу A1 tag+set за первый такт и offset за второй. И ожидает, когда кэш пришлет ответ на запрос. Для этого на шину C1 подается команда 1. После C1 назначается равным Z (высокоимпедансное состояние) и процессор ожидает ответ от кэша. Wait_cnt(int number) количество тактов ожидания

```
task wait_cnt(int cnt);
    for (j = 0; j < cnt; j += 1) begin
        @(posedge clk);
    end
endtask
```

Рисунок 2 – wait_cnt(), присутствует во всех модулях

Данные кэш-линии и их дополнительны данные хранятся отдельно. Cache_data – двумерный массив байт, cache_info – массив, в котором в том же порядке хранятся дополнительные данные (valid + dirty + tag + lru), блок это 2 последовательные линии (way = 2).

```
reg[7:0] cache_data[0:CACHE_LINE_COUNT - 1] [0:CACHE_LINE_SIZE - 1];
reg[CACHE_TAG_SIZE + 3 - 1:0] cache_info[0:CACHE_LINE_COUNT - 1]; // valid 1 dirty 1 tag 11 lru 1
```

Рисунок 3 – массивы, в которых хранятся данные кэш-линий

Кэш в свою обрабатывает поданную команду процессора. Это происходит при помощи блока always.

```
always @(posedge clk) begin
    if (C1 == 1) begin
        tag = A1[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE];
        set = A1[CACHE_SET_SIZE - 1:0];
        wait_cnt(1);
        offset = A1[CACHE_OFFSET_SIZE - 1:0];
        wait_cnt(1);
        search_cache_line(tag, set, index);
        _C1 = 7;
        _D1[15:8] = cache_data[set * CACHE_WAY + index][offset];
        wait_cnt(1);
        _C1 = 'dz;
        _D1 = 'dz;
    end
```

Рисунок 4 – always блок кэша

В первый такт кэш принимает по шине A1 tagset, во второй offset. После идет поиск кэш линии – метод search_cache_line(), где tag и set является входными параметрами, а index – выходным. Index – номер кэш-линии в блоке. Если кэш попадание, то передаем на выход нужный индекс. Если кэш промах, то выбираем линию по lru (обращение к которой было раньше всех), если выбранная линия валидная и хранит какие-то измененные значения (valid = 1 и dirty = 1), то передаем эту линию на запись в память – метод write_line_in_mem. После данные вымещенной кэш-линии заменяется текущей, и её данные считываются из памяти с помощью метода read_line_in_mem.

```
task search_cache_line(input reg[CACHE_TAG_SIZE - 1:0] _tag, input reg[CACHE_SET_SIZE - 1:0] _set, output reg index);
    line_info1 = cache_info[_set * CACHE_WAY];
    line_info2 = cache_info[_set * CACHE_WAY + 1];
    count += 1;
    if (line_info1[CACHE_TAG_SIZE + 3 - 1] == 1 && line_info1[CACHE_TAG_SIZE:1] == _tag) begin
        wait_cnt(6); // add 6
        hits += 1;
        index = 0;
    end
    else if (line_info2[CACHE_TAG_SIZE + 3 - 1] == 1 && line_info2[CACHE_TAG_SIZE:1] == _tag) begin
        wait_cnt(6); // add 6
        index = 1;
        hits += 1;
    end
    else begin
        miss += 1;
        wait_cnt(4);
        // read mem
        if (line_info2[0] == 1) begin
            index = 0;
        end else begin
            index = 1;
        end

        if (cache_info[_set * CACHE_WAY + index][CACHE_TAG_SIZE + 3 - 1] == 1
        && cache_info[_set * CACHE_WAY + index][CACHE_TAG_SIZE + 2 - 1] == 1) begin
            write_line_in_mem(cache_info[_set * CACHE_WAY + index][CACHE_TAG_SIZE:1], _set, index);
        end

        wait_cnt(1);
        read_line_in_mem(_tag, _set, index);
        cache_info[_set * CACHE_WAY + index][CACHE_TAG_SIZE + 3 - 1] = 1;
        cache_info[_set * CACHE_WAY + index][CACHE_TAG_SIZE + 2 - 1] = 0;
        cache_info[_set * CACHE_WAY + index][CACHE_TAG_SIZE:1] = _tag;
    end
    if (index == 0) begin
        cache_info[_set * CACHE_WAY][0] = 1;
        cache_info[_set * CACHE_WAY + 1][0] = 0;
    end else begin
        cache_info[_set * CACHE_WAY][0] = 0;
        cache_info[_set * CACHE_WAY + 1][0] = 1;
    end

endtask
```

Рисунок 5 – метод search_cache_line в кэше

Метод write_line_in_mem – принимает на вход tag, set и index и подает команду памяти на запись кэш линии. По шинам передаются значения C2 = 3, A2 = tag+set, D2 = данные кэш-линии. После ждет ответ от памяти об записи данных.

```
task write_line_in_mem(input reg[CACHE_TAG_SIZE - 1:0] _tag, input reg[CACHE_SET_SIZE - 1:0] _set, input reg index);
    _C2 = 3;
    _A2[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE] = _tag;
    _A2[CACHE_SET_SIZE - 1:0] = _set;
    for (iter = 0; iter < CACHE_LINE_SIZE - 1; iter += 2) begin
        _D2[15:8] = cache_data[_set * CACHE_WAY + index][CACHE_LINE_SIZE - iter - 1];
        _D2[7:0] = cache_data[_set * CACHE_WAY + index][CACHE_LINE_SIZE - iter - 2];
        wait_cnt(1);
    end
    _C2 = 'dz;
    _D2 = 'dz;
    wait (C2 == 1);
endtask
```

Рисунок 6 – метод write_line_in_mem

Метод read_line_in_mem – имеет схожую структуру с write_line_in_mem, но он получает данные от памяти. По шинам передается C2 = 2, A2 = tag+set. После метод ожидает ответ памяти и записывает полученные данные в выбранную кэш линию.

```
task read_line_in_mem(input reg[CACHE_TAG_SIZE - 1:0] _tag, input reg[CACHE_SET_SIZE - 1:0] _set, input reg index);
    _C2 = 2;
    _A2[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE] = _tag;
    _A2[CACHE_SET_SIZE - 1:0] = _set;
    wait_cnt(1);
    _C2 = 'dz;
    wait(C2 == 1);
    for (iter = 0; iter < CACHE_LINE_SIZE - 1; iter += 2) begin
        cache_data[_set * CACHE_WAY + index][CACHE_LINE_SIZE - iter - 1] = D2[15:8];
        cache_data[_set * CACHE_WAY + index][CACHE_LINE_SIZE - iter - 2] = D2[7:0];
        wait_cnt(1);
    end
endtask
```

Рисунок 7 – метод write_read_in_mem

Память обрабатывает запросы при помощи блока always, по шине A2 получает адрес и команду по шине C2, если команда на запись также получает данные линии по шине D2, каждый такт по 16 бит. Через 100 тактов память отправляет ответ кэшу.

```
always @(posedge clk) begin
    if (C2 === 2) begin
        tag = A2[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE];
        set = A2[CACHE_SET_SIZE - 1:0];
        wait_cnt(100); ///<100>
        _C2 = 1;
        addr[CACHE_ADDR_SIZE - 1:CACHE_OFFSET_SIZE] = A2;
        addr[CACHE_OFFSET_SIZE - 1:0] = (1 << CACHE_OFFSET_SIZE) - 1;
        for (i = 0; i < CACHE_LINE_SIZE; i+= 2) begin
            _D2[15:8] =  mem[addr - i];
            _D2[7:0] =  mem[addr - i - 1];
            wait_cnt(1);
        end
        _D2 = 'dz;
        _C2 = 'dz;
    end

    else if (C2 === 3) begin
        tag = A2[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE];
        set = A2[CACHE_SET_SIZE - 1:0];
        addr[CACHE_ADDR_SIZE - 1:CACHE_OFFSET_SIZE] = A2;
        addr[CACHE_OFFSET_SIZE - 1:0] = (1 << CACHE_OFFSET_SIZE) - 1;
        for (i = 0; i < CACHE_LINE_SIZE; i+= 2) begin
            mem[addr - i] = D2[15:8];
            mem[addr - i - 1] = D2[7:0];
            wait_cnt(1);
        end
        wait_cnt(100); ///<100>
        _C2 = 1;
        wait_cnt(1);
        _C2 = 'dz;
    end
end
```

Рисунок 8 – блок always в памяти

Выше были рассмотрена команда, читающая из кэша, рассмотрим команду записывающую в кэш. По шинам передаются значения C1 = 5, A1 = addr, D1 = данные. После процессор ждет ответ от кэша о выполнение команды.

```
task write_8(input reg[CACHE_ADDR_SIZE - 1:0] addr, input reg[7:0] data);
    _C1 = 5;
    _A1 = addr[CACHE_ADDR_SIZE - CACHE_OFFSET_SIZE - 1:CACHE_OFFSET_SIZE]; // tag+set
    _D1[15:8] = data;
    wait_cnt(1);
    _A1 = addr[CACHE_OFFSET_SIZE - 1:0]; // offset
    wait_cnt(1);
    _C1 = 'bz;
    _D1 = 'bz;
    wait(C1 == 7);
    wait_cnt(1);
endtask
```

Рисунок 9 – метод записи 8 бит

Кэш принимает tag+set за первый такт и данные на запись, а во второй offset. После ищет кэш-линию с помощью метода search_cache_line описанным выше. Изменяет данные и меняет dirty = 1, так как теперь данные кэш-линии не соответствуют данным в памяти

```
else if (C1 == 5) begin
    tag = A1[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE];
    set = A1[CACHE_SET_SIZE - 1:0];
    data8[7:0] = D1[15:8];
    wait_cnt(1);
    offset = A1[CACHE_OFFSET_SIZE - 1:0];
    wait_cnt(1);
    search_cache_line(tag, set, index);
    cache_info[set * CACHE_WAY + index][CACHE_TAG_SIZE + 2 - 1] = 1;
    cache_data[set * CACHE_WAY + index][offset] = data8;
    _C1 = 7;
    wait_cnt(1);
    _C1 = 'dz;
end
```

Рисунок 10 – обработка команды записи 8 бит

Команды Read8, Read16, Read32 отличаются только количеством передаваемых данных, но логикой не отличаются. Так же и команды Write8, Write16, Write32.

Оставшийся не рассмотренный метод invalidate_line, который вытесняет кэш-линию. Он подает на вход команду C1 = 4 и адрес через A1.

```
task invalidate_line(input reg[CACHE_ADDR_SIZE - 1:0] addr);
    _C1 = 4;
    _A1 = addr[CACHE_ADDR_SIZE - CACHE_OFFSET_SIZE - 1:CACHE_OFFSET_SIZE]; // tag+set
    wait_cnt(1);
    _A1 = addr[CACHE_OFFSET_SIZE - 1:0]; // offset
    wait_cnt(1);
    _C1 = 'bz;
    wait(C1 == 7);
    wait_cnt(1);
endtask
```

Рисунок 11 – метод инвалидации кэш линии

Кэш обрабатывает эту команду, так же как и другие при помощи always. И вызывается метод invalid_cache_line, которому на вход подаются tag и set. После выполнения метода кэш отсылает ответ процессору

```
else if (C1 == 4) begin
    tag = A1[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE];
    set = A1[CACHE_SET_SIZE - 1:0];
    wait_cnt(1);
    offset = A1[CACHE_OFFSET_SIZE - 1:0];
    wait_cnt(1);
    invalid_cache_line(tag, set);
    _C1 = 7;
    wait_cnt(1);
    _C1 = 'dz;
end
```

Рисунок 12 – обработка инвалидации в кэше

Если данная кэш-линия изменена (dirty = 1), то записываем её в память. После кэш-линия становится valid = 0 и становится линией использовавшейся раньше.

```
task invalid_cache_line(input reg[CACHE_TAG_SIZE - 1:0] _tag, input reg[CACHE_SET_SIZE - 1:0] _set);
    line_info1 = cache_info[_set * CACHE_WAY];
    line_info2 = cache_info[_set * CACHE_WAY + 1];
    if (line_info1[CACHE_TAG_SIZE + 2 - 1] == 1 && line_info1[CACHE_TAG_SIZE - 1:1] == tag
        && line_info1[CACHE_TAG_SIZE] == 1) begin
        write_line_in_mem(_tag, _set, 0);
    end
    else if (line_info2[CACHE_TAG_SIZE + 2 - 1] == 1 && line_info2[CACHE_TAG_SIZE - 1:1] == tag) begin
        write_line_in_mem(_tag, _set, 1);
    end
endtask
```

Рисунок 13 – метод invalid_cache_line

Модулируемая задача выглядит так

```
task mmul;
    addr_a = 0;
    addr_b = M * K;
    addr_c = addr_b + K * N * 2; // b[x] - 2 bytes
    $display("Start - %t", $time);
    for (y = 0; y < M; y += 1) begin
        for (x = 0; x < N; x+=1) begin
            addr_b_now = addr_b;
            s = 0;
            wait_cnt(1);
            for (k = 0; k < K; k+=1) begin
                read_8(addr_a + k, pa);
                read_16(addr_b_now + x * 2, pb);
                s += pa * pb;
                wait_cnt(5 + 1); // * - 5, + - 1;
                addr_b_now += 2 * N;
                wait_cnt(1);
            end
            write_32(addr_c + x * 4, s);
            wait_cnt(1);
        end
        addr_a += K;
        wait_cnt(1);
        addr_c += N * 4;
        wait_cnt(1);

        wait_cnt(1);
    end
    wait_cnt(1);
    $display("End - %t", $time);
    $finish;
endtask
```

Рисунок 14 – моделируемая задача

```
Initial memory
Start -                   10
    |   |   85
Count -      249600, Miss -       35839, Hits -      213761
End -             15394112
```

Рисунок 15 – итоговый лог

Tact = (15394112 – 10) / 2 = 7697051

```
initial begin
    _c_dump = 'dz;
    #15394112
    _c_dump = 1;
    #10
    $finish;
end
```

Рисунок 16 – моделируемый вывод в testbench

Сравнение результатов

Результаты отличаются на небольшое значение из-за ошибки в аналитическом решении.

Листинг Кода

CPU.sv

```
module CPU #(
    parameter MEM_SIZE = (1 << 20),
    parameter CACHE_SIZE = (1 << 10),
    parameter CACHE_LINE_SIZE = 32,
    parameter CACHE_LINE_COUNT = 32,
    parameter CACHE_WAY = 2,
    parameter CACHE_SETS_COUNT = 16,
    parameter CACHE_TAG_SIZE = 11,
    parameter CACHE_SET_SIZE = 4,
    parameter CACHE_OFFSET_SIZE = 5,
    parameter CACHE_ADDR_SIZE = 20,
    parameter ADDR1_BUS_SIZE = 15,
    parameter ADDR2_BUS_SIZE = 15,
    parameter DATA1_BUS_SIZE = 16,
    parameter DATA2_BUS_SIZE = 16,
    parameter CTR1_BUS_SIZE = 3,
    parameter CTR2_BUS_SIZE = 2
)

(
    output wire [ADDR1_BUS_SIZE - 1:0] A1,
    inout wire [DATA1_BUS_SIZE - 1:0] D1,
    inout wire [CTR1_BUS_SIZE - 1:0] C1,

    input clk,
    input reset
);
    reg[ADDR1_BUS_SIZE - 1:0] _A1;
    reg[DATA1_BUS_SIZE - 1:0] _D1;
    reg[CTR1_BUS_SIZE - 1:0] _C1;
```

```
    reg[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:0] test;
    reg[CACHE_OFFSET_SIZE - 1:0] test2;
    reg[31:0] now;
    reg[7:0] dtest;
    reg[15:0] dtest16;
    integer j = 0;

    assign A1 = _A1;
    assign D1 = _D1;
    assign C1 = _C1;

    // const
    reg[6:0] M = 64;
    reg[5:0] N = 60;
    reg[5:0] K = 32;
    // адреса массивов
    reg[CACHE_ADDR_SIZE - 1:0] addr_a = 0;
    reg[CACHE_ADDR_SIZE - 1:0] addr_b;
    reg[CACHE_ADDR_SIZE - 1:0] addr_c;

    reg[CACHE_ADDR_SIZE - 1:0] addr_b_now;

    int y = 0;
    int k = 0;
    int x = 0;
    int s;

    reg[7:0] pa;
    reg[15:0] pb;
    reg[31:0] pc;

    reg[7:0] testnew;

    initial begin
        _C1 = 'bz;
        _D1 = 'bz;
        _A1 = 'bz;

        #10
        mmul();
    end

    task read_8(input reg[CACHE_ADDR_SIZE - 1:0] addr, output reg[7:0] data);
        _C1 = 1;
        _A1 = addr[CACHE_ADDR_SIZE - CACHE_OFFSET_SIZE - 1:CACHE_OFFSET_SIZE]; //
tag+set
        wait_cnt(1);
        _A1 = addr[CACHE_OFFSET_SIZE - 1:0]; // offset
        wait_cnt(1);
        _C1 = 'bz;
        wait(C1 === 7);
        data = D1[15:8];
        wait_cnt(1);
    endtask

    task read_16(input reg[CACHE_ADDR_SIZE - 1:0] addr, output reg[15:0] data);
```

```
        _C1 = 2;
        _A1 = addr[CACHE_ADDR_SIZE - CACHE_OFFSET_SIZE - 1:CACHE_OFFSET_SIZE]; //
tag+set
        wait_cnt(1);
        _A1 = addr[CACHE_OFFSET_SIZE - 1:0]; // offset
        wait_cnt(1);
        _C1 = 'bz;
        wait(C1 === 7);
        data[7:0] = D1[15:8];
        data[15:8] = D1[7:0];
        //$display("Data rez %d", data);
        wait_cnt(1);
    endtask

    task read_32(input reg[CACHE_ADDR_SIZE - 1:0] addr, output reg[31:0] data);
        _C1 = 3;
        _A1 = addr[CACHE_ADDR_SIZE - CACHE_OFFSET_SIZE - 1:CACHE_OFFSET_SIZE]; //
tag+set
        wait_cnt(1);
        _A1 = addr[CACHE_OFFSET_SIZE - 1:0]; // offset
        wait_cnt(1);
        _C1 = 'bz;
        wait(C1 === 7);
        data[7:0] = D1[15:8];
        data[15:8] = D1[7:0];
        wait_cnt(1);
        data[23:16] = D1[15:8];
        data[31:24] = D1[7:0];
        wait_cnt(1);
    endtask

    task invalidate_line(input reg[CACHE_ADDR_SIZE - 1:0] addr);
        _C1 = 4;
        _A1 = addr[CACHE_ADDR_SIZE - CACHE_OFFSET_SIZE - 1:CACHE_OFFSET_SIZE]; //
tag+set
        wait_cnt(1);
        _A1 = addr[CACHE_OFFSET_SIZE - 1:0]; // offset
        wait_cnt(1);
        _C1 = 'bz;
        wait(C1 == 7);
        wait_cnt(1);
    endtask

    task write_8(input reg[CACHE_ADDR_SIZE - 1:0] addr, input reg[7:0] data);
        _C1 = 5;
        _A1 = addr[CACHE_ADDR_SIZE - CACHE_OFFSET_SIZE - 1:CACHE_OFFSET_SIZE]; //
tag+set
        _D1[15:8] = data;
        wait_cnt(1);
        _A1 = addr[CACHE_OFFSET_SIZE - 1:0]; // offset
        wait_cnt(1);
        _C1 = 'bz;
        _D1 = 'bz;
        wait(C1 == 7);
        wait_cnt(1);
    endtask
```

```
    task write_16(input reg[CACHE_ADDR_SIZE - 1:0] addr, input reg[15:0] data);
        _C1 = 6;
        _A1 = addr[CACHE_ADDR_SIZE - CACHE_OFFSET_SIZE - 1:CACHE_OFFSET_SIZE]; //
tag+set
        _D1[15:8] = data[7:0];
        _D1[7:0] = data[15:8];
        wait_cnt(1);
        _A1 = addr[CACHE_OFFSET_SIZE - 1:0]; // offset
        wait_cnt(1);
        _C1 = 'bz;
        _D1 = 'bz;
        wait(C1 == 7);
        wait_cnt(1);
    endtask

    task write_32(input reg[CACHE_ADDR_SIZE - 1:0] addr, input reg[31:0] data);
        _C1 = 7;
        _A1 = addr[CACHE_ADDR_SIZE - CACHE_OFFSET_SIZE - 1:CACHE_OFFSET_SIZE]; //
tag+set
        _D1[15:8] = data[7:0];
        _D1[7:0] = data[15:8];
        wait_cnt(1);
        _A1 = addr[CACHE_OFFSET_SIZE - 1:0]; // offset
        _D1[15:8] = data[23:16];
        _D1[7:0] = data[31:24];
        wait_cnt(1);
        _C1 = 'bz;
        _D1 = 'bz;
        wait(C1 == 7);
        wait_cnt(1);
    endtask

    task wait_cnt(int cnt);
        for (j = 0; j < cnt; j += 1) begin
            @(posedge clk);
        end
    endtask


    task mmul;
        addr_a = 0;
        addr_b = M * K;
        addr_c = addr_b + K * N * 2; // b[x] - 2 bytes
        $display("Start - %t", $time);
        for (y = 0; y < M; y += 1) begin
            for (x = 0; x < N; x+=1) begin
                addr_b_now = addr_b;
                s = 0;
                wait_cnt(1);
                for (k = 0; k < K; k+=1) begin
                    read_8(addr_a + k, pa);
                    read_16(addr_b_now + x * 2, pb);
                    s += pa * pb;
                    wait_cnt(5 + 1); // * - 5, + - 1;
                    addr_b_now += 2 * N;
                    wait_cnt(1);
                end
```

```
                write_32(addr_c + x * 4, s);
                wait_cnt(1);
            end
            addr_a += K;
            wait_cnt(1);
            addr_c += N * 4;
            wait_cnt(1);

            wait_cnt(1);
        end
        wait_cnt(1);
        $display("End - %t", $time);
        //$finish;
    endtask

endmodule
```

testbench.sv

```
`include "CPU.sv"
`include "Cache.sv"
`include "MemCTR.sv"

module testbench # (
    parameter MEM_SIZE = (1 << 20),
    parameter CACHE_SIZE = (1 << 10),
    parameter CACHE_LINE_SIZE = 32,
    parameter CACHE_LINE_COUNT = 32,
    parameter CACHE_WAY = 2,
    parameter CACHE_SETS_COUNT = 16,
    parameter CACHE_TAG_SIZE = 11,
    parameter CACHE_SET_SIZE = 4,
    parameter CACHE_OFFSET_SIZE = 5,
    parameter CACHE_ADDR_SIZE = 20,
    parameter ADDR1_BUS_SIZE = 15,
    parameter ADDR2_BUS_SIZE = 15,
    parameter DATA1_BUS_SIZE = 16,
    parameter DATA2_BUS_SIZE = 16,
    parameter CTR1_BUS_SIZE = 3,
    parameter CTR2_BUS_SIZE = 2
);
    reg clk = 1;

    wire[ADDR1_BUS_SIZE - 1:0] A1;
    wire[DATA1_BUS_SIZE - 1:0] D1;
    wire[CTR1_BUS_SIZE - 1:0] C1;

    wire[ADDR2_BUS_SIZE - 1:0] A2;
    wire[DATA2_BUS_SIZE - 1:0] D2;
    wire[CTR2_BUS_SIZE - 1:0] C2;

    wire reset = 0;
    wire C_DUMP = 0;
    wire M_DUMP = 0;

    reg _c_dump;
```

```
    reg _m_dump;

    assign C_DUMP = _c_dump;
    assign M_DUMP = _m_dump;

    CPU _cpu(
        .A1(A1),
        .D1(D1),
        .C1(C1),
        .clk(clk),
        .reset(reset)
        );

    Cache _cache (
        .A2(A2),
        .D1(D1),
        .C1(C1),
        .D2(D2),
        .C2(C2),
        .A1(A1),
        .clk(clk),
        .reset(reset),
        .C_DUMP(C_DUMP)
    );

    MemCTR _memctr(
        .D2(D2),
        .C2(C2),
        .A2(A2),
        .clk(clk),
        .reset(reset),
        .M_DUMP(M_DUMP)
    );



    initial begin
        _c_dump = 'dz;
        #15394112
        _c_dump = 1;
        #10
        $finish;
    end

    always #1 begin
        clk = ~clk;
    end

endmodule
```

Cache.sv

```
module Cache #(
    parameter MEM_SIZE = (1 << 20),
    parameter CACHE_SIZE = (1 << 10),
    parameter CACHE_LINE_SIZE = 32,
```

```
    parameter CACHE_LINE_COUNT = 32,
    parameter CACHE_WAY = 2,
    parameter CACHE_SETS_COUNT = 16,
    parameter CACHE_TAG_SIZE = 11,
    parameter CACHE_SET_SIZE = 4,
    parameter CACHE_OFFSET_SIZE = 5,
    parameter CACHE_ADDR_SIZE = 20,
    parameter ADDR1_BUS_SIZE = 15,
    parameter ADDR2_BUS_SIZE = 15,
    parameter DATA1_BUS_SIZE = 16,
    parameter DATA2_BUS_SIZE = 16,
    parameter CTR1_BUS_SIZE = 3,
    parameter CTR2_BUS_SIZE = 2

)

(
    output wire [ADDR2_BUS_SIZE - 1:0] A2,
    inout wire [DATA1_BUS_SIZE - 1:0] D1,
    inout wire [CTR1_BUS_SIZE - 1:0] C1,
    inout wire [DATA2_BUS_SIZE - 1:0] D2,
    inout wire [CTR2_BUS_SIZE - 1:0] C2,

    input wire [ADDR1_BUS_SIZE - 1:0] A1,
    input clk,
    input reset,
    input C_DUMP

);

    reg[ADDR2_BUS_SIZE - 1:0] _A2;
    reg[DATA2_BUS_SIZE - 1:0] _D2;
    reg[CTR2_BUS_SIZE - 1:0] _C2;

    reg[ADDR1_BUS_SIZE - 1:0] _A1;
    reg[DATA1_BUS_SIZE - 1:0] _D1;
    reg[CTR1_BUS_SIZE - 1:0] _C1;

    reg[7:0] cache_data[0:CACHE_LINE_COUNT - 1] [0:CACHE_LINE_SIZE - 1];
    reg[CACHE_TAG_SIZE + 3 - 1:0] cache_info[0:CACHE_LINE_COUNT - 1]; // valid 1
dirty 1 tag 11 lru 1

    assign D1 = _D1;
    assign C1 = _C1;
    assign A2 = _A2;
    assign D2 = _D2;
    assign C2 = _C2;



    reg[7:0] line[0:CACHE_LINE_SIZE - 1];
    integer iter = 0;
    integer j = 0;
    integer i = 0;
    integer dm = 0;
    integer bt = 0;
    integer cl = 0;
```

```
    reg[CACHE_TAG_SIZE - 1:0] tag;
    reg[CACHE_SET_SIZE - 1:0] set;
    reg[CACHE_OFFSET_SIZE - 1:0] offset;

    reg[7:0] line1[0:CACHE_LINE_SIZE - 1];
    reg[7:0] line2[0:CACHE_LINE_SIZE - 1];
    reg[CACHE_TAG_SIZE + 3 - 1:0] line_info1;
    reg[CACHE_TAG_SIZE + 3 - 1:0] line_info2;
    reg[7:0] data8;
    reg[15:0] data16;
    reg[31:0] data32;

    reg index = 0;
    int count = 0;
    int miss = 0;
    int hits = 0;

    initial begin
        _D1 = 'bz;
        _C1 = 'bz;
        _A2 = 'bz;
        _D2 = 'bz;
        _C2 = 'bz;

        // for (iter = 0; iter < CACHE_LINE_COUNT; iter += 1) begin
        //     for (i = 0; i < CACHE_LINE_SIZE; i += 1) begin
        //         cache_data[iter][i] = 0;
        //     end
        // end

        // for (iter = 0; iter < CACHE_LINE_COUNT; iter += 1) begin
        //     cache_info[iter] = 0;
        // end
        reser_cache();
    end

    task wait_cnt(int cnt);
        for (j = 0; j < cnt; j += 1) begin
            @(posedge clk);
        end
    endtask


    always @(posedge clk) begin
        if (C1 == 1) begin
            tag = A1[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE];
            set = A1[CACHE_SET_SIZE - 1:0];
            wait_cnt(1);
            offset = A1[CACHE_OFFSET_SIZE - 1:0];
            wait_cnt(1);
            search_cache_line(tag, set, index);
            _C1 = 7;
            _D1[15:8] = cache_data[set * CACHE_WAY + index][offset];
            wait_cnt(1);
            _C1 = 'dz;
            _D1 = 'dz;
```

```
        end

        else if (C1 == 2) begin
            tag = A1[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE];
            set = A1[CACHE_SET_SIZE - 1:0];
            wait_cnt(1);
            offset = A1[CACHE_OFFSET_SIZE - 1:0];
            wait_cnt(1);
            search_cache_line(tag, set, index);
            _C1 = 7;
            _D1[15:8] = cache_data[set * CACHE_WAY + index][offset];
            _D1[7:0] = cache_data[set * CACHE_WAY + index][offset + 1];
            wait_cnt(1);
            _C1 = 'dz;
            _D1 = 'dz;
        end

        else if (C1 == 3) begin
            tag = A1[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE];
            set = A1[CACHE_SET_SIZE - 1:0];
            wait_cnt(1);
            offset = A1[CACHE_OFFSET_SIZE - 1:0];
            wait_cnt(1);
            search_cache_line(tag, set, index);
            _C1 = 7;
            _D1[15:8] = cache_data[set * CACHE_WAY + index][offset];
            _D1[7:0] = cache_data[set * CACHE_WAY + index][offset + 1];
            wait_cnt(1);
            _D1[15:8] = cache_data[set * CACHE_WAY + index][offset + 2];
            _D1[7:0] = cache_data[set * CACHE_WAY + index][offset + 3];
            wait_cnt(1);
            _C1 = 'dz;
            _D1 = 'dz;
        end

        else if (C1 == 4) begin
            tag = A1[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE];
            set = A1[CACHE_SET_SIZE - 1:0];
            wait_cnt(1);
            offset = A1[CACHE_OFFSET_SIZE - 1:0];
            wait_cnt(1);
            invalid_cache_line(tag, set);
            _C1 = 7;
            wait_cnt(1);
            _C1 = 'dz;
        end

        else if (C1 == 5) begin
            tag = A1[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE];
            set = A1[CACHE_SET_SIZE - 1:0];
            data8[7:0] = D1[15:8];
            wait_cnt(1);
            offset = A1[CACHE_OFFSET_SIZE - 1:0];
            wait_cnt(1);
            search_cache_line(tag, set, index);
            cache_info[set * CACHE_WAY + index][CACHE_TAG_SIZE + 2 - 1] = 1;
            cache_data[set * CACHE_WAY + index][offset] = data8;
```

```
            _C1 = 7;
            wait_cnt(1);
            _C1 = 'dz;
        end

        else if (C1 == 6) begin
            tag = A1[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE];
            set = A1[CACHE_SET_SIZE - 1:0];
            data16[7:0] = D1[15:8];
            data16[15:8] = D1[7:0];
            wait_cnt(1);
            offset = A1[CACHE_OFFSET_SIZE - 1:0];
            wait_cnt(1);
            search_cache_line(tag, set, index);
            cache_info[set * CACHE_WAY + index][CACHE_TAG_SIZE + 2 - 1] = 1;
            cache_data[set * CACHE_WAY + index][offset] = data16[7:0];
            cache_data[set * CACHE_WAY + index][offset + 1] = data16[15:8];
            _C1 = 7;
            wait_cnt(1);
            _C1 = 'dz;
        end

        else if (C1 == 7) begin
            tag = A1[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE];
            set = A1[CACHE_SET_SIZE - 1:0];
            data32[7:0] = D1[15:8];
            data32[15:8] = D1[7:0];
            wait_cnt(1);
            offset = A1[CACHE_OFFSET_SIZE - 1:0];
            data32[23:16] = D1[15:8];
            data32[31:24] = D1[7:0];
            wait_cnt(1);
            search_cache_line(tag, set, index);
            cache_info[set * CACHE_WAY + index][CACHE_TAG_SIZE + 2 - 1] = 1;
            cache_data[set * CACHE_WAY + index][offset] = data32[7:0];
            cache_data[set * CACHE_WAY + index][offset + 1] = data32[15:8];
            cache_data[set * CACHE_WAY + index][offset + 2] = data32[23:16];
            cache_data[set * CACHE_WAY + index][offset + 3] = data32[31:24];
            _C1 = 7;
            wait_cnt(1);
            _C1 = 'dz;
        end
    end

    task search_cache_line(input reg[CACHE_TAG_SIZE - 1:0] _tag, input
reg[CACHE_SET_SIZE - 1:0] _set, output reg index);
        line_info1 = cache_info[_set * CACHE_WAY];
        line_info2 = cache_info[_set * CACHE_WAY + 1];
        count += 1;
        if (line_info1[CACHE_TAG_SIZE + 3 - 1] == 1 &&
line_info1[CACHE_TAG_SIZE:1] == _tag) begin
            wait_cnt(5); // add 6
            hits += 1;
            index = 0;
        end
        else if (line_info2[CACHE_TAG_SIZE + 3 - 1] == 1 &&
line_info2[CACHE_TAG_SIZE:1] == _tag) begin
```

```
            wait_cnt(5); // add 6
            index = 1;
            hits += 1;
        end
        else begin
            miss += 1;
            wait_cnt(3);
            // read mem
            if (line_info2[0] == 1) begin
                index = 0;
            end else begin
                index = 1;
            end

            if (cache_info[_set * CACHE_WAY + index][CACHE_TAG_SIZE + 3 - 1] == 1
                && cache_info[_set * CACHE_WAY + index][CACHE_TAG_SIZE + 2 - 1] ==
1) begin
                write_line_in_mem(cache_info[_set * CACHE_WAY +
index][CACHE_TAG_SIZE:1], _set, index);
            end

            wait_cnt(1);
            read_line_in_mem(_tag, _set, index);
            cache_info[_set * CACHE_WAY + index][CACHE_TAG_SIZE + 3 - 1] = 1;
            cache_info[_set * CACHE_WAY + index][CACHE_TAG_SIZE + 2 - 1] = 0;
            cache_info[_set * CACHE_WAY + index][CACHE_TAG_SIZE:1] = _tag;
        end
        if (index == 0) begin
            cache_info[_set * CACHE_WAY][0] = 1;
            cache_info[_set * CACHE_WAY + 1][0] = 0;
        end else begin
            cache_info[_set * CACHE_WAY][0] = 0;
            cache_info[_set * CACHE_WAY + 1][0] = 1;
        end

    endtask

    task read_line_in_mem(input reg[CACHE_TAG_SIZE - 1:0] _tag, input
reg[CACHE_SET_SIZE - 1:0] _set, input reg index);
        _C2 = 2;
        _A2[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE] = _tag;
        _A2[CACHE_SET_SIZE - 1:0] = _set;
        wait_cnt(1);
        _C2 = 'dz;
        wait(C2 == 1);
        for (iter = 0; iter < CACHE_LINE_SIZE - 1; iter += 2) begin
            cache_data[_set * CACHE_WAY + index][CACHE_LINE_SIZE - iter - 1] =
D2[15:8];
            cache_data[_set * CACHE_WAY + index][CACHE_LINE_SIZE - iter - 2] =
D2[7:0];
            wait_cnt(1);
        end
    endtask

    task write_line_in_mem(input reg[CACHE_TAG_SIZE - 1:0] _tag, input
reg[CACHE_SET_SIZE - 1:0] _set, input reg index);
        _C2 = 3;
```

```
        _A2[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE] = _tag;
        _A2[CACHE_SET_SIZE - 1:0] = _set;
        for (iter = 0; iter < CACHE_LINE_SIZE - 1; iter += 2) begin
            _D2[15:8] = cache_data[_set * CACHE_WAY + index][CACHE_LINE_SIZE -
iter - 1];
            _D2[7:0] = cache_data[_set * CACHE_WAY + index][CACHE_LINE_SIZE - iter
- 2];
            wait_cnt(1);
        end
        _C2 = 'dz;
        _D2 = 'dz;
        wait (C2 == 1);
    endtask

    task invalid_cache_line(input reg[CACHE_TAG_SIZE - 1:0] _tag, input
reg[CACHE_SET_SIZE - 1:0] _set);
        line_info1 = cache_info[_set * CACHE_WAY];
        line_info2 = cache_info[_set * CACHE_WAY + 1];
        if (line_info1[CACHE_TAG_SIZE + 3 - 1] == 1 &&
line_info1[CACHE_TAG_SIZE:1] == tag) begin
            if (line_info1[CACHE_TAG_SIZE + 3 - 2] == 1) begin
                write_line_in_mem(_tag, _set, 0);
            end
            cache_info[_set * CACHE_WAY][CACHE_TAG_SIZE + 3 - 1] = 0;
            cache_info[_set * CACHE_WAY][0] = 0;
            cache_info[_set * CACHE_WAY][0] = 1;
        end
        else if (line_info2[CACHE_TAG_SIZE + 3 - 1] == 1 &&
line_info2[CACHE_TAG_SIZE:1] == tag) begin
            if (line_info2[CACHE_TAG_SIZE + 3 - 2] == 1) begin
                write_line_in_mem(_tag, _set, 1);
            end
            cache_info[_set * CACHE_WAY + 1][CACHE_TAG_SIZE + 3 - 1] = 0;
            cache_info[_set * CACHE_WAY][0] = 1;
            cache_info[_set * CACHE_WAY + 1][0] = 0;
        end
    endtask

    always @(posedge C_DUMP) begin
        $display((count - miss) * 100 / count);
        $display("Count - %d, Miss - %d, Hits - %d", count, miss, hits);
        // for (dm = 0; dm < CACHE_LINE_COUNT; dm+=1) begin
        //     $display("Cache info number %d - %b", dm, cache_info[dm]);
        //     $display("Data line");
        //     for (bt = 0; bt < CACHE_LINE_SIZE; bt += 1) begin
        //         $display("%d - %b", bt, cache_data[dm][bt]);
        //     end
        //     $display("--------");
        // end
    end

    always @(posedge reset) begin
        reser_cache();
    end

    task reser_cache;
        for (cl = 0; cl < CACHE_LINE_COUNT; cl += 1) begin
```

```
            cache_info[cl] = 0;
        end
    endtask

endmodule
```

MemCTR

```
module MemCTR #(
    parameter MEM_SIZE = (1 << 20),
    parameter CACHE_SIZE = (1 << 10),
    parameter CACHE_LINE_SIZE = 32,
    parameter CACHE_LINE_COUNT = 32,
    parameter CACHE_WAY = 2,
    parameter CACHE_SETS_COUNT = 16,
    parameter CACHE_TAG_SIZE = 11,
    parameter CACHE_SET_SIZE = 4,
    parameter CACHE_OFFSET_SIZE = 5,
    parameter CACHE_ADDR_SIZE = 20,
    parameter ADDR1_BUS_SIZE = 15,
    parameter ADDR2_BUS_SIZE = 15,
    parameter DATA1_BUS_SIZE = 16,
    parameter DATA2_BUS_SIZE = 16,
    parameter CTR1_BUS_SIZE = 3,
    parameter CTR2_BUS_SIZE = 2,
    parameter _SEED = 225526
)
(
    inout wire [DATA2_BUS_SIZE - 1:0] D2,
    inout wire [CTR2_BUS_SIZE - 1:0] C2,

    input wire [ADDR2_BUS_SIZE - 1:0] A2,
    input clk,
    input reset,
    input M_DUMP
);
    reg[ADDR2_BUS_SIZE - 1:0] _A2;
    reg[DATA2_BUS_SIZE - 1:0] _D2;
    reg[CTR2_BUS_SIZE - 1:0] _C2;

    assign D2 = _D2;
    assign C2 = _C2;

    // initial memory
    reg[CACHE_TAG_SIZE - 1:0] tag;
    reg[CACHE_SET_SIZE - 1:0] set;
    reg[7:0] line [0:CACHE_LINE_SIZE - 1];
    reg[CACHE_ADDR_SIZE:0] addr = 0;
    reg[CACHE_ADDR_SIZE:0] addrend = 0;
```

```
    integer i = 0;
    integer j = 0;
    integer m = 0;

    integer SEED = _SEED;
    reg[7:0] mem[0:MEM_SIZE - 1];
    initial begin
        _D2 = 'bz;
        _C2 = 'bz;
        initila_memory();
        $display("Initial memory");
    end


    always @(posedge reset) begin
        initila_memory();
    end

    task initila_memory;
        for (i = 0; i < MEM_SIZE; i += 1) begin
            mem[i] = $random(SEED)>>16;
        end
    endtask

    task wait_cnt(int cnt);
        for (j = 0; j < cnt; j += 1) begin
            @(posedge clk);
        end
    endtask


    always @(posedge clk) begin
        if (C2 === 2) begin
            tag = A2[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE];
            set = A2[CACHE_SET_SIZE - 1:0];
            wait_cnt(100); ///<100>
            _C2 = 1;
            addr[CACHE_ADDR_SIZE - 1:CACHE_OFFSET_SIZE] = A2;
            addr[CACHE_OFFSET_SIZE - 1:0] = (1 << CACHE_OFFSET_SIZE) - 1;
            for (i = 0; i < CACHE_LINE_SIZE; i+= 2) begin
                _D2[15:8] =  mem[addr - i];
                _D2[7:0] =  mem[addr - i - 1];
                wait_cnt(1);
            end
            _D2 = 'dz;
            _C2 = 'dz;
        end

        else if (C2 === 3) begin
```

```
            tag = A2[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1:CACHE_SET_SIZE];
            set = A2[CACHE_SET_SIZE - 1:0];
            addr[CACHE_ADDR_SIZE - 1:CACHE_OFFSET_SIZE] = A2;
            addr[CACHE_OFFSET_SIZE - 1:0] = (1 << CACHE_OFFSET_SIZE) - 1;
            for (i = 0; i < CACHE_LINE_SIZE; i+= 2) begin
                mem[addr - i] = D2[15:8];
                mem[addr - i - 1] = D2[7:0];
                wait_cnt(1);
            end
            wait_cnt(100); ///<100>
            _C2 = 1;
            wait_cnt(1);
            _C2 = 'dz;
        end
    end

    always @(posedge M_DUMP) begin
        for (m = 0; m < MEM_SIZE; m+=1) begin
            $display("Mem %d - %b", m, mem[m]);
        end
    end


endmodule
```