МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Институт Информационных Технологий
Кафедра Вычислительной Техники (ВТ)

ОТЧЁТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 3

«Проектирование простого процессорного ядра»

по дисциплине

«Схемотехника устройств компьютерных систем»

Выполнил студент группы ИВБО-03-22 — Заливакин А.С.

Принял ассистент кафедры ВТ — Дуксин Н.А.

Практическая работа выполнена «__»_____2024 г.

«Зачтено» «__»_____2024 г.

Москва 2024

АННОТАЦИЯ

Данная работа включает в себя 3 рисунка, 3 листинга, 3 таблицы. Количество страниц в работе — 28.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: целью данной практической работы является создание процессорного ядра и разработка всех необходимых модулей и кодов, обеспечивающих выполнение базовых операций, обработку команд и взаимодействие с памятью.

Постановка задачи: определить архитектуру процессорного ядра, включая структуру регистров, арифметико-логического устройства (АЛУ), системы управления и блоков памяти. Создать и задокументировать набор команд для процессорного ядра, включая их форматы, семантику и функциональность. Разработать программные модули для реализации функционала процессора

Результат выполнения работы: код модулей на Verilog HDL, временные диаграммы, отражающие корректность работы спроектированных модулей.

1 ОСНОВНОЙ РАЗДЕЛ

1.1 Описание архитектуры процессора

Опишем формат команд по заданному варианту: сортировка массива пузырьком.

Сортировка пузырьком — это простой алгоритм сортировки, который многократно проходит по массиву, сравнивая соседние элементы и меняя их местами, если они находятся в неправильном порядке. Процесс повторяется, пока массив не будет отсортирован.

Память данных (mem) для хранения массива, который будет отсортирован. Размер: N = 3 (3 элементоа). Каждый элемент массива будет занимать 32 бита.

Память команд (prog) для хранения команд, которые будут выполняться процессором. Максимальный размер одной команды 44 бит. Потенциально общее количество команд – максимум 64 (6 бит).

Регистры общего назначения (РОН) включают в себя:

- Регистр 0 (константа 0);
- Регистр 1 (константа 1);
- Регистр 2 (количество элементов);
- Регистр 3 (индекс i в цикле сортировки);
- Регистр 4 (хранит результат вычитания в операции SUB, используется для проверки условия продолжения цикла);
- Регистр 5 (временное хранилище для обмена элементов при выполнении SWP);
- Регистр 6 (индекс j в цикле сортировки);
- Регистр 7 (элемент j в цикле сортировки);
- Регистр 8 (индекс j + 1 в цикле сортировки).

1.2 Описание формата команд

Каждая команда состоит из нескольких полей:

- Код операции (КОП) – определяет какую операцию выполнить (4 бита, позволяет иметь до 16 команд);
- Адреса регистров или памяти – указывают на места, откуда нужно брать данные или куда записывать результат;
- Литеральные значения – задают значения, которые используются в операциях.

Максимальное количество бит для команды 44 бит.

Адресация регистров осуществляется с помощью 4 бит, что позволяет адресовать до 15 регистров (адрес 1 + адрес 2 + адрес результата).

Для команд, которые используют переходы (например, JLZ и JMP), используются биты для хранения литералов.

- NOP (No Operation):
  - Код операции (КОП) 4 бита;
  - Общий формат: 00000000000000000000000000000000000000000000;
- INCR (Увеличение значения регистра на 1):
  - Код операции (КОП) 4 бита;
  - Адрес регистра для инкрементации: 4 бита;
  - Общий формат: 00010011000000000000000000000000000000000000
  - (увеличение регистра 3)
- ZTR (Обнуление регистра):
  - Код операции (КОП) 4 бита;
  - Адрес регистра, для обнуления: 4 бита;

    - Общий формат: 00100011000000000000000000000000000000000
    - (обнулить регистр 3).
- RTM (Загрузка значения регистра в память):
    - Код операции (КОП) 4 бита;
    - Адрес регистра, из которого писать: 4 бита;
    - Адрес памяти в которую писать: 4 бита.
    - Общий формат: 00110111011000000000000000000000000000000
    - (записывает из регистра 7 в память 6).
- MTR (Загрузка памяти в регистр):
    - Код операции (КОП) 4 бита;
    - Адрес памяти: 4 бита;
    - Адрес регистра: 4 бита;
    - Общий формат: 01000101011100000000000000000000000000000;
    - (записывает из ячейки памяти 5 в регистр 7)
- RTR (Загрузка из регистра в регистр):
    - Код операции (КОП) 4 бита;
    - Номер регистра из которого записывать: 4 бита;
    - Номер регистра в который записывать: 4 бита;
    - Общий формат: 01010101011000000000000000000000000000000
    - (записать значение из регистра 5 в регистр 6).
- LTR (Загрузка из литерала в регистр):
    - Код операции (КОП) 4 бита;
    - Литерал из которого записывать: 32 бита;

- Номер регистра в который записывать: 4 бита;
- Общий формат: 01100010000000000000000000000000000000011
- (записать значение литерала 3 в регистр 2).

- LTM (Загрузка литерала в память):
  - Код операции (КОП) 4 бита;
  - Номер регистра, в котором хранится адрес ячейки: 4 бита (косвенная адресация);
  - Значение литерала: 32 бита;
  - Общий формат: 00100110000000000000000000000001010 (записать значение из регистра 3 в ячейку памяти с адресом 10).
- SUB (Вычитание значений из двух регистров):
  - Код операции (КОП) 4 бита;
  - Адрес 1 регистра: 4 бита (прямая адресация);
  - Адрес 2 регистра: 4 бита (прямая адресация);
  - Адрес регистра, куда записывать результат: 4 бита;
  - Общий формат: 10000010001100000000000000000000000000000 (вычесть значение регистра 2 из регистра 3 и записать в регистр 4).
- JLZ (Условный переход, если результат меньше нуля):
  - Код операции (КОП) 4 бита;
  - Адрес ячейки памяти команд (куда переходить): 6 бит;
  - Общий формат: 10010000000000000000000000000000010010
  - (если результат в регистре оказался меньше нуля – перейти по адресу 18).

- JMP (Безусловный переход):
    - Код операции (КОП) 4 бита;
    - Адрес куда переходить: 4 бита;
    - Общий формат: 10100011000000000000000000000000000000000
    - (безусловный переход в 3).

1.3 Блок схема алгоритма

Алгоритм начинается с ввода количества элементов массива. Затем происходит инициализация счетчика i = 0 и ввод элементов массива в цикле.

После ввода всех элементов начинается основная часть сортировки: внешний цикл с счетчиком i, внутренний цикл с счетчиком j, сравнение соседних элементов массива (элемент j и элемент j + 1). Если текущий элемент больше следующего, происходит их обмен с помощью временной переменной t (Рисунок 1.1).

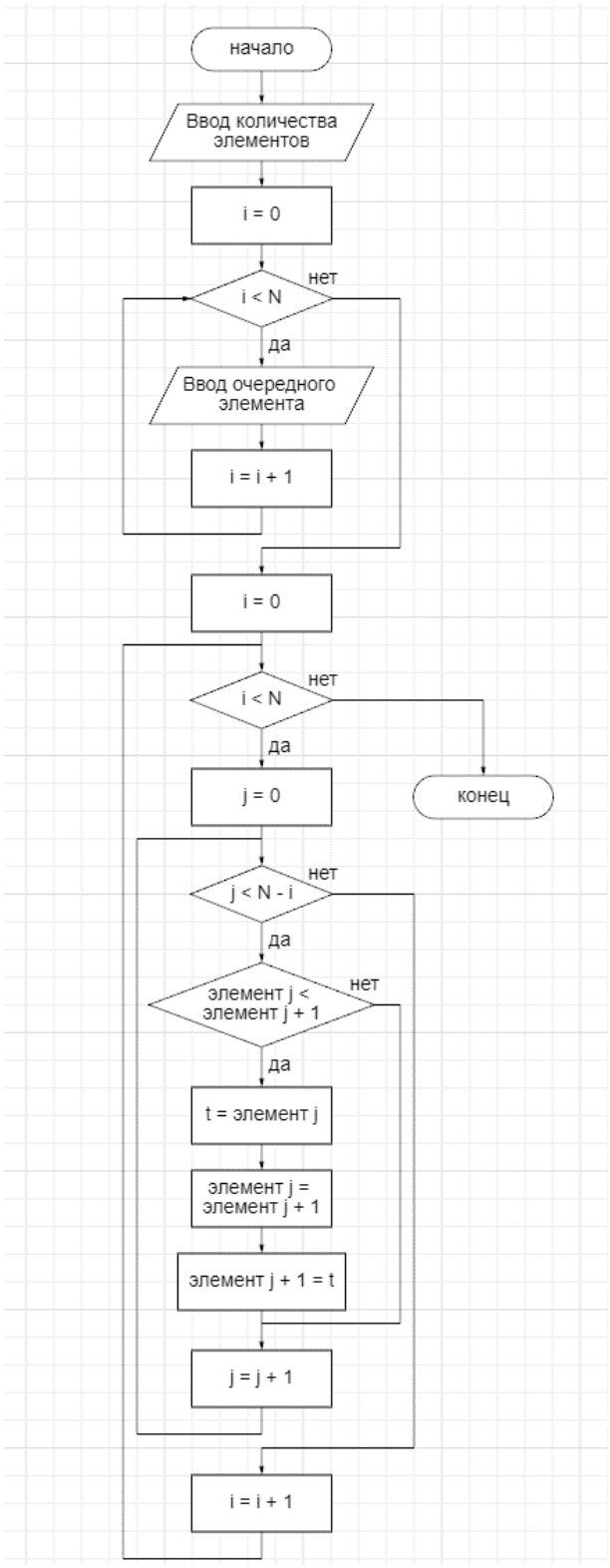

Рисунок 1.1 – Блок схема алгоритма сортировки пузырьком

Далее опишем покомандный алгоритм на основе наших команд, описанных выше.

На шаге 0 команда LTR 2, N загружает в регистр 2 длину массива (N). На шаге 1 команда ZTR 3 обнуляет регистр 3, который будет использоваться как индекс) для текущей позиции в массиве.

Шаг 2 выполняет безусловный переход к шагу 30 с помощью команды JMP 30, который инициализирует сортировку.

На шаге 3 команда ZTR 3 обнуляет регистр 3. Далее на шаге 4 выполняется операция сравнения: команда SUB 3, 2, 4 вычитает из текущего индекса длину массива. Это нужно для того, чтобы определить, достигнут ли конец массива.

Если результат отрицательный, команда JLZ 22 на шаге 5 переходит к завершению работы сортировки. Если нет, продолжается выполнение.

Внутри цикла выполняется операция сравнения соседних элементов (шаги 7-13). Если элемент в позиции j больше элемента в позиции j+1, они меняются местами.

После каждой итерации индекс (j) увеличивается на 1 с помощью команды INCR 5 на шаге 18.

После инкремента происходит безусловный переход на шаг 4 для повторного выполнения цикла.

Когда индекс (j) достигает значения (N - 1), программа переходит к завершению на шаге 22 и завершает сортировку (Рисунок 1.2).

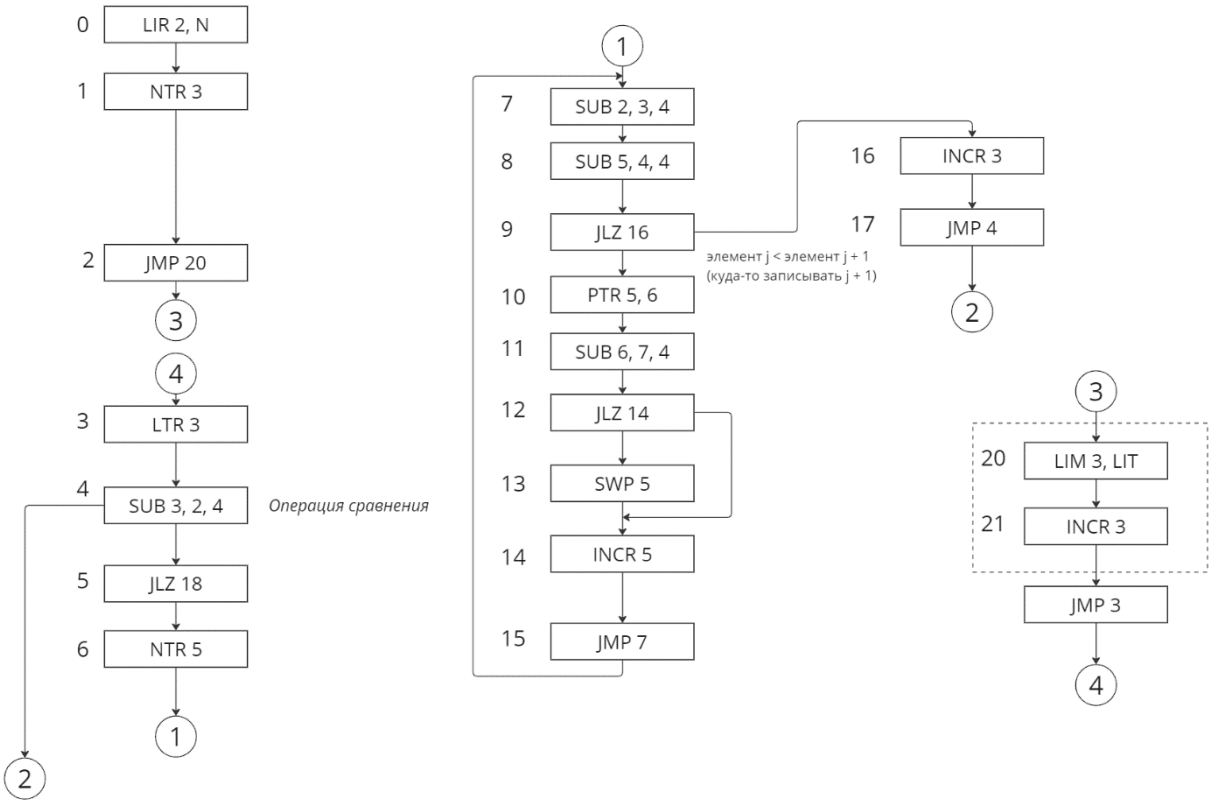

Рисунок 1.2 – Блок схема алгоритма сортировки пузырьком в процессорных командах

1.4 Описание стадий команд

Опишем выполнение каждой команды на различных стадиях выполнения (fetch, decode, execute, mem access, write back). Каждая команда проходит через 5 стадий — извлечение, декодирование, исполнение, доступ к памяти и запись результата (Таблица 1.1).

Таблица 1.1 – Описание стадий команд в форме текста

Команда	0 стадия fetch	1 стадия decode	2 стадия исполнение (execute)	3 стадия доступ к памяти (mem access)	4 стадия запись результата (write back)
NOP	Извлечение команды из памяти.	Команда декодируется как NOP.	-	-	-
INCR	Извлечение команды из памяти.	Декодирование команды, получение адреса регистра.	Операция инкремента: значение регистра увеличивается на 1.	-	Результат записывается в регистр.

ZTR	Извлечение команды из памяти.	Декодирование команды.	Запись нуля в регистр.	-	Результат записывается в регистр.
RTM	Извлечение команды из памяти.	Декодирование команды. Загрузка операндов.		-	Результат записывается в регистр.
MTR	Извлечение команды из памяти.	Декодирование команды, получение адреса памяти и регистра.	-	Загрузка значения из памяти по указанному адресу.	Результат записывается в регистр.
RTR	Извлечение команды из памяти.	Декодирование команды, получение операндов и адресов регистров.		-	Результат записывается в регистр.
LTR	Извлечение команды из памяти.	Декодирование команды, загрузка операндов.	Загрузка литерала в регистр.	-	-
LTM	Извлечение команды из памяти.	Декодирование команды, получение адреса памяти и регистра.	Взятие значения по указателю (косвенный доступ к памяти).	Доступ к памяти по косвенному адресу.	Результат записывается в регистр.
SUB	Извлечение команды из памяти.	Декодирование команды, получение адресов регистров.	Операция обмена значениями между регистрами.	-	Запись результатов обмена в регистры.
JLZ	Извлечение команды из памяти.	Декодирование команды, получение адреса перехода.	Переход на указанный адрес.	-	-
JMP					

Далее опишем данные таблице в форме команд (Таблица 1.2).

Таблица 1.2 – Описание стадий команд в форме команд

Команда	0: Fetch	1: Decode	2: Execute	3: Memory Access	4: Write Back
INCR	PC = CMD[SC]	OperandA <= cmd[39:36]; OperandB <= cmd[35:32];	res = alu1 + alu2	-	GPR[adr1] = res pc = pc + 1

LTR	PC = CMD[SC]	-	-	-	GPR[adr1] = lit pc = pc + 1
NTR	PC = CMD[SC]	-	res = 0	-	GPR[adr1] = res pc = pc + 1
LTM	PC = CMD[SC]	-	-	-	MEM[GPR[adr1]] = lit pc = pc + 1
SUB	PC = CMD[SC]	alu1 = GPR[adr1] alu2 = GPR[adr2]	res = alu1 - alu2 lz = res[31]	-	GPR[adr1] = res pc = pc + 1
JLZ	PC = CMD[SC]	-	if (lz) pc = ja; else pc = pc + 1;	-	-
PTR	PC = CMD[SC]	alu1 = GPR[adr1] alu2 = GPR[adr2]	res_h = alu1 + 1 res_l = alu2 + 1	-	GPR[adr1] = MEM[alu1 + 1] pc = pc + 1
SWP	PC = CMD[SC]	alu1 = GPR[adr1] alu2 = GPR[adr2]	-	-	GPR[adr1] = GPR[adr2] GPR[adr2] = alu1
JMP	PC = CMD[SC]	-	-	-	pc = ja

Далее опишем таблицу на языке Verilog (Таблица 1.3).

Таблица 1.3 – Описание стадий команд в форме команд Verilog

Команда	0: Fetch	1: Decode	2: Execute	3: Memory Access	4: Write Back
INCR	pr_next <= cmd[pc]	AdrA <= addr1_next, AdrB <= 1	res_next <= alu1 + alu2	-	DATA <= res[31:0], AdrWrite <= addr1, pc <= pc + 1
LTR	pr_next <= cmd[pc]	-	-	-	DATA <= lit, AdrWrite <= addr1, pc <= pc + 1
NTR	pr_next <= cmd[pc]	-	-	-	DATA <= 0, wen <= 1, AdrWrite <= addr1, pc <= pc + 1
LTM	pr_next <= cmd[pc]	AdrA <= addr1_next	-	-	mem[GPR[adr1]] <= lit, pc <= pc + 1

SUB	pr_next <= cmd[pc]	AdrA <= addr1_next, AdrB <= addr2_next	res_next <= alu1 - alu2, lz_next <= res[31]	-	DATA <= res[31:0], AdrWrite <= addr1, pc <= pc + 1
JLZ	pr_next <= cmd[pc]	-	if (~lz) pc <= ja else pc <= pc + 1	-	-
PTR	pr_next <= cmd[pc]	AdrA <= addr1_next, AdrB <= addr2_next	res_next <= {alu1 + 1, alu2 + 1}	DATA <= mem[alu1]	GPR[res[31:0]] <= mem[res[63:32]], pc <= pc + 1
SWP	pr_next <= cmd[pc]	-	res_next <= alu2 + 1	-	mem[alu2] <= mem[res], mem[res] <= mem[alu2],
JMP	pr_next <= cmd[pc]	-	-	-	pc <= ja

1.5 Код модулей на Verilog

Основной модуль процессорного ядра включает систему команд и стадии выполнения. В данном модуле используются следующие переменные и логика:

- clk и reset – входные сигналы тактового генератора и сброса.
- stage_counter – счетчик стадий, управляющий процессом выполнения команд.
- program_counter – регистр для хранения текущего адреса команды.
- cmd – регистр для хранения текущей команды.
- data – регистр для хранения данных.
- code_prog – регистр для хранения кода текущей команды.
- OperandA, OperandB, OperandC – регистры для хранения операндов.
- prog – память команд, загружаемая из внешнего файла "prog.mem".
- РОН – регистровый файл.
- mem – память данных.

Стадии выполнения команд:

1. **Стадия 0**:
    - Загружается команда из памяти команд в регистр cmd,

используя program_counter в качестве указателя.
    - Подготавливаются адреса регистров для следующей команды.
2. **Стадия 1**:
    - Определяются операнды, используя старшие биты команды cmd.
    - Операнды загружаются в регистры OperandA, OperandB, OperandC.
3. **Стадия 2**:
    - Выполняются арифметические и логические операции в зависимости от типа команды.
    - Результат сохраняется в регистр data или устанавливаются флаги (например, lz для команды SUB).
4. **Стадия 3**:
    - Для команд, требующих доступа к памяти (например, LTR), выполняется чтение или запись в память данных.
5. **Стадия 4**:
    - Записываются результаты в регистровый файл или обновляется память данных.
    - Счетчик команд program_counter также обновляется в зависимости от результатов выполнения команд (например, для условных переходов JLZ).

Пример выполнения команд:

- **NOP**: Ничего не делает.
- **INCR**: Увеличивает значение в регистре РОН на 1.
- **ZTR**: Обнуляет значение в регистре РОН.
- **RTM**: Записывает значение из регистра РОН в память mem.
- **MTR**: Загружает значение из памяти mem в регистр РОН.
- **RTR**: Копирует значение из одного регистра РОН в другой.
- **LTR**: Загружает значение в регистр РОН.
- **LTM**: Записывает значение в память mem.
- **SUB**: Вычитает значение одного регистра РОН из другого и сохраняет результат.
- **JLZ**: Переходит по адресу, если значение регистра меньше нуля.
- **JMP**: Переходит по указанному адресу.

Код описанного модуля представлен в Листинге 1.1.

Листинг 1.1 – Основной модуль процессорного ядра на Verilog

```
module cpu(
    input clk,
    input reset,
    output reg [1:0] stage_counter,
    output reg [63:0] program_counter,
    output reg [43:0] cmd,
    output reg [31:0] data,
    output reg [3:0] code_prog,
    output reg [3:0] OperandA,
    output reg [3:0] OperandB,
    output reg [3:0] OperandC
);
integer lz;
reg [43:0] prog [0:63];
reg [3:0] POH [0:8];
reg [31:0] mem [0:63];
integer i;
initial begin
    $readmemb("prog.mem", prog);
    stage_counter <= 2'b00;
    program_counter <= 64'b0;
    cmd <= 44'b0;
    data <= 32'b0;
    code_prog <= 4'b0;
    for (i = 0; i < 8; i = i + 1) begin
        POH[i] <= 4'b0;
    end
    POH[1] <= 4'b1;
    for (i = 0; i < 64; i = i + 1) begin
        mem[i] <= 32'b0;
```

Продолжение Листинга 1.1.

```
        end
    end
    localparam NOP = 4'b0000, INCR = 4'b0001, ZTR = 4'b0010, RTM = 4'b0011,
MTR = 4'b0100, RTR = 4'b0101, LTR = 4'b0110, LTM = 4'b0111, SUB = 4'b1000, JLZ
= 4'b1001, JMP = 4'b1010;
    always @(posedge clk) begin
        if (reset) begin
            stage_counter <= 2'b00;
            program_counter <= 64'b0;
            cmd <= 44'b0;
            data <= 32'b0;
            code_prog <= 4'b0;
        end
    end
    always @(posedge clk) begin
        if (stage_counter == 2'b11) begin
            stage_counter <= 2'b00;
            program_counter <= program_counter + 1;
            OperandA <= 4'b0;
            OperandB <= 4'b0;
            OperandC <= 4'b0;
        end else begin
            stage_counter <= stage_counter + 2'b01;
        end
    end
    integer dec;
    integer dec2;
    integer dec3;
    always @* begin
        case (stage_counter)
            2'b00: begin
```

Продолжение Листинга 1.1.

```
            cmd <= prog[program_counter];
        end
        2'b01: begin
            case (cmd[43:40])
                NOP: code_prog <= NOP;
                INCR: begin
                    code_prog <= INCR;
                    OperandA <= cmd[39:36];
                end
                ZTR: begin
                    code_prog <= ZTR;
                    OperandA <= cmd[39:36];
                end
                RTM: begin
                    code_prog <= RTM;
                    OperandA <= cmd[39:36];
                    OperandB <= cmd[35:32];
                end
                MTR: begin
                    code_prog <= MTR;
                    OperandA <= cmd[39:36];
                    OperandB <= cmd[35:32];
                end
                RTR: begin
                    code_prog <= RTR;
                    OperandA <= cmd[39:36];
                    OperandB <= cmd[35:32];
                end
                LTR: begin
```

Продолжение Листинга 1.1.

```
                    code_prog <= LTR;
                    OperandA <= cmd[39:36];
                    data <= cmd[27:0];
                end
                LTM: begin
                    code_prog <= LTM;
                    OperandA <= cmd[39:36];
                    data <= cmd[27:0];
                end
                SUB: begin
                    code_prog <= SUB;
                    dec <= cmd[39:36];
                    dec2 <= cmd[35:32];
                    dec3 <= cmd[31:28];
                end
                JLZ: begin
                    code_prog <= JLZ;
                    data <= cmd[27:0];
                end
                JMP: begin
                    code_prog <= JMP;
                    data <= cmd[27:0];
                end
                default: code_prog <= NOP;
            endcase
        end
        2'b10: begin
            case (code_prog)
                NOP: ;
               INCR: begin
```

Продолжение Листинга 1.1.

```
                dec = OperandA;
                data = POH[dec];
            end
            ZTR: begin
                dec = OperandA;
                POH[dec] <= 0;
            end
            RTM: begin
                dec = OperandA;
                dec2 = OperandB;
                mem[dec] <= POH[dec2];
            end
            MTR: begin
                dec = OperandA;
                dec2 = OperandB;
                POH[dec] <= mem[dec2];
            end
            RTR: begin
                dec = OperandA;
                dec2 = OperandB;
                POH[dec] <= POH[dec2];
            end
            LTR: begin
                dec = OperandA;
                POH[dec] <= data;
            end
            LTM: begin
                dec = OperandA;
                mem[dec] <= data;
            end
```

Продолжение Листинга 1.1.

```
                    SUB: begin
                        POH[dec3] <= POH[dec2] - POH[dec3];
                        lz <= dec3;
                    end
                    JMP: program_counter <= data;
                    default: ;
                endcase
            end
            2'b11: begin
                case (code_prog)
                    INCR: begin
                        dec = OperandA;
                        POH[dec] <= data + 1;
                    end
                    JLZ: if (POH[lz] < 0) program_counter <= data;
                    default: ;
                endcase
            end
            default: ;
        endcase
    end
    endmodule
```

Напишем тестовый модуль для проверки работы программы (Листинг 1.2).

Листинг 1.2 – Тестовый модуль на языке Verilog

```
module testbench;

    reg clk;

    reg reset;

    wire [1:0] stage_counter;

    wire [63:0] program_counter;

    wire [43:0] cmd;

    wire [31:0] data;

    wire [3:0] code_prog;

    wire [3:0] OperandA;

    wire [3:0] OperandB;

    wire [3:0] OperandC;

    cpu uut (

        .clk(clk),

        .reset(reset),

        .stage_counter(stage_counter),

        .program_counter(program_counter),

        .cmd(cmd),

        .data(data),

        .code_prog(code_prog),

        .OperandA(OperandA),

        .OperandB(OperandB),

        .OperandC(OperandC)

    );

    initial begin

        clk = 0;

        forever #5 clk = ~clk;

    end

    integer i;

    initial begin
```

Продолжение Листинга 1.2.

```
        reset = 1;
        #10;
        reset = 0;
        for (i = 0; i < 128; i = i + 1) begin
            #10;
            if (i == 37) begin
                $display("POH, DATA");
                $display("%b", uut.POH);
                $display("%b", uut.mem);
            end
        end
        $finish;
    end
endmodule
```

Опишем программу (инструкции в двоичном формате) через mem файл. (Листинг 1.4).

Листинг 1.3 – Программа для процессорного ядра в двоичном формате

```
01100010000000000000000000000000000000000011
00100011000000000000000000000000000000000000
10100000000000000000000000000000000000011110
00100011000000000000000000000000000000000000
10000011001001000000000000000000000000000000
10010000000000000000000000000000000000010110
00100101000000000000000000000000000000000000
10000010001101000000000000000000000000000000
10000101010001000000000000000000000000000000
10010000000000000000000000000000000000010100
01000101011100000000000000000000000000000000
01010101011000000000000000000000000000000000
00010110000000000000000000000000000000000000
01000110100000000000000000000000000000000000
10000111100001000000000000000000000000000000
10010000000000000000000000000000000000010010
00110111011000000000000000000000000000000000
00111000010100000000000000000000000000000000
00010101000000000000000000000000000000000000
10100111000000000000000000000000000000000000
00010011000000000000000000000000000000000000
10100100000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
01110011000000000000000000000000000000000011
00010011000000000000000000000000000000000000
01110100000000000000000000000000000000000001
00010011000000000000000000000000000000000000
01110101000000000000000000000000000000000000
00010011000000000000000000000000000000000000
10100000000000000000000000000000000000000011
```

1.6 Тестирование работы процессора

Протестируем работу процессора. Общая временная диаграмма показана ниже (Рисунок 1.3).

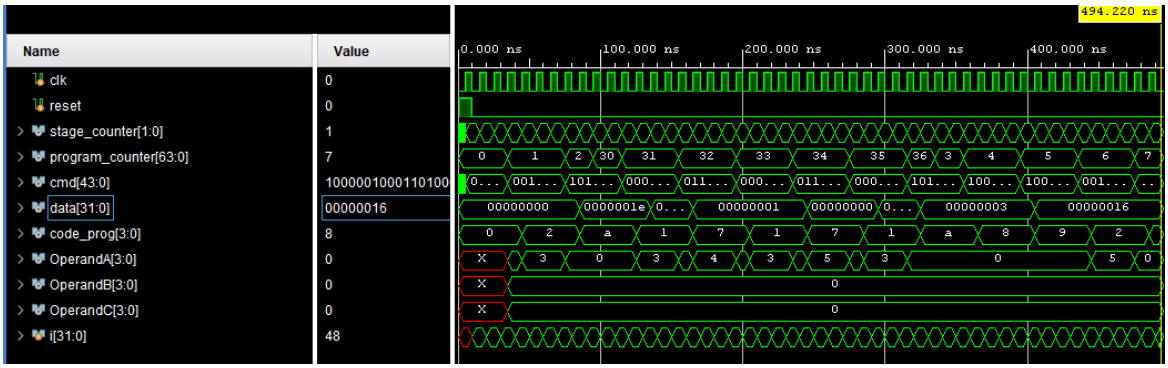

Рисунок 1.3 – Временная диаграмма тестового модуля

Рассмотрим диаграмму подробнее. Например, команда 1200000003 записывает в регистр 2 литерал 3 (Рисунок 1.4).

ЗАКЛЮЧЕНИЕ

В рамках данной практической работы спроектировали порстое процессорное ядро и разработали все необходимые модули и коды, обеспечивающих выполнение базовых операций, обработку команд и взаимодействие с памятью.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Методические указания по ПР № 1 — URL: https://online-edu.mirea.ru/mod/resource/view.php?id=405132 (Дата обращения: 23.09.2022).

2. Методические указания по ПР № 2 — URL: https://online-edu.mirea.ru/mod/resource/view.php?id=409130 (Дата обращения: 23.09.2022).

3. Смирнов С.С. Информатика [Электронный ресурс]: Методические указания по выполнению практических и лабораторных работ / С.С. Смирнов — М., МИРЭА — Российский технологический университет, 2018. — 1 электрон. опт. диск (CD-ROM).

4. Тарасов И.Е. ПЛИС Xilinx. Языки описания аппаратуры VHDL и Verilog, САПР, приемы проектирования. — М.: Горячая линия — Телеком, 2021. — 538 с.: ил.

5. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие / Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).

6. Антик М.И. Математическая логика и программирование в логике [Электронный ресурс]: Учебное пособие / Антик М.И., Бражникова Е.В.— М.: МИРЭА – Российский технологический университет, 2018. — 1 электрон. опт. диск (CD-ROM).

7. Жемчужникова Т.Н. Конспект лекций по дисциплине «Архитектура вычислительных машин и систем» — URL: https://drive.google.com/file/d/12OAi2_axJ6mRr4hCbXs-mYs8Kfp4YEfj/view?usp=sharing (Дата обращения: 23.09.2022).

8. Антик М.И. Теория автоматов в проектировании цифровых схем [Электронный ресурс]: Учебное пособие / Антик М.И., Казанцева Л.В. — М.: МИРЭА – Российский технологический университет, 2020. — 1 электрон. опт. диск (CD-ROM).