

Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Архитектура вычислительных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе  
на тему:

“Разработка простейшего 256-битного SIMD-АЛУ”

Выполнил: студент группы  
953505  
Басенко Кирилл  
Александрович

Проверил:  
Леченко Антон Владимирович

Минск 2021

## Введение

Время неумолимо, объем и сложность данных, обрабатываемых современными персональными компьютерами, растут в геометрической прогрессии, предъявляя невероятные требования к микропроцессорам. Между тем, производительность вычислений, которая может быть достигнута за счет увеличения тактовой частоты микропроцессора, приближается к физическим пределам, что делает архитектурные решения более заметными. В связи с этим умные люди создали важную архитектурную - SIMD (одна инструкция для нескольких данных), которая представляет собой набор инструкций, которые могут повысить производительность приложения, позволяя выполнять базовые операции над несколькими элементами данных параллельно с меньшим количеством инструкций.

## Single instruction, multiple data

Устройство SIMD обеспечивает неконкурентную параллельность на уровне данных. Такое устройство будет выполнять одну команду для разных данных. Инструкции такого устройства будут обрабатывать входные данные за одну операцию.

К примеру: нам нужно соответственно сложить  $n$  чисел с другими  $n$  числами. Первое, что приходит в голову -- это сложить соответственные числа:

A0	+	B0	=	res0
A1	+	B1	=	res1
...	...	...	...	...
An	+	Bn	=	resn

Инструкция SIMD это выполнит так:

A0	+	B0	=	res0
A1		B1		res1
...		...		...
An		Bn		resn

## Повторение -- мать учения

*“Симулякр — это вовсе не то, что скрывает собой истину, —  
это истина, скрывающая, что её нет.  
Симулякр есть истина.”  
-Экклезиаст*

В качестве примеров:

### Basic-SIMD-Processor-Verilog-Tutorial от zslwyuan

Данный пример представляет собой реализацию SIMD CPU.

Представленные типы данных: упакованные 4-битные числа, упакованные байты, упакованные слова.

На вход в ALU идут два 16-битных оператора и флаги типа данных: Q -- упакованные 4-битные числа, O -- упакованные байты, H -- упакованные слова. На выходе слово.

Реализованные операции: сложение/вычитание, умножение, правый/левый сдвиг.

### Intel's MMX Technology

Данный пример представляет собой расширение архитектуры Intel.

Представленные типы данных: упакованные байты, упакованные слова, упакованные двойные слова, четверное слово.

На вход в инструкции идут два четверных слова, на выходе четверное слово.

Реализованные инструкции: сложение/вычитание, умножение/деление, правый/левый сдвиг, сравнение на больше/меньше/равенство, etc.

## Простейшее 256-битное SIMD-АЛУ

Выполнено на языке Python с использованием модуля nmigen.

Представленные типы данных: упакованные байты, упакованные слова, упакованные двойные слова, упакованные четверные слова.

На вход в АЛУ идут сигналы: два 256-битных операнда, команда, тип данных. На выходе 256-битный результат.

```
0
7 class ALU(Elaboratable):
8     def __init__(self):
9         super().__init__()
10        self.op1: Signal = Signal(256, reset=0)
11        self.op2: Signal = Signal(256, reset=0)
12        self.data_type: Signal = Signal(DATA_TYPES, reset=0)
13        self.func: Signal = Signal(ALU_FUNCS, reset=0)
14        self.res: Signal = Signal(256, reset=0)
15
```

В зависимости от операции выполняется соответствующая логика.

```
16 def elaborate(self, platform) -> Module:
17     m = Module()
18
19     if platform is None:
20         m.d.sync += Signal().eq(1)
21
22     with m.Switch(self.func):
23         with m.Case(ALU_FUNCS.ADD, ALU_FUNCS.SUB):
24             m.d.comb += list(self.addsub_logic_gen())
25         with m.Case(ALU_FUNCS.EQ):
26             m.d.comb += list(self.equal_logic_gen())
27         with m.Case(ALU_FUNCS.MORE, ALU_FUNCS.LESS):
28             m.d.comb += list(self.moreless_logic_gen())
29         with m.Case(ALU_FUNCS.SHR, ALU_FUNCS.SHL):
30             m.d.comb += list(self.sh_logic_gen())
31
32     return m
33
34 > def moreless_logic_gen(self) -> Assign: ...
115
116 > def sh_logic_gen(self) -> Assign: ...
189
190 > def equal_logic_gen(self) -> Assign: ...
265
266 > def addsub_logic_gen(self) -> Assign: ...
300
```

Определены константы для типов данных, команд:

```
CONSTS.py > ...
1  from enum import Enum
2
3  class ALU_FUNCS(Enum):
4      NONE = 0
5
6      ADD = 1
7      SUB = 2
8      MUL = 3
9      DIV = 4
10
11     EQ  = 5
12     MORE = 6
13     LESS = 7
14
15     SHL = 8
16     SHR = 9
17
18
19  class DATA_TYPES(Enum):
20      pckd_b = 0
21      pckd_w = 1
22      pckd_dw = 2
23      pckd_qw = 3
```

Так же созданы тесты для команд АЛУ:

```

ALU_TEST.py > ...
390
391 def alu_test(alu: ALU) -> Assign:
392     global s, f
393
394     yield from alu_moreless_test(alu)
395     yield from alu_addsub_test(alu)
396     yield from alu_equal_test(alu)
397     yield from alu_sh_test(alu)
398
399     print(f'{s = }\n{f = }')
400
401
402 def main():
403     alu = ALU()
404
405     sim = Simulator(alu)
406     with sim.write_vcd(open('out.vcd', 'w')):
407         sim.add_clock(1e-6)
408         sim.add_sync_process(lambda: (yield from alu_test(alu)))
409         sim.run()
410
411
412 if __name__ == '__main__':
413     main()

```

```

ALU_TEST.py > alu_test
1 from ALU import ALU
2 from nmigen.back.pysim import *
3 from CONSTS import ALU_FUNCS, DATA_TYPES
4 from Utils import to_formatted_hex
5
6 s: int = 0
7 f: int = 0
8 def alu_ut( alu: ALU,
9             func: ALU_FUNCS,
10             op1: str,
11             op2: str,
12             data_type: DATA_TYPES,
13             expected: str) -> Assign:
14     global s, f
15
16     yield alu.op1      .eq(int(op1, 16))
17     yield alu.op2      .eq(int(op2, 16))
18     yield alu.data_type.eq(data_type)
19     yield alu.func      .eq(func)
20
21     yield Settle()
22
23     res = to_formatted_hex((yield alu.res))
24
25     if res == expected:
26         s += 1
27     else:
28         print(f'WRONG:\n{op1 = }\n{op2 = }\n{func = }\n{res = }\n{data_type = }\n{expected = }\n\n')
29         f += 1
30
31
32 > def alu_sh_test(alu: ALU) -> Assign: ...
172
173
174 > def alu_moreless_test(alu: ALU) -> Assign: ...
244
245
246 > def alu_equal_test(alu: ALU) -> Assign: ...
302
303
304 > def alu_addsub_test(alu: ALU) -> Assign: ...
394
395

```





## Заключение

`“_,_,i = i[:] = 'я смотрю кино ', 'про себя, в котором ', [[]]”`  
-Хаски

В результате выполнения курсовой работы:

- Рассмотрены статьи/проекты по текущей теме
- Повторены принципы SIMD архитектуры.
- Создано простейшее 256-битное SIMD-АЛУ на языке Python с использованием стороннего модуля `nmigen`.
- Проверена работа АЛУ на созданных тестах.

## Список используемой литературы

- SIMD: <https://en.wikipedia.org/wiki/SIMD6>, лекции / теоретические сведения 1й ЛР.
- Работа с nmigen: <https://github.com/RobertBaruch/nmigen-tutorial>, <https://vivonomicon.com/2020/06/13/lets-write-a-minimal-risc-v-cpu-in-nmigen> [Электронный доступ 17.12.2021]

## Приложение

Исходный код: [https://github.com/Lirosk/256b\\_SIMD\\_ALU](https://github.com/Lirosk/256b_SIMD_ALU)