PRAKTIKUM RANGKAIAN DIGITAL

RANGKAIAN LOGIKA

TUJUAN

1. Memahami berbagai kombinasi logika AND, OR, NAND atau NOR untuk mendapatkan gerbang dasar yang lain.
2. Menyusun suatu rangkaian kombinasi logika menggunakan satu jenis dan atau beberapa jenis gerbang logika untuk mendapatkan tabel kebenaran.

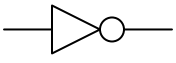
MATERI.

1. Operasi-operasi logika dasar
2. Tabel kebenaran
3. Gerbang-gerbang logika (Logic Gates)
4. Aljabar Boolean
5. Rangkaian ekivalen

TEORI DASAR

a. Operasi-operasi logika dasar

Ada beberapa operasi-operasi dasar pada suatu rangkaian logika dan untuk menunjukkan suatu perilaku dari operasi-operasi tersebut biasanya ditunjukkan dengan menggunakan suatu tabel kebenaran. Tabel kebenaran berisi statemen-statemen yang hanya berisi:

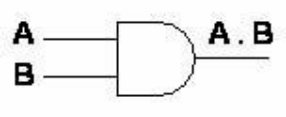
- Benar yang dilambangkan dengan huruf "T" kependekan dari "True" atau bisa juga dilambangkan dengan angka 1. atau
- Salah yang dilambangkan dengan huruf "F" kependekan dari "False" atau bisa juga dilambangkan dengan angka 0. Adapun operasi-operasi dasar logika adalah sebagai berikut:
- **Operasi INVERS :** 

  Operasi invers ini dilambangkan dengan tanda "¯" diatas variabel atau tanda single apostrope " ' ". Operasi invers ini akan mengubah logic benar/1 menjadi logic salah/0 dan begitu pula sebaliknya akan mengubah logic salah/0 menjadi logic benar/1. operasi ini dapat ditunjukkan dengan tabel kebenaran sebagai

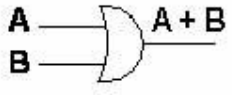
berikut:

A	A
0	1
1	0

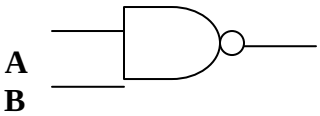
- AND



- OR

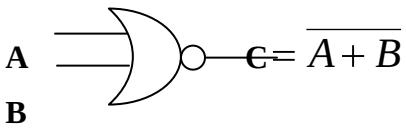


- NAND

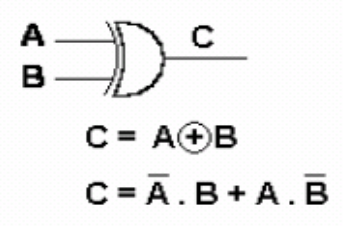


$$C = \overline{A.B}$$

- NOR



- XOR



Suatu gerbang logika dapat diperoleh dari satu atau beberapa gerbang logika lainnya yang disusun dalam suatu rangkaian kombinasi tertentu. Secara matematis kombinasi tersebut memenuhi kaidah aljabar Boole, antara lain :

b. Aljabar Boole :

Hukum Asosiatif : A.B.C = (A.B).C = (A.(B.C) = (A.C).B

Hukum Komutatif : A.B = B.A
A + B = B + A

Hukum perluasan : A = A.A.A. ....

A = A + A + A + ...

Hukum komplementasi : A . A’ = 0
A + A’ = 1

Hukum penjalinan dengan konstanta :
A . 0 = 0 A + 0 = A
A . 1 = A A + 1 = 1

Hukum pembalikan : A = (A’)’

Hukum Absorpsi : A.(A + B) = A
A + B.C = (A+B).(A+C)

Hukum distributif : A.B + A.C = A.(B+C)
(A+B).(A+C) = A + B.C

Hukum De Morgan : (A + B)’ = A’ B’
(AB)’ = A’ + B’

Dengan kaidah-kaidah tersebut, dapat disusun suatu fungsi gerbang logika dari satu atau beberapa gerbang lainnya. Misalnya karena A = A.A maka A’ = (A.A)’, berarti suatu gerbang NOT dapat dibangun dari sebuah gerbang NAND.

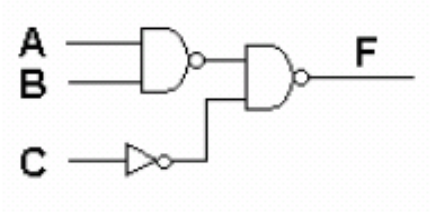
ALAT-ALAT PERCOBAAN

Dengan software DSCH2. Para mahasiswa harus mencoba fasilitas-fasilitas yang ada pada software tersebut. Karena petunjuk penggunaan tidak diberikan.

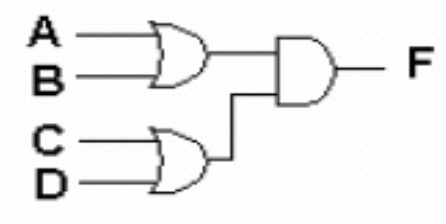
PRAKTIKUM PERTAMA

PERCOBAAN

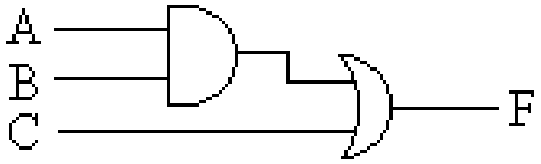
1. Buat gerbang-gerbang INVERS, AND, OR, NAND, NOR, XOR. Kemudian jalankan simulasi. Lihat hasilnya apakah sesuai dengan tabel kebenaran.
2. Dalam mendesain rangkaian logika seringkali diminta hanya untuk menggunakan gerbang-gerbang NAND atau NOR saja. Untuk itu buatlah rangkaian ekuivalen dari gerbang NAND dan NOR (yaitu NAND ekuivalen dengan INVERS-OR; NOR ekuivalen dengan INVERS-AND). Kemudian jalankan simulasinya dan cek apakah memang benar ekuivalen dari hasil simulasi tsb.
3. Ubah rangkaian di bawah ini menjadi rangkaian yang hanya terdiri dari gerbang NAND saja. Kemudian jalankan simulasi dan buat tabel kebenarannya.



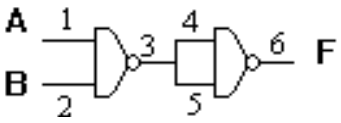
4. Ubahlah rangkaian di bawah ini menjadi rangkaian yang hanya terdiri dari gerbang NOR saja.

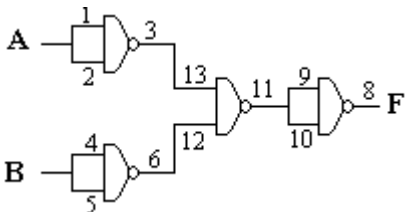


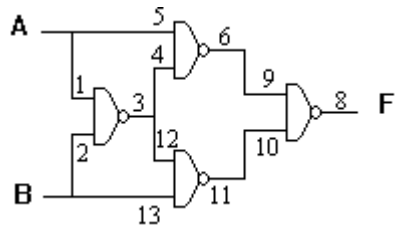
5. Ubahlah rangkaian berikut ini menggunakan tak lebih dari 4 buah NAND, dan susun tabel kebenarannya



6. Ubahlah rangkaian pada soal no.3 menggunakan tak lebih dari empat buah NOR, dan susun tabel kebenarannya.
7. Susun semua rangkaian di bawah ini dan kemudian jalankan simulasinya dan buat tabel kebenarannya.







PRAKTIKUM KEDUA

8. Membuat Rangkaian Kombinasional untuk implementasi tabel kebenaran yang diberikan..

   Tujuan

   * Mampu mengubah suatu fungsi aljabar menjadi bentuk yang paling sederhana dan menerapkan kedalam tabel kebenaran kemudian mengimplementasi-kannya ke rangkaian kombinasional.
   * Mampu merancang rangkaian kombinasional dari analisa tabel kebenaran.
   * Mampu mendesain rangkaian kombinasional dengan menggunakan IC seminimal mungkin.

   Ada kalanya suatu kasus logika disajikan dalam bentuk suatu fungsi logika atau suatu diagram gerbang-gerbang logika yang belum tersaji secara efisien. Oleh sebab itu penting bagi perancang rangkaian logika untuk mengerti bagaimana merancang suatu rangkaian logika dari setiap masalah yang dihadapi.

   Apabila demikian maka harus diteliti dahulu apakah fungsi logika yang disajikan tersebut sudah dalam bentuk yang paling sederhana/efisien ataukah masih dalam bentuk yang apa adanya(belum paling sederhana).

   Jika kita mempunyai semua gerbang bisa memenuhi semua gerbang logika yang ada pada fungsi tersebut, segeralah merancangnya. Tetapi jika kita ingin mengubah menjadi satu macam type gerbang saja seperti NAND atau NOR kita harus mengubah fungsi tersebut menjadi bentuk seperti berikut:

   Untuk membuat rangkaian hanya dari gerbang NAND:

   Fungsi: $F = B(A + \overline{C}) + D$

   ubahlah fungsi tersebut menjadi bentuk SOP (Sum of Product), sehingga menjadi:

   $$F = AB + B\overline{C} + D$$

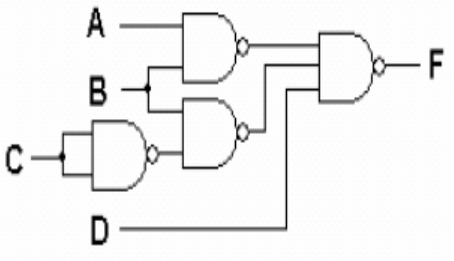
   Komplemenkan dua kali, fungsi menjadi :

$$F = \overline{\overline{AB + B\overline{C} + D}}$$

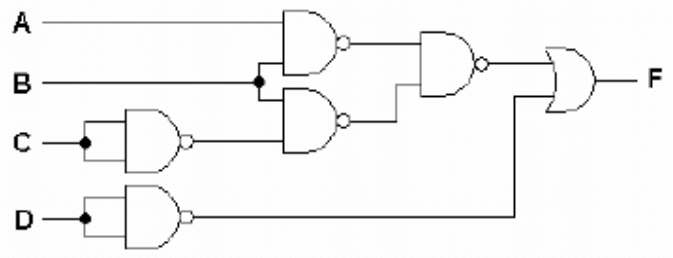
Operasikan komplemen tersebut sehingga menjadi :

$$F = \overline{\overline{AB}}.\overline{\overline{B\overline{C}}}.\overline{\overline{D}}$$

Rangkaian kombinasionalnya :



Jika hanya dibuat dengan gerbang NAND dengan 2 input saja dan gerbang OR



Untuk membuat rangkaian hanya dari gerbang NOR:

Fungsi: $F = (\overline{A} + B)(\overline{B} + \overline{C})(A + C)$

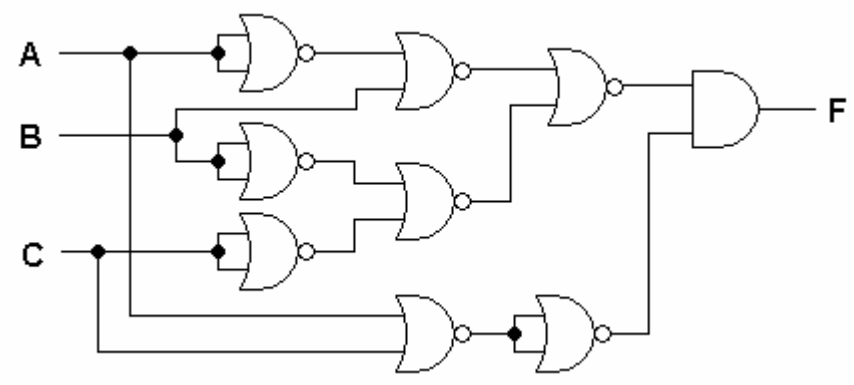
karena sudah dalam bentuk POS, maka langsung double komplemen fungsi tsb:

$$F = \overline{\overline{(\overline{A} + B)(\overline{B} + \overline{C})(A + C)}}$$

Operasikan komplemen yang bawah dari double komplemen sehingga menjadi :

$$F = \overline{\overline{(\overline{A} + B)} + \overline{(\overline{B} + \overline{C})} + \overline{(A + C)}}$$

Rangkaian kombinasionalnya jika hanya memakai NOR dengan 2 input :



Jika fungsi belum dalam bentuk yang paling sederhana, maka sederhanakan dahulu fungsi tersebut dengan metode penyederhanaan fungsi aljabar boolean atau dengan menggunakan K-Map (Karnaugh-Map). Sesudah itu rancanglah rangkaian kombinasionalnya. Sama seperti proses pembuatan yang sudah dijelaskan di atas.

Jika suatu kasus diberikan dalam bentuk tabel kebenaran, maka dapat dilakukan beberapa hal diantaranya:
Membentuk fungsi logika secara efisien (dengan cara penyederhanaan dengan KMap)
Membuatnya ekspresi fungsi logika ke arah SOP (Sum of Product) atau POS (Product of Sum).

Salah satu metode penyederhanaan fungsi logika untuk maksimal 4 variabel dapat dilakukan dengan metode K-Map (Karnaugh Map). Sebab jika lebih dari 4 variabel kita menggunakan metode Quine Mc Cluskey.
Contoh penyederhanaan fungsi logika dengan menggunakan K-Map:
Sederhanakan fungsi logika dengan 3 variabel berikut ini :

$$F = A\overline{B} + \overline{A}BC + B\overline{C} + \overline{B}$$

Karena bentuk ekspresi fungsi diatas adalah SOP maka pada matrik K-Map kita letakkan angka 1. Sehingga K -Map tersebut akan tampak seperti:

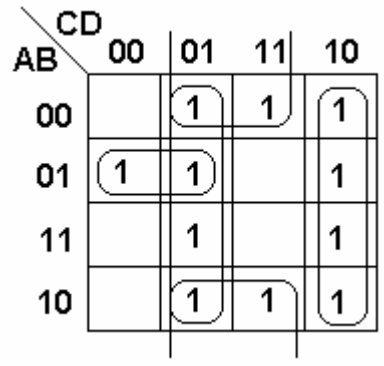
A \ BC	00	01	11	10
0			1	1
1	1	1	1	1

Dari K-map diperoleh penyederhanaan fungsi menjadi : $F = A + B$

Contoh: Sederhanakan fungsi Boole dengan 4 variabel :

$$F = \overline{A}B\overline{C} + \overline{B}D + \overline{A}B\overline{C}D + A\overline{C}D + C\overline{D}$$

Fungsi di atas dipetakan dengan Karnaugh-map diperoleh:



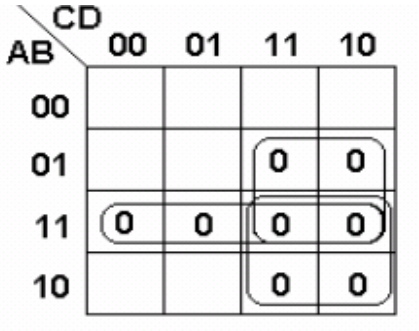
Sehingga penyederhanaan fungsi menjadi : $F = \overline{C}D + C\overline{D} + \overline{B}D + \overline{A}B\overline{C}$

Contoh: Sederhanakan fungsi Boole dengan 4 variabel

$$F = (\overline{A} + \overline{B} + C).(\overline{A} + \overline{C}).(\overline{B} + \overline{C})$$

Bentuk ekspresi fungsi di atas adalah POS, oleh sebab itu tempat 0 pada K-Map.

Sehingga K -Mapnya akan adalah sebagai berikut:



Hasil penyederhanaan K-Map : $F = (\overline{A} + \overline{B}).(\overline{B} + \overline{C}).(\overline{A} + \overline{C})$

Penyelesaian logika dari tabel kebenaran dengan menggunakan metode SOP dan POS dan implementasi pada rancangan rangkaian logikanya.

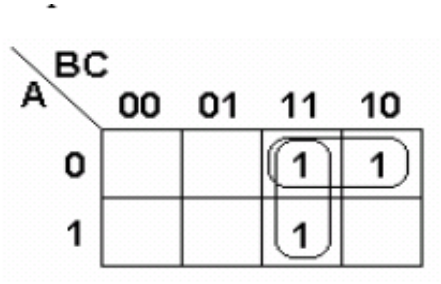
Jika diberikan suatu tabel kebenaran dari suatu kasus maka kita bisa menggunakan metode SOP atau POS untuk merancang suatu rangkaian kombinasionalnya. Seperti yang telah dijelaskan diatas. Untuk menentukan suatu rancangan biasanya kita menghendaki suatu rancangan yang paling efisien. Dengan adanya tabel kebenaran kita dapat menentukan mana diantara metode yang paling efisien untuk diimplementasikan. Untuk menentukan metode mana yang paling efisien, kita lihat bagian output pada tabel kebenaran tersebut. Jika jumlah output yang mempunyai nilai 1 lebih sedikit dari jumlah output yang mempunyai nilai 0, maka kita bisa menentukan bahwa metode SOP yang lebih efisien. Jika jumlah output yang mempunyai nilai 0 lebih sedikit dari jumlah output yang mempunyai nilai 1, maka kita bisa menentukan metode POS yang lebih efisien.

Contoh:

Buatlah rangkaian kombinasional untuk mengimplentasikan tabel kebenaran dibawah ini:

A	B	C	OUTPUT
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Karena output dengan nilai 1 lebih sedikit maka kita gunakan metode SOP. Dan untuk teknik penyederhanaannya kita langsung gunakan K-Map (karena masih 3 variabel). Sehingga K-Map akan berbentuk:

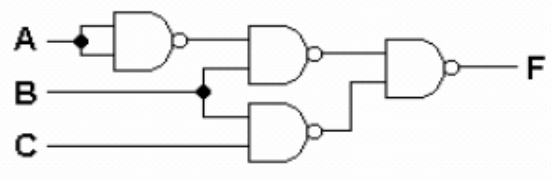


Ekspresi fungsi logikanya dari hasil K-Map tersebut adalah:

$$F = \overline{A}B + BC$$

Karena bentuk fungsi logikanya adalah SOP dapat dirancang rangkaian kombinasionalnya dari gerbang NAND saja, yaitu dengan cara memberi double bar pada fungsi tersebut kemudian operasikan bar yang terbawah. Fungsi akan menjadi:

$F = \overline{\overline{AB} + \overline{BC}} = \overline{\overline{AB}.\overline{BC}}$ sehingga rangkaian kombinasionalnya :

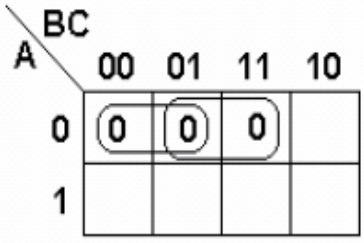


Coba rangkaian tersebut dan jalankan simulasinya cek apakah hasil tabel kebenarannya sama dengan aslinya.

Buatlah rangkaian kombinasional untuk mengimplentasikan tabel kebenaran berikut ini:

A	B	C	OUTPUT
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Karena output dengan nilai 0 lebih banyak maka kita gunakan metode POS.
Sehingga K-Map akan berbentuk:

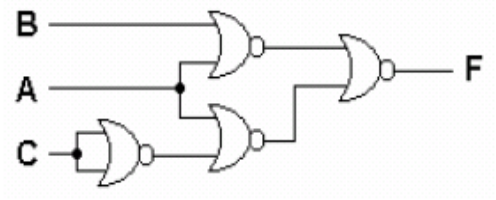


Ekspresi dari hasil K-Map adalah: $F = (A + B).(A + \overline{C})$

Dari fungsi logika tersebut kita dapat merancang rangkaian kombinasionalnya dari gerbang NOR saja dengan cara memberi double komplemen kemudian komplemen terbawah dioperasikan sehingga:

$$F = \overline{\overline{(A+B).(A+\overline{C})}} = \overline{\overline{(A+B)} + \overline{(A+\overline{C})}}$$

Dan rangkaian kombinasionalnya:



Susun rangkaian tersebut dan kemudian jalankan simulasinya. Cek hasil table kebenarannya dengan aslinya.

Kadangkala suatu hasil dari tabel disajikan dalam bentuk fungsi. Dan telah dikenal tentang lambang (symbol) "Σ" yang melambangkan operasi SOP sehingga yang ditampilkan adalah output yang mempunyai nilai 1 dan symbol "Π" melambangkan operasi POS sehingga yang ditampilkan adalah ouput yang mempunyai nilai 0. (lihat di kuliah).

Contoh:

F(A, B, C) = Σ (0, 3, 5, 7)

Maksud dari fungsi diatas adalah fungsi tersebut mempunyai 3 variabel input dan output yang mempunyai nilai 1 adalah 0, 3, 5, dan 7 (tanda Σ melambangkan SOP).

Jika fungsi yang disajikan adalah:

F(A, B, C) = Π (0, 3, 5, 7)

Maksudnya adalah fungsi tersebut mempunyai 3 variabel input dan output yang mempunyai nilai 0 adalah 0, 3, 5, dan 7 (tanda Π melambangkan POS).

Latihan

a. Buatlah rangkaian kombinasional dengan hanya menggunakan gerbang NOR untuk fungsi: F(A, B, C, D) = Π (2, 3, 6, 8, 10, 13, 14)

b. Buatlah rangkaian kombinasional untuk mengimplementasikan table kebenaran berikut ini dengan menggunakan gerbang seminimal mungkin. Maksimal 2 macam type gerbang.

A	B	C	OUTPUT 1	OUTPUT 2
0	0	0	0	1
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	1	0
1	1	0	0	1
1	1	1	0	1

Kemudian jalankan simulasinya dan cek apakah sudah sesuai dengan tabel kebenarannya dengan soalnya.

c. Diberikan $F_1 = A\overline{C} + ACE + AC\overline{E} + \overline{A}C\overline{D} + \overline{A}\overline{D}\overline{E}$, sederhanakan dan buat rangkaiannya dengan gerbang NAND

d. Buat rangkaian dari : $F = \overline{w}y\overline{z} + v\overline{w}\overline{z} + v\overline{w}x + \overline{v}wz + \overline{v}\overline{w}yz$ dengan gerbang AND dan OR

PRAKTIKUM KETIGA
RANGKAIAN ARITHMATIKA

1. Tujuan

a. Mampu memahami dan merancang rangkaian arithmatika

b. Mampu merancang rangkaian penambahan (adder) dan pengurangan (substractor)

2. Materi

a. Half Adder

b. Full Adder

c. Full Subtractor

d. Bilangan tak bertanda

e. Bilangan bertanda

f. Pengurangan dengan operasi complement

3. Teori

Pendahuluan:

Dalam penjumlahan bilangan biner, bilangan yang ditambah dan bilangan penambah tidak perlu dikomplementasikan. Sedangkan dalam proses pengurangan, bilangan pengurang harus dikomplementasikan terlebih dahulu dan hasil penjumlahannya mungkin harus dikomplementasikan lagi.

Komplementasi bilangan biner bisa merupakan komplemen 1 atau komplemen 2.

Komplemen 1 suatu bilangan biner adalah komplemen (kebalikan) dari bit-bit bilangan biner tersebut, sedangkan komplemen 2 adalah komplemen (kebalikan) dari bit-bit biner tersebut dan ditambah dengan satu. Dengan kata lain, komplemen 2 adalah komplemen 1 ditambah dengan satu.

Bilangan	komplemen 1	komplemen 2
0110	1001	1010
1100	0011	0100

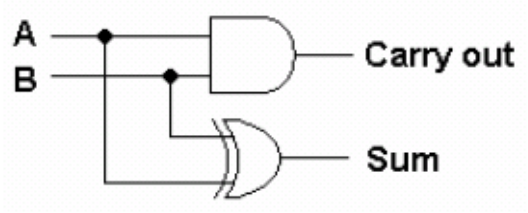
a. Half Adder

Adalah suatu operasi penjumlahan dua bit biner tanpa menyertakan carry-in nya.

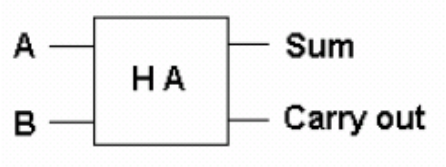
Half adder ini dapat dibuat tabel kebenarannya sebagai berikut:

A	B	SUM	Carry Out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Dari tabel kebenaran tersebut dapat dirancang rangkaian kombinasionalnya menjadi:



Sedangkan diagram menurut rangkaian kombinasional di atas, half adder tersebut menjadi:



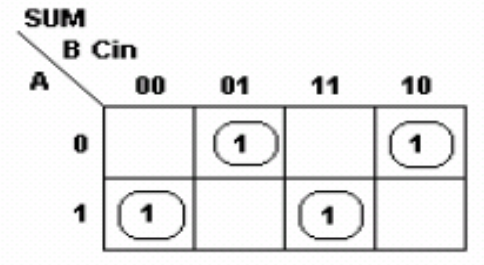
b. Full Adder

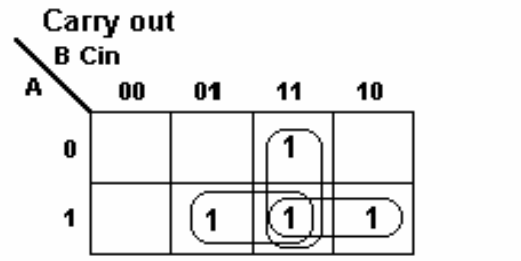
Adalah operasi penjumlahan dua bit biner dengan menyertakan carry-in nya.

Tabel kebenaran untuk full adder ini adalah sebagai berikut:

A	B	Carry in	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Dengan K-Map kita bisa merancang rangkaian full addernya sebagai berikut:



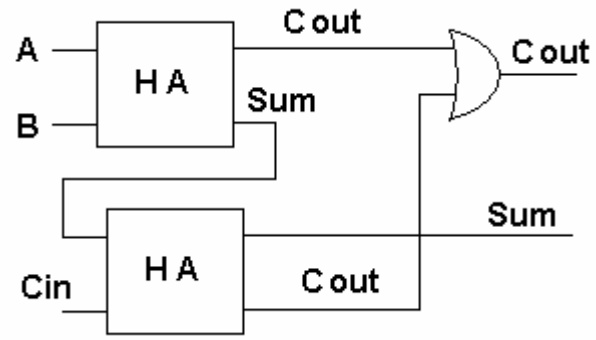


C = Carry in

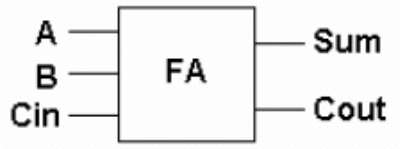
$Sum = A\bar{B}\bar{C} + \bar{A}\bar{B}C + ABC + \bar{A}B\bar{C}$

Carry out = AB + AC + BC

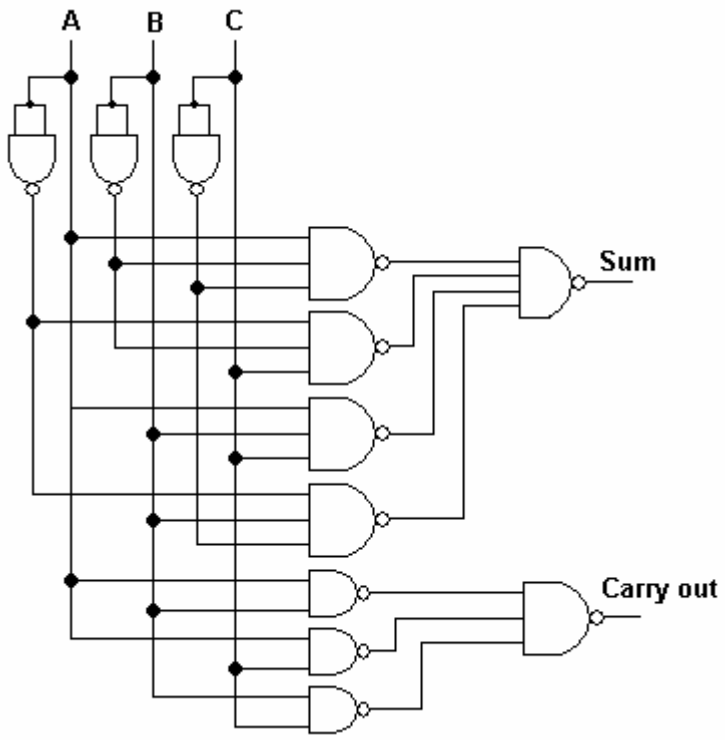
Rangkaian full adder dibuat dari 2 buah rangkaian half adder bentuknya adalah sebagai berikut:



Rangkaian tersebut dapat dibuat diagram logikanya menjadi:



Sedangkan rangkaian kombinasionalnya adalah:

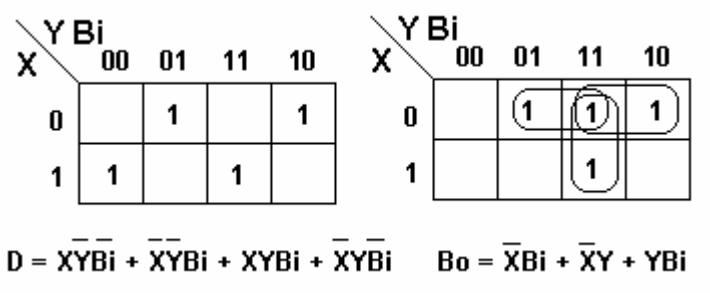


c. Full substractor

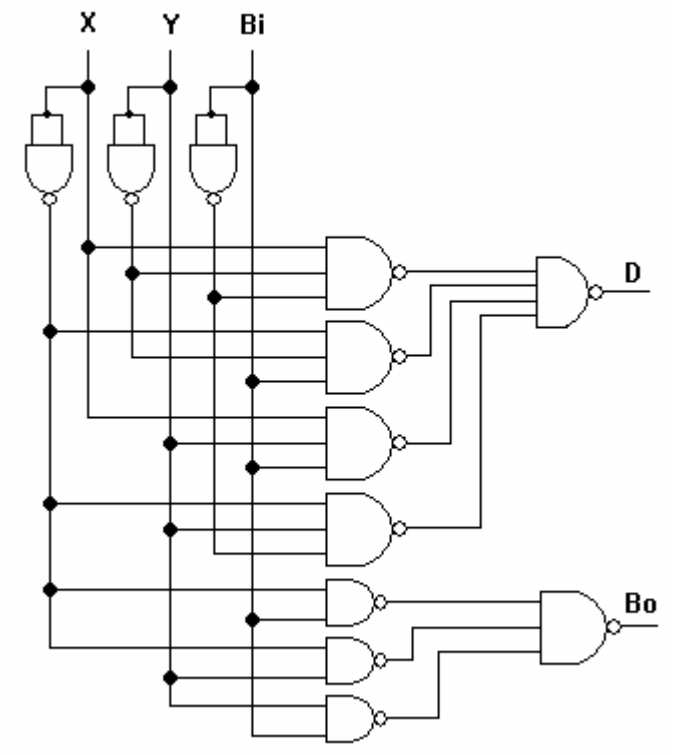
Rangkaian logika lainnya yang dapat dikelompokkan sebagai unit rangkaian arithmatika adalah full substractor. Full substractor ini merupakan operasi pengurangan dua bit biner yang mengikutsertakan borrow-in nya di dalam operasi pengurangannya. Tabel kebenaran dari full substractor ini adalah sebagai berikut:

X	Y	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

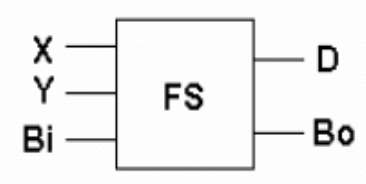
Dengan menggunakan K -Map untuk mencari fungsi logika yang paling sederhana.



Rangkaian kombinasional/logikanya adalah:



Rangkaian diatas dapat dibuat diagram logikanya sebagai berikut:



d. Bilangan tak bertanda

Pada beberapa aplikasi kadangkala tidak memperhatikan tanda negatif dan positip, karena hanya berkonsentrasi pada besaran nilai. Sebagai contoh pada angka biner 8 bit nilai besaran yang terkecil adalah 0000 0000 dan yang terbesar adalah 1111 1111. Maka dari itu total kisaran angka biner 8 bit adalah dari 0000 0000 (00H) s/d 1111 1111 (FFH). Besaran nilai biner 8 bit setara dengan decimal dari 0 s/d 255. seperti

yang anda lihat bahwa kita tidak menyertakan tanda + dan – pada bilangan decimal tersebut.

Jika kita menggunakan 16 bit, maka kisaran biner adalah:

0000 0000 0000 0000 (0000H) s/d 1111 1111 1111 1111 (FFFFH).

Yang setara dengan bilangan decimal dari 0 s/d 65.535

Contoh :

Penjumlahan bilangan decimal 150 dan 85 dengan menggunakan 8 bit.

Jawab:

150 → 1001 0110 → 96H

85 → 0101 0101 → 55H

1001 0110	96H
+ 0101 0101	+ 55H
1110 1011	EBH

1110 1011 → EBH → 235

Contoh:

Pengurangan bilangan decimal 150 dan 85 dengan menggunakan 8 bit.

Jawab:

1001 0110	96H
- 0101 0101	- 55H
0100 0001	41H

0100 0001 → 41H → 65

- Batasan-batasan

Microcomputer generasi pertama hanya dapat memproses 8 bit dalam satu waktu.

Oleh karena itu semua operasi-operasi arithmatika (baik penjumlahan atau pengurangan) haruslah menghasilkan besaran pada kisaran 0 s/d 255. Jika suatu besaran lebih besar dari 255, berarti kita harus menggunakan operasi arithmatika 16 bit. 8 bit pertama baru kemudian 8 bit berikutnya.

- *Overflow*

Pada penjumlahan 8 bit antara dua bilangan tak bertanda yang menghasilkan besaran lebih besar dari 255 akan menyebabkan overflow yaitu carry pada bit ke -9. Pada sebagian besar microprocessor mempunyai rangkaian logic yang disebut carry flag.

Rangkaian ini mendeteksi sebuah carry yang berada pada bit ke -9 dan memberi tanda bahwa hasil yang diperoleh pada operasi 8 bit tersebut invalid.

Contoh:

Tunjukkan dengan menggunakan operasi 8 bit untuk $(175)_{10}$ ditambah $(118)_{10}$.

Penyelesaian:

$$\begin{array}{r} 175 \\ +\ 118 \\ \hline 293 \end{array}$$

Karena jawabannya lebih dari 255, yang akan terjadi jika kita menggunakan operasi 8 bit adalah:

$$\begin{array}{rcl} \text{AFH} & & 1010\ 1111 \\ +\ \text{76H} & & +\ 0111\ 0110 \\ \hline 125\text{H} & \text{Overflow} \rightarrow & 1\ 0010\ 0101 \end{array}$$

Karena 8 bit, maka hanya hasil 8 bit kebawah saja yang digunakan. Sehingga:

0010 0101 → 25H → 37

seperti yang anda lihat, bahwa jawabannya salah.

e. Bilangan bertanda

Bilangan bertanda sangat penting artinya untuk operasi arithmatika yang lebih kompleks. Angka-angka decimal –1, -2, -3 dst yang menunjukkan besaran negative akan sama jika dilambangkan dengan bilangan biner -001, -010, dan -011 secara berurutan. Karena semua kode harus dilambangkan dengan 0 dan 1, maka tanda + dilambangkan dengan 0 dan – dilambangkan dengan 1. Sehingga -001, -010, dan -011 dikodekan sebagai 1001, 1010 dan 1011.

Pada angka -angka selanjutnya, MSB selalu menunjukkan tanda dan bit sisanya adalah besaran angka. Berikut ini adalah contoh konversi besaran bilangan bertanda:

+7 → 0000 0111

-16 → 1001 0000

+25 → 0000 0000 0001 1001

-128 → 1000 0000 1000 0000

- **Kisaran besaran bilangan bertanda**

  Seperti telah kita ketahui, bilangan tak bertanda 8 bit dapat mewakili 0 s/d 255. Jika kita menggunakan besaran bertanda nilai yang diwakili menjadi berkurang dari 255 menjadi 127, kar ena 1 bit dipergunakan sebagai wakil dari tanda besaran (+ atau -).

  Contoh:

  1000 0001 → (-1) 0000 0001 → (+1)

  1111 1111 → (-127) 0111 1111 → (+127)

- **1'S Complement**

  adalah bilangan biner yang dihasilkan dari menginvers-kan setiap bit-nya. 1'S complement dari biner 1000 adalah 0111

- **2'S Complement**

  adalah bilangan biner yang dihasilkan dari 1'S complement ditambah 1. jika dibuat rumus menjadi:

  2'S complement = 1'S complement + 1

  sebagai contoh, 2'S complement dari 1011 adalah:

  1011 → 0100 (1'S complement)

  0100 + 1 → 0101 (2'S complement)

- **Odometer biner**

  Adalah cara yang baik sekali untuk memahami gambaran tentang 2'S complement. Pada umumnya microcomputer menggunakan 2'S complement untuk menggambarkan bilangan negatif dan positif.

  Seperti telah kita ketahui bahwa tanda + diwakili dengan 0 dan – diwakili dengan 1 pada MSB bit-bit biner. Pada odometer, bilangan negatif merupakan 2'S complement dari bilangan positifnya, sehingga:

Besaran	Positif	Besaran	Negatif
1	0001	-1	1111
2	0010	-2	1110
3	0011	-3	1101
4	0100	-4	1100
5	0101	-5	1011
6	0110	-6	1010

7	0111	-7	1001
8	-	-8	1000

Kecuali untuk bilangan terakhir, bilangan negatif merupakan 2'S complement dari bilangan positifnya.

f. Pengurangan dengan complement

Pengurangan dua buah bilangan adalah sa ma dengan penjumlahan antara bilangan yang dikurangi dengan complement pengurangnya. Pada microcomputer selalu menggunakan biner untuk melakukan operasi arithmatika.

Contoh :

Hitunglah : 83 – 16 dengan menggunakan 8 bit.

Biner setiap bilangan:

83 → 0101 0011

16 → 0001 0000

16 akan dikirimkan 2'S complement-nya (karena minus) menjadi :

-16 → 1111 0000

sehingga :

83 0101 0011

\- 16 + 1111 0000

67 Carry don't care 1 0100 0011 → 0100 0011 → 43H → 67

Contoh:

Hitunglah : 14 – 108 dengan menggunaka n 8 bit.

Biner setiap bilangan:

\+ 14 → 0000 1110

\+ 108 → 0110 1100

108 akan dikirimkan 2'S complement-nya (karena minus) menjadi :

-108 → 1001 0100

sehingga:

14 0000 1110

\- 108 + 1001 0100

-94 Tanpa carry 1010 0010 → A2H → -94

Bahan

a. Buku praktikum

b. Perangkat lunak DSCH2

Prosedur Praktikum

a. Peserta telah membaca dan mempelajari materi praktikum.

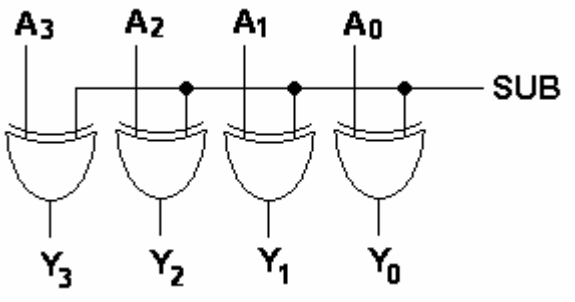
b. Peserta merancang rangkaian pada lembar kerja (buat dirumah) kemudian diimplementasikan dengan DSCH2.

Percobaan

Buatlah adder sekaligus substractor 4 bit untuk menjumlahkan dan mengurangkan dua bilangan biner 4 bit.

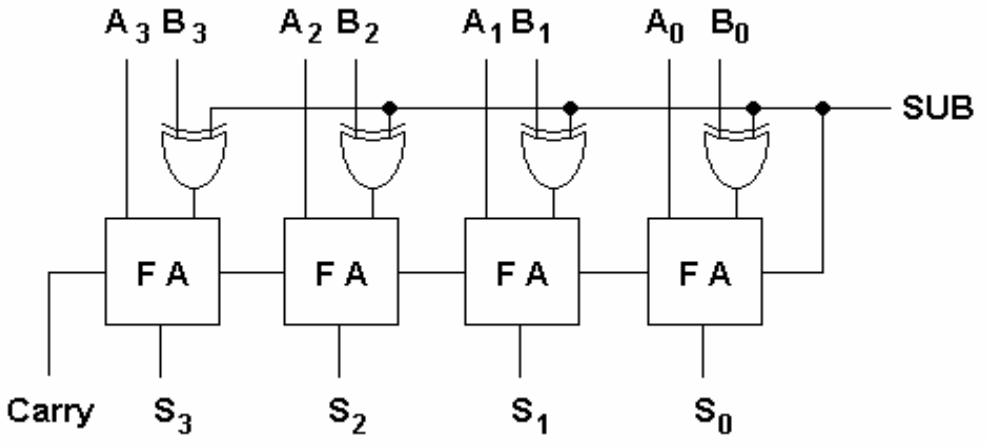
	A3 A2 A1 A0		A3 A2 A1 A0
+	B3 B2 B1 B0	-	B3 B2 B1 B0
	S3 S2 S1 S0		S3 S2 S1 S0

Rangkaian diagram logika 1'S complement untuk 4 bit:



jika SUB bernilai 0 maka $Y_3Y_2Y_1Y_0 = A_3A_2A_1A_0$ dan jika SUB bernilai 1 maka $Y_3Y_2Y_1Y_0 = A_3A_2A_1A_0$.

Rangkaian diagram Adder-Substractor (aplikasi paralel full adder) adalah:



Latihan.

a. Buatlah rangkaian gerbang logikanya untuk pengurangan 4 bit dengan menggunakan paralel full substractor.

b. Buatlah rangkaian yang outputnya adalah multiplication (perkalian) dengan bilangan 2 dari bilangan yang diinputkan. Jumlah input dan output adalah 4 bit.

PRAKTIKUM KEEMPAT
DATA PROCESSING CIRCUITS
(RANGKAIAN PEMROSES DATA)

1. Tujuan

Mahasiswa dapat mempelajari dan mengimplementasikan rangkaian logika untuk memproses data-data biner.

2. Materi

a. Multiplexer

b. Demultiplexer

c. Decoder

d. Encoder

3. Teori

a. MULTIPLEXER

Multiplexer adalah suatu rangkaian yang mempunyai banyak input dan hanya mempunyai satu output. Dengan menggunakan selector, kita dapat memilih salah satu inputnya untuk dijadikan output. Sehingga dapat dikatakan bahwa multiplexer ini mempunyai n input, m selector , dan 1 output. Biasanya jumlah inputnya adalah 2^m selectornya. Adapun macam dari multiplexer ini adalah sebagai berikut:

- Multiplexer 4x1 atau 4 to 1 multiplexe r
- Multiplexer 8x1 atau 8 to 1 multiplexer
- Multiplexer 16x1 atau 16 to 1 multiplexer dsb.

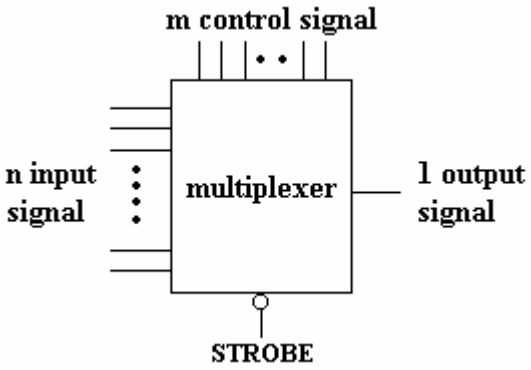
Gambar berikut adalah symbol dari multiplexer 4x1 yang juga disebut sebagai "data selector" karena bit output tergantung pada input data yang dipilih oleh selector. Input data biasanya diberi label D_0 s/d D_n. Pada multiplexer ini hanya ada satu input yang ditransmisikan sebagai output tergantung dari kombinasi nilai selectornya. Kita misalkan selectornya adalah S_1 dan S_0, maka jika nilai :

$$S_1\ S_0 = 00$$

Maka outputnya (kita beri label Y) adalah :

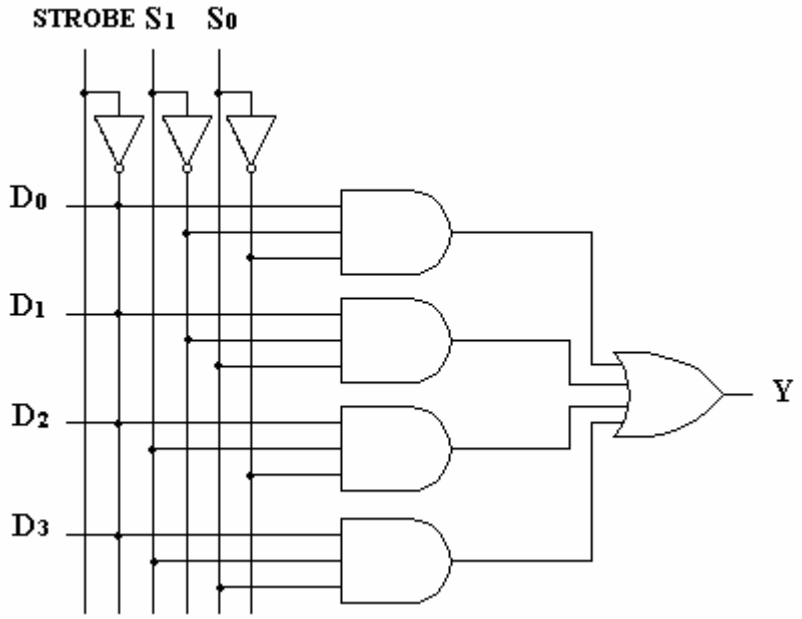
$$Y = D_0$$

Jika D_0 bernilai 0 maka Y akan bernilai 0, jika D_0 bernilai 1 maka Y akan bernilai 1.



Gambar symbol multiplexer

Adapun rangkaian multiplexer 4x1 dengan menggunakan strobe atau enable yaitu suatu jalur bit yang bertugas mengaktifkan atau menonaktifkan multiplexer, dapat kita lihat pada gambar berikut ini.



Gambar Rangkaian Multiplexer 4x1

True table multiplexer 4x1

Strobe	S1	S0	Output
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	X	X	0

- **LOGIKA MULTIPLEXER DAN IMPLEMENTASI FUNGSI BOOLEAN**

Suatu desain dari rangkaian logic biasanya dimulai dengan membuat table kebenaran. Seperti telah kita ketahui bahwa kita mengenal ada 2 macam metode yang diterapkan pada tabel kebenaran, yaitu metode *sum of product* (SOP) dan metode *product of sum* (POS). Nah pada bagian ini kita kenalkan dengan metode yang ketiga yaitu *multiplexer solution*.

Pada kenyataannya, kita dapat merancang suatu multiplexer 8x1 dari multiplexer 4x1 atau multiplexer 16x1 dari multiplexer 8x1 dan seterusnya. Jika kita anggap **selector** sebagai **n**, maka kita dapat membuat multiplexer 2nx1 dari multiplexer 2^{n-1} x1. Dengan kata lain kita memfungsikan multiplexer 2^{n-1}x1 sebagai multiplexer 2^nx1.

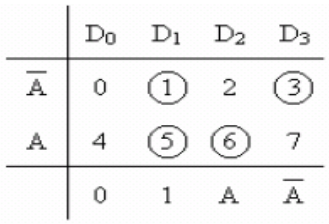
Jika kita menterjemahkan suatu kasus sebagai suatu fungsi F :

$$F(A, B, C) = \sum (1, 3, 5, 6)$$

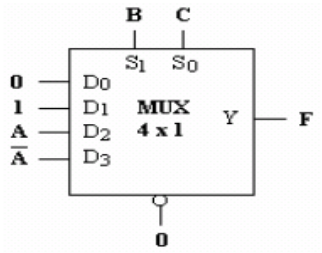
Dimana parameter fungsi tersebut A, B, C adalah merupakan selector dari multiplexer dan sisi sebelah kanan fungsi adalah output yang diinginkan dari multiplexer. Tanda ∑ beserta parameter berikutnya adalah merupakan bentuk SOP (Sum of Product), hal ini menandakan hanya minterm yang mempunyai nilai 1 saja yang diikutkan sebagai parameter. Tabel kebenaran dari fungsi di atas adalah sebagai berikut:

Tabel

Minterm	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0



(b) implementasi tabel

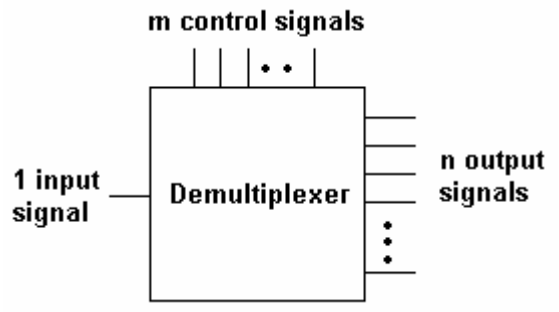


(c) Implementasi multiplexer.

b. DEMULTIPLEXER

Demultiplexer berarti satu ke banyak. Sebuah demultiplexer adalah suatu rangkaian logic yang mempunyai satu input dan mempunyai banyak output.

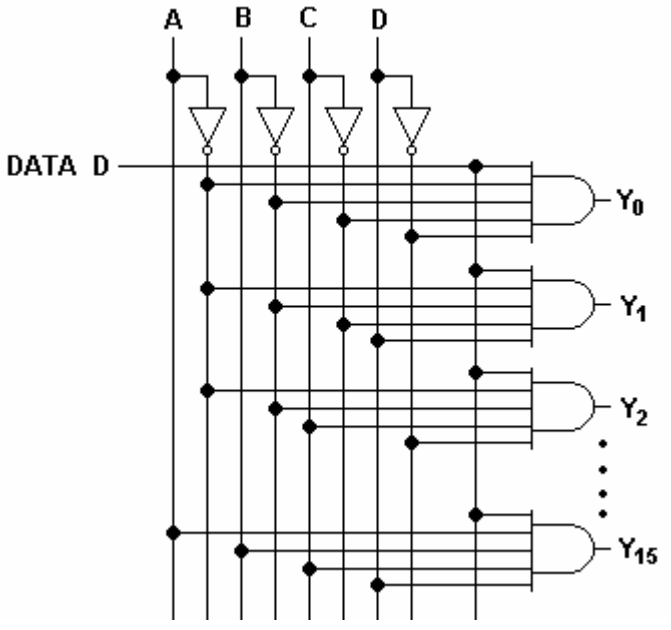
Dengan menggunakan control signal, kita dapat mengarahkan input signal ke salah satu outputnya. Gambar 4.3 mengilustrasikan ide dasar dari demultiplexer yang mempunyai 1 input signal, *m* control signal, dan *n* output signal.



Gambar Demultiplexer.

- **1 TO 16 DEMULTIPLEXER**

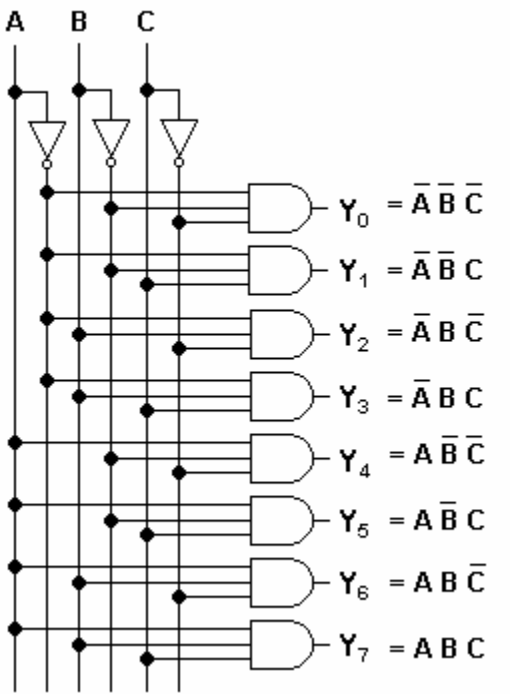
Gambar di bawah menunjukkan 1 to 16 demultiplexer. Input diberi label *D*. Bit data *D* ditransmisikan ke output tergantung pada nilai input control *ABCD*. Jika *ABCD* bernilai 0000, maka gerbang AND teratas enable/aktif dan gerbang AND lainnya akan disable/ tidak aktif. Oleh karena itu bit data *D* hanya ditransmisikan ke output Y_0, sehingga $Y_0=D$. Jika *D* bernilai 0, maka Y_0 bernilai 0. Jika *D* bernilai 1, maka Y_0 bernilai 1. Jika input control bernilai 1111, maka semua gerbang AND akan disable kecuali gerbang AND terbawah. Kemudian *D* hanya diransmisikan ke output Y_{15}, dan $Y_{15}=D$.



Gambar Rangkaian 1 to 16 demultiplexer.

c. DECODER

Jika kita perhatikan decoder ini sebenarnya mirip dengan demultiplexer, dengan satu pengecualian yaitu pada decoder ini tidak mempunyai data input. Input hanya digunakan sebagai data control.



Gambar Rangkaian 1 to 8 decoder.

Dalam hal ini adalah *ABCD*. Seperti yang ditunjukkan pada gambar 4.5, circuit logic ini disebut *1 of 8 decoder*, karena hanya 1 dari 8 jalur output yang bernilai 1.

Sebagai contoh, ketika *ABCD* = 0001, maka hanya output *Y1* yang akan bernilai 1. Begitu juga jika *ABCD* = 0100, maka hanya output *Y4* yang mempunyai output 1 dan seterusnya. Operasi pada decoder dapat dijelaskan lebih lanjut dari hubungan input-output, seperti pada tabel di bawah. Amatilah pada variabel output yang mana, satu sama lainnya saling eksklusif, karena ha nya ada satu output yang bernilai 1 pada satu waktu. Jalur output ditunjukkan dengan minterm yang ekivalen dengan angka biner.

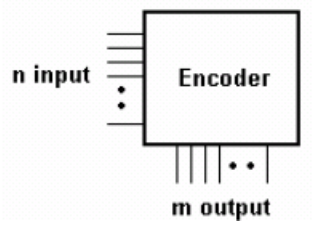
Tabel kebenaran 1 to 8 decoder.

Input			**Output**							
A	**B**	**C**	**D_0**	**D_1**	**D_2**	**D_3**	**D_4**	**D_5**	**D_6**	**D_7**
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0

A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

d. ENCODER

Sebuah encoder mengkonversikan input signal yang aktif menjadi output signal yang dikodekan. Pada gambar 4.6 mengilustrasikan suatu encoder. Dimana ada sejumlah n jalur input, dan hanya salah satunya yang aktif. Internal logic di dalam encoder mengkonversikan input yang aktif menjadi output kode-kode biner sebanyak m bit.



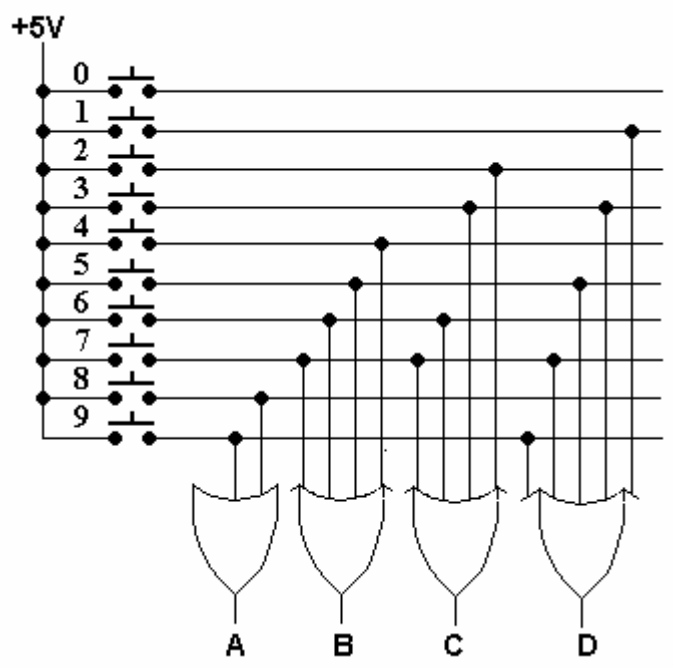
Gambar Encoder

- **DECIMAL TO BCD ENCODER**

Gambar 4.7 menunjukkan suatu type encoder yang sudah umum yaitu decimal to BCD encoder. Switch dengan penekan tombol mirip dengan tombol kalkulator dihubungkan dengan tegangan Vcc. Jika tombol 3 ditekan, maka gerbang-gerbang OR pada jalur C dan D akan mempunyai input bernilai 1. Oleh karena itu maka outputnya menjadi :

ABCD = 0011

Dan seterusnya.



Gambar Decimal to BCD encoder

4. Alat dan Bahan

a. Buku praktikum

b. Perangkat Lunak DSCH2

5. Prosedur Praktikum

a. Peserta telah membaca dan mempelajari materi praktikum.

b. Peserta merancang rangkaian pada lembar kerja (dibuat dirumah) kemudian diimplementasikan dengan DSCH2.

6. Percobaan

Buatlah rangkaian kombinasional untuk implementasi 1 to 8 demultiplexer.

Ubahlah rangkaian pada gambar 4.4 (1 to 16 demultiplexer) untuk menjadi 1 to 8 demultiplexer.

7. Latihan

a. Buatlah implementasi full adder dengan menggunakan 1 to 8 decoder dan 2 buah gerbang OR

b. Buatlah BCD to DECIMAL decoder