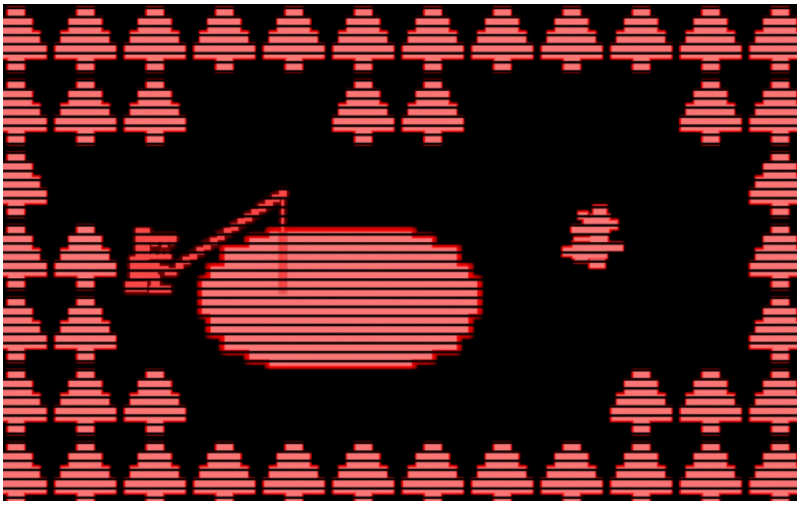
4-Bit Fisherman



“I'm sorry to say that you have gone too deep into the code. There is no way back out. Come have a seat, and let's fish for a while. You have nowhere else to go.”

Simulasi Komputer 4-Bit dalam Logisim

Ditulis Oleh:
M. Rayhan Farrukh

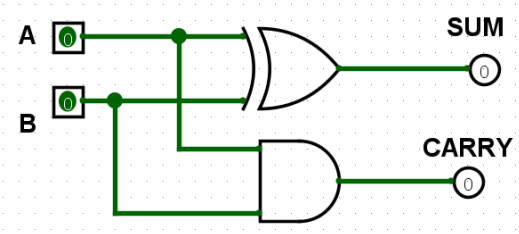
Daftar Isi

Daftar Gambar

A. CPU

Pada implementasi yang saya buat, CPU hanya terdiri atas ALU sebagai komponen penghitung aritmatika. Sebelumnya saya mencoba mengimplementasikan 4-bit register, tapi saya *skill issue* dan terlalu memakan banyak waktu. Karena di spesifikasi tidak disebutkan harus membuat register dan juga tidak terlalu berguna pada saat pengujian, saya memilih untuk tidak menggunakan *register*.

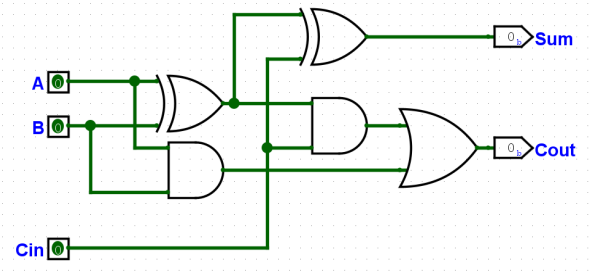
Half Adder



Gambar 1. Implementasi *Half adder*

Half adder merupakan unit terkecil dalam operasi aritmatika. Komponen ini menerima dua *input* 1-bit dan menghitung hasil penjumlahan beserta *carry*-nya. Implementasinya menggunakan satu XOR *gate* untuk *bit* hasil jumlah dan 1 AND *gate* untuk *carry*.

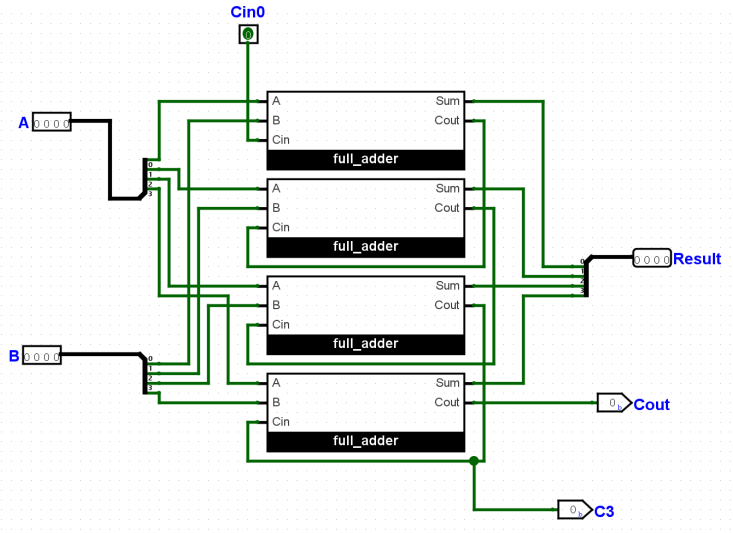
Full Adder



Gambar 2. Implementasi *Full adder*

Full adder adalah peningkatan dari *half adder,* yang bisa menerima 3 *input* untuk melakukan penjumlahan. Ini memungkinkannya untuk menerima *carry* dari penjumlahan lain sehingga bisa dirantai untuk menghitung angka lebih dari 1-bit. Implementasinya menggunakan 2 *half adder* serta satu OR *gate* untuk *carry output*.

Ripple-Carry Adder



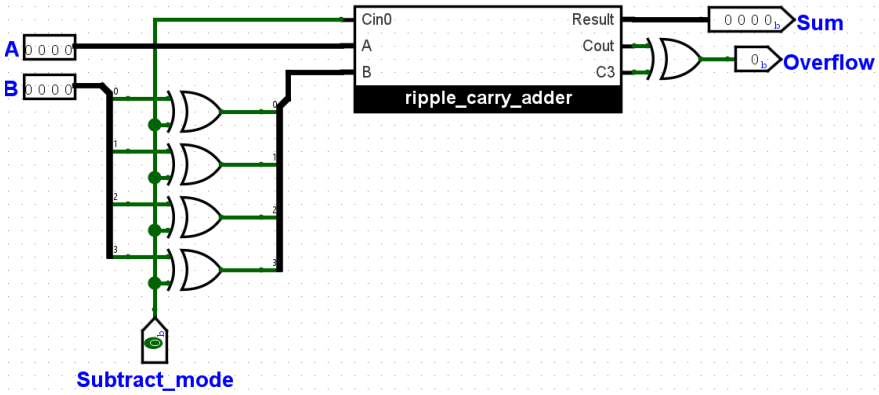
Gambar 3. Implementasi *Ripple-carry adder*

Ripple-carry adder adalah implementasi perantaian dari *full adder* untuk menghitung angka lebih dari 1-bit. Implementasinya menggunakan *full adder* sebanyak jumlah bit yang ingin dijumlahkan.

Untuk membuatnya saya menggunakan 2 4-bit *input pin* sebagai *input* dua angka yang ingin dijumlahkan, serta menggunakan satu 1-bit *input* sebagai *Cin*. *Input* 1-bit ini akan berguna nantinya untuk membalikkan tanda (+/-) menggunakan *two's complement*.

Output dari komponen ini ada 3, yaitu satu 4-bit *output* untuk hasil penjumlahan dan dua 1-bit *output* untuk menyimpan *carry out* yang akan digunakan sebagai *overflow flag*.

ALU



Gambar 4. Implementasi *Arithmetic Logic Unit*

ALU adalah komponen utama dalam aritmatika komputer ini. Secara sederhana, ALU adalah gabungan dari *Ripple-carry adder* dan sebuah *sign flipper* yang dapat mengubah tanda suatu angka sebagai operasi pengurangan.

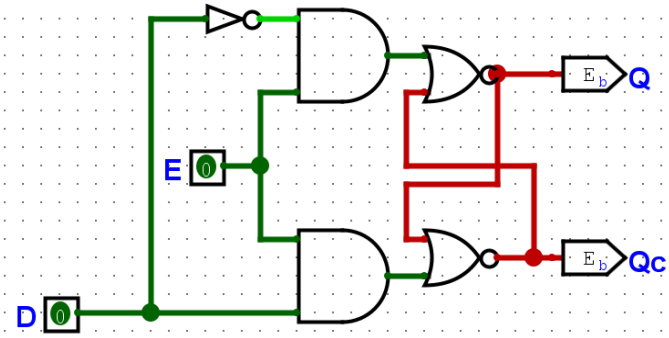
Seperti yang terlihat pada gambar, ALU menerima dua 4-bit *input* dan kemudian meneruskannya ke *Ripple-carry adder*. Namun, input B akan dilalui XOR *gate* bersama dengan 1-bit *subtract flag*.

Ketika *subtract mode* aktif, evaluasi pada XOR *gate* akan menghasilkan 0 untuk bit B yang bernilai 1, dan akan menghasilkan 1 untuk yang bernilai 0, ini merupakan operasi **inversi**. Nilai dari *subtract flag* sendiri akan dimasukkan ke Cin sebagai +1, sesuai formula *sign flipping two's complement* (-A = ~A + 1).

B. Memori

Memori yang saya implementasikan hanyalah memori sederhana untuk menyimpan satu angka 4-bit, yang dibuat menggunakan 4 *d-latch*.

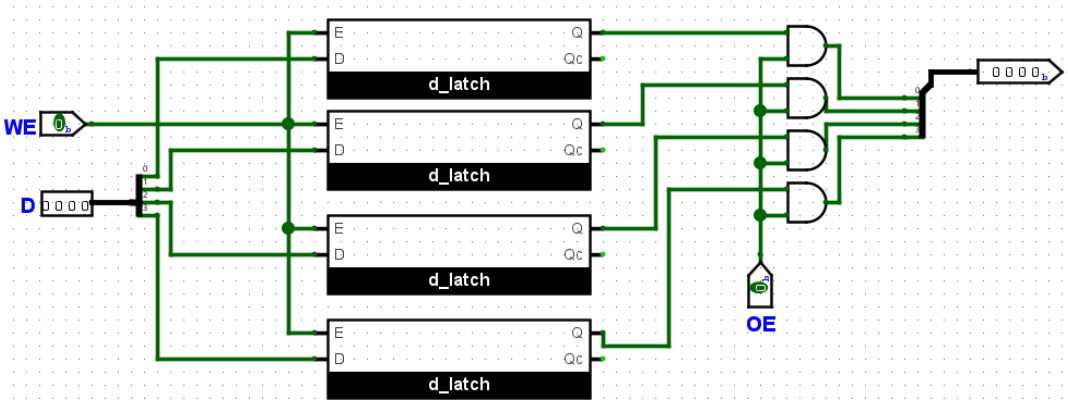
D-Latch



Gambar 5. Implementasi D-Latch

D-latch, atau yang juga dikenal sebagai *transparent latch*, adalah komponen memori sederhana yang dapat menyimpan satu bit data. *D-latch* menerima dua *input*, yaitu data dan kontrol. *D-latch* hanya menyimpan data ketika kontol aktif (*high signal*), dan akan mengabaikan data yang masuk jika kontrol nonaktif. Bisa dilihat pada gambar 5, data dilalui AND *gate* yang hanya *true* jika kontrol sedang aktif.

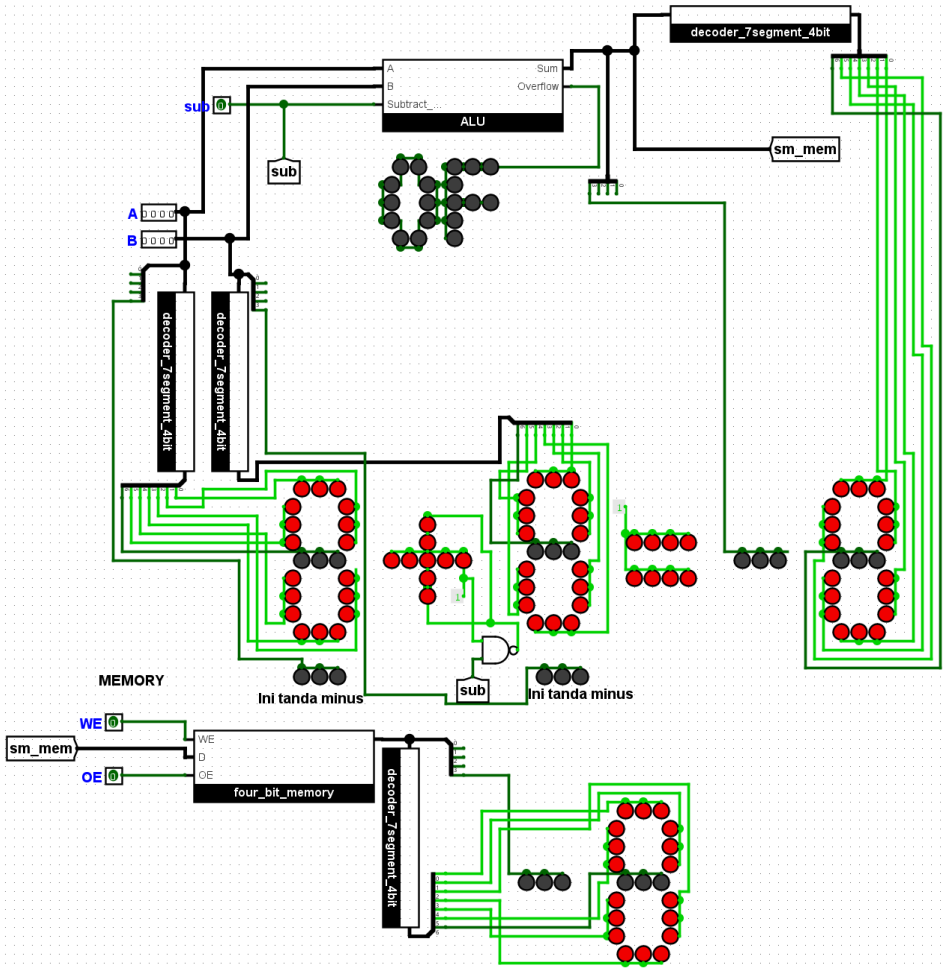
Memory



Gambar 6. Implementasi Memori

Seperti yang disebutkan sebelulmnya, memori yang dibuat hanyalah memori sederhana yang menyimpan satu angka 4-bit. Ini dibuat menggunakan 4 *D-latch*, yang masing-masing menyimpan satu bit data. Memori ini menerima tiga *input*, yaitu data, dan dua sinyal *write enable* dan *output enable*. *Write enable* akan mengaktifkan kontrol dari semua *D-latch* agar dapat menyimpan data, sedangkan *output enable* akan membuat data tersebut berhasil melewati sebuah *controlled buffer*. *Controlled buffer* dibuat dengan memasang AND *gate* pada *output* semua *D-latch*, yang akan menghasilkan *true* **hanya** jika *output enable* aktif.

C. Komputer Utama



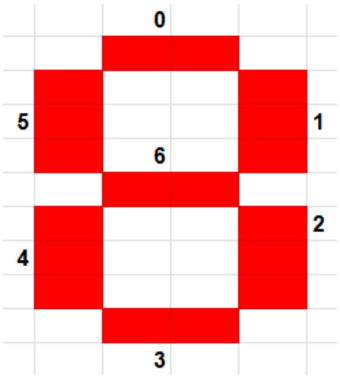
Gambar 7. Implementasi Komputer

Implementasi komputer ditunjukkan pada gambar 7. Komputer ini mampu melakukan operasi aritmatika dasar yaitu penjumlahan dan pengurangan. Komputer menerima 5 *input* untuk mengendalikan data yang diproses pada ALU dan data pada memori. ALU dan memori dihubungkan melalui sebuah *tunnel*, ini berfungsi untuk mempermudah *wiring* saja, tidak ada logika tambahan.

Aritmatika dilakukan pada ALU dan hasilnya diteruskan ke memori, yang bisa menyimpan nilai atau mengabaikannya menggunakan sinyal WE. Nilai yang disimpan pada memori dapat ditampilkan dengan mengaktifkan sinyal OE. Semua angka ditampilkan menggunakan *seven segment display* agar memudahkan dalam melihat angka. Jika ada *overflow*, maka LED OF (*overflow*) akan menyala.

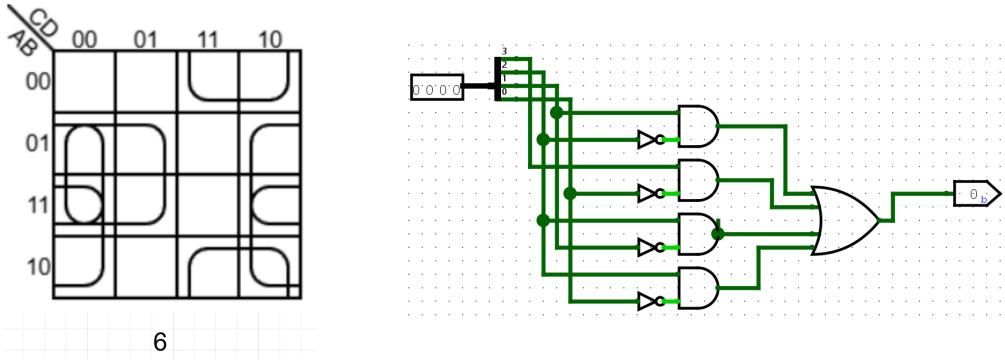
D.7 Segment Display

Seven segment display dibuat menggunakan komponen LED, yang masing-masing bagiannya diatur menggunakan ***combinational logic***.



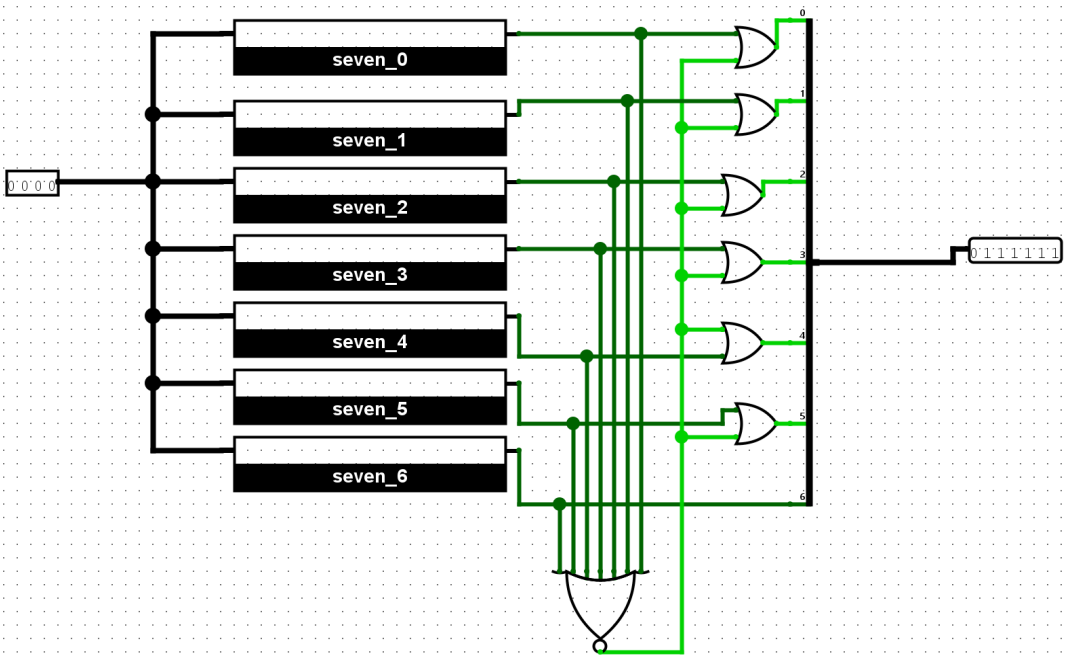
Gambar 8. Ilustrasi *seven segment display*

Masing-masing bagian dinomorkan dari 0 hingga 6, yang akan menjadi urutan bit dari hasil *decoder*. *Decoder* digunakan untuk mentranslasikan angka 4-bit menjadi data 7-bit menggunakan *combinational logic*. Untuk ini, saya membuat *truth table* untuk mencatat bagian nomor berapa saja yang menyala untuk setiap angka 4-bit, kemudian membuat 7 K-map untuk masing-masing bagian. Untuk mempermudah membuat dan menyelesaikan K-map, saya menggunakan situs ini link.



Gambar 9. Contoh K-map untuk segmen 6, dan implementasi *combinational logic*-nya

Untuk melihat *truth tables* dan diagram K-map secara lengkap bisa membuka *link* yang tersedia pada Lampiran. Catatan: ada kesalahan pada *truth table* dan K-map, seharusnya untuk 0, segmen dihidupkan juga, namun saya mati buat ada *truth table*. Karena sudah terlanjur membuat yang salah ini, dan akan memakan waktu jika harus *remake*, saya *handle* logikanya pada *decoder* utama, yang bisa dilihat pada gambar 10.



Gambar 10. Implementasi *decoder seven segment display*

Decoder **dibuat dengan menghubungkan 4-bit *input* pada semua *combinational logic*, yang masing-masing akan menghasilkan keluaran 1-bit. Keluaran ini kemudian melalui sebuah NOR *gate* yang akan menghasilkan *true* jika semua bit 0 dan akan meng-*handle special case* untuk angka 0 (karen kesalahan yang saya sebutkan tadi).**

Lampiran

- Video demo: youtube atau gdrive
- Sheets *truth table*: *link*
- Draw.io K-map: *link*