**Министерство образования Республики Беларусь**

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Проектирование цифровых систем на устройствах программируемой логики

# оТЧЕТ

по курсу лабораторных работ

«Реализация цифровых систем с микропрограммным управлением на основе классических вычислительных архитектур с использованием современных САПР цифровых устройств»

|  |  |  |
| --- | --- | --- |
| Выполнил |  | магистрант группы 956301  Лабоха А.К. |
| Проверил |  | профессор,  доктор технических наук  Иванюк А.А. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Минск, 2020

**1. Условие задания**

**Исходные данные**:

- Модели вычислительных устройств, имеющих стековую, регистровую и аккумуляторную архитектуру.

- Язык проектирования цифровой аппаратуры VHDL.

- САПР цифровых устройств: Active HDL, Xilinx ISE/Vivado.

**Задание**:

- Выбрать один из вычислительных алгоритмов для реализации (из предложенного списка).

- Разработать модель специализированного цифрового устройства без микропрограммного управления (hardware accelerator), реализующего выбранный алгоритм.

- Разработать архитектуру набора инструкций (ISA) для одной из перечисленных моделей вычислительных устройств (по выбору).

- Реализовать VHDL-проект вычислительного устройства.

- Реализовать бинарный исполняемый код на основе выбранного алгоритма для вычислительного устройства.

- Оценить ассемблерный и бинарный исполняемый код вычислительного алгоритма.

- Провести функциональное моделирование VHDL-проекта вычислительного устройств и оценить модельное время выполнения алгоритма.

- Провести RTL- и физический синтез VHDL-компонент вычислительных устройств для технологических норм ПЛИС Xilinx Artix-7. Провести сравнительный анализ аппаратурных затрат на реализацию вычислительных устройств с различными архитектурами.

- Подготовить отчет о выполненной работе.

**Поставленная задача**: нахождение числа ненулевых элементов целочисленного массива.

**2. Общие положения, принятые при проектировании**

Для всех архитектур была заданы по умолчанию следующие настройки.

Размер элементов RAM памяти: 8 бит

Количество элементов в RAM памяти: 26

Размер команды в ROM памяти: 4 бита

Размер операнды в ROM памяти: 8 бит (равен размеру элементов в RAM)

Количество элементов в RAM памяти: 25

Для всех трех архитектур был подготовлен код, который позволяет найти количество ненулевых элементов в массиве памяти. Границы массива (минимальный и максимальный адрес в памяти) передаются в RAM-памяти по заранее оговоренным адресам. В каждой архитектуре предварительно выполняются проверки:

- переданный адрес начала массива не меньше минимального допустимого;

- переданный адрес конца массива не больше максимально допустимого;

- переданный адрес начала массива не больше адреса конца массива;

Для всех архитектур приводятся инструкции и код, необходимый для решения поставленной задачи с выполнением описанных проверок. В разделе 6 также приводится сравнение времени решения поставленной задачи на различных архитектурах без выполнения проверок.

Результат выполнения (количество ненулевых элементов в памяти) должен быть записан в память по нулевому адресу; в случае наличия ошибок в заданном условии (заданы недопустимые границы массива) в память по адресу 1 устанавливается значение «1», которое указывает на наличие ошибок при выполнении поставленной задачи.

Для адекватного сравнения всех архитектуры, необходимо было найти количество ненулевых элементов в памяти в адресном диапазоне [7; 63].

Для оценки аппаратных элементов были введены интегральные коэффициенты для следующих категорий элементов:

BELS  1;

FlipFlops/Latches: 2

RAM32X1S: 64

Целевое устройство для синтеза:

FPGA XILINX xc7a100t, package csg324

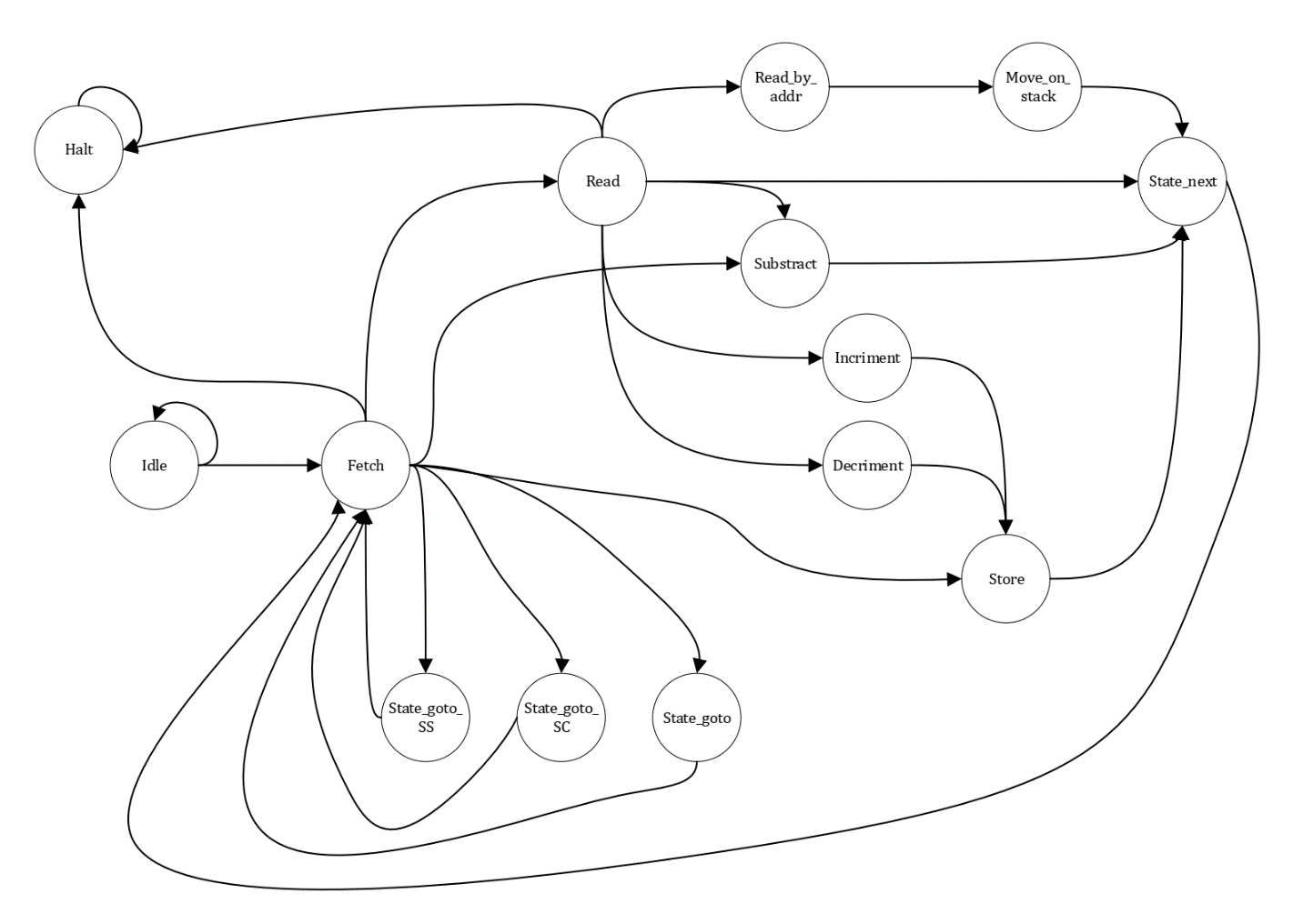
**3. Архитектура LIFO**

Таблица 3.1 – Набор инструкций для архитектуры LIFO

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Команда | Операнд | Действие | Состояний |
| 1 | PUSH\_R | ADDR | TOS <= RAM[ADDR] | 3 |
| 2 | INC\_R | ADDR | RAM[ADDR] <= RAM[ADDR] + 1 | 5 |
| 3 | DECR\_R | ADDR | RAM[ADDR] <= RAM[ADDR] - 1 | 5 |
| 4 | SUB\_L | LITERAL | ALU\_ZERO <= ((TOS – LITERAL) = 0)  ALU\_CARRY<= ((TOS – LITERAL) < 0) | 3 |
| 5 | SUB | ADDR | ALU\_ZERO <= (RAM[ADDR] – TOS = 0)  ALU\_CARRY<= (RAM[ADDR] – TOS < 0) | 4 |
| 6 | BTFSC | BIT | IF (BIT = 0 AND ALU\_CARRY = 0)  OR (BIT = 1 AND ALU\_ZERO = 0)  THEN IC <= IC + "10";  ELSE IC <= IC + "1"; | 2 |
| 7 | BTFSS | BIT | IF (BIT = 0 AND ALU\_CARRY = 1)  OR (BIT = 1 AND ALU\_ZERO = 1)  THEN IC <= IC + "10";  ELSE IC <= IC + "1"; | 2 |
| 8 | GOTO | ADDR | IC <= ADDR | 2 |
| 9 | POP | ADDR | RAM[ADDR] <= TOS | 3 |
| 10 | CLR\_R | ADDR | RAM[ADDR] <= 0 | 3 |
| 11 | LOAD\_R | ADDR | TOS <= RAM[RAM[ADDR]] | 5 |
| 12 | END |  | HALT execution | 2 |

Размерность ROM-памяти: 12 бит (4 бит на команду, 8 бит на операнду)

Содержимое ROM-памяти:

constant ROM\_data\_amount: ROM\_inner\_data := (

|  |  |
| --- | --- |
| 0 => CMD\_CLR\_R & "00000000",  1 => CMD\_CLR\_R & "00000001",  2 => CMD\_PUSH\_R & "00000011",  3 => CMD\_SUB\_L & "00000101",  4 => CMD\_BTFSC & "00000000",  5 => CMD\_GOTO & "00011001",  6 => CMD\_PUSH\_R & "00000100",  7 => CMD\_SUB\_L & "01000000",  8 => CMD\_BTFSS & "00000000",  9 => CMD\_GOTO & "00011001",  10 => CMD\_PUSH\_R & "00000011",  11 => CMD\_SUB & "00000100",  12 => CMD\_BTFSC & "00000000",  13 => CMD\_GOTO & "00011001",  14 => CMD\_PUSH\_R & "00000100",  15 => CMD\_POP & "00000010",  --cycle  16 => CMD\_LOAD\_R & "00000010",  17 => CMD\_SUB\_L & "00000000",  18 => CMD\_BTFSS & "00000001",  19 => CMD\_INC\_R & "00000000",  20 => CMD\_DECR\_R & "00000010",  21 => CMD\_SUB & "00000010",  22 => CMD\_BTFSS & "00000000",  23 => CMD\_GOTO & "00010000",  24 => CMD\_GOTO & "00011010",  25 => CMD\_INC\_R & "00000001",  26 => CMD\_END & "00000000", | 3 т.  3 т.  3 т.  3 т.  2 т.  2 т.  3 т.  3 т.  2 т.  2 т.  3 т.  4 т.  2 т.  2 т.  3 т.  3 т.  5 т.  3 т.  2 т.  5 т.  5 т.  4 т.  2 т.  2 т.  2 т.  5 т.  2 т. |

others => (others => '0')

);

Содержимое RAM-памяти:

constant RAM\_data\_amount: RAM\_inner\_data := (

0 => "00000000", --result

1 => "00000000", --error\_code

2 => "00000000", --current address

3 => "00000111", --min\_address

4 => "00111111", --max\_address

6 => "00110000",

7 => "00010011",

10 => "00000111",

11 => "10000000",

12 => "00001010",

others => (others => '0')

);

Основные метрики

Время выполнения: 6862,5 – 12,5 = 6850 нс

Частота тактового генератора: 200 МГц

Время выполнения в системных циклах: 1370 циклов

Среднее время выполнения инструкции 80/27=2.96 [3] с.ц.

15 нс на одну инструкцию при частоте 200 МГц

MIPS при 200 МГц: 66,67

0,(3) инструкции на системный цикл

BELS : 459

FlipFlops/Latches : 123

RAMS : 8

Условная оценка аппаратных затрат:

459\*1 + 123\*2 + 8\*64 = 1217

Максимальная частота: 274.198MHz

MIPS при этой частоте: 91,39 MIPS

Алгоритм в машинных циклах. Для проверки массива длинной N понадобится: 43 + 28\*N + 2 = 45 + 28\*N.

Оценка эффективности использования ROM:

Общий объем памяти: 32\*12 бит.

Микропрограмма занимает: 27\*12 бит

Использование памяти: 84.375 %

Микропрограмма занимает 324 бита, но фактически используется только 281 бит (с учетом неиспользуемых полей аргументов, в том числе 7 бит на каждую проверку BTFSx).

Эффективность двоичного кода микропрограммы составляет 86,7%.

**4. Архитектура Accumulator**

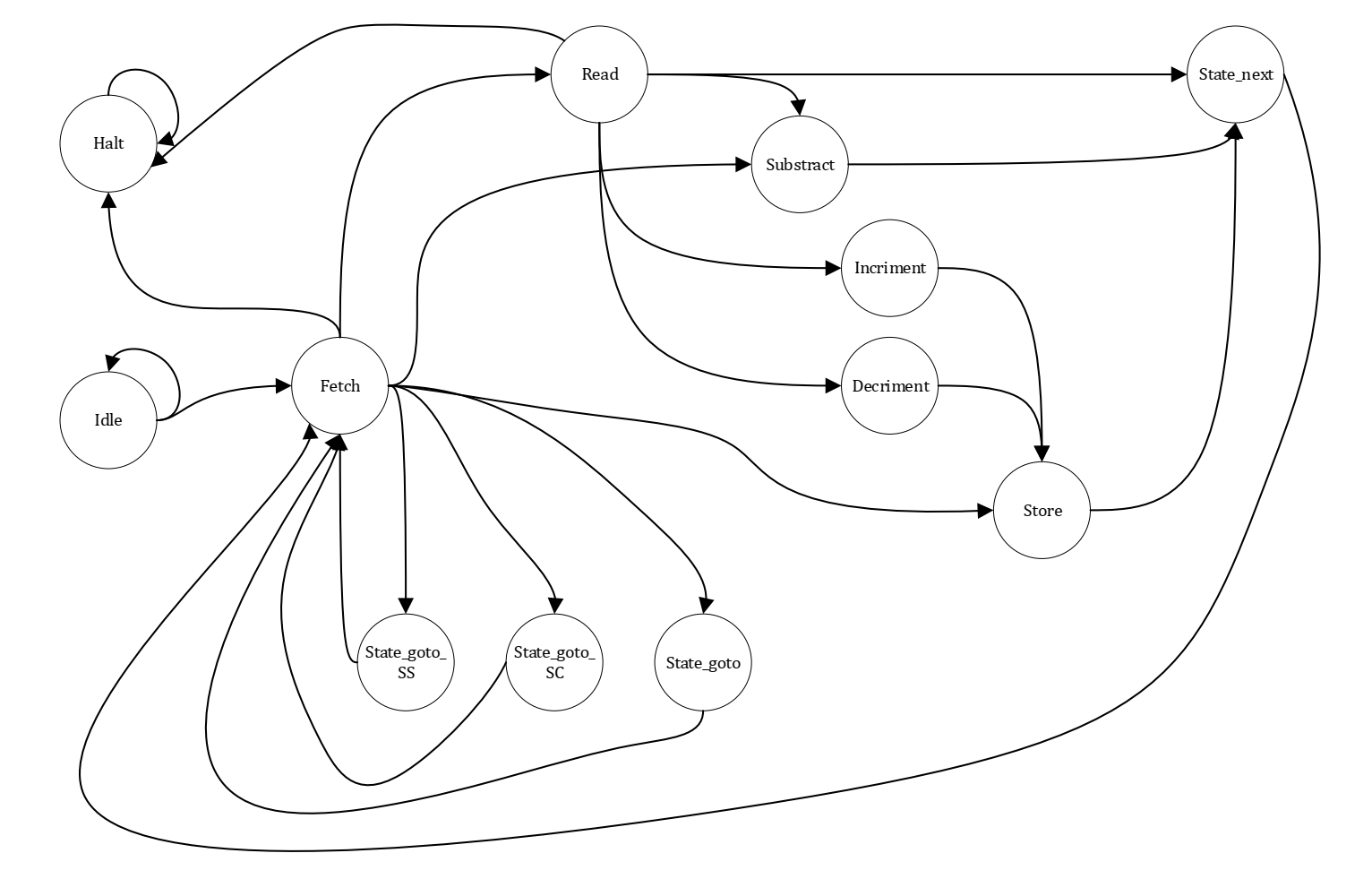


Таблица 4.1 – Набор инструкций для архитектуры Accumulator

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Команда | Операнд | Действие | Состояний |
| 1 | CLRF | ADDR | RAM[ADDR] <= (others => '0') | 3 |
| 2 | MOVF | ADDR | WREG <= RAM[ADDR] | 3 |
| 3 | MOVWF | ADDR | RAM[ADDR] <= WREG | 3 |
| 4 | SUBLW | LITERAL | ALU\_ZERO <= ((WREG – LITERAL) = 0)  ALU\_CARRY<= ((WREG – LITERAL) < 0) | 3 |
| 5 | SUBFW | ADDR | ALU\_ZERO <= (WREG – RAM[ADDR] = 0)  ALU\_CARRY<= (WREG – RAM[ADDR] < 0) | 4 |
| 6 | BTFSC | BIT | IF (BIT = 0 AND ALU\_CARRY = 0)  OR (BIT = 1 AND ALU\_ZERO = 0)  THEN IC <= IC + "10";  ELSE IC <= IC + "1"; | 2 |
| 7 | BTFSS | BIT | IF (BIT = 0 AND ALU\_CARRY = 1)  OR (BIT = 1 AND ALU\_ZERO = 1)  THEN IC <= IC + "10";  ELSE IC <= IC + "1"; | 2 |
| 8 | GOTO | ADDR | IC <= ADDR | 2 |
| 9 | INCF | ADDR | RAM[ADDR] <= RAM[ADDR] + 1 | 5 |
| 10 | DECF | ADDR | RAM[ADDR] <= RAM[ADDR] – 1 | 5 |
| 11 | END | ADDR | HALT execution | 2 |

Размерность ROM-памяти: 12 бит (4 бит на команду, 8 бит на операнду)

Содержимое ROM-памяти:

constant ROM\_data\_amount: ROM\_inner\_data := (

|  |  |
| --- | --- |
| 0 => CMD\_CLRF & "00000000",  1 => CMD\_CLRF & "00000001",  2 => CMD\_MOVF & "00000101",  3 => CMD\_SUBLW & "00000111",  4 => CMD\_BTFSC & "00000000",  5 => CMD\_GOTO & "00011100",  6 => CMD\_MOVF & "00000110",  7 => CMD\_SUBLW & "01000000",  8 => CMD\_BTFSS & "00000000",  9 => CMD\_GOTO & "00011100",  10 => CMD\_MOVF & "00000110",  11 => CMD\_SUBFW & "00000101",  12 => CMD\_BTFSC & "00000000",  13 => CMD\_GOTO & "00011100",  14 => CMD\_MOVF & "00000110",  15 => CMD\_MOVWF & "00000100",  -- cycle  16 => CMD\_MOVF & "00000100",  17 => CMD\_MOVWF & FSR\_register,  18 => CMD\_MOVF & INDF\_register,  19 => CMD\_SUBLW & "00000000",  20 => CMD\_BTFSS & "00000001",  21 => CMD\_INCF & "00000000",  22 => CMD\_DECF & "00000100",  23 => CMD\_MOVF & "00000100",  24 => CMD\_SUBFW & "00000101",  25 => CMD\_BTFSS & "00000000",  26 => CMD\_GOTO & "00010000",  27 => CMD\_GOTO & "00011101",  28 => CMD\_INCF & "00000001",  29 => CMD\_END & "00000000", | 3 т.  3 т.  3 т.  3 т.  2 т.  2 т.  3 т.  3 т.  2 т.  2 т.  3 т.  4 т.  2 т.  2 т.  3 т.  3 т.  3 т.  3 т.  3 т.  3 т.  2 т.  5 т.  5 т.  3 т.  4 т.  2 т.  2 т.  2 т.  5 т.  2 т. |

others => (others => '0')

);

Содержимое RAM-памяти:

constant RAM\_data\_amount: RAM\_inner\_data := (

0 => "00000000", --result

1 => "00000000", --error\_code

--2 and 3 addresses reserved for FSR and INDF

4 => "00000000", --current address

5 => "00000111", --min\_address

6 => "00111111", --max\_address

--data array 0 to 7

7 => "00010011",

10 => "00000111",

11 => "10000000",

12 => "00001010",

others => (others => '0')

);

Основные метрики

Время выполнения: 8857,5 – 12,5 = 8845 нс

Частота тактового генератора: 200 МГц

Время выполнения в системных циклах: 1769 циклов

Среднее время выполнения инструкции 87/30=2.9 [3] с.ц.

15 нс на одну инструкцию при частоте 200 МГц

MIPS при 200 МГц: 66,67

0,(3) инструкции на системный цикл

BELS : 167

FlipFlops/Latches : 70

RAMS : 8

Условная оценка аппаратных затрат:

169\*1 + 70\*2 + 8\*64 = 821

Максимальная частота: 272.546MHz

MIPS при этой частоте: 90,84 MIPS

Алгоритм в машинных циклах. Для проверки массива длинной N понадобится: 43 + 35\*N + 2 = 45 + 35\*N.

Оценка эффективности использования ROM:

Общий объем памяти: 32\*12 бит.

Микропрограмма занимает: 30\*12 бит

Использование памяти: 93.75 %

Микропрограмма занимает 360 бит, но фактически используется только 317 бит (с учетом неиспользуемых полей аргументов, в том числе 7 бит на каждую проверку BTFSx).

Эффективность двоичного кода микропрограммы составляет 88,05%.

**5. Архитектура GPR**

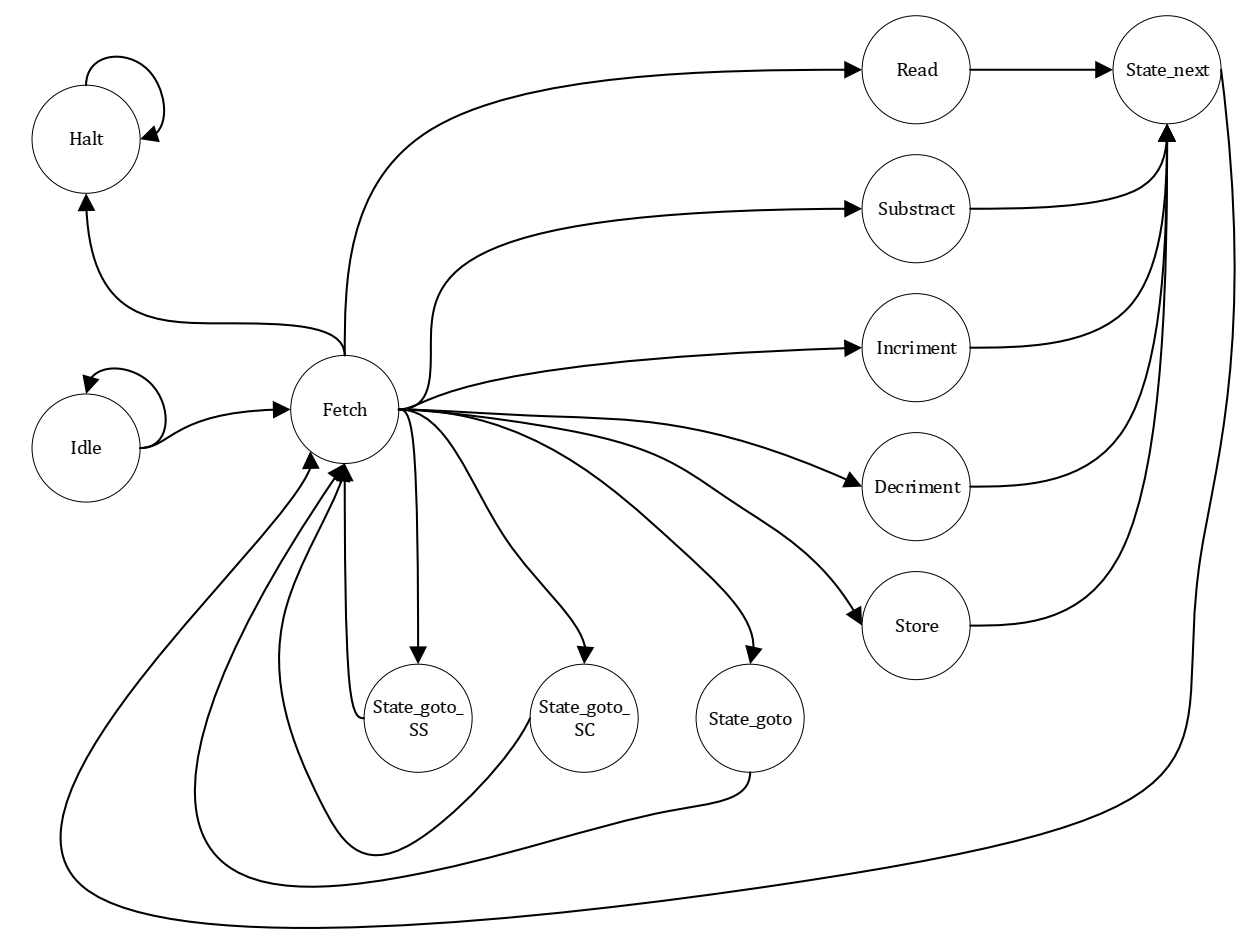


Таблица 5.1 – Набор инструкций для архитектуры GPR

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Команда | Операнд 1 | Операнд 2 | Действие | Состояний |
| 1 | MOVF | ADDR | REGx | REGx <= RAM[ADDR] | 3 |
| 2 | MOVLF | ADDR | LITERAL | RAM[ADDR] <= LITERAL | 3 |
| 3 | MOVRF | ADDR | REGx | RAM[ADDR] <= REGx | 3 |
| 4 | SUBLR | LITERAL | REGx | ALU\_ZERO <= ((REGx – LITERAL) = 0)  ALU\_CARRY<= ((REGx – LITERAL) < 0) | 3 |
| 5 | SUBRR | REGx | REGy | ALU\_ZERO <= (REGx – REGy = 0)  ALU\_CARRY<= (REGx – REGy < 0) | 3 |
| 6 | BTFSC\_GO | BIT | ADDR | IF (BIT = 0 AND ALU\_CARRY = 0)  OR (BIT = 1 AND ALU\_ZERO = 0)  THEN IC <= IC + "1";  ELSE IC <= ADDR; | 2 |
| 7 | BTFSS\_GO | BIT | ADDR | IF (BIT = 0 AND ALU\_CARRY = 1)  OR (BIT = 1 AND ALU\_ZERO = 1)  THEN IC <= IC + "1";  ELSE IC <= ADDR; | 2 |
| 8 | BTFSC\_INC | BIT | REGx | IF (BIT = 0 AND ALU\_CARRY = 0)  OR (BIT = 1 AND ALU\_ZERO = 0)  THEN REGx <= REGx + "1"; | 3 |
| 9 | GOTO | ADDR | – | IC <= ADDR | 2 |
| 10 | DECR | value | REGx | REGx <= REGx - value | 3 |
| 11 | MOVF\_REGADDR | REGx | REGy | REGy <= RAM[REGx] | 3 |
| 12 | END | – | – | HALT execution | 2 |

Размерность ROM-памяти: 20 бит (4 бит на команду, 8 бит на операнду №1, 8 бит на операнду №2)

Содержимое ROM-памяти:

constant ROM\_data\_amount: ROM\_inner\_data := (

|  |  |
| --- | --- |
| 0 => CMD\_MOVLF & "00000000" & "00000000",  1 => CMD\_MOVLF & "00000001" & "00000000",  2 => CMD\_MOVF & "00000010" & RA0,  3 => CMD\_SUBLR & "00000100" & RA0,  4 => CMD\_BTFSC\_GO & "00000000" & "00010010",  5 => CMD\_MOVF & "00000011" & RA1,  6 => CMD\_SUBLR & "01000000" & RA1,  7 => CMD\_BTFSS\_GO & "00000000" & "00010010",  8 => CMD\_SUBRR & RA1 & RA0,  9 => CMD\_BTFSC\_GO & "00000000" & "00010010",  --cycle  10 => CMD\_MOVF\_REGADDR & RA1 & RA2,  11 => CMD\_SUBLR & "00000000" & RA2,  12 => CMD\_BTFSC\_INC & "00000001" & RESULT,  13 => CMD\_DECR & "00000001" & RA1,  14 => CMD\_SUBRR & RA1 & RA0,  15 => CMD\_BTFSS\_GO & "00000000" & "00001010",  16 => CMD\_MOVRF & "00000000" & RESULT,  17 => CMD\_GOTO & "00010011" & NO\_REG,  18 => CMD\_MOVLF & "00000001" & "00000001",  19 => CMD\_END & "00000000" & NO\_REG, | 3 т.  3 т.  3 т.  3 т.  2 т.  3 т.  3 т.  2 т.  3 т.  2 т.  3 т.  3 т.  3 т.  3 т.  3 т.  2 т.  3 т.  2 т.  3 т.  2 т. |

others => (others => '0')

);

Содержимое RAM-памяти:

constant RAM\_data\_amount: RAM\_inner\_data := (

0 => "00000000", --result

1 => "00000000", --error\_code

2 => "00000111", --min\_address

3 => "00111111", --max\_address

--data array 0 to 7

7 => "00010011",

10 => "00000111",

11 => "10000000",

12 => "00001010",

others => (others => '0')

);

Основные метрики

Время выполнения: 4942,5 – 12,5 = 4930 нс

Частота тактового генератора: 200 МГц

Время выполнения в системных циклах: 986 циклов

Среднее время выполнения инструкции 54/20=2.7 [3] с.ц.

15 нс на одну инструкцию при частоте 200 МГц

MIPS при 200 МГц: 66,67

0,(3) инструкции на системный цикл

BELS : 257

FlipFlops/Latches : 92

RAMS : 8

Условная оценка аппаратных затрат:

257\*1 + 92\*2 + 8\*64 = 953

Максимальная частота: 282.087MHz

MIPS при этой частоте: 94,03 MIPS

Алгоритм в машинных циклах. Для проверки массива длинной N понадобится: 27 + 17\*N + 7 = 35 + 17\*N.

Оценка эффективности использования ROM:

Общий объем памяти: 32\*20 бит.

Микропрограмма занимает: 20\*20 бит

Использование памяти: 62.5 %

Микропрограмма занимает 640 бит, но фактически используется только 589 бит (с учетом неиспользуемых полей аргументов, в том числе 7 бит на каждую проверку BTFSx).

Эффективность двоичного кода микропрограммы составляет 92,03%.

**6. Сравнение архитектур**

Таблица 6.1 – Сравнение архитектур

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | LIFO | Accumulator | GPR |
| Количество инструкций | 12 | 11 | 12 |
| Строк кода, необходимых для решения задачи (с проверками) | 27 | 30 | 20 |
| Строк кода, необходимых для решения задачи (без проверок) | 14 | 16 | 12 |
| Время сортировки (57 элементов, с проверками), нс | 6850 | 8845 | 4930 |
| Время сортировки (121 элемент, с проверками), нс | 14430 | 18665 | 10450 |
| Время сортировки (121 элемент, без проверок), нс | 14310 | 18530 | 10365 |
| Среднее время выполнения инструкции | 2.9 | 2.96 | 2.7 |
| Масштабируемость алгоритма | 45 + 28\*N | 45 + 35\*N | 35 + 17\*N |
| BELS | 459 | 167 | 257 |
| FlipFlops/Latches | 123 | 70 | 92 |
| RAMS | 8 | | |
| Условная оценка аппаратных затрат | 1217 | 821 | 953 |
| Максимальная частота, MHz | 274.198 | 272.546 | 282.087 |
| MIPS (при максимальной частоте) | 91,39 | 90,84 | 94,03 |

По результатам проведенного анализа можно сделать вывод, что наиболее оптимальной архитектурой является архитектура регистров общего назначения (GPR). Это обусловлено тем, что при выполнении задачи значительная доля машинного времени приходится работу с памятью (преимущественно чтение значений); в указанной архитектуре эти операции выполняются напрямую, без использования промежуточных элементов, что позволяет получить наименьшее время выполнения. При этом архитектура GPR требует лишь немного больше аппаратных ресурсов по сравнению с аккумуляторной архитектурой, при этом увеличивая скорость выполнения задачи примерно на 44%.

Таким образом, именно архитектура регистров общего назначения наиболее компромиссным и приемлемым вариантом с точки зрения аппаратных затрат и времени выполнения поставленной задачи. Использование аккумуляторной архитектуры позволяет максимально снизить затраты на изготовление микроконтроллера, однако это приводит к уменьшению его эффективности. Стековая архитектура является компромиссным вариантом с точки зрения эффективности, однако требует гораздо больших вложений в аппаратную часть.

Также хотелось бы обратить внимание, что при достаточно больших размерах сортируемого массива для оценки времени работы кода на выбранной архитектуре допустимо использовать оценку алгоритма в машинных циклах. В частности, соотношение времени сортировки массива размером 121 элемент тремя архитектурами примерно соответствует соотношению коэффициентов при множителе N в оценке масштабируемости алгоритма.

Таблица 6.2 – Сравнение времени выполнения алгоритма при достаточно большом числе элементов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Критерий | LIFO | Accumulator | GPR |
| 1 | Время сортировки (121 элемент, без проверок), нс | 14310 | 18530 | 10365 |
| 2 | Масштабируемость алгоритма (коэффициент при множителе N) | 28 | 35 | 17 |
| 3 | Коэффициент при множителе N, полученный по времени сортировки | 23,47 | 30,1 | 17 |
| 4 | Диапазон колебаний коэффициента при множителе N | 23 –28 | 30 – 35 | 17 |

Если пропорционально изменять время сортировки, то можно получить практическое значение этого коэффициента, приняв коэффициент для архитектуры GPR равным 17, т.е. коэффициенту при множителе N (строка 3 таблицы 6.2). Можно заметить, что практические значения коэффициента для других архитектур оказались меньше расчетных. Это объясняется тем, что в исходном массиве из 121 элементов только 4 были ненулевыми; поэтому увеличение числа ненулевых элементов – операцию увеличения значения в памяти – необходимо было выполнить только 4 раза. В коде архитектур LIFO и Accumulator этой операции соответствуют команды INC\_R и INC соответственно, время выполнения которых как раз и составляет 5 тактов. Благодаря этому можно оценить примерное фактическое соотношение времени выполнения различных архитектур в случае отсутствия проверок на корректность диапазона адресов массива (строка 4 таблицы 6.2). В случае наличия проверок необходимо дополнительно суммировать время на выполнение проверок до цикла.