DIGITAL SYSTEM DESIGN FINAL PROJECT REPORT
DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITAS INDONESIA

PROGRAMMABLE BASIC CPU

GROUP PA-07

RIVI YASHA HAFIZHAN	**2306250535**
MAXWELL ZEFANYA GINTING	**2306221200**
MUHAMAD DZAKY MAULANA	**2306264401**
REICHAN ADHIGUNO	**2106703273**

PREFACE

Kami berterima kasih kepada Tuhan Yang Maha Kuasa atas rahmat dan anugerah-Nya sehingga kami berhasil menyelesaikan laporan akhir proyek ini yang berjudul "CPU Basic Programmable". Laporan ini disusun sebagai bagian dari tugas akhir dalam pembelajaran Perancangan Sistem Digital, dan dimaksudkan untuk memberikan pemahaman dan penerapan konsep dasar terhadap arsitektur komputer yang telah kami pelajari sebelumnya.

Tujuan proyek ini adalah untuk membuat CPU sederhana yang dapat membaca kode mesin dari berkas input, mendekode setiap perintah, dan menghasilkan output yang disimpan dalam file terpisah. CPU ini juga dapat menjalankan beberapa instruksi dasar menggunakan opcode dalam format biner. Untuk memastikan semua proses sinkronis, CPU menggunakan komponen penting seperti register 16-bit, flag identik, dan mekanisme kontrol berbasis clock saat beroperasi.

Kami berterima kasih kepada dosen yang telah memberikan bimbingan dan asisten laboratorium yang telah mendampingi dan membantu kami selama praktikum ini. Asisten laboratorium yang berfungsi sebagai pembimbing teknis dan membantu menyelesaikan masalah. Menyelesaikan proyek ini dengan baik memerlukan bimbingan dan arahan yang tepat.

Kami berharap laporan ini bermanfaat sebagai sumber pembelajaran dan dasar untuk pengembangan sistem digital yang lebih canggih. Selain itu, kami menyadari bahwa laporan ini memiliki kelemahan dan kekurangan, dan kami terbuka untuk kritik dan saran yang membangun untuk membantu memperbaikinya di masa mendatang.

Depok, December 12, 2024

Group PA-07

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Di era yang sudah berkembang ini, perkembangan teknologi digital pastinya semakin pesat, terutama pada bidang komputasi dan teknologi. Salah satu elemen penting dalam teknologi komputasi adalah Central Processing Unit (CPU), yang berperan sebagai otak dari sistem komputer itu sendiri. Memahami cara kerja CPU menjadi salah satu fondasi penting dalam studi sistem digital dan arsitektur komputer.

Namun, pemahaman mengenai CPU sering kali terbatas pada teori tanpa disertai implementasi langsung yang memungkinkan user untuk mengeksplorasi desain dan cara kerja sebuah CPU secara praktis. Untuk itu, dirancang sebuah proyek yang bertujuan untuk menciptakan CPU sederhana yang mampu menjalankan sejumlah instruksi dasar.

CPU yang dirancang dalam proyek ini memiliki kemampuan untuk menjalankan delapan instruksi, yaitu CMP, JUMP, ADD, SUB, LOAD, STORE, AND, dan NOT. Dengan jumlah instruksi yang terbatas, user dapat mempelajari konsep dasar arsitektur komputer secara lebih mendalam dan terarah. Selain itu, penggunaan file sebagai input instruksi dan output hasil eksekusi menambah fleksibilitas dalam memahami alur kerja CPU secara komprehensif.

1.2 PROJECT DESCRIPTION

Proyek ini bertujuan merancang dan melaksanakan sebuah CPU programmable sederhana yang dirancang untuk memahami konsep dasar dari arsitektur komputer. CPU ini memiliki tombol enable dan reset untuk mengatur operasinya. Sama seperti pada CPU biasanya, semua proses akan disinkronkan melalui clock.

Di dalam CPU terdapat empat komponen utama yang hanya dapat diakses melalui instruksi opcode, yaitu tiga register 16-bit dan satu bendera equals yang digunakan untuk operasi perbandingan. CPU ini memanfaatkan cara pembacaan kode mesin dari sebuah berkas masukan yang ditulis dalam format bilangan bulat.

Cara kerja CPU dimulai dengan membaca berkas yang memuat kode secara berurutan, lalu melaksanakan instruksi-instruksi itu sesuai dengan urutannya. Proyek ini menawarkan metode praktis untuk memahami cara CPU memproses instruksi, dari tahap pembacaan hingga eksekusi, dengan desain yang sederhana tetapi tetap mencerminkan cara kerja CPU secara umum.

1.3 OBJECTIVES

Proyek ini memiliki tiga tujuan utama, yaitu sebagai berikut:

1. **Memahami Konsep Dasar CPU**
    a. Memahami cara kerja CPU sederhana, termasuk penggunaan tombol enable dan reset, sinkronisasi operasi melalui clock, dan pemanfaatan komponen seperti register 16-bit dan flag equals.
2. **Implementasi Sistem yang Berbasis Opcode**
    a. Mengimplementasikan mekanisme pembacaan machine code dalam bentuk integer melalui file input dan eksekusi instruksi berdasarkan opcode yang diakses oleh CPU.
3. **Simulasi Proses CPU dengan Sinkronisasi Clock**
    a. Mensimulasikan alur kerja CPU secara lebih sederhana, mulai dari pembacaan machine code secara sekuensial hingga eksekusi instruksi, dengan seluruh proses sinkronisasi melalui clock.

1.4 ROLES AND RESPONSIBILITIES

The roles and responsibilities assigned to the group members are as follows:

Roles	Responsibilities	Person
Role 1	Membuat PPT. Membuat ALU bagian register	Rivi Yasha Hafizhan

	handling & 4 instruksi ALU	
Role 2	Menghubungkan keempat komponen & 4 instruksi ALU. Membuat Sintesis rangkaian	Maxwell Zefanya Ginting
Role 3	Membuat fetcher & 4 instruksi ALU. Membuat laporan	Muhamad Dzaky Maulana
Role 4	Membuat decoder & 4 instruksi ALU. Membuat laporan	Reichan Adhiguno

Table 1. Roles and Responsibilities

CHAPTER 2

IMPLEMENTATION

2.1 EQUIPMENT

1. MODEL SIM

Tools pertama yang digunakan untuk simulasi dan implementasi pada proyek kali ini adalah **ModelSim**. **ModelSim** digunakan untuk mensimulasikan dan memverifikasi desain CPU sederhana yang dirancang. Beberapa Kegunaan dari **ModelSim** pada proyek ini antara lain :

a. Simulasi Sinkronisasi Clock

i. ModelSim memungkinkan simulasi sinkronisasi semua operasi CPU melalui clock. Ini penting untuk memastikan bahwa CPU bekerja sesuai dengan logika sekuensial yang telah dirancang.

b. Pengujian Komponen Internal CPU

i. Dengan ModelSim, komponen internal CPU seperti tiga register 16-bit dan flag equals dapat diuji secara terpisah maupun bersamaan. Simulasi memastikan bahwa setiap komponen bekerja sesuai dengan fungsinya saat diakses melalui opcode.

c. Simulasi Eksekusi Instruksi

i. ModelSim digunakan untuk mensimulasikan kode VHDL yang mendeskripsikan CPU. Simulasi ini mencakup pembacaan VHDL code dari file input, decoding opcode, dan eksekusi instruksi secara berurutan yang hasilnya nanti akan ditampilkan dalam bentuk wave.

2. QUARTUS

Selanjutnya *tools* kedua yang digunakan pada proyek kali ini yaitu **Quartus. Quartus** sendiri digunakan sebagai *software* utama untuk melakukan proses synthesis dan juga validasi desain CPU sederhana yang sudah dirancang menggunakan VHDL. Melalui **Quartus**, kode VHDL yang ditulis dapat diubah menjadi sebuah skema logika digital yang terstruktur dan dapat divisualisasikan dalam bentuk diagram rangkaian.

Proses synthesis ini memungkinkan setiap modul CPU, seperti ALU, Register File, Fetcher, dan juga Decoder, dapat diintegrasikan menjadi sebuah top-level entity yang merepresentasikan fungsi CPU secara keseluruhan. Dengan visualisasi ini, hubungan antar komponen dapat dianalisis untuk memastikan desain sesuai spesifikasi.

3. VISUAL STUDIO CODE

Untuk **Visual Studio Code** ini digunakan sebagai untuk membuat file markdown yang nantinya ada di repository github proyek ini. Di Dalamnya terdapat cara kerja dari CPU, Metode Implementasi, dan juga tabel instruksi dari CPU ini sendiri. Dengan adanya **Visual Studio Code** proses penulisan Markdown menjadi lebih efisien.

2.2 IMPLEMENTATION

IMPLEMENTASI MODUL

Jadi untuk implementasinya CPU sederhana yang dalam proyek ini yang menggunakan atau memanfaatkan modul yang diberikan, antara lain sebagai berikut :

1. **Behavioral Style Programming (Modul 3)**
   Pada **behavioral style programming** digunakan dalam mendeskripsikan fungsi utama CPU. Dalam VHDL, **behavioral style programming** ini memungkinkan pemrograman logika berdasarkan perilaku sistem, seperti bagaimana CPU membaca file machine code, mendecode instruksi, dan menjalankan operasi berdasarkan opcode.
2. **Testbench (Modul 4)**
   Testbench digunakan untuk menguji program ini. Testbench mensimulasikan skenario kerja nyata, seperti pembacaan file machine code dan eksekusi setiap instruksi, untuk memastikan seluruh komponen CPU berfungsi sesuai apa yang kami kerjakan.
3. **Structural Style Programming (Modul 5)**
   Desain CPU menggunakan **structural style programming** untuk menghubungkan komponen-komponen utama CPU, seperti register 16-bit, unit decoding, dan flag equals. Dengan pendekatan ini, setiap blok modul dalam CPU diintegrasikan secara hierarkis.
4. **Looping Construct (Modul 6)**
   Penggunaan **looping construct** diimplementasikan untuk proses pembacaan dan pengolahan file machine code secara berurutan. Misalnya, instruksi dalam file dibaca dalam loop hingga semua instruksi selesai dijalankan.
5. **Finite State Machine (Modul 8)**
   Desain CPU mengikuti pendekatan **finite state machine (FSM)** untuk mengelola transisi antar status, seperti status membaca file, mendecode instruksi, dan mengeksekusi operasi. Pendekatan FSM memastikan alur kerja CPU berjalan secara terstruktur dan terkontrol.
6. **Microprogramming (Modul 9)**
   Konsep **microprogramming** digunakan untuk mendefinisikan instruksi CPU dalam level yang lebih rendah. Setiap opcode yang dibaca dikaitkan dengan serangkaian

operasi mikro yang dijalankan oleh CPU untuk menyelesaikan instruksi tersebut. CPU sendiri akan memiliki 16 instruksi yang bisa digunakan, yang akan dijabarkan pada tabel dibawah ini. Bagian format menjabarkan bagaimana kode secara keseluruhan harus ditulis. Pada tabel ini, register destinasi dilambangkan sebagai “XX”, register asal dilambangkan sebagai “YY”, dan value immediate sebagai “YYYYYY”. Pilihan register sendiri dibatasi menjadi 3 buah. Oleh karena itu, address register yang valid hanyalah “00”, “01”, dan “10” saja.

INSTRUKSI	OPCODE	PENJELASAN	FORMAT
CMP-I	0000	Bandingkan apabila isi register dengan immediate (zero-fill) sama, hanya mengefek flag "equals"	0000XXYYYYYY
CMP-R	0001	Bandingkan apabila isi dua register sama, hanya mengefek flag "equals"	0001XXYY0000
ADD-I	0010	Register X = Register X + Immediate	0010XXYYYYYY
ADD-R	0011	Register X = Register X + Register Y	0011XXYY0000
SUB-I	0100	Register X = Register X - Immediate	0100XXYYYYYY
SUB-R	0101	Register X = Register X - Register Y	0101XXYY0000
SAL	0110	Logical bit-shift (bukan arithmetic). Register X << Immediate (Unsigned)	0110XXYYYYYY
SAR	0111	Logical bit-shift (bukan arithmetic). Register X >> Immediate (Unsigned)	0111XXYYYYYY
ISEQ	1000	Akan mengeksekusi instruksi berikutnya apabila flag "equals" = 1. Bila tidak, akan melompati eksekusi satu	100000000000

		instruksi	
NOEQ	1001	Akan mengeksekusi instruksi berikutnya apabila flag "equals" = 0. Bila tidak, akan melompati eksekusi satu instruksi	100100000000
LOD-I	1010	Register X = Immediate	1010XXYYYYYY
STO-R	1011	Menulis isi register kedalam file "Outputs.txt" dalam format integer (signed)	1011XX000000
AND	1100	Register X = Register X && Register Y	1100XXYY0000
OR	1101	Register X = Register X || Register Y	1101XXYY0000
NOT	1110	Register X = !(Register X)	1110XX000000
CLF	1111	Clear flag yang ada. Equals = 0	111100000000

Table 2. CPU Instruction Set

IMPLEMENTASI PROGRAM

1. **ALU (Arithmetic Logic Unit)**

   ALU merupakan unit utama dalam CPU yang bertugas menjalankan operasi aritmatika dan logika sesuai instruksi yang diberikan. ALU menerima input dari register atau nilai immediate melalui Decoder dan menghasilkan output yang disimpan kembali ke register atau dikirim ke modul lain untuk diproses lebih lanjut.

2. **CPU**

   CPU berfungsi untuk menangani berbagai status seperti FETCH, DECODE, EXECUTE, dan STORE. CPU berinteraksi dengan komponen lain seperti ALU, Decoder, dan Fetcher.

3. **Fetcher**

   Fetcher bertugas membaca instruksi secara sekuensial dari file input berdasarkan urutan yang diatur oleh program counter. Instruksi yang diambil oleh Fetcher diteruskan dalam bentuk mentah ke Decoder untuk diproses lebih lanjut. Modul ini memastikan alur pengambilan instruksi berjalan sinkron dan sesuai urutan.

4. **Decoder**

   Decoder berfungsi menerjemahkan instruksi mentah dari Fetcher menjadi sinyal kontrol untuk modul lain seperti ALU dan register. Modul ini memisahkan opcode dari operand dan menentukan operasi yang akan dilakukan sesuai dengan instruksi yang diterima.

CHAPTER 3

TESTING AND ANALYSIS

3.1 TESTING

Testing dilakukan secara langsung dengan menggunakan Modelsim. Untuk input, hanya diberikan 3 input saja, yaitu CLK, ENABLE, dan RESET. Sesuai dengan kerjanya, CPU hanya akan berjalan pada saat ada input CLK, ENABLE diset menjadi ‘1’, RESET menjadi ‘0’, dan terdapat file yang relevan. Pada saat kondisi tersebut dipenuhi, maka CPU akan berjalan secara otomatis dan membaca instruksi yang terdapat didalam file “Instructions.txt”, selama file tersebut belum berakhir, CPU akan terus membaca instruksi yang berada di dalam file tersebut. Bila ada instruksi yang sesuai, file juga akan bisa menuliskan hasil yang ada di register kedalam file “Outputs.txt”. CPU akan berhenti secara otomatis pada saat file Instructions sudah berakhir.

Proses testing sendiri bisa dilakukan dengan mengubah sinyal CLK, ENABLE, atau RESET secara manual atau dengan menggunakan testbench. Bila menggunakan testbench, maka signal yang relevan tidak akan muncul di waveform, tetapi keseluruhan fungsionalitas

CPU tetap berjalan secara normal, termasuk penulisan di Outputs dan interaksi antar register. Hasil dari testing ini bisa diverifikasi dengan menggunakan instruksi “STO-R” yang ada dalam CPU, dimana register yang dipilih akan dimasukkan kedalam file Outputs.

Untuk memverifikasi desain, maka program yang dibuat adalah program yang menuliskan “Hello World!” (ASCII) dalam bentuk integer. Hasil yang ada akan dituliskan kedalam file Outputs.txt dan register yang digunakan adalah register A. Verifikasi desain juga akan dilakukan dengan mensintesis desain menggunakan Quartus, dimana keduanya akan dijabarkan pada bagian berikutnya.

3.2 RESULT

Berikut merupakan waveform hasil simulasi.

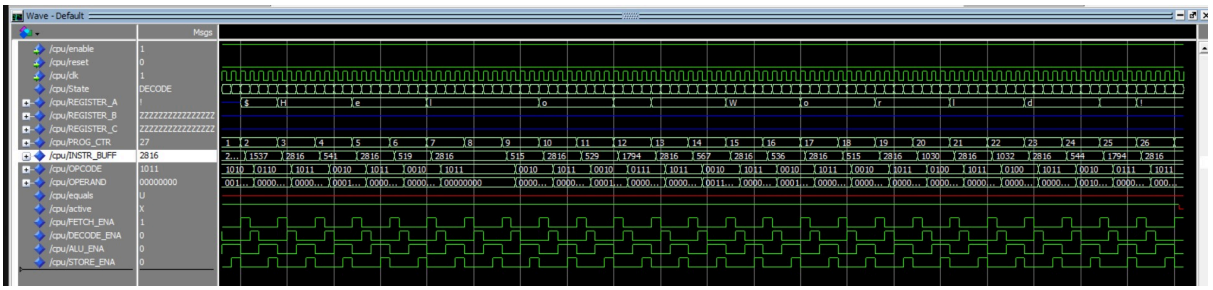

Fig 1. Testing Result

1. **Clock Signal (/cpu/clk)**
   Berfungsi stabil secara periodik dengan gelombang persegi, memastikan siklus sistem berjalan baik.
2. **Reset dan Enable Signals (/cpu/reset, /cpu/enable)**
   Sinyal reset aktif di awal simulasi dan sinyal enable diatur ke 1 untuk mengaktifkan CPU.
3. **Program Counter (/cpu/PROG_CTR)**
   Bertambah secara berurutan, menunjukkan instruksi diproses dalam urutan yang benar.
4. **Instruction Buffer (/cpu/INSTR_BUFF)**
   Menyimpan instruksi yang sedang diproses, seperti 2816, 1537, dan 567.
5. **Opcode dan Operand (/cpu/OPCODE, /cpu/OPERAND)**

Opcode berubah sesuai instruksi yang dijalankan, sedangkan operand tetap 0 selama simulasi.

6. **State CPU (/cpu/State)**
   Berubah sesuai tahapan siklus instruksi, seperti FETCH dan DECODE.
7. **Enable Signals (/cpu/FETCH_ENA, /cpu/DECODE_ENA, /cpu/ALU_ENA, /cpu/STORE_ENA)**
   /cpu/FETCH_ENA aktif saat instruksi diambil, dan /cpu/DECODE_ENA aktif saat instruksi diterjemahkan.

Berikut adalah hasil dari sintesis, lengkap dengan petunjuk komponen yang ada didalamnya

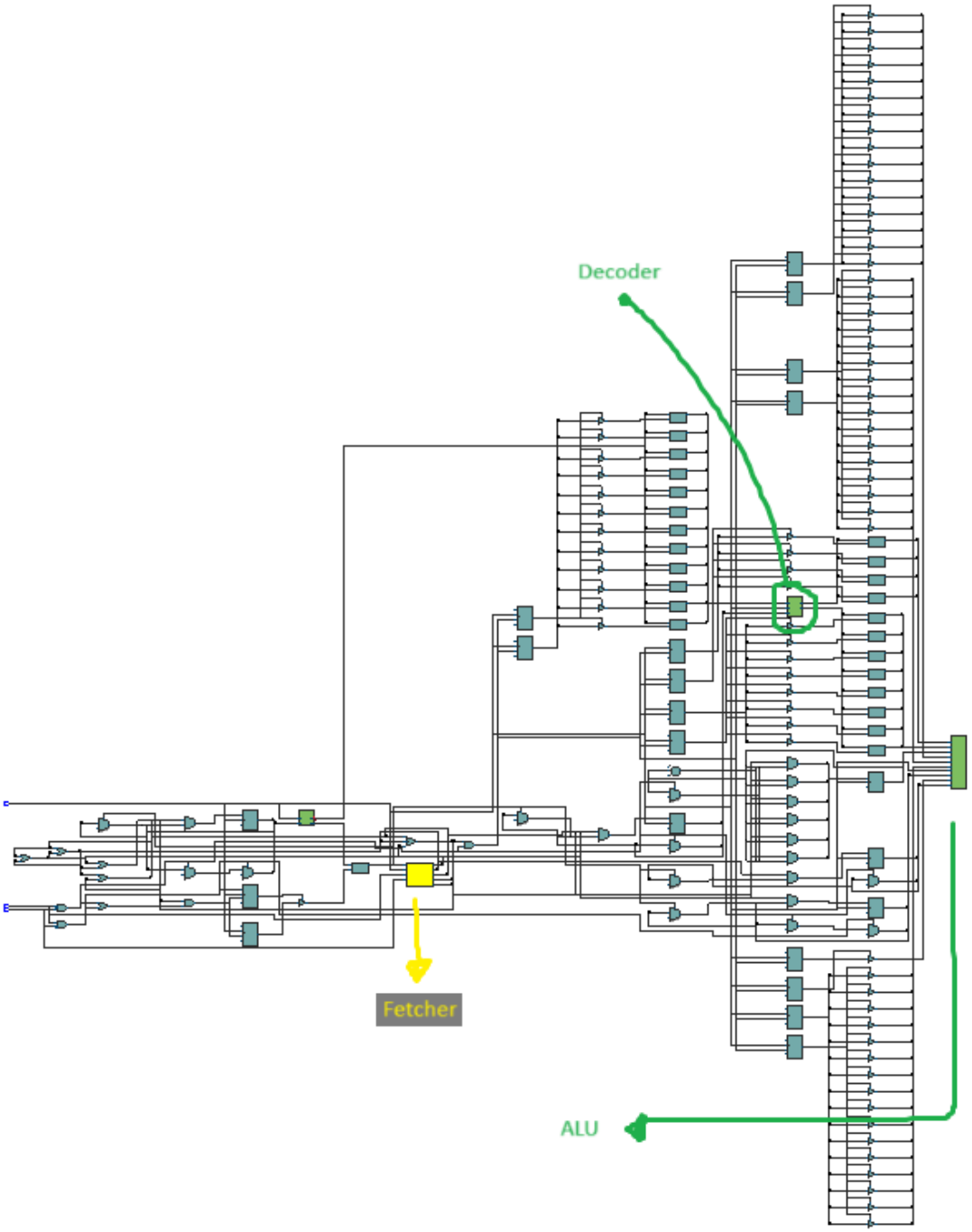

Fig 2. Synthesis Result

3.3 ANALYSIS

Dari hasil simulasi gelombang dapat disimpulkan bahwa sistem CPU sederhana telah berjalan dengan benar. Setiap komponen yang digunakan dalam sistem ini, seperti state machine, program counter, register, dan control signal, dapat bekerja sama dan menghasilkan

output yang sesuai dengan alur eksekusi instruksi yang diharapkan. Sama seperti desain awal, CPU memproses instruksi dari tahap fetch, decode, hingga execute secara berurutan.

Pada simulasi ini, state machine menunjukkan transisi yang sesuai, dimulai dari state FETCH, dilanjutkan ke DECODE, dan selanjutnya menuju state EXECUTE. Hal ini menandakan bahwa proses pembacaan dan interpretasi instruksi berhasil dilakukan. Program counter (PROG_CTR) terus meningkat secara otomatis pada setiap siklus, yang mencerminkan bahwa CPU membaca instruksi secara berurutan dari memori eksternal.

Instruksi yang telah di-fetch disimpan dalam instruction buffer (INSTR_BUFF), yang kemudian dipecah menjadi opcode (OPCODE) dan operand (OPERAND). Opcode pada simulasi ini adalah 1011, menunjukkan jenis operasi yang sedang didekodekan. Operand berada dalam nilai default (00000000), sesuai dengan instruksi masukan.

Selama tahap DECODE, sinyal kontrol seperti FETCH_ENA, DECODE_ENA, dan ALU_ENA aktif dan nonaktif sesuai dengan urutan proses. Hal ini menunjukkan bahwa sistem kontrol CPU bekerja sesuai dengan desain. Register-register internal (REGISTER_A, REGISTER_B, dan REGISTER_C) berada dalam kondisi high-impedance (ZZZZZZZZZZZZZZZZ) karena tidak aktif selama simulasi ini. Hal ini sesuai dengan desain, di mana register hanya akan digunakan pada state EXECUTE.

CHAPTER 4

CONCLUSION

Pengembangan CPU dengan Sistem Instruksi berhasil dirancang dengan solusi arsitektur yang efektif dan efisien. Sistem ini tidak hanya memastikan eksekusi instruksi secara berurutan melalui program counter, tetapi juga dilengkapi dengan mekanisme yang memungkinkan pengambilan dan penguraian instruksi yang tepat. Keamanan dan kinerja sistem ditingkatkan dengan penggunaan sinyal enable yang mengontrol tahap-tahap seperti FETCH, DECODE, dan instruksi lainnya. Implementasi ini menghasilkan CPU yang handal, mampu memproses instruksi dengan tepat dan stabil, serta siap untuk digunakan dalam aplikasi yang lebih kompleks. Solusi ini dapat diterapkan secara luas untuk meningkatkan

kinerja perangkat keras dalam berbagai lingkungan komputasi dan memberikan platform yang dapat diandalkan untuk pengembangan sistem digital yang lebih lanjut.

REFERENCES

[1] BBC, “Opcodes and operands - programming languages and Integrated Development Environments - OCR - GCSE computer science revision - OCR - BBC Bitesize,” BBC News, https://www.bbc.co.uk/bitesize/guides/z6x26yc/revision/4 (accessed Dec. 8, 2024).

[2] A. Singh, “Instruction cycle,” University of Lucknow, https://udrc.lkouniv.ac.in/Content/DepartmentContent/SM_85eb2f65-0a76-4e8c-96b5-e82fe740137e_58.pdf (accessed Dec. 8, 2024).

[3] R. Merrick, “Create Tri-State Buffer in VHDL and Verilog,” Nandland, https://nandland.com/create-tri-state-buffer-in-vhdl-and-verilog/#google_vignette (accessed Dec. 8, 2024).

[4] R. Merrick, “VHDL Example Code of file Io,” Nandland, https://nandland.com/file-input-output/ (accessed Dec. 8, 2024).

[5] J. J. Jensen, “How to use port map instantiation in VHDL,” VHDLwhiz, https://vhdlwhiz.com/port-map/ (accessed Dec. 8, 2024).