Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.04 Программная инженерия
Дисциплина «Функциональная схемотехника»

Лабораторная работа №3

Вариант FIFO

Выполнили:
Комягин Д.А.
Чмурова М.В.
Кузенина В.Н.
P3332

Преподаватель:
Кустарёв П.В.

Санкт-Петербург
2024 г.

Задание

В лабораторной работе вам предлагается разобраться во внутреннем устройстве простейшего процессорного ядра архитектуры RISC-V. Результатом изучения микроархитектуры процессорного ядра и системы команд RISC-V станут ваши функциональные и нефункциональные модификации ядра.

Основное задание:

1. Модифицировать процессорное ядро, в соответствии с вашим вариантом;
2. Подготовить тестовое окружение системного уровня и убедиться в корректности вашей реализации путём запуска симуляционных тестов.

Примечание:

При непосредственном описании ваших модификаций в коде проекта, запрещено использовать несинтезируемые конструкции и арифметические операции, отличные от сложения и вычитания (то есть, умножение, деление и возведение в степень реализуйте сами посредством описания любого, понравившегося вам, алгоритма). Однако, в тестовом окружении использовать несинтезируемые конструкции и всевозможные арифметические операции можно (и даже нужно).

Варианты с буффером/очередью должны реализовать две команды.

Первая команда - **push xN** - должна загружать данные из младшей части регистра **xN** в буффер/очередь.

Вторая команда - **pop xN** - должна выгружать данные в младшую часть регистра **xN**.

Микроархитектурная диаграмма

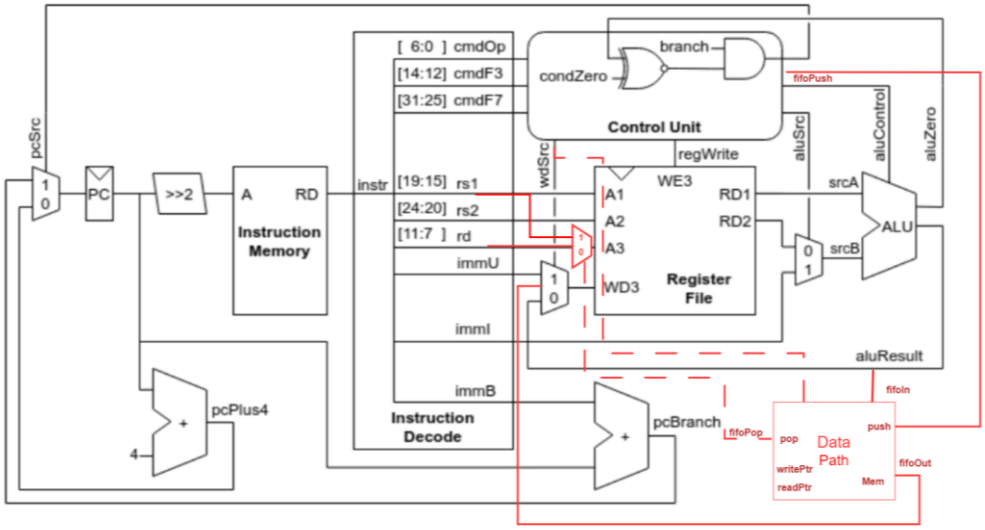


Схема 1. Микроархитектура с модулем fifo

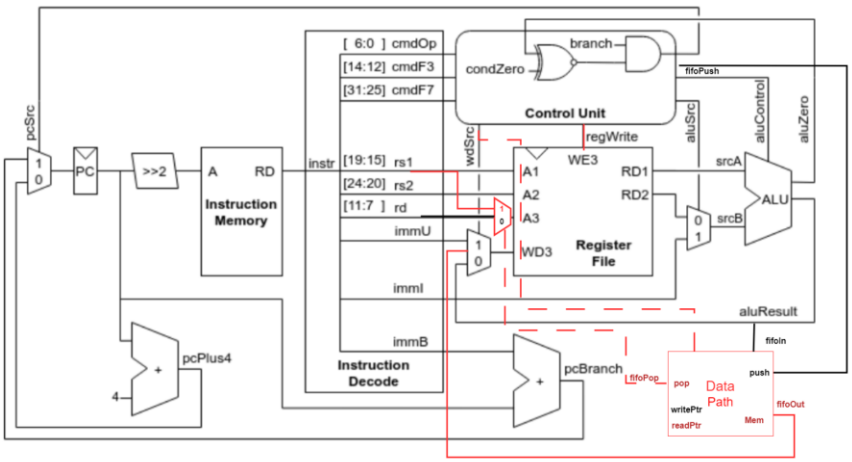


Схема 2. Выделенные пути при команде pop (lw)

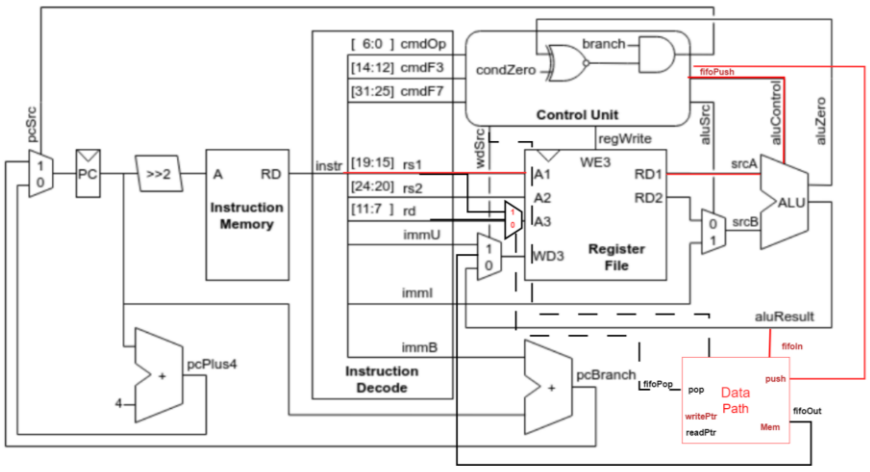


Схема 2. Выделенные пути при команде push (sw)

instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero	fifoPop	fifoPush
PUSH	.0100011	.010	???	.0	101	0	0	0	0	0	wdSrc	1
POP	.0000011	.010	???	.0	.000	2	1	0	0	0	wdSrc	0

Описание микроархитектурной схемы

Основные компоненты схемы:

1. PC, Instruction Memory, Instruction Decoder, Register File, ALU, Сумматоры и Мультиплексоры
2. Control Unit - интерпретирует управляющие сигналы
3. Data Path- используется для хранения данных и содержит логику работы fifo

- Однотактная реализация
- Словная адресация памяти команд
- 9 инструкций: add, or, srl, sltu, sub, addi, lui, beq, bne

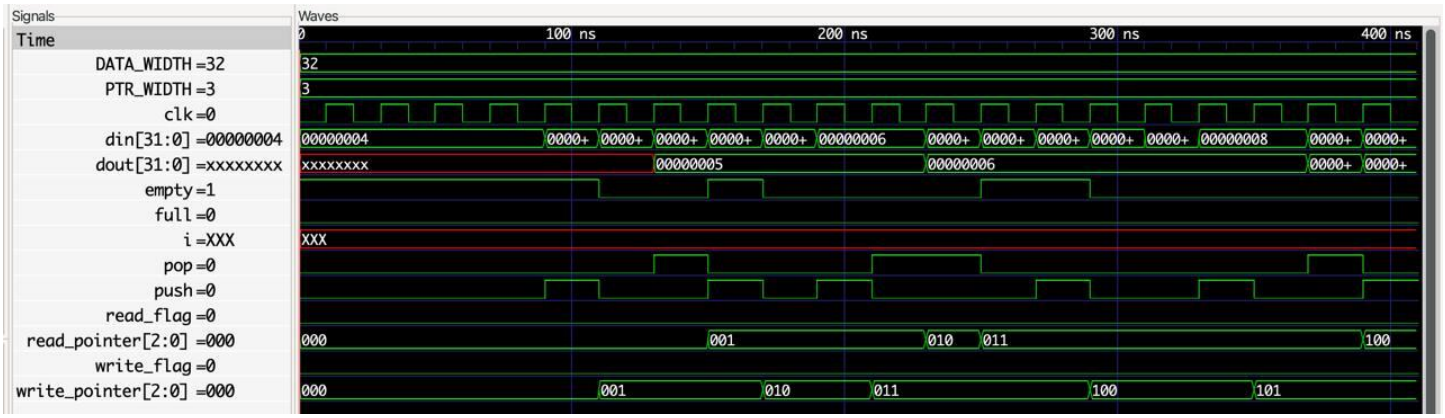
Описание алгоритма функционирования микропроцессорного ядра в части исполнения инструкций;

1. Data Path:
    - Содержит счетчики для контроля памяти read_ptr и write_ptr
    - Флаги для контроля команды pop и push, read_flag и write_flag
    - Область памяти
2. Обновление Control Unit:
    - Распознаёт команды *push* и *pop* на основе кодов операций opcode
    - Генерирует управляющие сигналы *fifoPush* (для реализации записи в fifo - push) и *wdSrc* (для выбора источника данных для записи в регистр при pop)
3. Обновление управляющих сигналов
    - Мультиплексор *wdSrc*: управляет выбором источника данных для записи в регистр (ALU, константы, данные из FIFO)

- Контроль записи (*fifoPush*): активирует механизм записи в стек.

4. Реализация инструкций:
   - sw (push) - данные из ALU передаются в стек (fifo), генерируется сигнал *fifoPuh* для записи.
   - lw (pop) - данные извлекаются из стека и записываются в регистр, управляющий сигнал *wdSrc* указывает на источник данных (fifo).

Временная диаграмма



Основная программа для тестирования

```
 .text

start:      li a0, 1              # t0 = 1

push_full:  li a0, 1              # t0 = 1
            sw a0, 0(a0)          # push t0
            li a0, 2              # t0 = 2
            sw a0, 0(a0)          # push t0
            li a0, 3              # t0 = 3
            sw a0, 0(a0)          # push t0
            li a0, 4              # t0 = 4
            sw a0, 0(a0)          # push t0
            li a0, 5              # t0 = 5
            sw a0, 0(a0)          # push t0

pop_empty: lw a0, 0(a0)       # pop t0
            lw a0, 0(a0)          # pop t0
            lw a0, 0(a0)          # pop t0
            lw a0, 0(a0)          # pop t0
```

```
alternating: li a0, 6          # t0 = 6
        sw a0, 0(a0)           # push t0
        lw a0, 0(a0)           # pop t0
        li a0, 7               # t0 = 7
        sw a0, 0(a0)           # push t0
        lw a0, 0(a0)           # pop t0
        li a0, 8               # t0 = 8
        sw a0, 0(a0)           # push t0
        lw a0, 0(a0)           # pop t0

loop_test:   beq zero, zero, push_full
```

Результат работы

```
0  pc = 00000000 instr = 00100513 a0 = x   addi  $10, $0, 0x00000001
   1  pc = 00000000 instr = 00100513 a0 = 1   addi  $10, $0, 0x00000001
   2  pc = 00000000 instr = 00100513 a0 = 1   addi  $10, $0, 0x00000001
   3  pc = 00000000 instr = 00100513 a0 = 1   addi  $10, $0, 0x00000001
   4  pc = 00000000 instr = 00100513 a0 = 1   addi  $10, $0, 0x00000001
   5  pc = 00000004 instr = 00100513 a0 = 1   addi  $10, $0, 0x00000001
   6  pc = 00000008 instr = 00a52023 a0 = 1   push  $10
   7  pc = 0000000c instr = 00200513 a0 = 1   addi  $10, $0, 0x00000002
   8  pc = 00000010 instr = 00a52023 a0 = 2   push  $10
   9  pc = 00000014 instr = 00300513 a0 = 2   addi  $10, $0, 0x00000003
  10  pc = 00000018 instr = 00a52023 a0 = 3   push  $10
  11  pc = 0000001c instr = 00400513 a0 = 3   addi  $10, $0, 0x00000004
  12  pc = 00000020 instr = 00a52023 a0 = 4   push  $10
  13  pc = 00000024 instr = 00500513 a0 = 4   addi  $10, $0, 0x00000005
  14  pc = 00000028 instr = 00a52023 a0 = 5   push  $10
  15  pc = 0000002c instr = 00052503 a0 = 5   pop  $10
  16  pc = 00000030 instr = 00052503 a0 = 1   pop  $10
  17  pc = 00000034 instr = 00052503 a0 = 2   pop  $10
  18  pc = 00000038 instr = 00052503 a0 = 3   pop  $10
  19  pc = 0000003c instr = 00600513 a0 = 4   addi  $10, $0, 0x00000006
  20  pc = 00000040 instr = 00a52023 a0 = 6   push  $10
  21  pc = 00000044 instr = 00052503 a0 = 6   pop  $10
  22  pc = 00000048 instr = 00700513 a0 = 6   addi  $10, $0, 0x00000007
  23  pc = 0000004c instr = 00a52023 a0 = 7   push  $10
  24  pc = 00000050 instr = 00052503 a0 = 7   pop  $10
  25  pc = 00000054 instr = 00800513 a0 = 7   addi  $10, $0, 0x00000008
  26  pc = 00000058 instr = 00a52023 a0 = 8   push  $10
  27  pc = 0000005c instr = 00052503 a0 = 8   pop  $10
  28  pc = 00000060 instr = fa0002e3 a0 = 8   beq   $0, $0, 0xffffffa4 (-92)
  29  pc = 00000004 instr = 00100513 a0 = 8   addi  $10, $0, 0x00000001
  30  pc = 00000008 instr = 00a52023 a0 = 1   push  $10
  31  pc = 0000000c instr = 00200513 a0 = 1   addi  $10, $0, 0x00000002
  32  pc = 00000010 instr = 00a52023 a0 = 2   push  $10
  33  pc = 00000014 instr = 00300513 a0 = 2   addi  $10, $0, 0x00000003
  34  pc = 00000018 instr = 00a52023 a0 = 3   push  $10
```

```
 35  pc = 0000001c instr = 00400513 a0 = 3   addi  $10, $0, 0x00000004
 36  pc = 00000020 instr = 00a52023 a0 = 4   push  $10
 37  pc = 00000024 instr = 00500513 a0 = 4   addi  $10, $0, 0x00000005
 38  pc = 00000028 instr = 00a52023 a0 = 5   push  $10
 39  pc = 0000002c instr = 00052503 a0 = 5   pop  $10
 40  pc = 00000030 instr = 00052503 a0 = 1   pop  $10
 41  pc = 00000034 instr = 00052503 a0 = 2   pop  $10
 42  pc = 00000038 instr = 00052503 a0 = 3   pop  $10
 43  pc = 0000003c instr = 00600513 a0 = 4   addi  $10, $0, 0x00000006
 44  pc = 00000040 instr = 00a52023 a0 = 6   push  $10
 45  pc = 00000044 instr = 00052503 a0 = 6   pop  $10
 46  pc = 00000048 instr = 00700513 a0 = 6   addi  $10, $0, 0x00000007
 47  pc = 0000004c instr = 00a52023 a0 = 7   push  $10
 48  pc = 00000050 instr = 00052503 a0 = 7   pop  $10
 49  pc = 00000054 instr = 00800513 a0 = 7   addi  $10, $0, 0x00000008
 50  pc = 00000058 instr = 00a52023 a0 = 8   push  $10
 51  pc = 0000005c instr = 00052503 a0 = 8   pop  $10
 52  pc = 00000060 instr = fa0002e3 a0 = 8   beq   $0, $0, 0xffffffa4 (-92)
 53  pc = 00000004 instr = 00100513 a0 = 8   addi  $10, $0, 0x00000001
 54  pc = 00000008 instr = 00a52023 a0 = 1   push  $10
 55  pc = 0000000c instr = 00200513 a0 = 1   addi  $10, $0, 0x00000002
 56  pc = 00000010 instr = 00a52023 a0 = 2   push  $10
 57  pc = 00000014 instr = 00300513 a0 = 2   addi  $10, $0, 0x00000003
 58  pc = 00000018 instr = 00a52023 a0 = 3   push  $10
 59  pc = 0000001c instr = 00400513 a0 = 3   addi  $10, $0, 0x00000004
 60  pc = 00000020 instr = 00a52023 a0 = 4   push  $10
Timeout
```

Вывод

В ходе данной лабораторной работы в процессорное ядро SchoolRiscV была добавлена реализация команд push и pop с помощью ассемблерных команд lw и sw и блока data memory для хранения данных в памяти.

К достоинствам данной реализации можно отнести очевидную простоту за счет использования указателей и флагов для поддержания кругового буфера в fifo. Кроме того, очевидно, что данная реализация расширяет функциональность процессорного ядра, добавляя в него новые команды и новую логику для работы с ними. Интеграция fifo была произведена без значительного усложнения работы тракта данных и устройства управления - по сути новые сигналы были добавлены поверх реализации, не касаясь логики работы уже существующих команд. Таким образом, можно сделать вывод о хорошей интеграции fifo с уже существующим процессором.

Тем не менее, недостатками этой реализации является разделение отработки fifo на два блока - синхронный и асинхронный, что было сделано для обеспечения корректности работы команд. Это было необходимо, так как изменения указателей и флагов происходит в синхронном формате, но при этом чтение из памяти выводятся независимо от их состояния. Асинхронный доступ безопасен, так как данные в памяти обновляются строго в синхронном блоке, а указатели чтения и записи синхронизированы с тактовым сигналом.

Альтернативными решениями для реализации этих команд могли бы послужить более грамотное объединение всего кода в один синхронный always блок, что могло бы повысить предсказуемость работы fifo

Однако выбранное решения для реализации отличается простотой логики работы и демонстрирует корректную обработку при тестировании всех крайних случаев тестовой системы, так как альтернативные подходы

требуют более сложных условий и обработки сигналов. Кроме того, выбранный подход легко интегрируется в существующую архитектуру процессора.