PROGRAMMABLE BASIC CPU

GROUP PA-07
RIVI YASHA HAFIZHAN 2306250535
MAXWELL ZEFANYA GINTING 2306221200
MUHAMAD DZAKY MAULANA 2306264401
REICHAN ADHIGUNO 2106703273

Latar Belakang

Teknologi digital yang terus berkembang menempatkan CPU sebagai elemen penting dalam sistem komputer. Sebagai otak komputer, memahami cara kerja CPU menjadi fondasi dalam studi arsitektur komputer. Namun, pemahaman ini sering terbatas pada teori tanpa praktik.

Proyek ini bertujuan untuk merancang CPU sederhana yang mampu menjalankan 8 instruksi dasar: CMP, JUMP, ADD, SUB, LOAD, STORE, AND, NOT. CPU dirancang menggunakan file sebagai input dan output untuk mempermudah pemahaman alur kerja secara komprehensif. Dengan desain ini, proyek memberikan pengalaman praktis dan terarah dalam mempelajari konsep dasar arsitektur komputer.

Deskripsi Proyek

Proyek ini bertujuan merancang CPU programmable sederhana untuk memahami konsep arsitektur komputer. CPU dilengkapi tombol enable dan reset, dengan semua proses disinkronkan melalui clock. Komponen utamanya terdiri dari 3 register 16-bit dan 1 bendera equals untuk operasi perbandingan, yang hanya diakses melalui opcode.

Cara kerjanya dimulai dengan membaca berkas masukan berformat bilangan bulat, kemudian menjalankan instruksi secara berurutan. Desain sederhana ini memberikan metode praktis untuk memahami cara kerja CPU dari tahap pembacaan hingga eksekusi.

Tujuan

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.

Alat yang digunakan

1. ModelSim
2. Intel Quartus Prime
3. Visual Studio Code

Implementasi

1. Modul
2. Program

Modul

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.

Modul

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.

Modul

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.

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 (Central Processing Unit) - CPU berfungsi untuk menangani berbagai status seperti FETCH, DECODE, EXECUTE, dan STORE. CPU berinteraksi dengan komponen lain seperti ALU, Decoder, dan Fetcher.

Program

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.

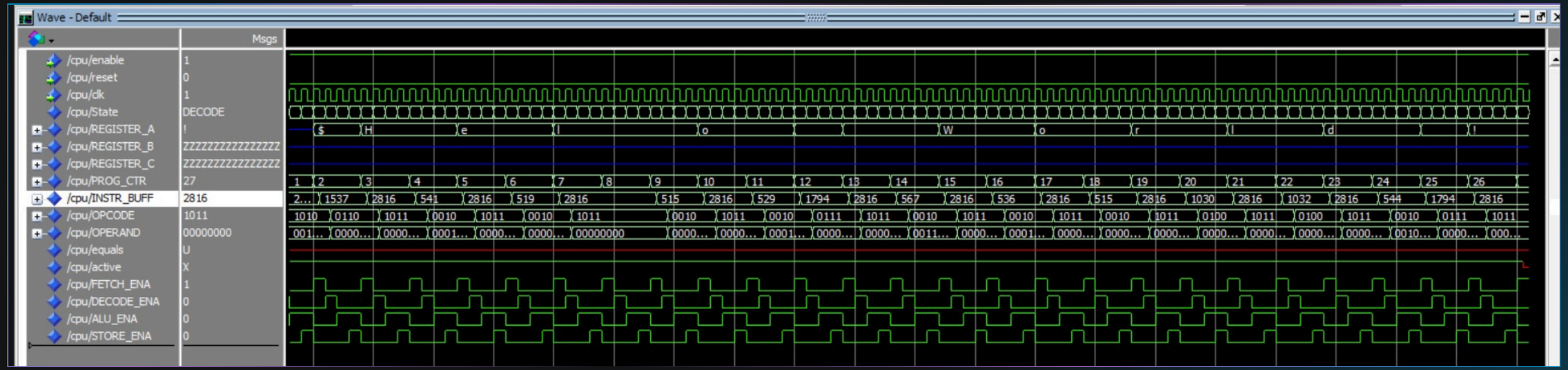
4. Decoder - Decoder berfungsi menerjemahkan instruksi mentah dari Fetcher menjadi sinyal kontrol untuk modul lain seperti ALU dan register. Modul ini memisahkan opcode dari operand dan menentukan operasi yang akan dilakukan sesuai dengan instruksi yang diterima.

Percobaan

Testing dilakukan menggunakan Modelsim dengan input berupa CLK, ENABLE, dan RESET. CPU berjalan saat CLK aktif, ENABLE = 1, RESET = 0, dan terdapat file "Instructions.txt". CPU membaca instruksi dalam file tersebut hingga selesai, lalu menulis hasil ke "Outputs.txt" jika terdapat instruksi sesuai. Pengujian dilakukan secara manual dengan mengubah sinyal atau menggunakan testbench.

Dengan testbench, sinyal tidak muncul di waveform, tetapi semua fungsi CPU, termasuk penulisan output dan interaksi antar register, tetap berjalan normal. Verifikasi dilakukan dengan instruksi "STO-R" untuk menulis data register ke file output. Sebagai contoh, program yang menuliskan "Hello World!" dalam bentuk integer menggunakan register A akan memvalidasi fungsi CPU. Desain juga diuji dengan sintesis menggunakan Quartus untuk memastikan keakuratan.

Hasil Percobaan

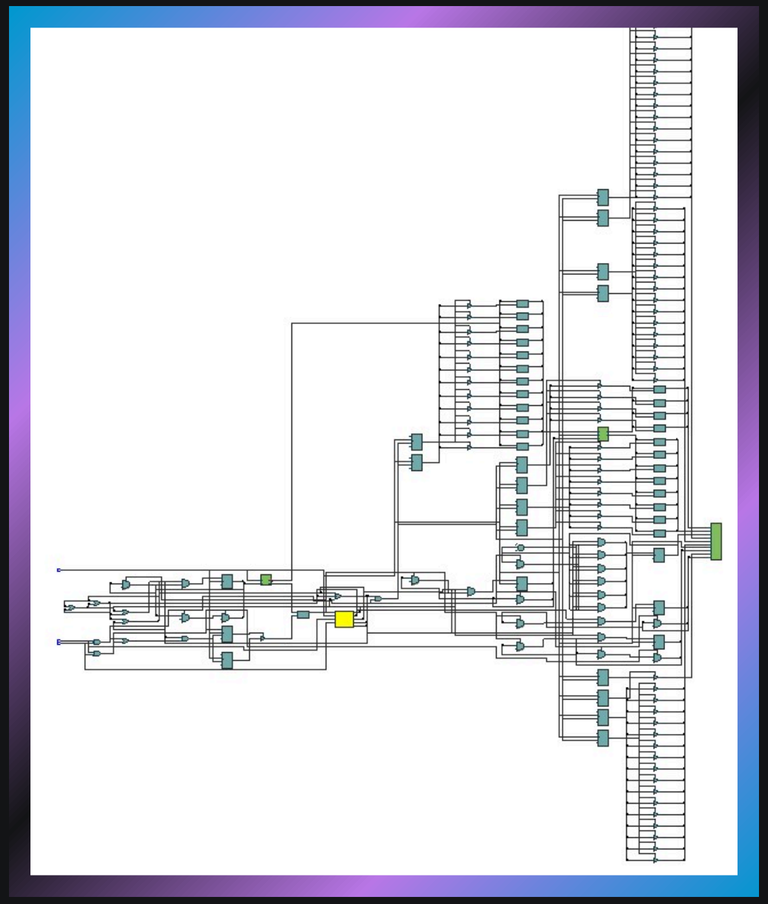


Analisis

Hasil simulasi gelombang menunjukkan bahwa sistem CPU sederhana berfungsi dengan benar. Komponen seperti state machine, program counter, register, dan control signal bekerja sama untuk menghasilkan output sesuai alur instruksi. CPU memproses instruksi secara berurutan dari tahap FETCH, DECODE, hingga EXECUTE, dengan program counter (PROG_CTR) otomatis meningkat pada setiap siklus untuk membaca instruksi dari memori eksternal. Instruksi yang di-fetch disimpan dalam instruction buffer (INSTR_BUFF) dan dipecah menjadi opcode (1011) serta operand (00000000).

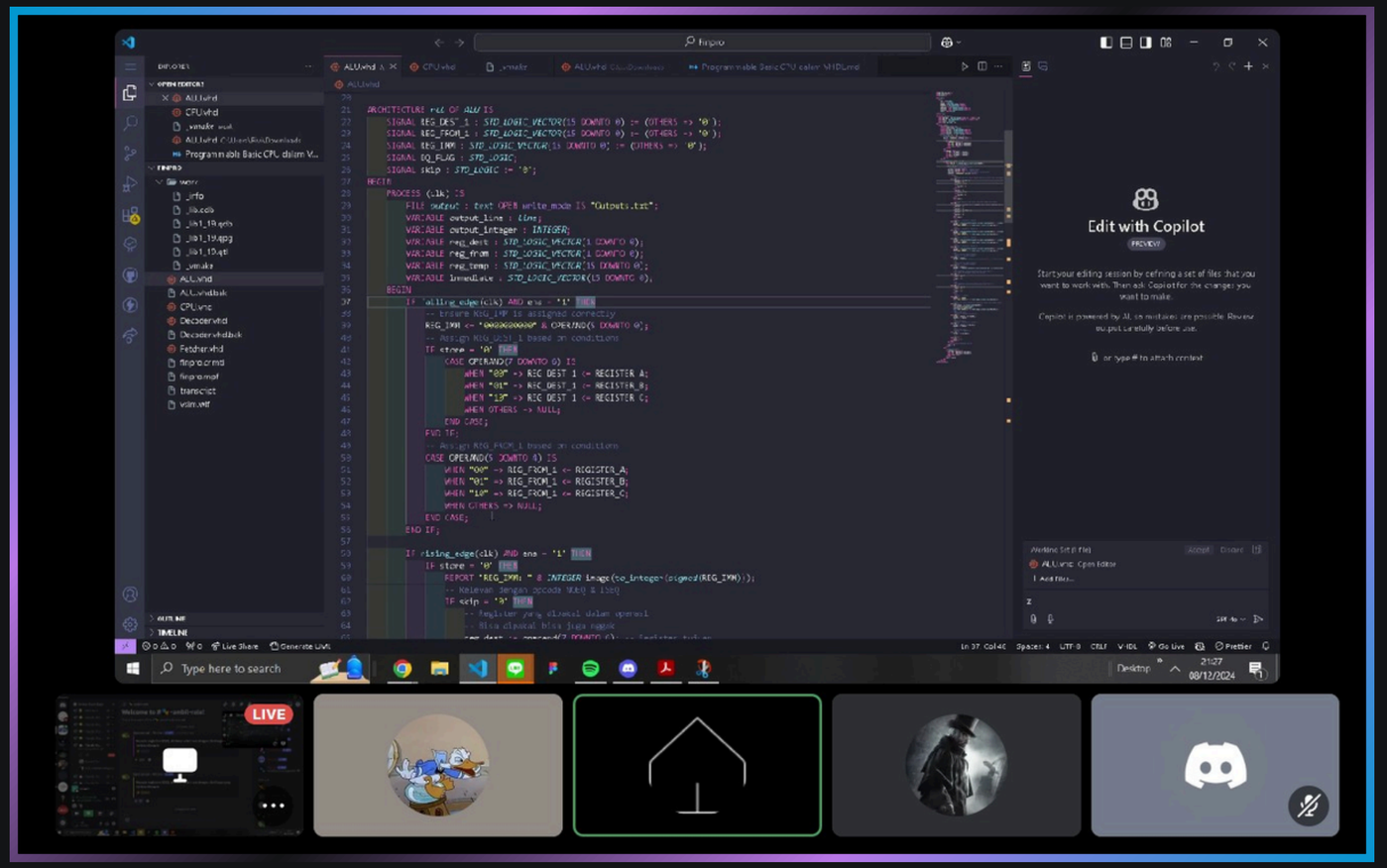
Pada tahap DECODE, sinyal kontrol seperti FETCH_ENA, DECODE_ENA, dan ALU_ENA aktif sesuai urutan proses, menandakan sistem kontrol bekerja sesuai desain. Register internal (REGISTER_A, REGISTER_B, REGISTER_C) berada dalam kondisi high-impedance selama simulasi, karena tidak aktif kecuali pada state EXECUTE. Hal ini sesuai dengan desain awal CPU.

Appendix



Project Schematic

Appendix



Dokumentasi Pekerjaan

Kesimpulan

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.

Referensi

[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).

Thank You