Innopolis University

SYSTEM AND NETWORKING ENGINEERING

Essential Skills

LABORATORY REPORT 5

Computer Architecture

Student Name	**Student ID**
Sergey Grebennikov	47611

Lecturer:

Stanislav Litvinov

Submission Date : January 22, 2019

Contents

1 Introduction

In computer engineering, computer architecture is a set of rules and methods that describe the functionality, organization, and implementation of computer systems. Some definitions of architecture define it as describing the capabilities and programming model of a computer but not a particular implementation. In other definitions computer architecture involves instruction set architecture design, microarchitecture design, logic design, and implementation.

2 Digital Circuit

a) The digital circuit of an 4-bit incrementer is shown in the figure 1

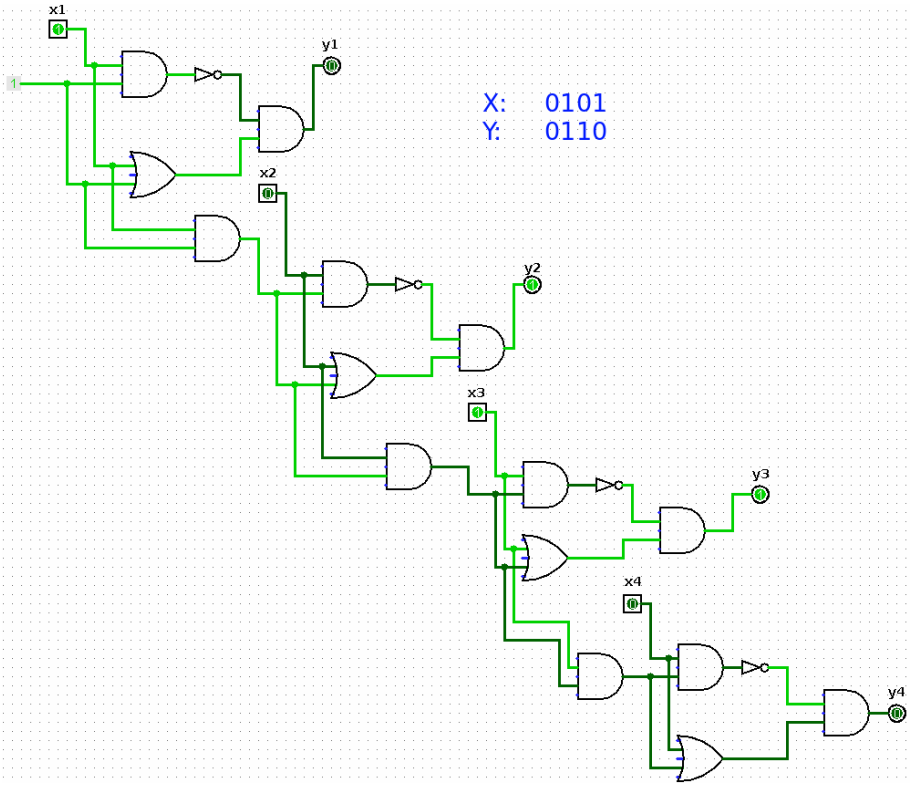

Figure 1: Incrementer with input values of **0101**

b) An incrementer with the minimal number of operation needed is shown in the figure 2

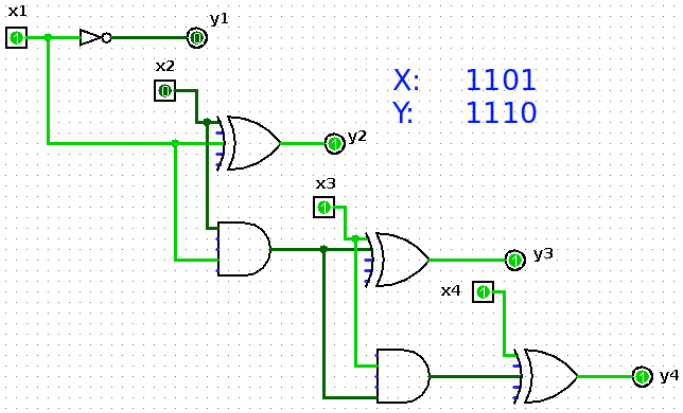

Figure 2: Simplified digital circuit of a 4-bit incrementer with input values of **1101**

c) A digital circuit of an incrementer with **XOR** elements represents the minimal depth of this circuit equal to two (**AND**, **XOR**).

3 Experiments Performance measurement

a) SIM-PL

- Downloaded the SIM-PL
- Downloaded the lab components
- Watched how works the following program:
  - **addition.wasm** - this program adds two numbers
  - **forloop.wasm** - this program shows how the loop works
  - **squares.wasm** - this program squares the first 16 numbers (from 0 to 15) and stores them in different parts of memory

b) Measured the following parameters:

- The number of instructions that have been executed:
  - Single Cycle Architecture:
    - addition.wasm - **6** instructions and **6** clock cycles
    - forloop.wasm - **45** instructions and **45** clock cycles
    - squares.wasm - **101** instructions and **101** clock cycles
  - Multi Cycle Architecture:
    - addition.wasm - **6** instructions and **24** clock cycles
    - forloop.wasm - **48** instructions and **192** clock cycles
    - squares.wasm - **119** instructions and **354** clock cycles
  - Pipelined Architecture:
    - addition.wasm - **15** instructions and **15** clock cycles
    - forloop.wasm - **135** instructions and **135** clock cycles
    - squares.wasm - **297** instructions and **297** clock cycles
- CPI: $\frac{ClockCycles}{InstructionCount}$
  - Single Cycle Architecture:
    - addition.wasm - **1**
    - forloop.wasm - **1**
    - squares.wasm - **1**
  - Multi Cycle Architecture:
    - addition.wasm - **4**
    - forloop.wasm - **4**
    - squares.wasm **3**
  - Pipelined Architecture:
    - addition.wasm - **1**
    - forloop.wasm - **1**
    - squares.wasm - **1**
- CPUtime: $ClockCycles * ClockCycleTime$
  - Single Cycle Architecture ($ClockCycleTime = 1ns$):
    - addition.wasm - ***6 ns***
    - forloop.wasm - ***45 ns***
    - squares.wasm - ***101 ns***

- Multi Cycle Architecture ($ClockCycleTime = 0.25ns$):
  - addition.wasm - ***6 ns***
  - forloop.wasm - ***48 ns***
  - squares.wasm - ***88.5 ns***
- Pipelined Architecture ($ClockCycleTime = 0.2ns$):
  - addition.wasm - ***3 ns***
  - forloop.wasm - ***27 ns***
  - squares.wasm - ***59.4 ns***

			Program		
			addition	forloop	squares
Architecture	Syngle Cycle	Instruction	6	45	101
		ClockCycles	6	45	101
		CPI	1	1	1
		ClockCyclesTime, *(ns)*	1		
		CPUtime, *(ns)*	**6**	**45**	**101**
	Multi Cycle	Instruction	6	48	119
		ClockCycles	24	192	354
		CPI	4	4	2.9
		ClockCyclesTime, *(ns)*	0.25		
		CPUtime, *(ns)*	**6**	**48**	**88.5**
	Pipelined	Instruction	15	135	297
		ClockCycles	15	135	297
		CPI	1	1	1
		ClockCyclesTime, *(ns)*	0.2		
		CPUtime, *(ns)*	**3**	**27**	**59.4**

Figure 3: computational parameters of different architectures

c) Explaining

- In accordance with the above measurements, the best performance is shown by the Pipelined Architecture, because it uses parallelism at the instruction level.
- A program counter is a processor register that points which command should be executed next.
- When executing instructions, the values of the program counter and memory changed.

d) Instructions:

- **ADDI** - adds an immediate value to the source register and saves its value in the destination register (e.g., **ADDI $1, $2, 0x3719**)
- **NOP** - instruction that does nothing and do not change the state of any of the programmer-accessible registers, status flags, or memory. It often takes a well-defined number of clock cycles to execute. (e.g., **NOP**)
- **LI** - loads a specific numeric value into the register (e.g., **LI $3, 17**)
- **BNE** - branches if the two registers are not equal (e.g., **BNE $5, $7, L1**)

e) A program that subtracts two numbers:

```
@include "Mips.wasm"

.data MyRegisters : REGISTERS
```

```
0x00 : WORD 0x00

.data MyMemory : DATAMEM
0x00 : WORD 0x00
.code MyCode: MIPS, MyMemory

LI  $4, 1
ADD $1, $4, $0

LI  $5, 2
ADD $2, $5, $0

SUB $6, $1, $2
ADD $3, $6, $0
```

After executing these instructions, the register **$6** will store the value -1, which is represented in the computer's memory as $0xFFFFFFFF$

f) The clock rate of a CPU is not a good benchmark for the performance of a CPU because the amount of work different CPUs can do in one cycle varies. For example, subscalar CPUs or use of parallelism can also affect the performance of the computer regardless of clock rate.

4 Assembler

The assembler program (for the Harvard machine) that converts memory data from little endian to big endian:

```
@include "Mips.wasm"

.data MyRegiters : REGISTERS
0x00 : WORD 0x00

.data MyMemory : DATAMEM
0x00 : WORD

.code MyCode: MIPS, MyMemory

LI  $1, 0x55aa
LI  $3, 0x8
SRL $2, $1, $3
LI  $3, 0x18
SLL $4, $1, $3
LI  $3, 0x10
SRL $4, $4, $3
ADD $1, $2, $4
```

In the result, the value in the register **$1** written in a little endian format is converted to a big endian format.

5 Conclusion

This laboratory work consists of tasks that allow to learn more about computer architecture.

6 References

[1] Computer Architecture. ing. Edwin Steffens Practical lecturer – Informatics Institute - Faculty of Science. UvA [https://staff.fnwi.uva.nl/e.h.steffens/Downloads/Arco/Innopolis/Lab%201%20-%20Introduction.pdf]

[2] SIM-PL instruction manual [https://staff.fnwi.uva.nl/e.h.steffens/Downloads/Arco/Innopolis/SIM-PL%20instruction%20manual%202.0%20-%20UK.pdf]

[3] Computer Architecture. Introduction. Taco Walstra. UvA. [https://staff.fnwi.uva.nl/t.r.walstra/innopolis/lect01.pdf]