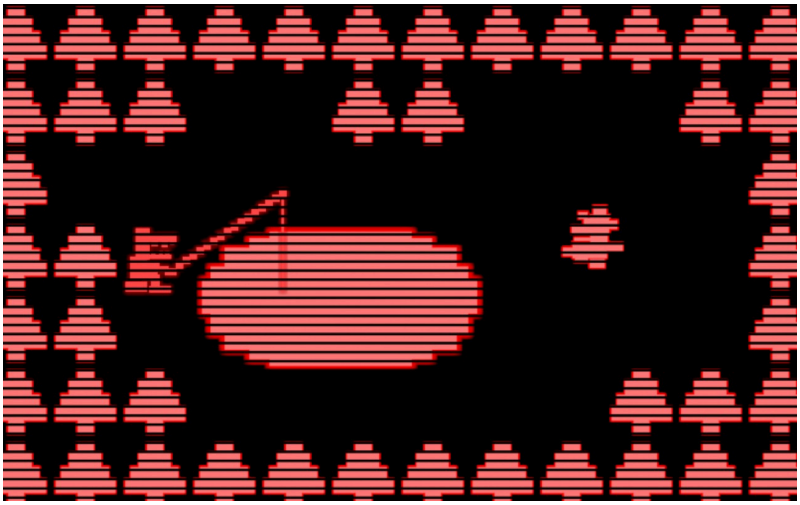
4-Bit Fisherman



“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

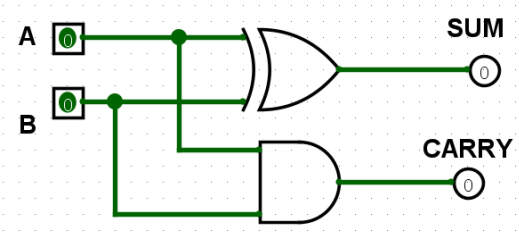
Daftar Gambar

CPU

Pada implementasi yang saya buat, CPU terdiri dari dua register untuk menyimpan nilai *input*, sebuah ALU untuk penjumlahan atau pengurangan, lalu komponen perkalian. Berikut penjelasan komponen-komponen yang diimplementasikan dan dipakai untuk membentuk CPU.

A. ALU

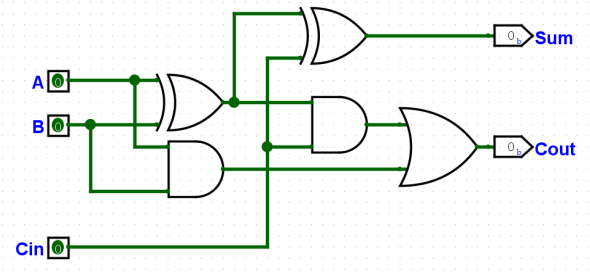
Half Adder



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*.

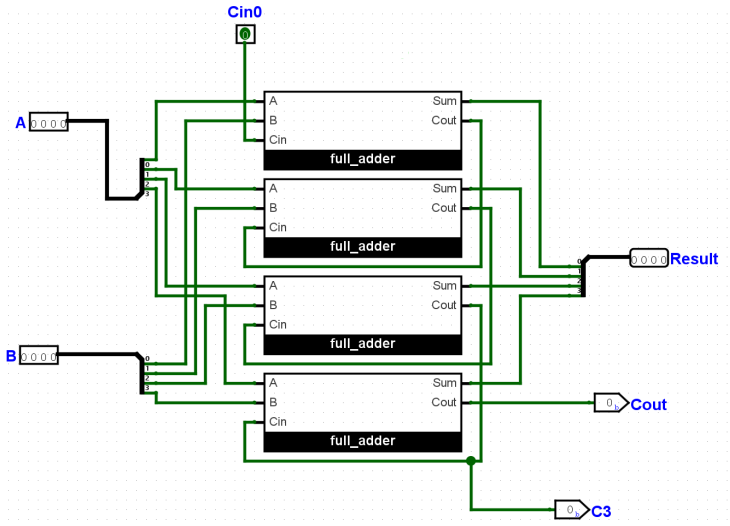
Full Adder



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



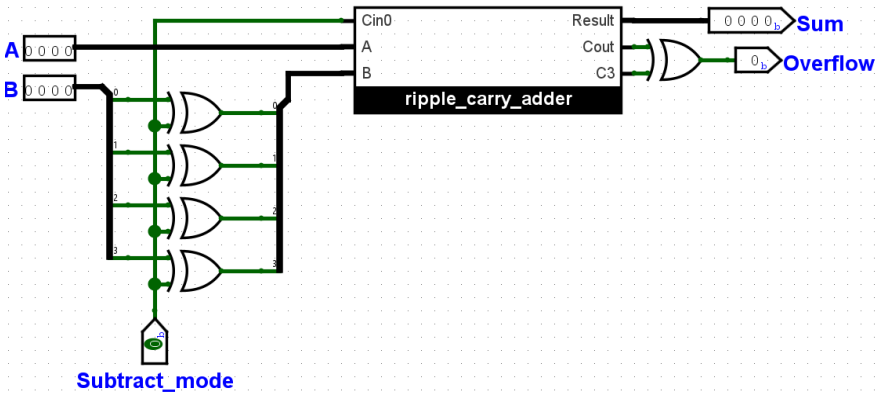
Gambar 3. Implementasi *Ripple-carry adder*

Ripple-carry adder adalah 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*

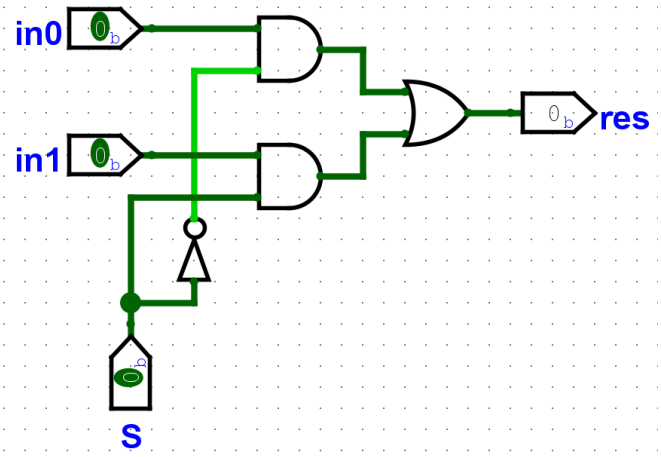
Secara sederhana, ALU adalah gabungan dari *Ripple-carry adder* dan sebuah *sign flipper* untuk mengubah tanda suatu angka ketika operasi pengurangan.

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. Register

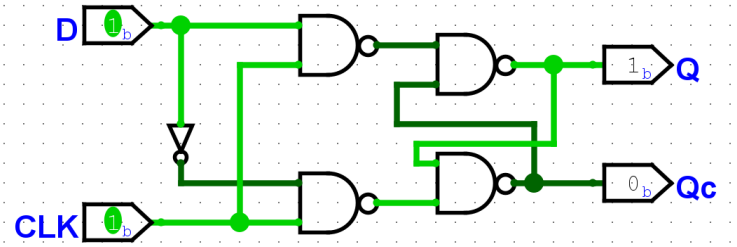
Multiplexer 2:1



Gambar 5. Implementasi *Multiplexer*

Multiplexer adalah komponen sederhana yang berfungsi untuk mengambil satu *input* dari dua *input* yang dimasukkan. Cara kerjanya dengan menggunakan dua AND *gate* dan satu NOT *gate* yang dipasangkan dengan sebuah *select bit* untuk menentukan input mana yang menghasilkan nilai 1 pada AND *gate* dan lanjut ke *output*.

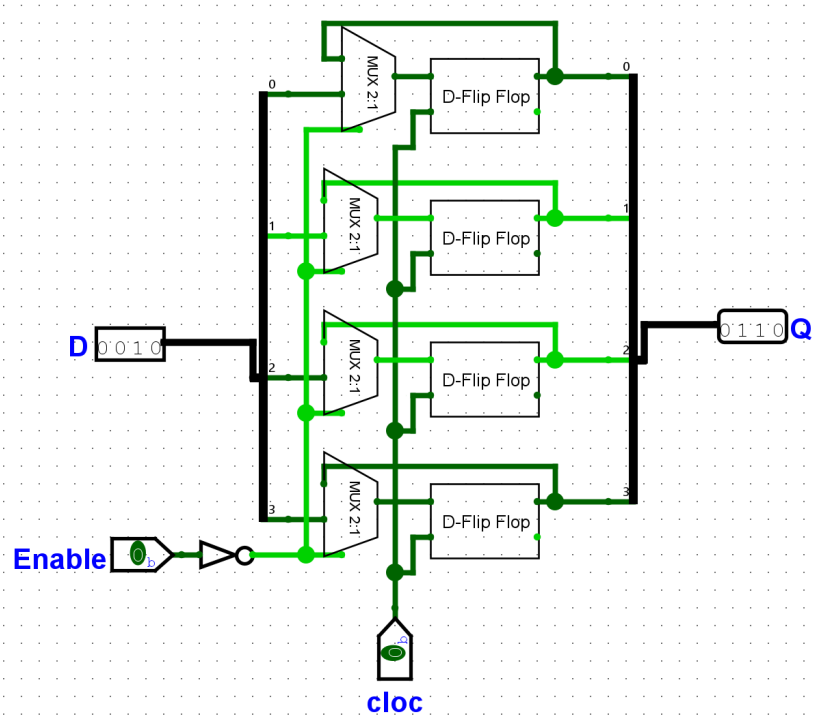
D-Flip Flop



Gambar 6. Implementasi D-Flip Flop

D-Flip Flop adalah komponen memori dasar yang berfungsi menyimpan satu bit data. D-Flip Flop menerima dua *input*, yaitu data dan kontrol melalui *clock*. D-Flip Flop akan menyimpan data ketika *clock* berubah nilai menjadi 1, dan akan mengabaikannya ketika *clock* bernilai 0. Bisa dilihat pada gambar 5, data akan melalui AND *gate*, yang hanya bernilai *true* jika kontrol sedang aktif. *Output* dari D-Flip Flop adalah bit data yang disimpan dan komplemen dari data tersebut.

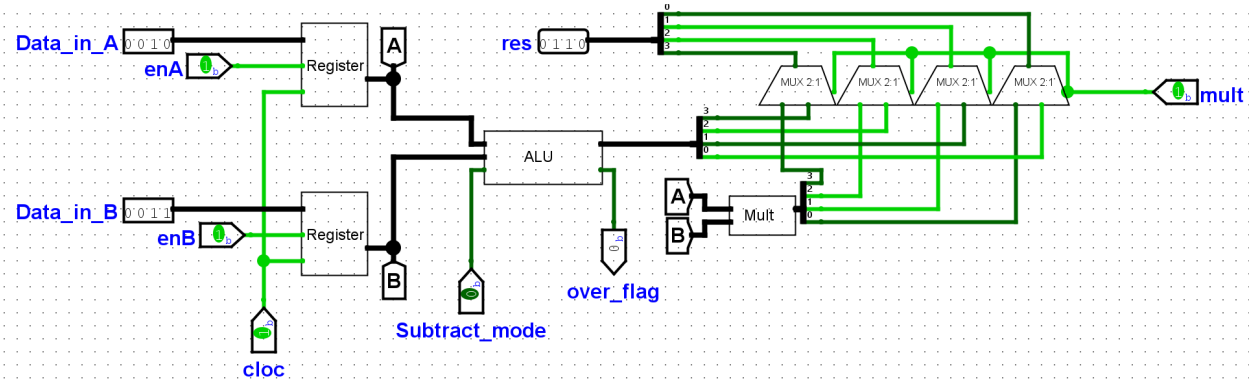
Register



Gambar 7. Implementasi *Register* 4-bit

Register adalah memori pada CPU yang cepat dan berfungsi untuk menyimpan data-data yang akan diolah oleh CPU. *Register* yang saya buat menggunakan empat D-Flip Flop untuk menyimpan data 4-bit dan empat *multiplexer* untuk menjaga data yang disimpan. Pada *register*, *switch enable* berfungsi sebagai *select bit* dari *multiplexer*, sehingga ketika aktif, *multiplexer* akan meneruskan data ke D-Flip Flop untuk disimpan. Sedangkan ketika *enable* tidak aktif, *multiplexer* akan meneruskan data yang sedang disimpan pada D-Flip Flop, sehingga data tidak berubah dari sebelumnya.

C. CPU



Gambar 8. Implementasi *CPU*

Saya membuat CPU dengan menggabungkan *register* dan komponen-komponen aritmatika. *Register* menyimpan data *input* yang dimasukkan sebelum diteruskan untuk perhitungan aritmatika. Komponen perhitungan dibuat terpisah, yang mana penjumlahan dilakukan pada ALU, sedangkan perkalian memiliki komponennya sendiri.

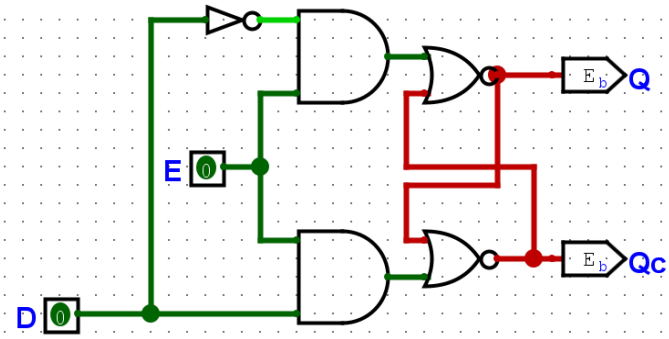
Karena pemisahan komponen perhitungan, saya membuat sebuah sistem untuk menyaring dan memilih *output* dari operasi mana yang akan dikeluarkan dengan menggunakan *multiplexer*, dengan *switch* 'mult' sebagai *select bit*.

Catatan: Komponen perkalian akan dijelaskan pada bagian Bonus.

RAM

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

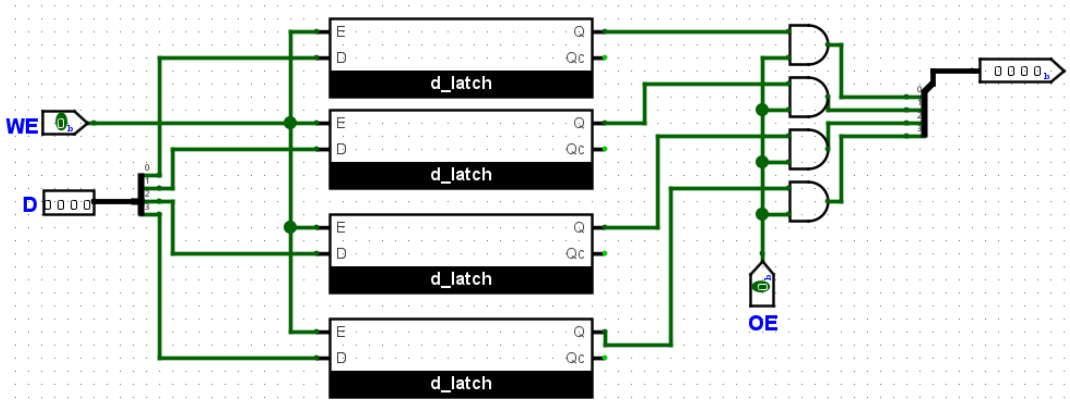
D-Latch



Gambar 9. Implementasi D-Latch

D-latch, atau yang juga dikenal sebagai *transparent latch*, adalah komponen memori sederhana yang dapat menyimpan satu bit data. *D-latch* bekerja serupa dengan D-Flip Flop, bedanya adalah D-Latch bersifat *level triggered*, dan *clock* digantikan dengan *switch enable*.

RAM

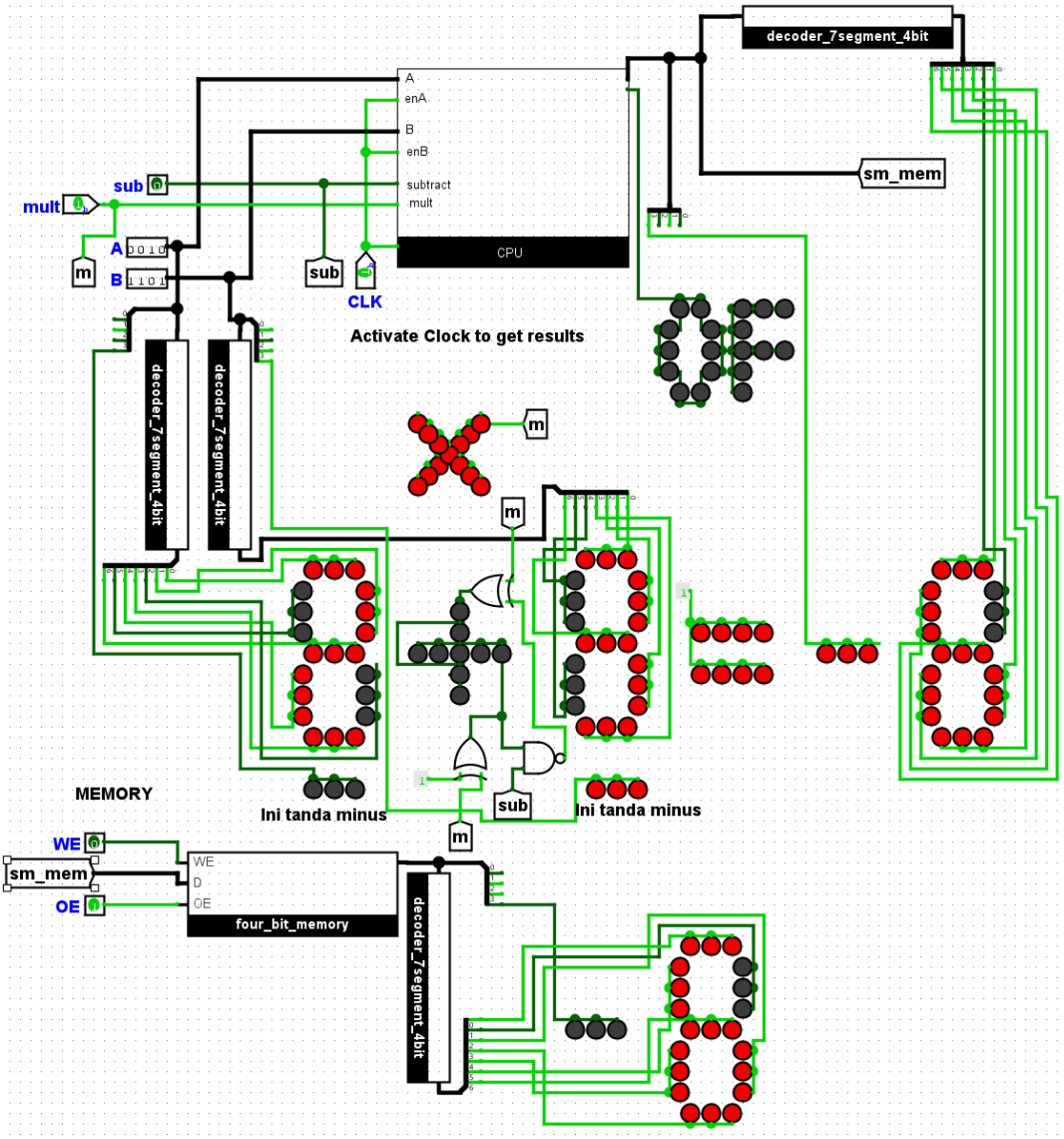


Gambar 10. Implementasi Memori

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.

Komputer Utama



Gambar 11. Implementasi Komputer

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

Aritmatika dilakukan pada CPU dan hasilnya diteruskan ke memori, yang bisa menyimpan nilai atau mengabaikannya menggunakan sinyal WE. Nilai yang disimpan

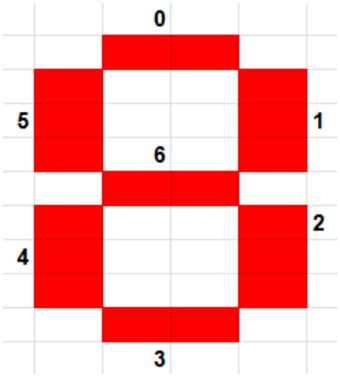
pada memori dapat ditampilkan dengan mengaktifkan sinyal OE. Semua angka ditampilkan menggunakan *seven segment display* agar memudahkan dalam melihat angka. Jika ada *overflow*, maka LED OF (*overflow*) akan menyala.

Untuk melakukan operasi, masukkan nilai *input* A dan B, kemudian tekan *clock*. Karena *register* bersifat *edge triggered*, operasi akan dilakukan ketika nilai *clock* berubah dari 0 menjadi 1. Lalu, jika ingin menyimpan nilai hasilnya, hidupkan *switch* 'WE', lalu matikan agar operasi berikutnya tidak meng-*overwrite* nilai yang disimpan. Untuk melakukan pengurangan, hidupkan *switch* 'sub', dan untuk perkalian, hidupkan *switch* 'mult'.

Bonus

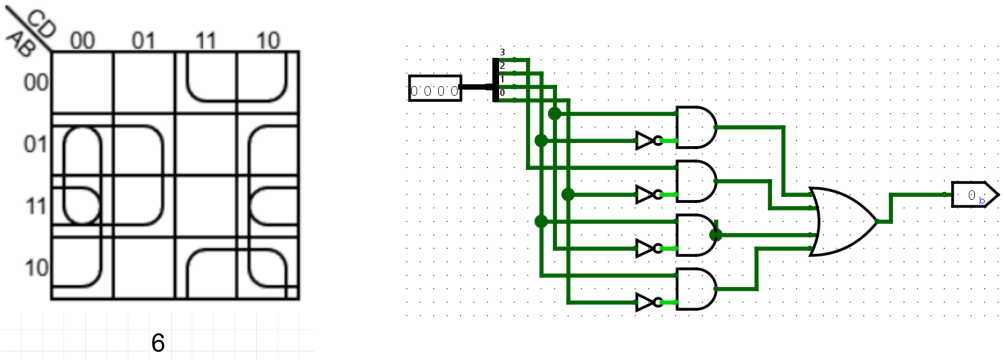
A. 7-Segment Display

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



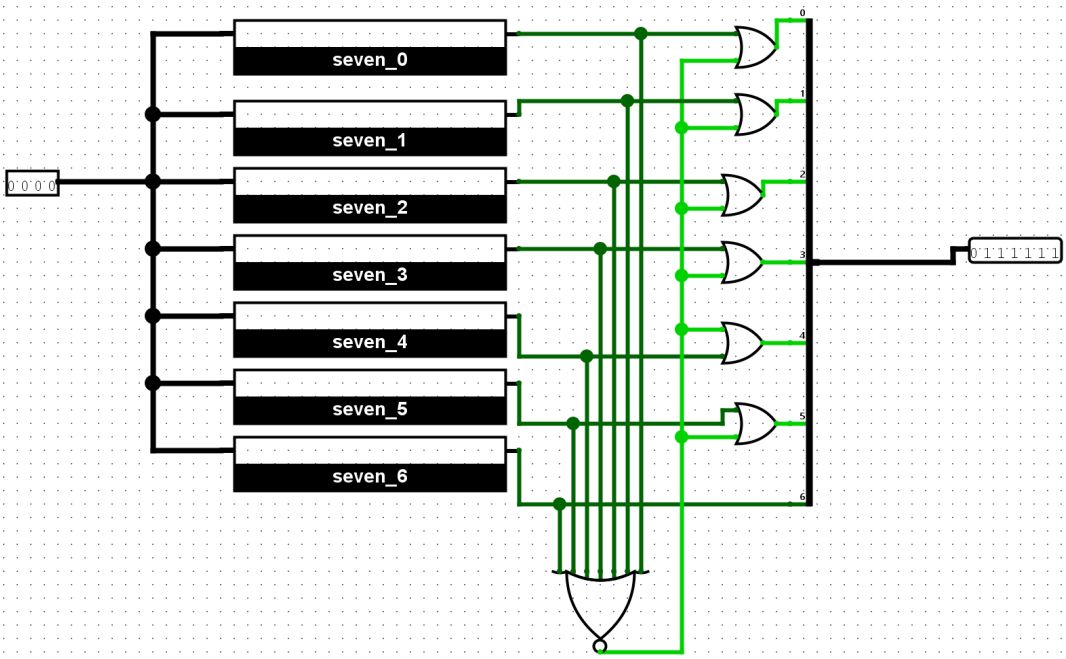
Gambar 12. Ilustrasi *seven segment display*

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

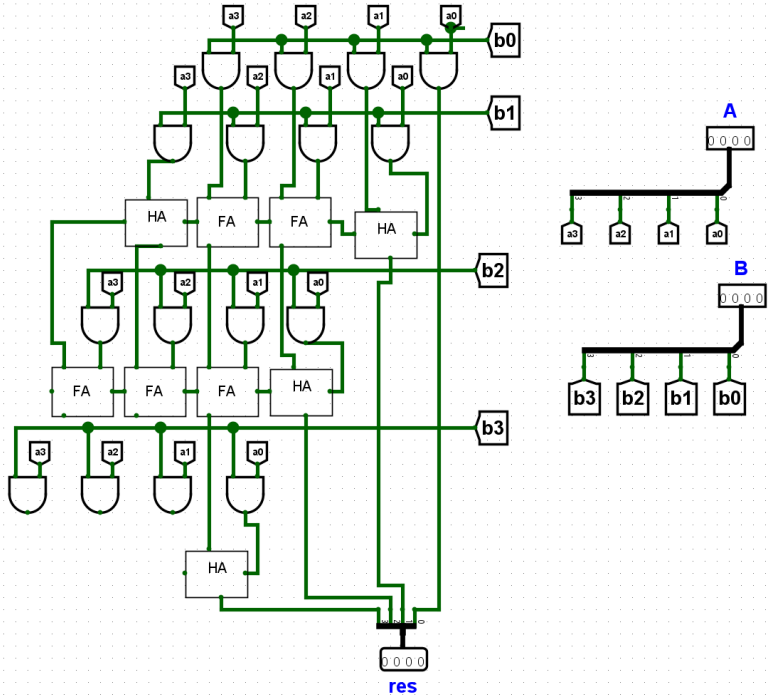
Untuk melihat *truth tables* dan diagram K-map secara lengkap bisa membuka *link* yang tersedia pada Lampiran. Catatan: ada kesalahan pada *truth table* dan K-map, seharusnya untuk 0, segmen dihidupkan juga, namun saya mati buat ada *truth table*. Karena sudah terlanjur membuat yang salah ini, dan akan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



Gambar 14. 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).**

B. Perkalian



Gambar 15. Implementasi *multiplier*

Untuk implementasi ini, saya mengikuti video berikut: youtube.com.
Komponen ini bekerja dengan cara meniru proses perkalian manual. AND *gate*, digunakan untuk menghitung *partial product* dan *adders* untuk menjumlahkan *partial product* tersebut. Peletakan komponen yang semakin ke bawah semakin condong ke kiri berfungsi sebagai *shift*. Ilustrasi proses perkalian sebagai berikut:

```
  0 1 0 1   ←— A
  0 1 1 0   ←— B
  0 0 0 0   ←— Hasil baris AND gate pertama (A x B1)
+ 0 1 0 1   ←— Hasil baris AND gate kedua (A x B2
  0 1 0 1 0
+ 0 1 0 1   ←— Hasil baris AND gate ketiga (A x B3
  0 1 1 1 1 0
+ 0 0 0 0   ←— Hasil baris AND gate keempat (A x B4)
  0 0 0 1 1 1 1 0
    ↑
    └— Bagian ini tidak diambil
```

Gambar 16. Ilustrasi perkalian

Perlu dicatat bahwa, perkalian ini hanya mengambil *lower 4 bits* dari hasil akhirnya, karena komputer hanya mendukung 4-bit. Oleh karena itu, angka yang bisa dikalikan terbatas, dengan hasil maksimum 6 (2x3, -6x-1, dll.) dan hasil minimum -8 (-2x4, 8x-1, dll.) Selain itu, *multiplier* ini sebenarnya bersifat *unsigned*, namun karena di-*truncate* ke 4-bit pertama saja, hasilnya dapat diinterpretasikan sebagai *signed* number.

Lampiran

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