# VERY SIMPLE CPU

**by**
**Orhun Aka**

**CSE 224 Introduction to Digital Systems**
**Term Project Report**

**Yeditepe University**
**Faculty of Engineering**
**Department of Computer Engineering**
**2023**

# DETAILS OF THE PROJECT

## What is a CPU?

A CPU (Central Processing Unit) is an electronic component of a computer which performs processing by carrying out instructions of a program such as basic arithmetic, logical, control and input/output (I/O) operations. Often referred to as the "brain" of the computer, a CPU handles the instructions it receives from the hardware and software. A group of commands which is provided to the processor is called an instruction set. An example of an instruction set can be given as x86 instruction set, which is common among many processor.

## What is a Very Simple CPU?

VerySimpleCPU (will be referred as VSCPU in the rest of this report), is a simple CPU, that can only do 16 instructions that are used for arithmetic calculations, data transfer and program control. These 16 instructions can be used to write nearly every program in a way that would be recognized by VSCPU. The 16 instructions are written in assembly code and will be explained in detail in the following sections.

## Goal of the Project

The goal of this project is to design and implement a VerySimpleCPU (VSCPU), a basic and straightforward Central Processing Unit capable of performing a limited set of instructions for arithmetic calculations, data transfer, and program control. The VSCPU will be designed to support a specific instruction set comprising 16 instructions, which will be described in detail in the subsequent sections. By creating the VSCPU, we aim to provide a simplified CPU architecture that can effectively execute a wide range of programs written in assembly code compatible with the VSCPU instruction set. This project seeks to demonstrate the fundamental workings of a CPU by showcasing its ability to process instructions, perform basic operations, and handle input/output operations. The VSCPU will serve as a valuable educational tool, enabling users to gain practical insights into CPU functionality and programming concepts.

# INSTRUCTION SET ARCHITECTURE

Each VerySimpleCPU instruction word has a fixed length of 32-bit, and contains 4 important fields:

- [31:29] -> Opcode
- [28] -> IM (Immediate Bit)
- [27:14] -> Operand A
- [13:0] -> Operand B

Also, the VSCPU includes 4 registers:

- IW -> Instruction Word (32-bit)
- PC -> Program Counter (14-bit)
- (**Optional**) R1 -> General Purpose Register (32-bit)
- (**Optional**) R2 -> General Purpose Register (32-bit)

# INSTRUCTION SET

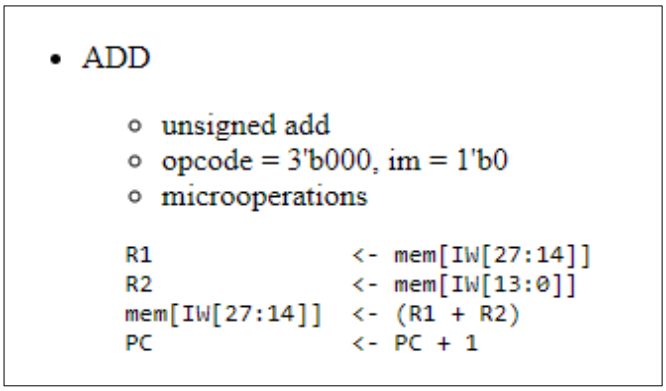
## ARITHMETIC & LOGIC INSTRUCTIONS

### *ADD Instruction:*

It is basically an addition operation. The opcode is: 3'b000, and the immediate bit is: 1'b0.

The steps are as follows:

1. Read two source operands from memory.
2. Add the values of the two operands together.
3. Store the result in the memory location that is given.
4. Update the necessary registers.

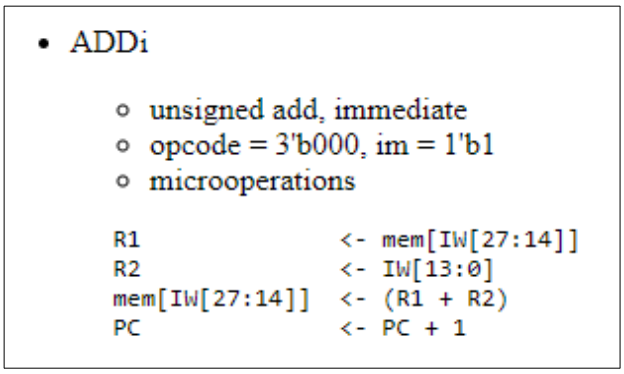


### *ADDi Instruction:*

It is an addition operation that is done immediately. The opcode is: 3'b000, and the immediate bit is: 1'b1.

The steps are as follows:

1. Read two source operands from memory and the register.
2. Add the values of the two operands together.
3. Store the result in the memory location that is given.
4. Update the necessary registers.

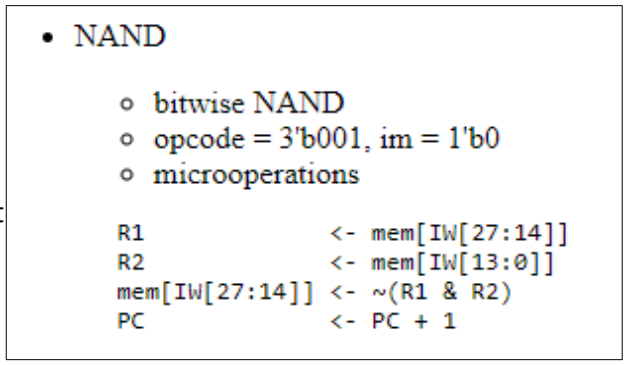



### *NAND Instruction:*

It is an instruction that returns the complement of an AND operation. The opcode is: 3'b001, and the immediate bit is: 1'b0.

The steps are as follows:

1. Read two source operands from memory.
2. AND the values, then take the complement.
3. Store the result in the memory location that is given.
4. Update the necessary registers.

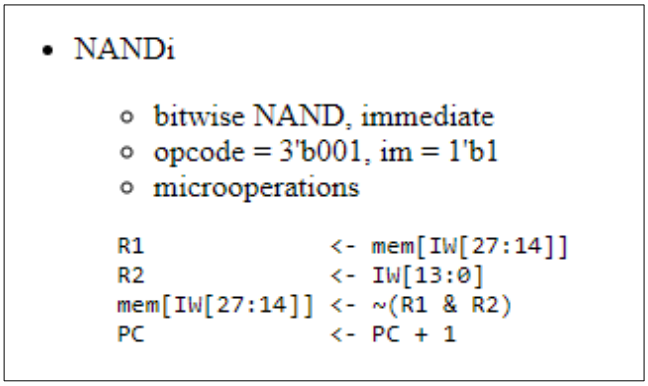



### *NANDi Instruction:*

It is an instruction that returns the complement of an AND operation, but is done immediately. The opcode is: 3'b001, and the immediate bit is: 1'b1.

The steps are as follows:

1. Read two source operands from memory and the register.
2. AND the values, then take the complement.
3. Store the result in the memory location that is given.
4. Update the necessary registers.



### *SRL Instruction:*

It is a Shift Right/Left operation . The opcode is: 3’b010, and the immediate bit is: 1’b0.

The steps are as follows:

1. Read two source operands from memory.
2. Complete the shift operation.
3. Store the result in the memory location that is given.
4. Update the necessary registers.

- SRL
  - shift right/left
  - opcode = 3'b010, im = 1'b0
  - microoperations

```
R1              <- mem[IW[27:14]]
R2              <- mem[IW[13:0]]
mem[IW[27:14]] <- (R2 < 32) ? (R1 >> R2) : (R1 << (R2-32))
PC              <- PC + 1
```

### *SRLi Instruction:*

It is a Shift Right/Left operation that is done immediately. The opcode is: 3’b010, and the immediate bit is: 1’b1.

The steps are as follows:

1. Read two source operands from memory and the register.
2. Complete the shift operation.
3. Store the result in the memory location that is given.
4. Update the necessary registers.

- SRLi
  - shift right/left, immediate
  - opcode = 3'b010, im = 1'b1
  - microoperations

```
R1              <- mem[IW[27:14]]
R2              <- IW[13:0]
mem[IW[27:14]] <- (R2 < 32) ? (R1 >> R2) : (R1 << (R2-32))
PC              <- PC + 1
```

### *LT Instruction:*

It is a compare and set operation. The opcode is: 3’b011, and the immediate bit is: 1’b0.

The steps are as follows:

1. Read two source operands from memory.
2. Compare the two operands.
3. Set the data of the memory accordingly.
4. Update the necessary registers.

- LT
  - compare and set
  - opcode = 3'b011, im = 1'b0
  - microoperations

```
R1              <- mem[IW[27:14]]
R2              <- mem[IW[13:0]]
mem[IW[27:14]] <- (R1 < R2) ? 1 : 0
PC              <- PC + 1
```

### *LTi Instruction:*

It is a compare and set operation that is done immediately. The opcode is: 3'b011, and the immediate bit is: 1'b1.

The steps are as follows:

1. Read two source operands from memory and the register.
2. Compare the two operands.
3. Set the data of the memory accordingly.
4. Update the necessary registers.

- LTi
  - compare and set, immediate
  - opcode = 3'b011, im = 1'b1
  - microoperations

```
R1              <- mem[IW[27:14]]
R2              <- IW[13:0]
mem[IW[27:14]] <- (R1 < R2) ? 1 : 0
PC              <- PC + 1
```

### *MUL Instruction:*

It is a multiplication operation. The opcode is: 3'b111, and the immediate bit is: 1'b0.

The steps are as follows:

1. Read two source operands from memory.
2. Multiply the two operands.
3. Store the data to the memory..
4. Update the necessary registers.

- MUL
  - unsigned multiply
  - opcode = 3'b111, im = 1'b0
  - microoperations

```
R1              <- mem[IW[27:14]]
R2              <- mem[IW[13:0]]
mem[IW[27:14]] <- (R1 * R2)
PC              <- PC + 1
```

### *MULi Instruction:*

It is a multiplication operation that is done immediately. The opcode is: 3'b111, and the immediate bit is: 1'b1.

The steps are as follows:

1. Read two source operands from memory and the register.
2. Multiply the two operands.
3. Store the data to the memory..
4. Update the necessary registers.

- MULi
  - unsigned multiply, immediate
  - opcode = 3'b111, im = 1'b1
  - microoperations

```
R1              <- mem[IW[27:14]]
R2              <- IW[13:0]
mem[IW[27:14]] <- (R1 * R2)
PC              <- PC + 1
```

## DATA TRANSFER INSTRUCTIONS

### *CP Instruction:*

It is a copy operation. The opcode is: 3'b100, and the immediate bit is: 1'b0.

The steps are as follows:

1. Read a source operand from memory.
2. Copy the data.
3. Update the necessary registers.

- CP
  - copy data
  - opcode = 3'b100, im = 1'b0
  - microoperations

```
R2              <- mem[IW[13:0]]
mem[IW[27:14]] <- R2
PC              <- PC + 1
```

### *CPi Instruction:*

It is a copy operation that is done immediately. The opcode is: 3'b100, and the immediate bit is: 1'b1.

The steps are as follows:

1. Read a source operand from memory.
2. Copy the data that is given in the IW.
3. Update the necessary registers.

- CPi
  - copy data, immediate
  - opcode = 3'b100, im = 1'b1
  - microoperations

```
R2              <- IW[13:0]
mem[IW[27:14]] <- R2
PC              <- PC + 1
```

### *CPI Instruction:*

It is a copy operation that is done indirectly. The opcode is: 3'b101, and the immediate bit is: 1'b0.

The steps are as follows:

1. Read a source operand from memory.
2. Copy the data.
3. Update the necessary registers.

- CPI
  - copy data indirect
  - opcode = 3'b101, im = 1'b0
  - microoperations

```
R1              <- mem[IW[13:0]]
R2              <- mem[R1]
mem[IW[27:14]] <- R2
PC              <- PC + 1
```

### *CPIi Instruction:*

It is a copy operation that is done indirectly and immediately. The opcode is: 3'b101, and the immediate bit is: 1'b1.

The steps are as follows:

1. Read a source operand from memory.
2. Copy the data that is given in the IW.
3. Update the necessary registers.

- CPIi
  - copy data indirect, immediate
  - opcode = 3'b101, im = 1'b1
  - microoperations

```
R1          <- mem[IW[27:14]]
R2          <- mem[IW[13:0]]
mem[R1]     <- R2
PC          <- PC + 1
```

## PROGRAM CONTROL INSTRUCTIONS

### *BZJ Instruction:*

It is a if branch zero, jump operation. The opcode is: 3'b110, and the immediate bit is: 1'b0.

The steps are as follows:

1. Read the source operands from memory.
2. Compare the second operand with zero.
3. Set the PC accordingly.
4. Update the necessary registers.

- BZJ
  - branch on zero
  - opcode = 3'b110, im = 1'b0
  - microoperations

```
R1          <- mem[IW[27:14]]
R2          <- mem[IW[13:0]]
PC          <- (R2 == 0) ? R1 : (PC + 1)
```

### *BZJi Instruction:*

It is a if branch zero, jump operation that is done immediately. The opcode is: 3'b110, and the immediate bit is: 1'b1.

The steps are as follows:

1. Read the source operands from memory and the register.
2. Add the two operands together.
3. Set the PC accordingly.
4. Update the necessary registers.

- BZJi
  - unconditional branch
  - opcode = 3'b110, im = 1'b1
  - microoperations

```
R1          <- mem[IW[27:14]]
R2          <- IW[13:0]
PC          <- (R1 + R2)
```

# CONCLUSION AND REPORT

## Design Summary

| verysimplecpu Project Status (05/31/2023 - 13:44:40) | | | |
|---|---|---|---|
| Project File: | VerySimpleCPU.xise | Parser Errors: | No Errors |
| Module Name: | verysimplecpu | Implementation State: | Synthesized |
| Target Device: | xc7a100t-3csg324 | • Errors: | No Errors |
| Product Version: | ISE 14.7 | • Warnings: | 0 Warnings |
| Design Goal: | Balanced | • Routing Results: | |
| Design Strategy: | Xilinx Default (unlocked) | • Timing Constraints: | |
| Environment: | System Settings | • Final Timing Score: | |

| Device Utilization Summary (estimated values) | | | [-] |
|---|---|---|---|
| Logic Utilization | Used | Available | Utilization |
| Number of Slice Registers | 83 | 126800 | 0% |
| Number of Slice LUTs | 964 | 63400 | 1% |
| Number of fully used LUT-FF pairs | 50 | 997 | 5% |
| Number of bonded IOBs | 82 | 210 | 39% |
| Number of BUFG/BUFGCTRLs | 1 | 32 | 3% |
| Number of DSP48E1s | 5 | 240 | 2% |

| Detailed Reports | | | | | [-] |
|---|---|---|---|---|---|
| Report Name | Status | Generated | Errors | Warnings | Infos |
| Synthesis Report | Current | Çar 31. May 13:44:40 2023 | 0 | 0 | 0 |
| Translation Report | | | | | |
| Map Report | | | | | |
| Place and Route Report | | | | | |
| Power Report | | | | | |
| Post-PAR Static Timing Report | | | | | |
| Bitgen Report | | | | | |

| Secondary Reports | | [-] |
|---|---|---|
| Report Name | Status | Generated |

# SYNTHESIS REPORT

==========

***  Advanced HDL Synthesis  ***

==========

==========

**Advanced HDL Synthesis Report**

**Macro Statistics**

# Multipliers : 2

32x14-bit multiplier : 1

32x32-bit multiplier : 1

# Adders/Subtractors : 6

14-bit adder : 2

15-bit subtractor : 1

32-bit adder : 2

32-bit subtractor : 1

# Registers : 80

Flip-Flops : 80

# Comparators : 4

14-bit comparator greater : 1

32-bit comparator greater : 3

# Multiplexers : 42

1-bit 2-to-1 multiplexer : 6

1-bit 32-to-1 multiplexer : 1

14-bit 2-to-1 multiplexer : 12

14-bit 6-to-1 multiplexer : 1

32-bit 2-to-1 multiplexer : 22

# Logic shifters : 4

32-bit shifter logical left : 2

32-bit shifter logical right : 2

# FSMs : 1

==========

==========

**Timing Report**

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
GENERATED AFTER PLACE-and-ROUTE.

**Clock Information:**

---

| Clock Signal | Clock buffer(FF name) | Load |
|---|---|---|
| clk | BUFGP | 87 |

Asynchronous Control Signals Information:

---

No asynchronous control signals found in this design

Timing Summary:

---

Speed Grade: -3

Minimum period: 2.540ns (Maximum Frequency: 393.670MHz)
Minimum input arrival time before clock: 3.679ns
Maximum output required time after clock: 6.818ns
Maximum combinational path delay: 7.229ns

Timing Details:

---

All values displayed in nanoseconds (ns)

==========

==========

***  Design Summary  ***

==========

Top Level Output File Name : VerySimpleCPU.ngc

**Primitive and Black Box Usage:**

---

# BELS : 1241

#  GND : 1

#  INV : 27

#  LUT1 : 32

#  LUT2 : 97

#  LUT3 : 142

#  LUT4 : 76

#  LUT5 : 139

#  LUT6 : 451

#  MUXCY : 144

#  MUXF7 : 12

#  VCC : 1

#  XORCY : 119

# FlipFlops/Latches : 83

#  FD : 33

#  FDE : 32

#  FDR : 3

#  FDRE : 15

# Clock Buffers : 1

#  BUFGP : 1

# IO Buffers : 81

#  IBUF : 34

#  OBUF : 47

# DSPs : 5

#  DSP48E1 : 5

**Device utilization summary:**

---

Selected Device : 7a100tcsg324-3

**Slice Logic Utilization:**

| | | | |
|---|---|---|---|
| Number of Slice Registers: | 83 | out of 126800 | 0% |
| Number of Slice LUTs: | 964 | out of 63400 | 1% |
| Number used as Logic: | 964 | out of 63400 | 1% |

**Slice Logic Distribution:**

| | | | |
|---|---|---|---|
| Number of LUT Flip Flop pairs used: | 997 | | |
| Number with an unused Flip Flop: | 914 | out of 997 | 91% |
| Number with an unused LUT: | 33 | out of 997 | 3% |
| Number of fully used LUT-FF pairs: | 50 | out of 997 | 5% |
| Number of unique control sets: | 6 | | |

**IO Utilization:**

| | | | |
|---|---|---|---|
| Number of IOs: | 82 | | |
| Number of bonded IOBs: | 82 | out of 210 | 39% |

**Specific Feature Utilization:**

| | | | |
|---|---|---|---|
| Number of BUFG/BUFGCTRLs: | 1 | out of 32 | 3% |
| Number of DSP48E1s: | 5 | out of 240 | 2% |

==========