



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени
Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**Лабораторная работа № 2
по дисциплине «Архитектура ЭВМ»**

Тема Изучение принципов работы микропроцессорного ядра RISC-V

Студент Шахнович Дмитрий Сергеевич

Группа ИУ7-52Б

Вариант №20

Преподаватели Попов А., Калитвенцев М.

Содержание

ВВЕДЕНИЕ	2
1 Основная часть	3
1.1 Общие теоретические сведения	3
1.2 Задание №1	3
1.3 Задание №2	7
1.4 Задание №3	8
1.5 Задание №4	9
1.6 Задание №5	11
ЗАКЛЮЧЕНИЕ	15

ВВЕДЕНИЕ

Основной целью работы является ознакомление с принципами функционирования, построения и особенностями архитектуры суперскалярных конвейерных микропроцессоров. Дополнительной целью работы является знакомство с принципами проектирования и верификации сложных цифровых устройств с использованием языка описания аппаратуры SystemVerilog и ПЛИС.

В ходе работы необходимо выполнить следующие задачи:

- 1) Изучить и скомпилировать программу на ассемблере по варианту;
- 2) Изучить временную диаграмму стадий выборки и диспетчеризацию команды по заданному адресу по варианту;
- 3) Изучить временную диаграмму стадий декодирования и планирования команды по заданному адресу по варианту;
- 4) Изучить временную диаграмму стадии выполнения команды по заданному адресу по варианту;
- 5) Составить трассу выполнения программы по варианту. Оптимизировать программу, составить трассу оптимизированной программы, сравнить её с исходной программой.

1 Основная часть

1.1 Общие теоретические сведения

RISC-V является открытым современным набором команд, который может использоваться для построения как микроконтроллеров, так и высокопроизводительных микропроцессоров. В связи с такой широкой областью применения в систему команд введена вариативность. Таким образом, термин RISC-V фактически является названием для семейства различных систем команд, которые строятся вокруг базового набора команд, путём внесения в него различных расширений.

В данной работе исследуется набор команд RV32I, который включает в себя основные команды 32-битной целочисленной арифметики кроме умножения и деления. В рамках данного набора команд мы не будем рассматривать системные команды, связанные с таймерами, системными регистрами, управлением привилегиями, прерываниями и исключениями.

1.2 Задание №1

Варианту №20 соответствует следующая программа:

Листинг 1.1 — Программа варианта №20

```
.section .text
.globl _start;
len = 9 #Размер массива
enroll = 2 #Количество обрабатываемых элементов за одну итерацию
elem_sz = 4 #Размер одного элемента массива

_start:
la x1, _x
addi x20, x0, (len-1)/enroll
lw x31, 0(x1)
addi x1, x1, elem_sz*1
lp:
lw x2, 0(x1)
lw x3, 4(x1)
add x1, x1, elem_sz*enroll
bltu x2, x31, lt1
add x31, x0, x2 #!
lt1:    bltu x3, x31, lt2
add x31, x0, x3
lt2:
addi x20, x20, -1
bne x20, x0, lp
```

```

lp2: j lp2

.section .data
_x:    .4byte 0x1
.4byte 0x2
.4byte 0x3
.4byte 0x4
.4byte 0x8
.4byte 0x6
.4byte 0x7
.4byte 0x5
.4byte 0x4

```

Исходя из листинга (1.1) можно понять, что это программа которая ищет максимум из элементов массива, при этом обрабатывая по 2 элемента за раз. Исходя из этого можно сказать, что в конце программы значение в регистре x31 будет равно максимальному элементу массива, то есть 8. Данной программе можно соотнести псевдокод на листинге (1.2)

Листинг 1.2 — Псевдокод программы

```

#define len 9 \\ Размер массива
#define enroll 2 \\ Количество элементов за итерацию
#define elem_sz 4 \\ Размер элемента массива
int _x[len] = {1, 2, 3, 4, 8, 6, 7, 5, 4};
void _start() {
    int *x1 = _x;
    int x20 = x0 + (len-1)/enroll \\ В регистре x0 всегда 0;
    int x31 = x1[0];
    x1 += elem_sz; \\ С учётом сложение адресов как char*
    do {
        int x2 = x1[0]
        int x3 = x1[elem_sz] \\ если учитывать, что адреса складываются к
                           // ак char*
        x1 += elem_sz * enroll;
        if !(x2 < x31) {
            x31 = x2;
        }
        if !(x3 < x31) {
            x31 = x3;
        }
        x20--;
    } while (x20 != 0);
    while(1) {};
}

```

```
}
```

На листинге (1.3) представлен дизассемблированный код с листинга (1.1)

Листинг 1.3 — Дизассемблированный код

SYMBOL TABLE:

80000000 1	d	.text	00000000 .text
8000003c 1	d	.data	00000000 .data
00000000 1	df	*ABS*	00000000 myvar.o
00000009 1		*ABS*	00000000 len
00000002 1		*ABS*	00000000 enroll
00000004 1		*ABS*	00000000 elem_sz
8000003c 1		.data	00000000 _x
80000014 1		.text	00000000 lp
80000028 1		.text	00000000 lt1
80000030 1		.text	00000000 lt2
80000038 1		.text	00000000 lp2
80000000 g		.text	00000000 _start
80000060 g		.data	00000000 _end

Disassembly of section .text:

80000000 <_start>:	
80000000: 00000097	auipc x1,0x0
80000004: 03c08093	addi x1,x1,60 # 8000003c <_x>
80000008: 00400a13	addi x20,x0,4
8000000c: 0000af83	lw x31,0(x1)
80000010: 00408093	addi x1,x1,4
80000014 <lp>:	
80000014: 0000a103	lw x2,0(x1)
80000018: 0040a183	lw x3,4(x1)
8000001c: 00808093	addi x1,x1,8
80000020: 01f16463	bltu x2,x31,80000028 <lt1>
80000024: 00200fb3	add x31,x0,x2
80000028 <lt1>:	
80000028: 01f1e463	bltu x3,x31,80000030 <lt2>
8000002c: 00300fb3	add x31,x0,x3

```
80000030 <lt2>:  
80000030: fffa0a13           addi   x20 ,x20 ,-1  
80000034: fe0a10e3           bne   x20 ,x0 ,80000014 <lp>  
  
80000038 <lp2>:  
80000038: 0000006f          jal    x0 ,80000038 <lp2>
```

Disassembly of section .data:

```
8000003c <_x>:  
8000003c: 0001           .insn 2, 0x0001  
8000003e: 0000           .insn 2, 0x  
80000040: 0002           .insn 2, 0x0002  
80000042: 0000           .insn 2, 0x  
80000044: 00000003       lb    x0,0(x0) # 0 <enroll -0x2>  
80000048: 0004           .insn 2, 0x0004  
8000004a: 0000           .insn 2, 0x  
8000004c: 0008           .insn 2, 0x0008  
8000004e: 0000           .insn 2, 0x  
80000050: 0006           .insn 2, 0x0006  
80000052: 0000           .insn 2, 0x  
80000054: 00000007       .insn 4, 0x0007  
80000058: 0005           .insn 2, 0x0005  
8000005a: 0000           .insn 2, 0x  
8000005c: 0004           .insn 2, 0x0004
```

1.3 Задание №2

Для варианта №20 требуется проследить стадии выборки и диспетчеризации для команды по адресу 8000002c, при этом 2-й итерации. Эти стадии представлены на рисунке (1.1)

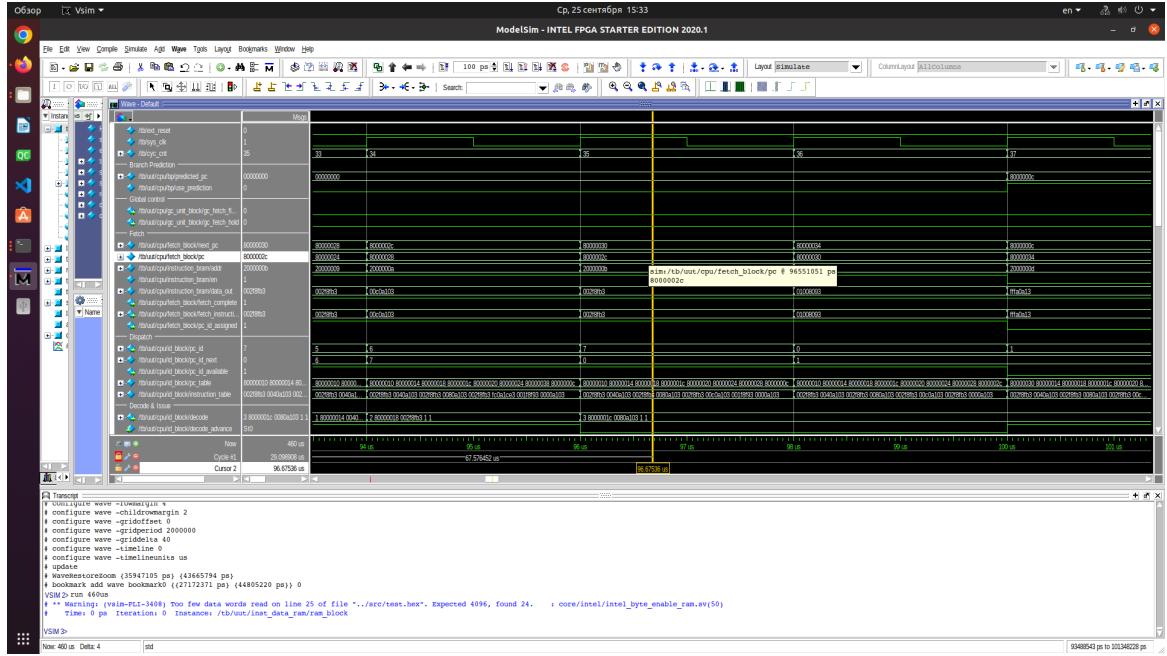


Рисунок 1.1 — Стадии выборки и диспетчеризации для 2-й итерации 8000002с

1.4 Задание №3

Для варианта №20 требуется проследить стадии декодирования и планирования для команды по адресу 80000038, при этом 2-й итерации. Эти стадии представлены на рисунках (1.2) и (1.3), так как стадии выполнялись с задержкой в такт.

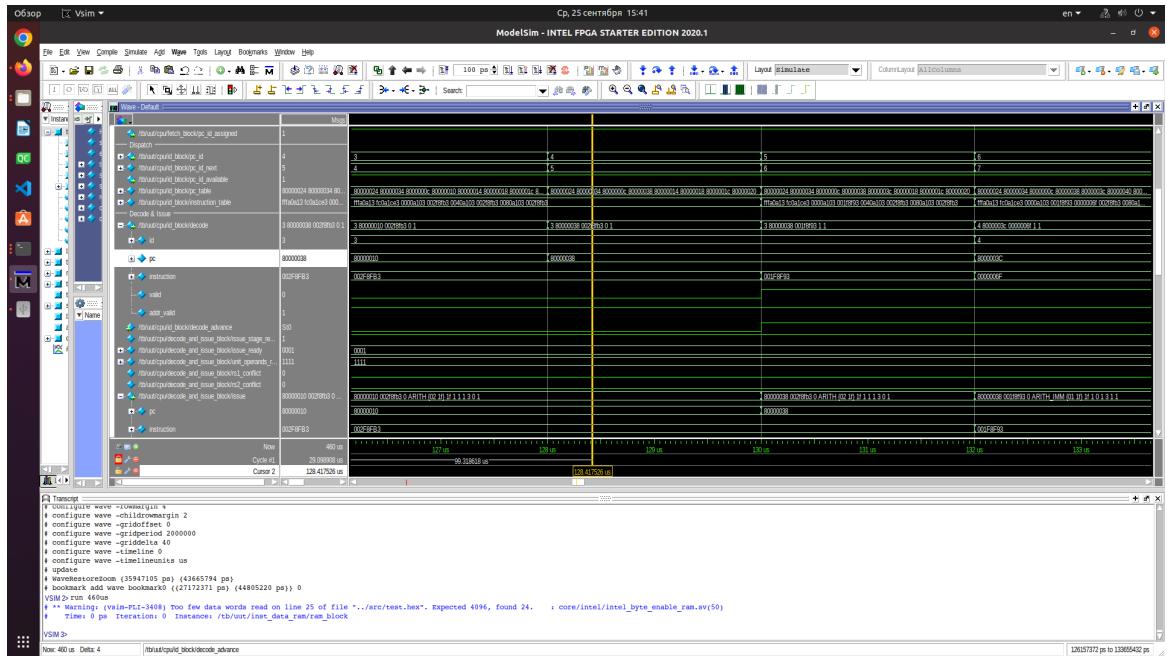


Рисунок 1.2 — 1-й такт стадии декодирования и планирования для 2-й итерации 80000038

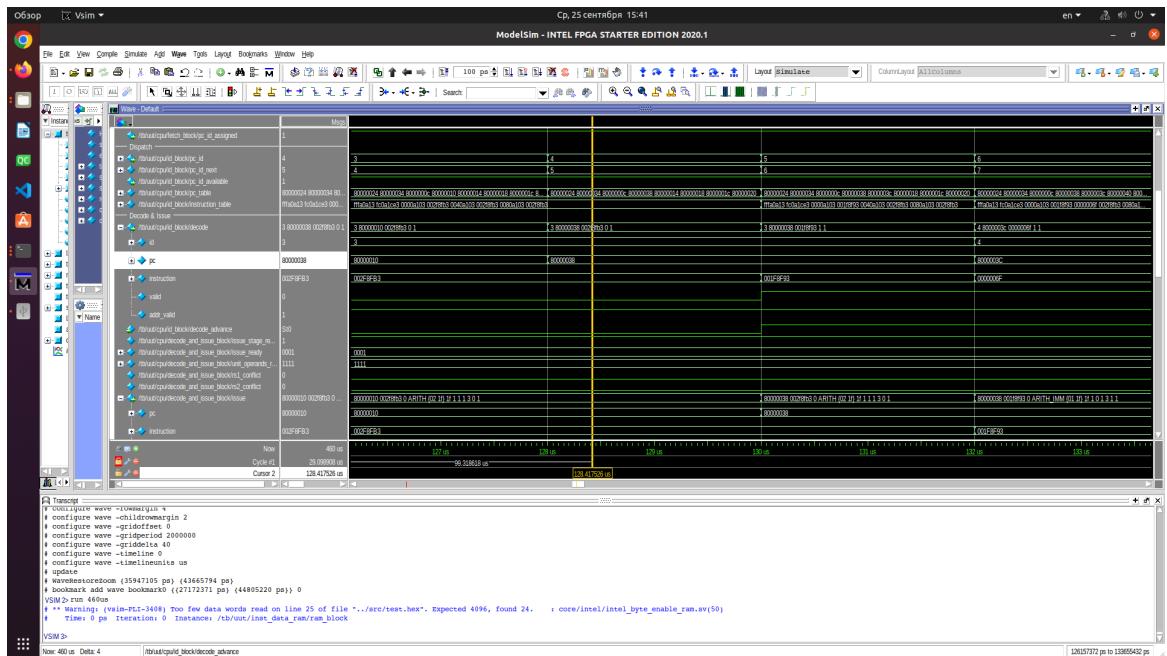


Рисунок 1.3 — 1-й такт стадии декодирования и планирования для 2-й итерации 80000038

1.5 Задание №4

Для варианта №20 требуется проследить стадию выполнения для команды по адресу 800000024, при этом 2-й итерации.

На рисунке (1.4) представлена стадия декодирования, из которой видно, что команда имеет $id=5$, это команда загрузки, значит дальше команда поступит в блок LSU, а также, что команда загрузки будет идти в 3 такта.

На рисунках (1.5) и (1.6) представлены 1-й и 3-й такты выполнения. На последнем видно, что команда загрузки с $id=5$ закончила выполнения.

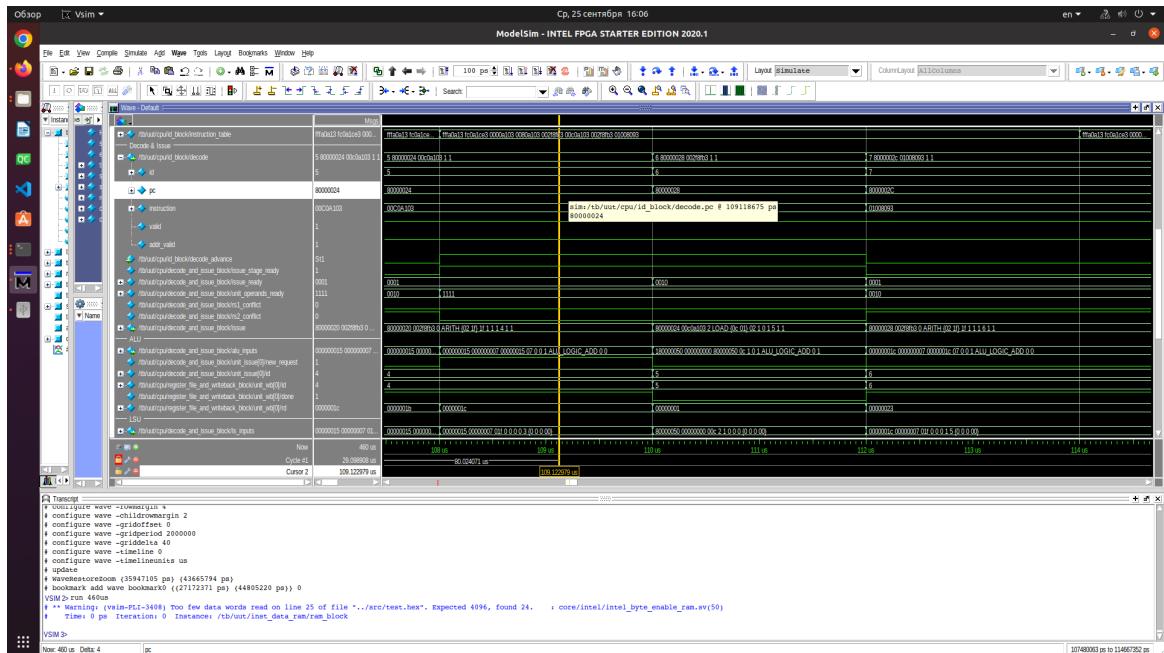


Рисунок 1.4 — Стадия декодирования для 2-й итерации 800000024

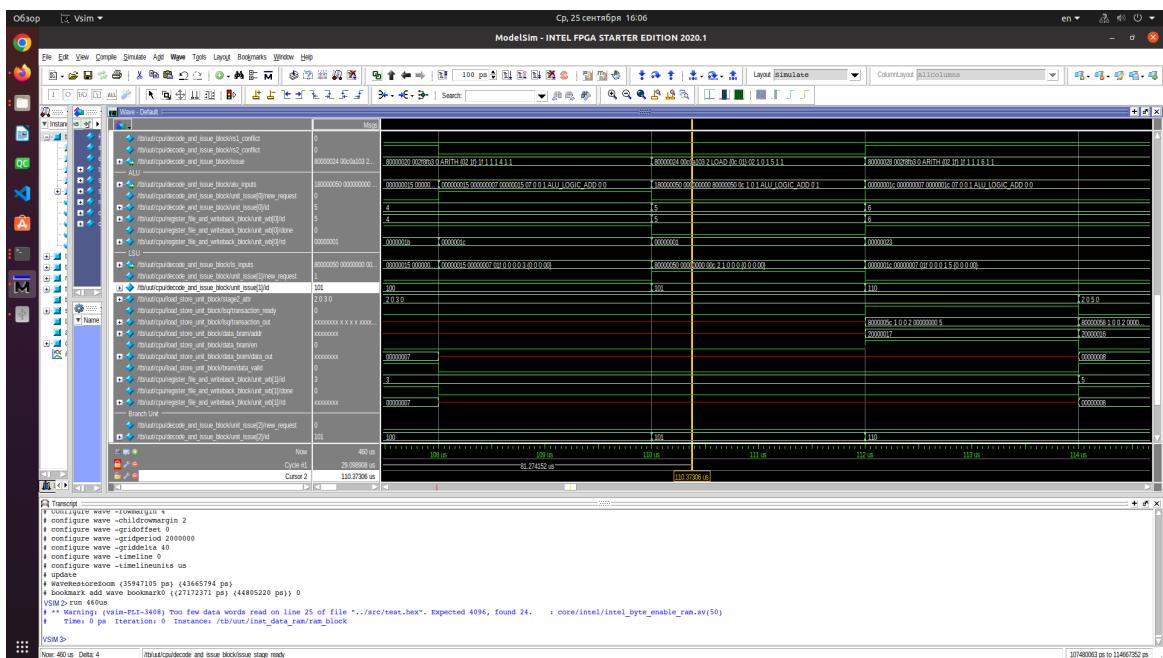


Рисунок 1.5 — 1-й такт стадии выполнения для 2-й итерации 80000024

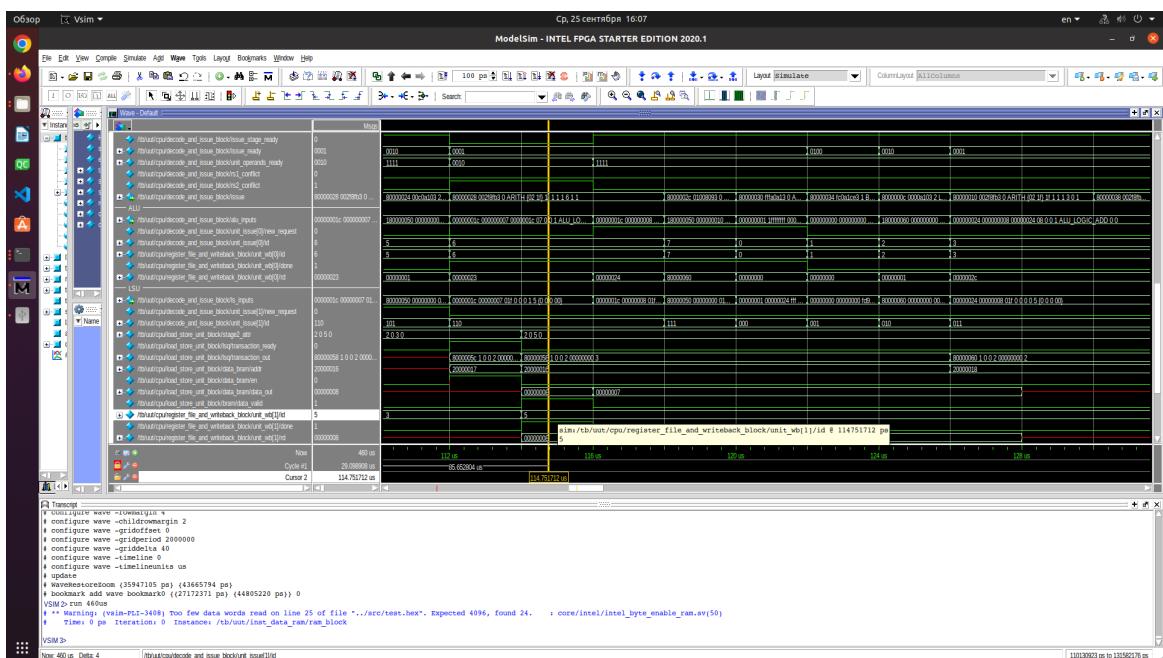


Рисунок 1.6 — 3-й такт стадии выполнения для 2-й итерации 80000024

1.6 Задание №5

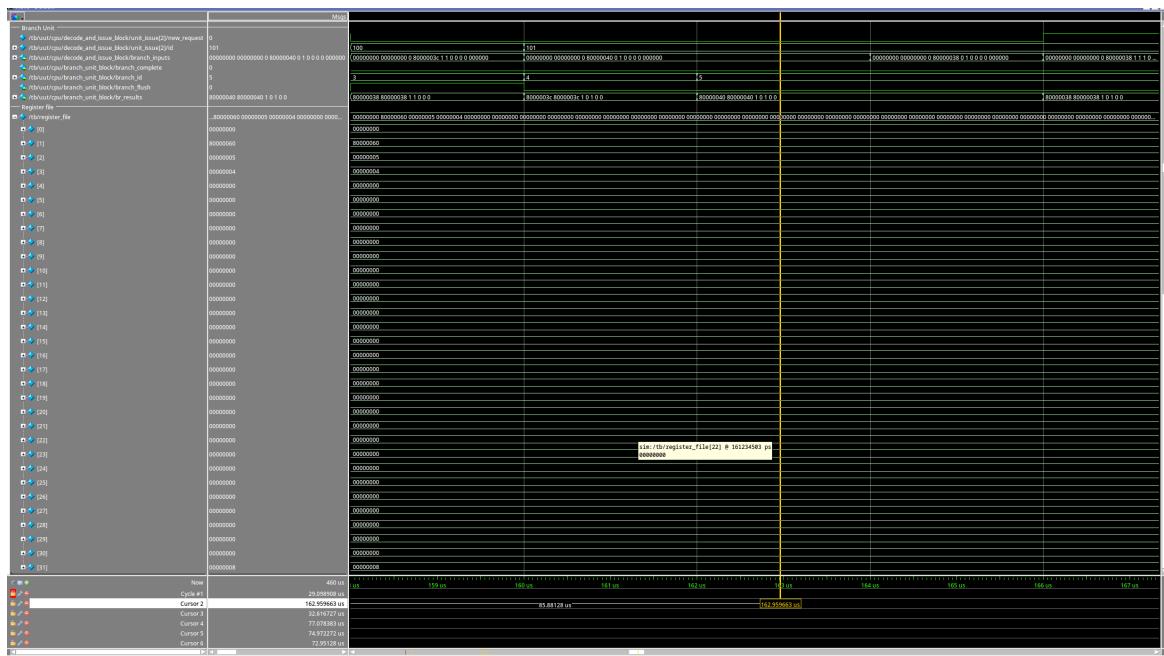


Рисунок 1.7 — Значение регистров после окончания работы программы

На рисунке (1.7) можно увидеть, что значение регистра $x31$ на момент окончания программы равно 8, как и было предсказано в задании 1.



Рисунок 1.8 — Трасса команды отмеченной #! в листинге 1.1

На рисунке (1.8) представлены все стадии команды отмеченной #! в листинге 1.1.

На рисунке (1.9) представлена трасса программы по варианту. Как видно программа достаточно хорошо оптимизирована, так как в ней нет конфликтов при обращения к ресурсам, а

Рисунок 1.9 — Трасса программы в листинге 1.1

значит и нет задержек. Единственное что можно оптимизировать - количество сбросов конвейера из-за условных переходов.

Для этого заменим поиск максимума через сравнение с условным переходом на конструкцию:

Листинг 1.4 — Поиск максимума через логические и арифметические операции.

```
c = b - a  
k = 1 if a < b else 0  
k-- (c переполнением)  
max = b - d & c
```

Данная конструкция легко реализуется с помощью команд данной архитектуры. Оптимизированный листинг представлен на (1.5).

Листинг 1.5 — Оптимизированная программа по варианту

```
.section .text
.globl _start;
len = 9 #Размер массива
enroll = 2 #Количество обрабатываемых элементов за одну итерацию
elem_sz = 4 #Размер одного элемента массива

_start:
la x1, _x
```

```

    addi x20, x0, (len-1)/enroll
    lw x31, 0(x1)
    addi x1, x1, elem_sz*1
lp:
    lw x2, 0(x1)
    lw x3, 4(x1)
    add x1, x1, elem_sz*enroll

    sub x4, x31, x2
    slt x5, x2, x31 # x5 = 1 if x2 < x31
    addi x5, x5, -1 # x5 = 11....11 if x2 > x31
    and x4, x4, x5 # x4 = 0 if x2 < x31
    sub x31, x31, x4

    sub x4, x31, x3
    slt x5, x3, x31 # x5 = 1 if x3 < x31
    addi x5, x5, -1 # x5 = 11....11 if x3 > x31
    and x4, x4, x5 # x4 = 0 if x3 < x31
    sub x31, x31, x4

    addi x20, x20, -1
    bne x20, x0, lp
lp2: j lp2

.section .data
_x: .4byte 0x1
.4byte 0x2
.4byte 0x3
.4byte 0x4
.4byte 0x8
.4byte 0x6
.4byte 0x7
.4byte 0x5
.4byte 0x4

```

Трасса оптимизированной программы на рисунке (??)

Как видно в программе стало меньше сбросов конвейера, тем не менее количество тактов стало больше(79 и 67), так как поиск максимума без условий требует больше команд. Однако в другой конфигурации массива, где блок предсказателя чаще ошибался, возможно оптимизированная программа была бы эффективная.

Адрес	Код	Команда	Номер тела
80000000<_start>	95400931	bne x20,x0,80000014 <>	0
80000004	95400931	addi x1,x1,84 # 80000054 <	1
80000008	95400931	addi x20,x0,4	2
8000000C	95400931	lw x31,0(x1)	3
80000010	95400931	addi x1,x1,4	4
80000014<-cp>	9000a103	lw x2,0(x1)	5
80000018	9000a103	lw x3,0(x1)	6
8000001C	9000a103	addi x1,x1,8	7
80000020	90282333	sub x2,x3,x2	8
80000024	9112263	slt x5,x2,x3	9
80000028	9728233	addi x5,x5,-1	10
8000002C	90282333	and x5,x5,1	11
80000030	9048fb3	sub x31,x31,x4	12
80000034	9038233	sub x4,x31,x3	13
80000038	9112263	slt x5,x3,x31	14
8000003C	9728233	addi x5,x5,-1	15
80000040	90327233	and x4,x4,x5	16
80000044	9048fb3	sub x31,x31,x4	17
80000048	9038233	addi x4,x4,-1	18
8000004C	f0a14e3	bne x20,x0,80000014 <>	19
80000050<-p2>	9000a001	jal x0,80000059 <>	20
80000054	9000a001	<invalid operation>	21
80000058	9000a002	<invalid operation>	22
8000005C	9000a003	<invalid operation>	23
80000060	9000a004	<invalid operation>	24
80000064	9000a005	<invalid operation>	25
80000068	9000a006	<invalid operation>	26
80000072	9000a007	<invalid operation>	27
80000076	9000a008	<invalid operation>	28
80000080	9000a009	addi x1,x1,8	29
80000084	90282333	sub x4,x31,x2	30
80000088	9112263	slt x5,x2,x31	31
80000092	9728233	addi x5,x5,-1	32
80000096	9048fb3	and x4,x4,x5	33
800000A0	90327233	sub x31,x31,x4	34
800000A4	9038233	addi x4,x4,-1	35
800000A8	90327233	and x4,x4,x5	36
800000B2	90282333	sub x31,x31,x4	37
800000B6	9038233	addi x4,x4,-1	38
800000C0	90327233	and x4,x4,x5	39
800000C4	90282333	sub x31,x31,x4	40
800000C8	9038233	addi x4,x4,-1	41
800000D2	90327233	and x4,x4,x5	42
800000D6	90282333	sub x31,x31,x4	43
800000D8	9038233	addi x4,x4,-1	44
800000E2	90327233	and x4,x4,x5	45
800000E6	90282333	sub x31,x31,x4	46
800000F0	9038233	addi x4,x4,-1	47
800000F4	90327233	and x4,x4,x5	48
800000F8	90282333	sub x31,x31,x4	49
800000FA	9038233	addi x4,x4,-1	50
800000FC	90327233	and x4,x4,x5	51
800000D0	90282333	sub x31,x31,x4	52
800000D4	9038233	addi x4,x4,-1	53
800000D8	90327233	and x4,x4,x5	54
800000E2	90282333	sub x31,x31,x4	55
800000E6	9038233	addi x4,x4,-1	56
800000F0	90327233	and x4,x4,x5	57
800000F4	90282333	sub x31,x31,x4	58
800000F8	9038233	addi x4,x4,-1	59
800000FA	90327233	and x4,x4,x5	60
800000D0	90282333	sub x31,x31,x4	61
800000D4	9038233	addi x4,x4,-1	62
800000D8	90327233	and x4,x4,x5	63
800000E2	90282333	sub x31,x31,x4	64
800000E6	9038233	addi x4,x4,-1	65
800000F0	90327233	and x4,x4,x5	66
800000F4	90282333	sub x31,x31,x4	67
800000F8	9038233	addi x4,x4,-1	68
800000FA	90327233	and x4,x4,x5	69
800000D0	90282333	sub x31,x31,x4	70
800000D4	9038233	addi x4,x4,-1	71
800000D8	90327233	and x4,x4,x5	72
800000E2	90282333	sub x31,x31,x4	73
800000E6	9038233	addi x4,x4,-1	74
800000F0	90327233	and x4,x4,x5	75
800000F4	90282333	sub x31,x31,x4	76
800000F8	9038233	addi x4,x4,-1	77
800000FA	90327233	and x4,x4,x5	78
800000D0	90282333	sub x31,x31,x4	79
800000D4	9038233	addi x4,x4,-1	80
800000D8	90327233	and x4,x4,x5	81
800000E2	90282333	sub x31,x31,x4	82
800000E6	9038233	addi x4,x4,-1	83
800000F0	90327233	and x4,x4,x5	84
800000F4	90282333	sub x31,x31,x4	85
800000F8	9038233	addi x4,x4,-1	86
800000FA	90327233	and x4,x4,x5	87
800000D0	90282333	sub x31,x31,x4	88
800000D4	9038233	addi x4,x4,-1	89
800000D8	90327233	and x4,x4,x5	90
800000E2	90282333	sub x31,x31,x4	91
800000E6	9038233	addi x4,x4,-1	92
800000F0	90327233	and x4,x4,x5	93
800000F4	90282333	sub x31,x31,x4	94
800000F8	9038233	addi x4,x4,-1	95
800000FA	90327233	and x4,x4,x5	96
800000D0	90282333	sub x31,x31,x4	97
800000D4	9038233	addi x4,x4,-1	98
800000D8	90327233	and x4,x4,x5	99
800000E2	90282333	sub x31,x31,x4	100
800000E6	9038233	addi x4,x4,-1	101
800000F0	90327233	and x4,x4,x5	102
800000F4	90282333	sub x31,x31,x4	103
800000F8	9038233	addi x4,x4,-1	104
800000FA	90327233	and x4,x4,x5	105
800000D0	90282333	sub x31,x31,x4	106
800000D4	9038233	addi x4,x4,-1	107
800000D8	90327233	and x4,x4,x5	108
800000E2	90282333	sub x31,x31,x4	109
800000E6	9038233	addi x4,x4,-1	110
800000F0	90327233	and x4,x4,x5	111
800000F4	90282333	sub x31,x31,x4	112
800000F8	9038233	addi x4,x4,-1	113
800000FA	90327233	and x4,x4,x5	114
800000D0	90282333	sub x31,x31,x4	115
800000D4	9038233	addi x4,x4,-1	116
800000D8	90327233	and x4,x4,x5	117
800000E2	90282333	sub x31,x31,x4	118
800000E6	9038233	addi x4,x4,-1	119
800000F0	90327233	and x4,x4,x5	120
800000F4	90282333	sub x31,x31,x4	121
800000F8	9038233	addi x4,x4,-1	122
800000FA	90327233	and x4,x4,x5	123
800000D0	90282333	sub x31,x31,x4	124
800000D4	9038233	addi x4,x4,-1	125
800000D8	90327233	and x4,x4,x5	126
800000E2	90282333	sub x31,x31,x4	127
800000E6	9038233	addi x4,x4,-1	128
800000F0	90327233	and x4,x4,x5	129
800000F4	90282333	sub x31,x31,x4	130
800000F8	9038233	addi x4,x4,-1	131
800000FA	90327233	and x4,x4,x5	132
800000D0	90282333	sub x31,x31,x4	133
800000D4	9038233	addi x4,x4,-1	134
800000D8	90327233	and x4,x4,x5	135
800000E2	90282333	sub x31,x31,x4	136
800000E6	9038233	addi x4,x4,-1	137
800000F0	90327233	and x4,x4,x5	138
800000F4	90282333	sub x31,x31,x4	139
800000F8	9038233	addi x4,x4,-1	140
800000FA	90327233	and x4,x4,x5	141
800000D0	90282333	sub x31,x31,x4	142
800000D4	9038233	addi x4,x4,-1	143
800000D8	90327233	and x4,x4,x5	144
800000E2	90282333	sub x31,x31,x4	145
800000E6	9038233	addi x4,x4,-1	146
800000F0	90327233	and x4,x4,x5	147
800000F4	90282333	sub x31,x31,x4	148
800000F8	9038233	addi x4,x4,-1	149
800000FA	90327233	and x4,x4,x5	150
800000D0	90282333	sub x31,x31,x4	151
800000D4	9038233	addi x4,x4,-1	152
800000D8	90327233	and x4,x4,x5	153
800000E2	90282333	sub x31,x31,x4	154
800000E6	9038233	addi x4,x4,-1	155
800000F0	90327233	and x4,x4,x5	156
800000F4	90282333	sub x31,x31,x4	157
800000F8	9038233	addi x4,x4,-1	158
800000FA	90327233	and x4,x4,x5	159
800000D0	90282333	sub x31,x31,x4	160
800000D4	9038233	addi x4,x4,-1	161
800000D8	90327233	and x4,x4,x5	162
800000E2	90282333	sub x31,x31,x4	163
800000E6	9038233	addi x4,x4,-1	164
800000F0	90327233	and x4,x4,x5	165
800000F4	90282333	sub x31,x31,x4	166
800000F8	9038233	addi x4,x4,-1	167
800000FA	90327233	and x4,x4,x5	168
800000D0	90282333	sub x31,x31,x4	169
800000D4	9038233	addi x4,x4,-1	170
800000D8	90327233	and x4,x4,x5	171
800000E2	90282333	sub x31,x31,x4	172
800000E6	9038233	addi x4,x4,-1	173
800000F0	90327233	and x4,x4,x5	174
800000F4	90282333	sub x31,x31,x4	175
800000F8	9038233	addi x4,x4,-1	176
800000FA	90327233	and x4,x4,x5	177
800000D0	90282333	sub x31,x31,x4	178
800000D4	9038233	addi x4,x4,-1	179
800000D8	90327233	and x4,x4,x5	180
800000E2	90282333	sub x31,x31,x4	181
800000E6	9038233	addi x4,x4,-1	182
800000F0	90327233	and x4,x4,x5	183
800000F4	90282333	sub x31,x31,x4	184
800000F8	9038233	addi x4,x4,-1	185
800000FA	90327233	and x4,x4,x5	186
800000D0	90282333	sub x31,x31,x4	187
800000D4	9038233	addi x4,x4,-1	188
800000D8	90327233	and x4,x4,x5	189
800000E2	90282333	sub x31,x31,x4	190
800000E6	9038233	addi x4,x4,-1	191
800000F0	90327233	and x4,x4,x5	192
800000F4	90282333	sub x31,x31,x4	193
800000F8	9038233	addi x4,x4,-1	194
800000FA	90327233	and x4,x4,x5	195
800000D0	90282333	sub x31,x31,x4	196
800000D4	9038233	addi x4,x4,-1	197
800000D8	90327233	and x4,x4,x5	198
800000E2	90282333	sub x31,x31,x4	199
800000E6	9038233	addi x4,x4,-1	200
800000F0	90327233	and x4,x4,x5	201
800000F4	90282333	sub x31,x31,x4	202
800000F8	9038233	addi x4,x4,-1	203
800000FA	90327233	and x4,x4,x5	204
800000D0	90282333	sub x31,x31,x4	205
800000D4	9038233	addi x4,x4,-1	206
800000D8	90327233	and x4,x4,x5	207
800000E2	90282333	sub x31,x31,x4	208
800000E6	9038233	addi x4,x4,-1	209
800000F0	90327233	and x4,x4,x5	210
800000F4	90282333	sub x31,x31,x4	211
800000F8	9038233	addi x4,x4,-1	212
800000FA	90327233	and x4,x4,x5	213
800000D0	90282333	sub x31,x31,x4	214
800000D4	9038233	addi x4,x4,-1	215
800000D8	90327233	and x4,x4,x5	216

ЗАКЛЮЧЕНИЕ

В результате работы были изучены принципы функционирования, построения и особенности архитектуры суперскалярных конвейерных микропроцессоров.

Были выполнены следующие задачи:

- 1) Изучена и скомпилирована программа на ассемблере по варианту;
- 2) Изучена временную диаграмму стадий выборки и диспетчеризацию команды по заданному адресу по варианту;
- 3) Изучена временную диаграмму стадий декодирования и планирования команды по заданному адресу по варианту;
- 4) Изучена временную диаграмму стадии выполнения команды по заданному адресу по варианту;
- 5) Составлены трассы исходной и оптимизированной программ, проведено их сравнение.

В результате оптимизации, не удалось уменьшить количество требуемых тактов для решения задачи, так как исходная программа была написана без конфликтов доступа к ресурсам и в ней не было задержек. В оптимизированной программе было уменьшено число сбросов конвейера из-за неудачных предсказаний блока предсказателя путём замены поиска максимума условным переходом на логические операции, что однако увеличило число тактов. Тем не менее для другого массива оптимизированная программа может быть эффективнее исходной.

Все цели и задачи лабораторной работы выполнены.