**İSTANBUL TEKNİK ÜNİVERSİTESİ BİLGİSAYAR VE BİLİŞİM FAKÜLTESİ**

**FPGA ÜZERİNDE BİR İŞLEMCİ TASARIMI**

**Bitirme Ödevi Ara Rapor**

**Tuğrul Yatağan**

**040100117**

**Bölüm : Bilgisayar Mühendisliği**

**Anabilim Dalı : Bilgisayar Bilimleri**

**Danışman: Prof. Dr. Ahmet Coşkun Sönmez**

Şubat 2015

**İSTANBUL TEKNİK ÜNİVERSİTESİ BİLGİSAYAR VE BİLİŞİM FAKÜLTESİ**

**FPGA ÜZERİNDE BİR İŞLEMCİ TASARIMI**

**Bitirme Ödevi Ara Rapor**

**Tuğrul Yatağan**

**040100117**

**Bölüm : Bilgisayar Mühendisliği**

**Anabilim Dalı : Bilgisayar Bilimleri**

**Danışman: Prof. Dr. Ahmet Coşkun Sönmez**

Şubat 2015

**Özgünlük Bildirisi**

1. Bu çalışmada, başka kaynaklardan yapılan tüm alıntıların, ilgili kaynaklar referans gösterilerek açıkça belirtildiğini,
2. Alıntılar dışındaki bölümlerin, özellikle projenin ana konusunu oluşturan teorik çalışmaların ve yazılım/donanımın benim tarafımdan yapıldığını

bildiririm.

İstanbul, 16.02.2015

Tuğrul Yatağan

İmza"

**FPGA ÜZERİNDE BİR İŞLEMCİ TASARIMI**

**( ÖZET )**

FPGA’lerin dinamik sayısal tasarım yeteneğinden faydalanılarak bütün işlemci mimarileri ve organizasyonları FPGA’ler üzerinde denenebilir. Bu farklı işlemci mimari tasarımlarının avantaj ve dezavantajlarının kıyaslanabilmesini sağlar. Ayrıca FPGA’ler yeni işlemci mimarisi tasarımlarının kolaylıkla geliştirilebilmesini ve üretimden önce test edilebilmesini sağlar. Bu projede özel tasarım mimari ve komut setine sahip bir RISC işlemci tasarlanıp, uygulamaya konulacaktır. Bu yeni mimari tasarımı için piyasada bulunan RISC işlemci mimarilerinden ilham alınmıştır. İşlemci tasarımı ve işlemcilerin çalışma mekanizmasının derinlemesine öğrenilmesi bu bitirme projesinin asıl amacıdır. Proje bitiminde komut işleyebilen tüm fonksiyonları yerinde FPGA üzerinde çalışabilen bir sanal (yazılımsal) işlemci tasarlanmış olacaktır.

Bu projedeki sanal işlemci Verilog donanım tanımlama dilinde (HDL), Altera FPGA üzerinde geliştirilecektir. Tasarım olarak THUMB ve MIPS mimarilerinden esinlenilmiştir bu yüzden RISC mimarisi kullanılmıştır. İşlemci Harvard mimari yapısında tasarlanmıştır yani ayrı veri ve komut belleğine sahiptir. Ayrıca sabit uzunlukta Yükle/Kaydet (Load/Store) komut seti mimarisine sahiptir yani yalnızca Yükle ve Kaydet komutları veri belleğine erişebilir. Bu daha basit bir iş hattı tasarımının yapılmasına ve tek çevrimde komut işlenmesine olanak sağlamaktadır. İş hattı mimarisi için klasik RISC iş hattı mimarisi seçilmiştir yani işlemci 5 aşamalı iş hattına sahiptir. [4] İşlemci 16 Bit genişliğinde CPU, ALU, yazmaç ve veri yolu mimarisine sahiptir. Veri ve komut belleği için adres yolu genişliği 12 bittir. Bu yüzden en fazla adreslenebilir veri ve komut belleği boyutu 4K kelime uzunluğunda yani 8KB boyutundadır. Bu kısıtlamanın sebebi FPGA içerisindeki en fazla kullanılabilir dâhili bellek boyutunun kısıtlı olmasındandır.

Şimdiye kadar, tüm donanımsal ve yazılımsal kaynaklar temin edilip proje uygunluğu için test edilmiştir. Ayrıca tüm teorik ve mimari tasarım tamamlanmıştır. Proje geliştirme ve uygulama aşamasındadır.

**A PROCESSOR DESIGN ON FPGA**

**( SUMMARY )**

Using advantage of field programmable gate arrays (FPGA) dynamic digital design capability, every processor architecture and organization can be simulated on it. This gives a chance of compare different processor architecture design’s advantages and disadvantages. Also new architecture designs can be tested on FPGA’s before production. In this project a RISC processor will be implemented with a newly designed architecture and instruction set. Commercial processor architectures were examined and mixture of their design inspired the new architecture. Learning, design & working mechanism of a processor is main aim of this project. Final result of this project is a soft processor which will be able to execute instruction like fully functional normal microprocessor.

In this project a soft processor is designed and will be implemented on Altera FPGA in Verilog hardware description language. Mixture of THUMB and MIPS architectures were used to inspire new architecture so RISC architecture was chosen as main architecture. The processor is in Harvard architecture so it has separate instruction and data memory. Also it has fixed length Load/Store instruction set architecture which means only load and store instructions access to data memory. This leads to single cycle instuction execution capability with easier pipeline design. The classic RISC pipeline architecture is used for pipeline design so processor’s pipeline has 5 stages. [4] The processor has 16 Bit CPU, ALU, register and data bus architecture. Address bus for data and instruction memory is 12 bit so maximum addressable instruction and data memory is 4K word and it is equal to 8KB. This restriction is due to the FPGA’s usable internal memory size.

So far, all the hardware and software resources were obtained and tested, all theoretical research and architectural design were done.

**TABLE OF CONTENTS**

[1. ıntroductıon 5](#_Toc411874383)

[2. Project decrıptıon and plan 6](#_Toc411874384)

[2.1. Project Description 6](#_Toc411874385)

[2.2. Project Plan 6](#_Toc411874386)

[3. STUDIES AND RESULTS 7](#_Toc411874387)

[3.1. Architecture 7](#_Toc411874388)

[3.1.1. Organization 7](#_Toc411874389)

[3.1.2. Instruction Set 9](#_Toc411874390)

[3.1.2.1. Shift Instructions 11](#_Toc411874391)

[3.1.2.2. Add/Subtract Instructions 11](#_Toc411874392)

[3.1.2.3. Add/Subtract Register Immediate Instructions 12](#_Toc411874393)

[3.1.2.4. Move/Compare/Add/Subtract Immediate Instructions 12](#_Toc411874394)

[3.1.2.5. ALU Instructions 13](#_Toc411874395)

[3.1.2.6. Load/Store with Register Offset Instructions 14](#_Toc411874396)

[3.1.2.7. Load/Store with Immediate Offset Instructions 14](#_Toc411874397)

[3.1.2.8. Conditional Branch Instructions 15](#_Toc411874398)

[3.1.2.9. Unconditional Branch Instruction 16](#_Toc411874399)

[3.1.2.10. Inherent Instructions 16](#_Toc411874400)

[3.2. Implementation 17](#_Toc411874401)

[3.2.1. FPGA 17](#_Toc411874402)

[3.2.2. Simulation 18](#_Toc411874403)

[3.2.2.1. Modelsim 18](#_Toc411874404)

[3.2.2.2. Logisim 18](#_Toc411874405)

[4. TO DO LIST 19](#_Toc411874406)

[5. References 20](#_Toc411874407)

# ıntroductıon

In this project a RISC processor will be implemented with a newly designed architecture and instruction set on FPGA. Final result of this project is a soft processor which will be able to execute instruction like normal microprocessor. Project started October 2014 and will finished May 2015. Detailed project plan is described in chapter 2.

Mixture of THUMB and MIPS architectures were used to inspire new architecture so RISC architecture was chosen as main architecture. The processor is in Harvard architecture so it has separate instruction and data memory. The classic RISC pipeline architecture is used for pipeline design so processor’s pipeline has 5 stages. [4] The processor has 16 Bit CPU, ALU, register and data bus architecture. Address bus for data and instruction memory is 12 bit. Detailed architectural design is described in chapter 3.1. Also processor has fixed length Load/Store instruction set architecture which means only load and store instructions access to data memory. Detailed instruction set design is described in chapter 3.1.2.

In this project the soft processor is designed and will be implemented on Altera FPGA in Verilog hardware description language. Detailed hardware and software resources are described in chapter 3.2.

So far, all the hardware and software resources were obtained and tested, all theoretical research and architectural design were done. All the work performed and left jobs to perform are described in chapter 4.

There are mainly three books to be a reference in this area. Also some online books and lecture notes are used for project’s research. All sources which are used for research are referenced in chapter 5 according to APA style and cited in the text as square brackets ([#]).

# Project decrıptıon and plan

## Project Description

Project consists of three main phases. Research, design and implementation. So far, research and design stages are done. All hardware and software resources are ready and tested for implementation stage.

## Project Plan

It is planed that the project will be finished in May 2015. This graduation project has these major steps:

* Organization design
* Architectural design
* Instructions set design
* Choosing and understanding how to use FPGA kit
* Understanding how to program FPGA in Verilog
* ALU implementation
* Register interaction and BUS implementation
* Organization and pipeline implementation
* RAM and ROM access on FPGA
* Executing instructions in RAM
* Writing assembler for new instruction set

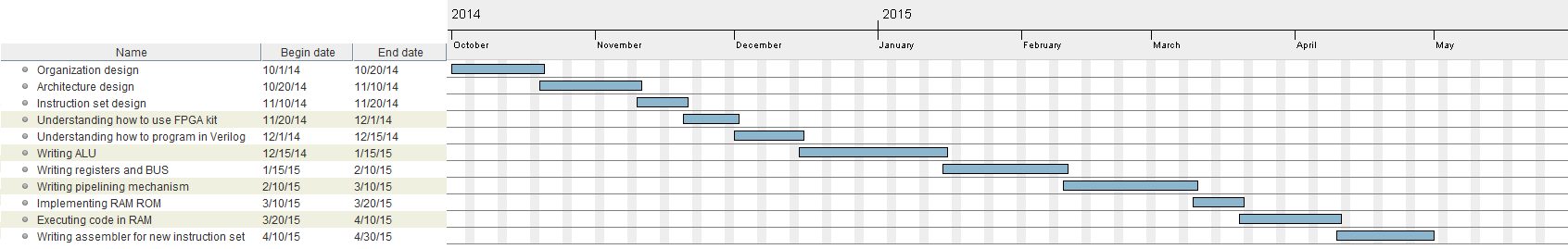
Schedule of these steps are shown in Figure 1.

Figure 1: Gantt Diagram of the project schedule

# STUDIES AND RESULTS

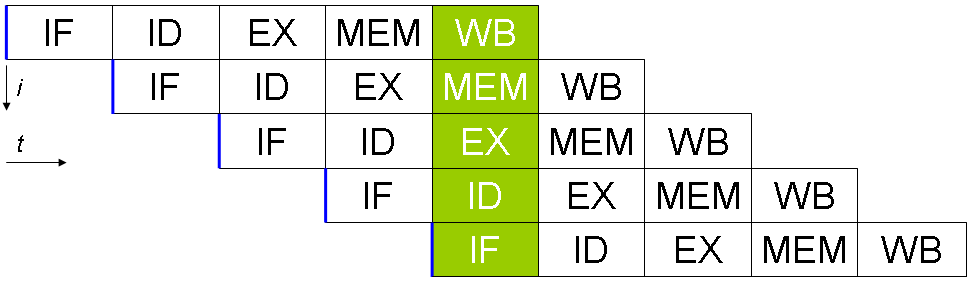
## Architecture

Mixture of THUMB and MIPS architectures were used to inspire new architecture so RISC architecture was chosen as frame architecture. The processor is in Harvard architecture so it has separate instruction and data memory. Also it has fixed length Load/Store instruction set architecture which means only load and store instructions access to data memory. This leads to single cycle instruction execution capability with easier pipeline design. The classic RISC pipeline architecture is used for pipeline design so processor’s pipeline has 5 stages. [4]

Classic RISC architecture pipeline has 5 stages. [4] Which are;

* Instruction fetch
* Instruction decode
* Execute (ALU)
* Memory access
* Write back

Figure 2: 5 stage classic RISC pipeline timing diagram [5]

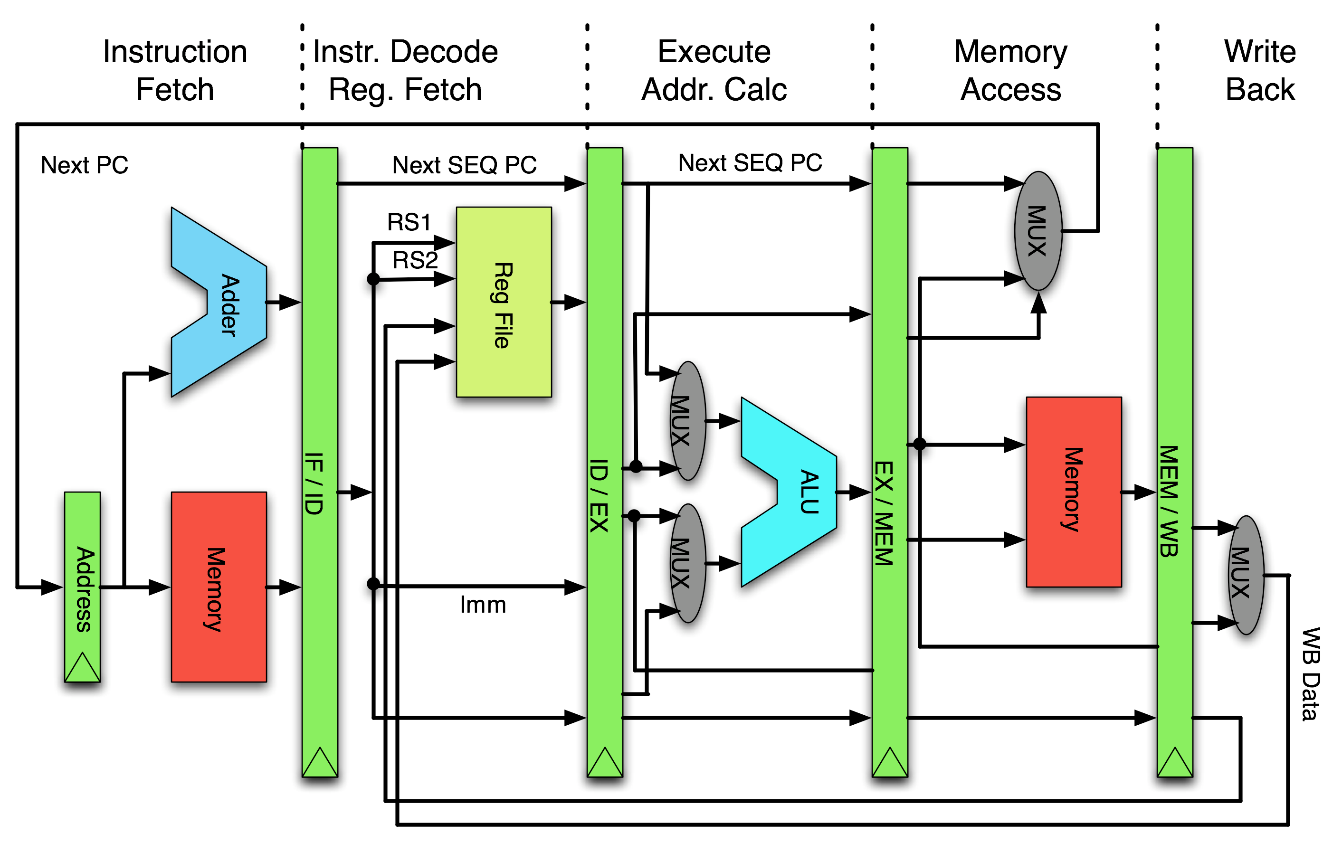


The processor has 16 Bit CPU, ALU, register and data bus architecture. Address bus for data and instruction memory is 12 bit so maximum addressable instruction and data memory is 4K word and it is equal to 8KB

### Organization

Slightly modified MIPS architecture is used for organizational design. Organization of major blocks with 5 stage pipeline is shown in Figure 3. [5]

Figure 3: Classic 5 stage RISC pipeline organization [5]



There are 8 general purpose 16 bit registers in register file. Also there are 16 bit instruction and condition code register, 12 bit program counter and address register. All general purpose register accessible and they can transfer data between them. PC can only be modifiable via branch instructions, its content is not accessible. IR is not accessible. CCR’s some condition bits can be changed via some instructions and they can be usable via only branch instructions.

**Registers**

|  |  |  |
| --- | --- | --- |
| **Abbreviation** | **Equivalent** | **Bit** |
| R0 | General purpose register 0 | 16 |
| R1 | General purpose register 1 | 16 |
| R2 | General purpose register 2 | 16 |
| R3 | General purpose register 3 | 16 |
| R4 | General purpose register 4 | 16 |
| R5 | General purpose register 5 | 16 |
| R6 | General purpose register 6 | 16 |
| R7 | General purpose register 7 | 16 |
| IR | Instruction register | 16 |
| PC | Program counter | 12 |
| AR | Address register | 12 |
| CCR | Condition code register | 16 |

**Condition code register**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
|  |  |  |  |  |  |  |  |  | IRQ | I |  | N | Z | C | V |

|  |  |
| --- | --- |
| **Bit** | **Assignment** |
| N | Negative/Less than |
| Z | Zero |
| C | Carry/Borrow |
| V | Overflow |
| I | IRQ Disable |
| IRQ | IRQ Status |

### Instruction Set

All instructions are16 Bit fixed length instruction. There are 10 type of instruction. Load/Store instruction set so only load and store instructions can access to data memory.

**Instruction Types**

1. Shift Instructions
2. Add/Subtract Register Instructions
3. Add/Subtract Register Immediate Instructions
4. Move/Compare/Add/Subtract Immediate Instructions
5. ALU Instructions
6. Load/Store with Register Offset Instructions
7. Load/Store with Immediate Offset Instructions
8. Conditional Branch Instructions
9. Unconditional Branch Instruction
10. Inherent Instructions

**Instruction Set format**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **15** | **14** | **13** | **12** | **11** | | **10** | | **9** | | **8** | | **7** | | **6** | | **5** | **4** | **3** | **2** | **1** | **0** |
| **1** | 0 | 0 | 0 | 0 | Op | | | | Off4 | | | | | | | | Rs | | | Rd | | |
| **2** | 0 | 0 | 1 | Op | Off3 | | | | | | Rn | | | | | | Rs | | | Rd | | |
| **3** | 0 | 1 | 0 | Op | Off6 | | | | | | | | | | | | Rs | | | Rd | | |
| **4** | 0 | 1 | 1 | Op | | | Rd | | | | | | Off8 | | | | | | | | | |
| **5** | 1 | 0 | 0 | Op | | | | | | | Rn | | | | | | Rs | | | Rd | | |
| **6** | 1 | 0 | 1 | Op | Off3 | | | | | | Ro | | | | | | Rb | | | Rd | | |
| **7** | 1 | 1 | 0 | Op | Off6 | | | | | | | | | | | | Rb | | | Rd | | |
| **8** | 1 | 1 | 1 | 0 | Cond | | | | | | | | Off8 | | | | | | | | | |
| **9** | 0 | 0 | 0 | 1 | Off12 | | | | | | | | | | | | | | | | | |
| **10** | 1 | 1 | 1 | 1 | X | X | | X | | X | | X | | X | | X | | X | X | Op | | |

**Operand Table**

|  |  |  |
| --- | --- | --- |
| **Abbreviation** | **Equivalent** | **Bit** |
| Rd | Destination Register | 3 |
| Rs | Source Register | 3 |
| Rb | Base Register | 3 |
| Ro | Offset Register | 3 |
| Off# | Immediate Offset | # |
| Op | Operation | X |
| Cond | Condition | X |

#### Shift Instructions

Immediate shift operation instructions with two register operands. Logical left, logical right, arithmetic right, circular right shift operations are implemented.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 0 | Op | | Off4 | | | | Rs | | | Rd | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Op** | **Assembler** | **Action** | **Updates** |
| 00 | LSL Rd, Rs, #Off4 | Logic shift left by #Off4 | N Z C |
| 01 | LSR Rd, Rs, #Off4 | Logic shift right by #Off4 | N Z C |
| 10 | ASR Rd, Rs, #Off4 | Arithmetic shift right by #Off4 | N Z C |
| 11 | CSR Rd, Rs, #Off4 | Circular shift right by #Off4 | N Z C |

**Example Instruction:**

LSR R2, R5, #10

#### Add/Subtract Instructions

Add and subtract operations with three register operands and immediate positive 3 bit offset.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 1 | Op | Off3 | | | Rn | | | Rs | | | Rd | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Op** | **Assembler** | **Action** | **Updates** |
| 0 | ADD Rd, Rs, #Off3 | Rd := Rs + Rn + #Off3 | N Z C V |
| 1 | SUB Rd, Rs, #Off3 | Rd := Rs – Rn + #Off3 | N Z C V |

**Example Instructions:**

ADD R2, R3, R4, #6

SUB R1, R4, R2 // offset is #0

#### Add/Subtract Register Immediate Instructions

Add and subtract operations with two register operands and immediate positive 6 bit offset.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 1 | 0 | Op | Off6 | | | | | | Rs | | | Rd | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Op** | **Assembler** | **Action** | **Updates** |
| 0 | ADD Rd, Rs, #Off3 | Rd := Rs + #Off6 | N Z C V |
| 1 | SUB Rd, Rs, #Off3 | Rd := Rs - #Off6 | N Z C V |

**Example Instruction:**

ADD R1, R4, #60

#### Move/Compare/Add/Subtract Immediate Instructions

Move, compare, add and subtract operations with one register operands and immediate positive 8 bit offset. Compare operations only updates condition code register, no write back stage.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 1 | 1 | Op | | Rd | | | Off8 | | | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Op** | **Assembler** | **Action** | **Updates** |
| 00 | MOV Rd, #Off8 | Rd := #Off8 | N Z |
| 01 | CMP Rd, #Off8 | Rd - #Off8 | N Z C V |
| 10 | ADD Rd, #Off8 | Rd := Rd + #Off8 | N Z C V |
| 11 | SUB Rd, #Off8 | Rd := Rd + #Off8 | N Z C V |

**Example Instructions:**

MOV R1, #130

CMP R2, #25

ADD R5, #40

SUB R3, #200

#### ALU Instructions

These instructions performs ALU operations like logical, shift and compare instructions with two or three register operands. Bitwise logical operations, add with carry and subtract with borrow operations, shift operations are implemented with three register operands. Complement and negations operations, compare and test operations are implemented with two register operands. Operation table for ALU instructions is incomplete, it can be extend.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 1 | 0 | 0 | Op | | | | Rn | | | Rs | | | Rd | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Op** | **Assembler** | **Action** | **Updates** |
| 0000 | AND Rd, Rs, Rn | Rd := Rs AND Rn | N Z |
| 0001 | OR Rd, Rs, Rn | Rd := Rs OR Rn | N Z |
| 0010 | XOR Rd, Rs, Rn | Rd := Rs XOR Rn | N Z |
| 0011 | LSL Rd, Rs, Rn | Rd := Rs << Rn | N Z C |
| 0100 | LSR Rd, Rs, Rn | Rd := Rs >> Rn | N Z C |
| 0101 | ASR Rd, Rs, Rn | Rd := Rs ASR Rn | N Z C |
| 0110 | CSR Rd, Rs, Rn | Rd := Rs CSR Rn | N Z C |
| 0111 | ADC Rd, Rs, Rn | Rd := Rs + Rn with Carry | N Z C V |
| 1000 | SBC Rd, Rs, Rn | Rd := Rs - Rn with Carry | N Z C V |
| 1001 | NEG Rd, Rs | Rd := -Rs | N Z |
| 1010 | NOT Rd, Rs | Rd := NOT Rs | N Z |
| 1011 | CMP Rd, Rs | Rd – Rs | N Z C V |
| 1100 | TST Rd, Rs | Rd AND Rs | N Z C V |
| 1101 |  |  |  |
| 1110 |  |  |  |
| 1111 |  |  |  |

**Example Instructions:**

XOR R2, R5, R3

CSR R1, R3, R4

ADC R4, R2, R6

NOT R5, R6

CMP R3, R5

#### Load/Store with Register Offset Instructions

These instructions transfer 16 bit word between registers and memory. Memory address operands are base register, offset register and 3 bit immediate offset.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 1 | 0 | 1 | Op | Off3 | | | Ro | | | Rb | | | Rd | | |

|  |  |  |
| --- | --- | --- |
| **Op** | **Assembler** | **Action** |
| 0 | LD Rd, [Rb, Ro, #Off3] | Rd := MEM[Rb + Ro + #Off3] |
| 1 | STR Rd, [Rb, Ro, #Off3] | MEM[Rb + Ro + #Off3] := Rd |

**Example Instructions:**

LD R5, [R1, R2, #4]

LD R3, [R5, R2] // offset is #0

STR R1, [R5, R3, #5]

#### Load/Store with Immediate Offset Instructions

These instructions transfer 16 bit word between registers and memory. Memory address operands are base register and 6 bit immediate offset.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | | **4** | **3** | **2** | | **1** | **0** |
| 1 | 1 | 0 | Op | Off6 | | | | | | | Rb | | | | Rd | | |

|  |  |  |
| --- | --- | --- |
| **Op** | **Assembler** | **Action** |
| 0 | LD Rd, [Rb, #Off6] | Rd := MEM[Rb + #Off6] |
| 1 | STR Rd, [Rb, #Off6] | MEM[Rb + #Off6] := Rd |

**Example Instructions:**

LD R3, [R5, #10]

STR R1, [R5, #26]

STR R1, [R3] // offset is #0

#### Conditional Branch Instructions

These instructions perform a conditional branch depending on the state of the condition code N Z C V bits with 8 bit offset from PC register. [2]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 1 | 1 | 1 | 0 | Cond | | | | Off8 | | | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Cond** | **Assembler** | **Condition Logic** | **Condition** |
| 0000 | BEQ #Off8 | Z = 1 | equal |
| 0001 | BNE #Off8 | Z = 0 | not equal |
| 0010 | BCS #Off8 | C = 1 | unsigned higher or same |
| 0011 | BCC #Off8 | C = 0 | unsigned lower |
| 0100 | BMI #Off8 | N = 1 | negative |
| 0101 | BPL #Off8 | N = 0 | positive or zero |
| 0110 | BVS #Off8 | V = 1 | overflow |
| 0111 | BVC #Off8 | V = 0 | no overflow |
| 1000 | BHI #Off8 | Z' \* C | unsigned higher |
| 1001 | BLS #Off8 | Z + C' | unsigned lower or same |
| 1010 | BGE #Off8 | (N \* V) +  (N' \* V') | signed greater than or equal |
| 1011 | BLT #Off8 | N XOR V | signed less than |
| 1100 | BGT #Off8 | (N \* Z' \* V) +  (N' \* Z' \* V') | signed greater than |
| 1101 | BLE #Off8 | Z + (N XOR V) | signed less than or equal |
| 1110 |  |  |  |
| 1111 |  |  |  |

**Example Instructions:**

BEQ #50 // BEQ Label

BGT #25 // BGT Label

#### Unconditional Branch Instruction

This instruction performs an unconditional branch with 12 bit offset from PC register.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 0 | 0 | 0 | 1 | Off12 | | | | | | | | | | | |

|  |  |
| --- | --- |
| **Assembler** | **Action** |
| BAL #Off12 | PC := PC + #Off6 |

**Example Instruction:**

BAL Label // Label may be #5324 offset

#### Inherent Instructions

These instructions are implied (inherent) instructions which means they have not any operand. Operation table for inherent instructions is incomplete, it can be extend.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 1 | 1 | 1 | 1 | X | X | X | X | X | X | X | X | X | Op | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Op** | **Assembler** | **Action** | **Updates** |
| 000 | NOP | No operation |  |
| 001 | ION | Interrupts on | I |
| 010 | IOF | Interrupts off | I |
| 011 | RTI | Return from interrupt |  |
| 100 | CLRC | Clear N Z C V bits | N Z C V |
| 101 |  |  |  |
| 110 |  |  |  |
| 111 |  |  |  |

**Example Instructions:**

NOP

ION

## Implementation

Physical implementation will be done on Altera DE0-Nano FPGA education board. Altera Quartus 13.0sp1 development environment is used for development. Altera Modelsim simulation software and Logisim digital design tool are used for simulate the design.

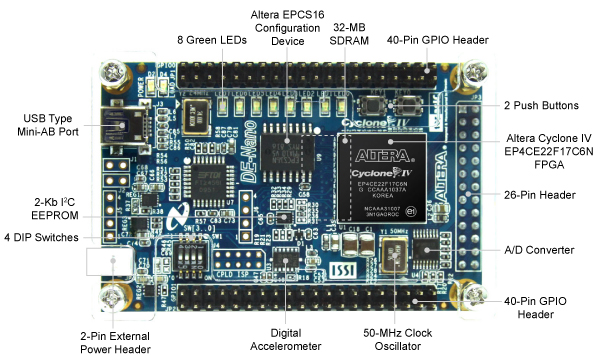
### FPGA

Altera DE0-Nano development and education board has Altera Cyclone® IV EP4CE22F17C6N FPGA which has:

* 22,320 Logic elements (LEs)
* 594 Embedded memory (Kbits)
* 153 Maximum FPGA I/O pins, some pins hardwired to LEDs and switches.

These features are more than enough for implementation of the processor.

Figure 4: Altera DE0-Nano development and education board [1]



Altera DE0-Nano development board enables to test new designs on FPGA via USB port, also its memory content can be assign before run and its memory content can be monitored during run time.

### Simulation

Altera Modelsim simulation software and Logisim digital design tool are used for simulate the design before FPGA run.

#### Modelsim

Altera FPGA’s default simulation software. It has gate level and logic level simulation capabilities. Detailed timing simulation can be done. It gives actual results as like FPGA run.

#### Logisim

Logisim is an educational tool for designing and simulating digital logic circuits. It can be used to design and simulate entire CPUs. It can give detailed look to processor before the FPGA implementation.

# TO DO LIST

Jobs to do for project until end of May 2015 can be list as:

* Understanding how to program in Verilog on Altera FPGA effectively
* ALU implementation with Logisim and Verilog
* Register interaction and BUS implementation with Logisim and Verilog
* Organization and pipeline implementation with Logisim and Verilog
* RAM and ROM access on FPGA with Verilog
* Executing instructions in RAM
* Testing and correcting instruction execution
* Writing assembler for new instruction set with Python
* Testing whole system with assembler
* Getting results and making experiments on processor

# References

1. Altera Corporation. (2012). *DE0-Nano User Manual.* Retrieved from

<ftp://ftp.altera.com/up/pub/Altera_Material/13.0/Boards/DE0-Nano/DE0_Nano_User_Manual.pdf>

1. Berger, A. S. (2005). *Hardware and Computer Organization: The Software Perspective.* Burlington: Elsevier Inc.
2. Harris, D. & Harris, S. (2012). *Digital Design and Computer Architecture* (2nd ed.). San Francisco: Morgan Kaufmann Publishers Inc.
3. Hennessy, J. L. & Patterson, D. A. (2011). *Computer Architecture: A Quantitative Approach* (5th ed.). San Francisco: Morgan Kaufmann Publishers Inc.
4. Microprocessor Design. (n.d.). In *Wikibooks*. Retrieved from <http://en.wikibooks.org/wiki/Microprocessor_Design/Print_Version>
5. University of Waterloo. (2012). *THUMB Instruction Set*. Retrieved from University of Waterloo website:

<https://ece.uwaterloo.ca/~ece222/ARM/ARM7-TDMI-manual-pt3.pdf>

1. Weatherspoon, H. (2011). *CS 3410: Computer System Organization and Programming* [Lecture Slides]. Retrieved from Cornell University website: <http://www.cs.cornell.edu/courses/CS3410/2011sp/schedule.html>