Dr. Ir. Agus Wibowo, M.Kom, M.Si, MM

YAYASAN PRIMA AGUS TEKNIK

DASAR KOMPUTER DIGITAL

Dasar Komputer Digital

Penulis :
Dr. Ir. Agus Wibowo, M.Kom., M.Si., MM.

ISBN : 9 786235 734637

Editor :
Dr. Joseph Teguh Santoso, S.Kom., M.Kom.

Penyunting :
Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

Desain Sampul dan Tata Letak :
Irdha Yunianto, S.Ds., M.Kom.

Penebit :
Yayasan Prima Agus Teknik Bekerja sama dengan
Universitas Sains & Teknologi Komputer (Universitas STEKOM)

Redaksi :
Jl. Majapahit no 605 Semarang
Telp. (024) 6723456
Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

Distributor Tunggal :
Universitas STEKOM
Jl. Majapahit no 605 Semarang
Telp. (024) 6723456
Fax. 024-6710144
Email : info@stekom.ac.id

KATA PENGANTAR

Puji Syukur penulis panjatkan atas selesainya buku yang berjudul ***"Dasar Komputer Digital".*** Buku ini dibagi menjadi tiga bagian: Desain Digital, Pengantar Arsitektur dan Memori Komputer, dan Arsitektur ARM dan Bahasa Perakitan. Materi dalam buku ini diperkaya dengan teori prinsip-prinsip sistem digital yang diperlukan dalam membangun sebuah sistem yang berbasis mikroprosesor.

Buku ini ditulis terutama untuk mata kuliah teknik Digital atau dasar komputer Digital sehingga materi buku ini telah disajikan sedemikian rupa agar para pembaca yang tidak memilik pengetahuan dasar komputer, tetap dapat memahami buku ini. Buku yang berjudul Dasar Komputer Digital ini terbagi dalam 10 bab, yang terperinci sebagai berikut: Bab 1 tentang Sistem Sinyal dan Angka: Sinyal Analog, Sinyal Digital, Bilangan Biner, Penambahan dan Pengurangan bilangan biner, representasi Titik Mengambang IEEE 754, ASCII, Unicode, Transmisi Serial, dan Transmisi Paralel. Bab 2 besisi Logika Boolean dan Gerbang Logika: Logika Boolean, Teorema Aljabar Boolean, Gerbang Logika, Sirkuit Integrasi (IC), Fungsi Boolean, Tabel Kebenaran suatu fungsi dan menggunakan Teorema Boolean untuk menyederhanakan Fungsi Boolean. Bab 3 Minterms, Maxterms, Karnaugh Map (K-Map) dan Gerbang Universal: Minterms, Maxterms, Karnaugh Map (K-Map) untuk menyederhanakan Fungsi Boolean, Kondisi Jangan Peduli dan Gerbang Universal.

Selanjutnya, Bab 4 akan membahas Logika Kombinasi: Analisis Logika Kombinasi, Desain Logika Kombinasi, Decoder, Encoder, Multiplexer, Half Adder, Full Adder, Binary Adder, Binary Subtractor, Merancang Arithmetic Logic Unit (ALU) dan BCD to Seven Segment Decoder. Bab 5 berisi Logika Sekuensial Sinkron: Logika Sekuensial seperti S-R Latch, D-Flip Flop, J-K Flip Flop, T-Flip Flop, Register, Shift Register, Analisis Logika Sekuensial, Diagram Status, Tabel Status, Tabel Eksitasi Flip Flop dan Perancangan Menangkal.

Bab 6 mengenai Pengantar Arsitektur Komputer: Komponen Komputer Mikro, Teknologi CPU, Arsitektur CPU, Eksekusi Instruksi, Pipelining, PCI, PCI Express, USB, dan HDMI. Bab 7 tentang Memori: Memori termasuk RAM, SRAM, DISK, SSD, Hirarki Memori, Memori Cache, Metode Pemetaan Memori Cache, Memori Virtual, Tabel Halaman, dan organisasi memori komputer. Bab 8 membahas Arsitektur dan Instruksi ARM Bagian I: Arsitektur Prosesor ARM, dan Kumpulan Instruksi ARM seperti Pemrosesan Data, Shift, Rotate, Instruksi Tanpa Syarat dan Instruksi Bersyarat, Operasi Stack, Cabang, Instruksi Multiply dan beberapa contoh konversi HLL ke bahasa Majelis. Bab 9 berisi Instruksi ARM Bagian II: Bab ini merupakan lanjutan dari Bab 8 yang mencakup Instruksi Load and Store, Instruksi Pseudo, Mode Pengalamatan ARM dan representasi data di memori. Bab 10 tentang Pemrograman Bahasa Perakitan ARM Menggunakan Alat Pengembangan Keil: Mencakup cara menggunakan perangkat lunak pengembangan Keil untuk menulis bahasa rakitan menggunakan Instruksi ARM, Mengkompilasi Bahasa Rakitan, dan Debugging. Pada setiap Bab dilengkapi dengan kegiatan praktek mandiri yang dapat dilakukan sendiri di rumah atau di lab. Teknik Digital serta ada solusi dari setiap masalah yang ada di bab itu. Akhir kata semoga buku ini berguna bagi para pembaca.

Semarang, Mei 2022
Penulis

Dr. Agus Wibowo, M.Kom, M.Si, MM

DAFTAR ISI

BAB 1
SISTEM SINYAL DAN ANGKA

Tujuan: Setelah menyelesaikan bab ini, Anda diharapkan dapat:

- Menjelaskan komponen dasar komputer.
- Bedakan antara sinyal analog dan digital.
- Pelajari karakteristik sinyal.
- Konversi bilangan desimal ke biner dan sebaliknya.
- Pelajari penambahan dan pengurangan bilangan biner.
- Mewakili angka mengambang dalam biner.
- Konversi dari biner ke heksadesimal dan sebaliknya.
- Bedakan antara transmisi serial dan paralel.

1.1 PENDAHULUAN

Nilai numerik telah menjadi bagian integral dari kehidupan kita sehari-hari. Nilai numerik dapat diwakili oleh analog atau digital; contohnya termasuk jam tangan analog, jam tangan digital, atau termometer. Berikut ini adalah kelebihan representasi digital nilai numerik dibandingkan dengan representasi analog:

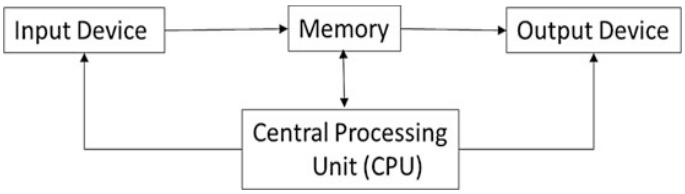
1. Representasi digital lebih akurat.
2. Informasi digital lebih mudah disimpan
3. Sistem digital lebih mudah dirancang.
4. Kebisingan memiliki efek yang lebih kecil.
5. Sistem digital dapat dengan mudah dibuat dalam sirkuit terpadu.

Sinyal digital adalah sinyal diskrit (langkah demi langkah), dan sinyal analog adalah sinyal kontinu. Sistem digital banyak digunakan dan aplikasinya dapat dilihat pada komputer, kalkulator, dan telepon seluler. Dalam sistem digital, informasi ditransfer antar komponen sistem digital dalam bentuk sinyal digital. Komputer terdiri dari dua komponen: perangkat keras dan perangkat lunak. Hardware mengacu pada komponen fisik komputer seperti keyboard, CPU, dan memori. Perangkat lunak mengacu pada program yang dijalankan oleh CPU termasuk sistem operasi dan program aplikasi. Komputer dapat datang dalam beberapa bentuk yang berbeda seperti desktop, laptop, tablet, server, dan iPhone. Terlepas dari bentuknya, semua komputer terdiri dari struktur dasar yang sama. Gambar 1.1 menunjukkan komponen dasar komputer.

Perangkat Input Perangkat input digunakan untuk memasukkan informasi ke dalam memori. Contoh perangkat input termasuk keyboard, mouse, layar panel sentuh, pena ringan, pembaca kode batang, dan pemindai. Perangkat input mengubah informasi menjadi bit, dan bit disimpan dalam memori.

Perangkat Keluaran Memori komputer mentransfer informasi ke perangkat keluaran dalam bentuk bit. Perangkat output mengubah bit menjadi karakter, gambar, dan suara yang dapat diinterpretasikan oleh manusia. Memori Memori digunakan untuk menyimpan informasi dan program. Memori datang dalam bentuk elektronik solid-state seperti RAM,

ROM, flash drive, atau hard disk. Central Processing Unit (CPU) CPU digunakan untuk melakukan perhitungan informasi.

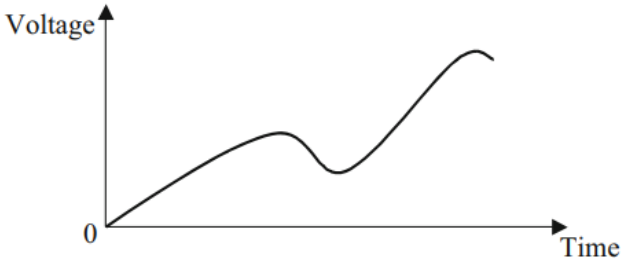


Gambar 1.1 Komponen Dasar Komputer

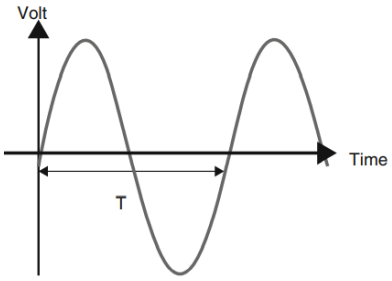
1.2 SINYAL ANALOG

Sinyal analog adalah sinyal yang amplitudonya merupakan fungsi waktu dan berubah secara bertahap seiring dengan perubahan waktu. Sinyal analog dapat diklasifikasikan sebagai sinyal nonperiodik dan periodik. Sinyal Nonperiodik Dalam sinyal nonperiodik, tidak ada pola berulang dalam sinyal seperti yang ditunjukkan pada Gambar 1.2. Sinyal Periodik Suatu sinyal yang mengulang suatu pola dalam suatu periode waktu yang terukur disebut sinyal periodik, dan penyelesaian suatu pola penuh disebut siklus. Sinyal periodik paling sederhana adalah gelombang sinus, yang ditunjukkan pada Gambar. 1.3. Dalam domain waktu, amplitudo gelombang sinus a(t) dapat direpresentasikan secara matematis sebagai a(t) A Sin(ωt + θ) di mana A adalah amplitudo maksimum, ω adalah frekuensi sudut, dan θ adalah sudut fase.

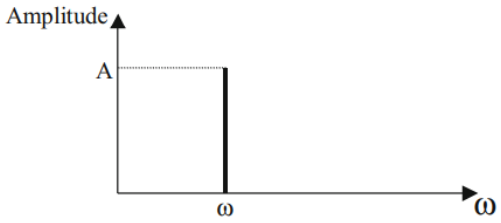
Sinyal periodik juga dapat direpresentasikan dalam domain frekuensi di mana sumbu horizontal adalah frekuensi dan sumbu vertikal adalah amplitudo sinyal. Gambar 1.4 menunjukkan representasi domain frekuensi dari sinyal gelombang sinus. Sinyal listrik, biasanya mewakili suara, suhu, atau suara musik, terbuat dari beberapa bentuk gelombang. Sinyal-sinyal ini memiliki satu frekuensi dasar dan beberapa frekuensi yang disebut harmonik.



Gambar 1.2 Representasi dari sinyal analog nonperiodik



Gambar 1.3 Representasi domain waktu dari gelombang sinus



Gambar 1.4 Representasi frekuensi gelombang sinus

Karakteristik Sinyal Analog

Karakteristik sinyal analog periodik adalah frekuensi, amplitudo, dan fase. Frekuensi Frekuensi (F) adalah jumlah siklus dalam 1 s, F = $\frac{1}{T}$, direpresentasikan dalam Hz (Hertz). Jika setiap siklus sinyal analog diulang setiap 1 detik, frekuensi sinyal adalah 1 Hz. Jika setiap siklus sinyal analog diulang 1000 kali setiap detik (sekali setiap milidetik), frekuensinya adalah

Tabel 1.1 Satuan khas frekuensi dan periode

Satuan frekuensi	Nilai numerik	Satuan periode	Nilai numerik
Hertz (Hz)	1 Hz	Kedua	1 s
Kilohertz (kHz)	10^3 Hz	Milidetik (md)	10^{-3} s
Megahertz (MHz)	10^6 Hz	Mikrodetik (µs)	10^{-6} s
Gigahertz (GHz)	10^9 Hz	Nanodetik (ns)	10^{-9} s
Terahertz (THz)	10^{12} Hz	Pikodetik (ps)	10^{-12} s

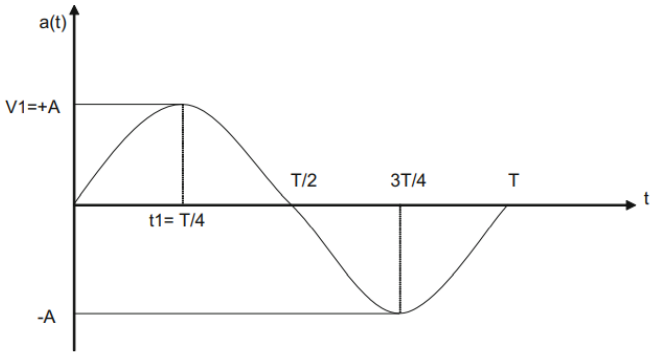
$$f = \frac{1}{T} = \frac{1}{10^{-3}} = 1000 \text{ Hz} = 1 \text{ kHz}$$

Tabel 1.1 menunjukkan nilai yang berbeda untuk frekuensi dan periode yang sesuai.

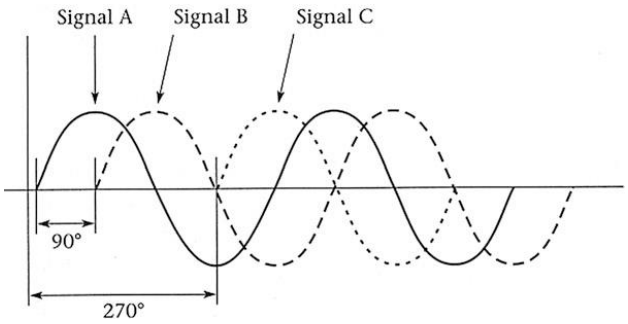
Amplitudo. Amplitudo sinyal analog adalah fungsi waktu seperti ditunjukkan pada Gambar 1.5 dan dapat dinyatakan dalam volt (satuan tegangan). Dengan kata lain, amplitudo adalah nilai tegangannya pada waktu tertentu. Pada waktu t1, amplitudo sinyal adalah V_1.

Fase Dua sinyal dengan frekuensi yang sama dapat berbeda fase. Ini berarti bahwa salah satu sinyal dimulai pada waktu yang berbeda dari yang lain. Perbedaan ini dapat direpresentasikan dalam derajat (0◦ hingga 360◦) atau dengan radian. Sudut fase 0◦ menunjukkan bahwa gelombang sinus dimulai pada waktu 0, dan sudut fase 90 menunjukkan bahwa sinyal dimulai pada 90◦ seperti yang ditunjukkan pada Gambar 1.6.

Contoh 1.1 Carilah persamaan untuk sinyal gelombang sinus dengan frekuensi 10 Hz, amplitudo maksimum 20 volt, dan sudut fasa 0◦:



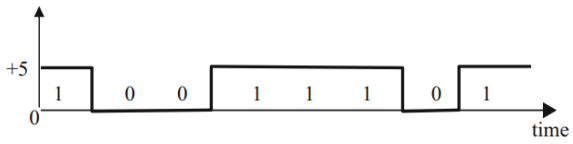
Gambar 1.5 Sinyal gelombang sinus selama satu siklus



Gambar 1.6 Tiga gelombang sinus dengan fase yang berbeda

1.3 SINYAL DIGITAL

Komputer modern berkomunikasi dengan menggunakan sinyal digital. Sinyal digital diwakili oleh dua tegangan: satu tegangan mewakili angka 0 dalam biner, dan tegangan lainnya mewakili angka 1 dalam biner. Contoh sinyal digital ditunjukkan pada Gambar 1.7, di mana 0 volt mewakili 0 dalam biner dan +5 volt mewakili 1, 0 atau 1 disebut bit dan 8 bit disebut byte.



Gambar 1.7 Sinyal digital

$$\omega = 2\pi f = 2 \times 3.1416 \times 10 = 62.83 \frac{rad}{\text{sec}}$$
$$a(t) = 20 \sin (62.83\, t)$$

1.4 SISTEM ANGKA

Angka dapat direpresentasikan dalam basis yang berbeda. Basis sepuluh disebut desimal. Dalam contoh, di bawah ini pertimbangkan 356 dalam desimal:

$$356 = 6 + 50 + 300 = 6*10^0 + 5*10^1 + 3*10^2$$

Secara umum, suatu bilangan dapat direpresentasikan dalam bentuk:

$$(a_5 a_4 a_3 a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3})_r,$$

Di mana r adalah basis bilangan dan ai harus lebih kecil dari r. $(10011)_2$ adalah angka yang valid tetapi $(211.01)_2$ tidak. Persamaan 1.1 dapat digunakan untuk mengonversi bilangan dalam basis tertentu ke desimal:

$$(\underbrace{a_5a_4a_3a_2a_1a_0}_{\text{Integer}}.\underbrace{a_{-1}a_{-2}a_{-3}}_{\text{Fraction}})_r = a_0 \times r^0 + a_1 \times r^1 + a_2 \times r^2 + a_3 \times r^3 + \ldots + a_{-1} \times r^{-1} + a_{-2} \times r^{-2} + \cdot a_{-2} \times r^{-3} \ldots \quad (1.1)$$

Contoh 1.2 Ubah (27.35)8 ke basis 10.

$$(27.35)_8 = 7*8^0 + 2*8^1 + 3*8^{-1} + 5*8^{-2} = 7 + 16 + .375 + .078125 = (23.45)_{10}$$

Contoh 1.3 Ubahlah 1101111 menjadi desimal.

$$(1101111)_2 = 1*2^0 + 1*2^1 + 1*2^2 + 1*2^3 + 0*2^4 + 1*2^5 + 1*2^6 = 1 + 2 + 4 + 8 + 32 + 64 = (111)_{10}$$

Mengonversi dari Biner ke Desimal

Persamaan 1.2 mewakili bentuk umum dari bilangan biner:

$$(a_5a_4a_3a_2a_1a_0.a_{-1}a_{-2}a_{-3})_2 \quad (1.2)$$

di mana ai adalah digit biner atau bit (baik 0 atau 1).

Persamaan 1.2 dapat dikonversi ke bilangan desimal dengan menggunakan Persamaan. 1.1:

$$(\underbrace{a_5a_4a_3a_2a_1a_0}_{\text{Integer}}.\underbrace{a_{-1}a_{-2}a_{-3}}_{\text{Fraction}})_2 = a_0 \times 2^0 + a_1 \times 2^1 + a_2 \times 2^2 + a_3 \times 2^3 + \ldots + a_{-1} \times 2^{-1} + a_{-2} \times 2^{-2} + \ldots \quad (1.3)$$

$$(a_5a_4a_3a_2a_1a_0.a_{-1}a_{-2}a_{-3})_2 = a_0 + 2\,a_1 + 4\,a_2 + 8\,a_3 + 16\,a_4 + 32\,a_5 + 64\,a_6 + \frac{1}{2}*a_{-1} + \frac{1}{4}*a_{-2} + \frac{1}{8}*a_{-3}$$

Contoh 1.4 Konversi $(110111.101)_2$ ke desimal.

$$(110111.101)_2 = 1*2^0 + 1*2^1 + 1*2^2 + 0*2^3 + 1*2^4 + 1*2^{*5} + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} = 55.625$$

Atau

2^5	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}
1	1	0	1	1	1	.	1	0	1

$$32 + 16 + 0 + 4 + 2 + 1 + 1/2 + 0 + 1/8$$

Jika nilai biner terdiri dari n bit satuan, maka nilai desimalnya adalah 2^n — 1.

Contoh 1.5

$$11 = 2^2 - 1 = 3$$
$$111 = 2^3 - 1 = 7$$
$$1111 = 2^4 - 1 = 15$$
$$11111 = 2^5 - 1 = 31$$
$$111111 = 2^6 - 1 = 63$$

Biner, atau basis dari 2 angka, diwakili oleh 0s dan 1s. Digit biner, 0 atau 1, disebut bit, 8 bit disebut byte, 16 bit disebut setengah kata, dan 4 byte disebut kata.

Mengonversi dari Bilangan Bulat Desimal ke Biner

Untuk mengonversi bilangan bulat dari desimal ke biner, bagilah bilangan desimal dengan basis baru (2 untuk biner), yang akan menghasilkan hasil bagi dan sisa (baik 0 atau 1). Sisa pertama akan menjadi bit paling signifikan dari bilangan biner. Bagi hasil bagi dengan basis baru secara terus menerus, sambil mengambil sisanya sebagai setiap bit berikutnya dalam bilangan biner, sampai hasil bagi menjadi 0.

Contoh 1.6 Konversikan 34 dalam desimal ke biner.

	Hasil bagi	**Sisa**
34/2 ¼	17	0 ¼ a_0
17/2¼	8	1 ¼ a_1
8/2	4	0 ¼ a_2
4/2	2	0 ¼ a3
2/2	1	0 ¼ a_4
1/2	0	1 ¼ a_5

Karena itu 34 ¼ $(100010)_2$

Jika bilangan biner dibuat dari semua, maka dengan menggunakan persamaan 2^n - 1 dapat diubah menjadi desimal.

Contoh

bilangan biner	**2^n — 1**	**Angka desimal**
11	2^2–1	3
111	2^3–1	7
1111	2^4–1	15
11111	2^5–1	32

Bilangan biner diwakili oleh a_5 a_4 a_3 a_2 a_1 a_0 di mana a0 adalah 20, a1 adalah 2^1, dan a_5 adalah 2^5. Tabel 1.2 menunjukkan 2^n.

Mengonversi Pecahan Desimal ke Biner

Representasi bilangan desimal $(0.XY)_{10}$ dapat diubah menjadi basis 2 dan diwakili oleh $(0.a_{-1}, a_{-2}, a_{-3}$, dst.$)_2$. Bilangan pecahan dikalikan 2, hasil bagian bilangan bulat adalah a_{-1} dan bagian pecahan dikalikan 2, kemudian pisahkan bagian bilangan bulat dari pecahan, bagian bilangan bulat mewakili a_{-2}; proses ini berlanjut sampai pecahan menjadi 0.

0.35*2	¼	0.7	¼	0	þ	0.7	a_{-1} ¼ 0
0.7*2	¼	1.4	¼	1	þ	0.4	a_{-2} ¼ 1
0.4*2	¼	0.8	¼	0	þ	0.8	a_{-3} ¼ 0
0.8*2	¼	1.6	¼	1	þ	0.6	a_{-4} ¼ 1
0.6*2	¼	1.2	¼	1	þ	0.2	a_{-5} ¼ 1

$(0,35)_{10}$ = ($)_2$

Terkadang, pecahan tidak mencapai 0 dan jumlah bit yang digunakan untuk pecahan tergantung pada akurasi yang ditentukan pengguna, oleh karena itu 0,35 = 0,010011 dalam biner.

Tabel 1.2 2^n dengan nilai n yang berbeda

2^n	Nilai desimal	2^n	Nilai desimal	2^n	Nilai desimal
2^0	1	2^8	256	2^{16}	65,536 ¼ 64 K
2^1	2	2^9	512	2^{17}	131,072 ¼ 128 K
2^2	4	2^{10}	1024 ¼ 1 K	2^{18}	262,144 ¼ 256 K
2^3	8	2^{11}	2048 ¼ 2 K	2^{19}	524,288 ¼ 512 K
2^4	16	2^{12}	4096 ¼ 4 K	2^{20}	1,048,576 ¼ 1 M
2^5	32	2^{13}	8192 ¼ 8 K	2^{21}	2 M
2^6	64	2^{14}	16,384 ¼ 16 K	2^{22}	4 M
2^7	128	2^{15}	32,768 ¼ 32 K	2^{23}	8 M

Tabel 1.3 Bilangan Desimal dengan Setara Biner dan Heksadesimal

Desimal	Biner (basis 2)	Heksadesimal (basis 16) atau HEX
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A

11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Sistem bilangan heksadesimal memiliki basis 16 dan oleh karena itu memiliki 16 simbol (0 hingga 9, dan A hingga F). Tabel 1.3 menunjukkan angka desimal, nilai binernya dari 0 hingga 15, dan ekuivalen heksadesimalnya.

Mengonversi dari Hex ke Biner

Tabel 1.3 juga dapat digunakan untuk mengonversi bilangan dari heksadesimal ke biner dan dari biner ke heksadesimal.

Contoh 1.7 Ubahlah bilangan biner 001010011010 menjadi heksadesimal. Setiap 4 bit dikelompokkan dari kanan ke kiri. Dengan menggunakan Tabel 1.3, setiap grup 4-bit dapat dikonversi ke ekuivalen heksadesimalnya.

0010	1001	1010
2	9	A

Contoh 1.8 Konversi (3D5)16 ke biner. Dengan menggunakan Tabel 1.3, hasil dalam biner adalah:

3	D	5
0011	1101	0101

Bilangan biner yang dihasilkan adalah 001111010101.

Contoh 1.9 Konversi 6DB dari heksadesimal ke biner. Dengan menggunakan Tabel 1.3, hasil dalam biner adalah:

6	D	B
0110	1101	1011

Bilangan biner yang dihasilkan adalah 011011011011.

Contoh 1.10 Konversi $(110111.101)_2$ ke desimal.

$$\begin{aligned}(110111.101)_2 &= 1*2^0 + 1*2^1 + 1*2^2 + 0*2^3 + 1*2^4 + 1*2^{*5} + 1*2^{-1} \\ &\quad + 0*2^{-2} + 1*2^{-3} \\ &= 55.625\end{aligned}$$

Penambahan Biner

$$1 + 0 = 1, 1 + 1 = 10,$$

Membawa bit

```
  11 1
  10101
+ 01101
-------
 100010
```

Dalam bilangan biner, bit pertama dari kiri angka disebut bit paling signifikan (MSb), dan bit pertama dari kanan angka disebut bit paling signifikan (LSb).

$$\underline{\text{MSb}} \rightarrow 1\,0\,0\,1\,0 \leftarrow \underline{\text{LSb}}$$

1.5 PELENGKAP DAN PELENGKAP DUA

Komplemen 1 adalah 0 dan komplemen 0 adalah 1. Komplemen bilangan biner dihitung dengan melengkapi setiap bit bilangan tersebut.

Contoh 1.11 Komplemen dari 101101 adalah 010010.

Komplemen Dua dari suatu bilangan = Komplemen suatu bilangan 1

Contoh 1.12 Komplemen dua dari 101011 adalah

010100(komplomen) + 1 = 010101

Contoh 1.13 Temukan komplemen dua dari 10.000:

01111(komplomen) + 1 = 10000

Pengurangan Bilangan Tanpa Tanda Menggunakan Pelengkap Dua

Prosedur berikut digunakan untuk mengurangkan B = $b_5\ b_4\ b_3\ b_2\ b_1\ b_0$ dari A = $a_5\ a_4\ a_3\ a_2\ a_1\ a_0$:

(a) Tambahkan komplemen dua B ke A.

(b) Periksa apakah hasilnya menghasilkan carry:

1. Jika hasilnya menghasilkan carry, buang carry dan hasilnya positif.
2. Jika hasilnya tidak menghasilkan carry, ambil komplemen dua dari hasilnya, dan hasilnya negatif.

Contoh 1.14 Kurangi B = 101010 dari A =110101.

01010 1 = Pelengkap B

Komplemen dua dari B = 010101 + 1 = 010110

Tambahkan komplemen dua dari B ke A.

```
  110101
+ 010110
----------
 10 01011
```

Carry, buang carry dan hasilnya adalah +001011.

Contoh 1.15 Kurangi B = 110101 dari A = 101010.
Komplemen dua B adalah 001010 + 1 = 001011.
Tambahkan komplemen dua B ke A.

$$\begin{array}{r} 001011 \\ +\ 101010 \\ \hline 110101 \end{array}$$

Seperti yang kita lihat, menambahkan dua angka 6-bit menghasilkan jawaban 6-bit. Tidak ada barang bawaan; kita ambil saja komplemen keduanya dari hasilnya.

Komplemen Dua dari 110101 = 001010 + 1 = −001011

Tidak Bertanda, Besaran Bertanda, Dan Bilangan Biner Komplemen Dua Bertanda

Bilangan biner dapat direpresentasikan dalam bentuk bilangan tak bertanda atau bilangan bertanda atau komplemen dua bertanda, + tanda dilambangkan dengan 0 dan tanda – dilambangkan dengan 1.

Unsigned Number Dalam unsigned number, semua bit angka digunakan untuk mewakili angka, tetapi dalam angka yang ditandatangani, bit paling signifikan dari angka mewakili tanda. Angka 1 pada posisi angka paling signifikan mewakili tanda negatif, dan 0 pada posisi angka paling signifikan mewakili tanda positif.

Nilai 1101 yang tidak ditandatangani adalah 13.

Bilangan Magnitudo Bertanda Dalam bilangan bertanda, bit paling signifikan mewakili tanda, di mana 1101 = −5 atau 0101 = +5

Dalam nomor tak bertanda, 1101 = 13.

Komplemen Dua Bertanda Komplemen dua bertanda berlaku untuk bilangan negatif. Jika tanda bilangan itu satu, maka bilangan tersebut dilambangkan dengan komplemen dua yang bertanda.

Contoh 1.16 Mewakili $(-5)_{10}$ dengan 4 bit dalam komplemen dua bertanda. $(-5)_{10}$ pada bilangan bertanda adalah 1101, maka komplemen dua dari 101 adalah 011, dan dengan menambahkan bit tanda menghasilkan 1011 yang mewakili -5 pada komplemen dua bertanda.

Contoh 1.17 Nyatakan $(-23)_{10}$ dengan komplemen dua bertanda 8-bit. $(23)_{10}$ $(1\ 0\ 1\ 1\ 1)_2$ di unsigned base-2

$(1\ 00\ 10111)_2$ – Diperluas hingga 8 bit yang ditandatangani (perhatikan MSb adalah 1) komplemen dua dari $(0010111)_2$ adalah $(1101001)_2$

$(11101001)_2$ adalah $(-23)_{10}$ dalam **komplemen dua bertanda.**

Penjumlahan Biner Menggunakan Pelengkap Dua Bertanda

Contoh berikut menunjukkan hasil penjumlahan dua bilangan bertanda:

(a) (+3) + (+4)

Mewakili kedua angka dalam biner, dan bit paling signifikan mewakili tanda, dan hasilnya positif.

0011 + 0100 = 0111= +7

(b) $(+3)_{10} + (—4)_{10}$

$(—4)_{10}$ dalam komplemen dua bertanda adalah 1100, maka

0011 1100 1111; hasil tidak menghasilkan carry maka ambil komplemen dua dari hasil yaitu -1

(—3) + (+5) = 1101 + 0101 = 1 0010 (buang carry dan hasilnya +2)

(c) $(—7)_{10} + (—5)_{10}$

Dengan merepresentasikan kedua bilangan dalam komplemen dua bertanda 4-bit,

$$(-7)_{10} = 1001$$
$$(-5)_{10} = 1011$$
$$1001 + 1011 = 10100$$
$$\uparrow$$

tanda tangan

Penambahan dua bilangan negatif menghasilkan positif, dan ini disebut overflow:

$$(+7) + (+6) = (0111) + (0110) = 1101$$

Penjumlahan dua bilangan positif menghasilkan negatif, dan disebut overflow:

Contoh 1.18

A. Penjumlahan berikut menggunakan komplemen dua bertanda 8 $(—38)_{10} + (44)_{10}$

$(—38)_{10}$ dalam komplemen dua s bertanda = 11011010

$(+44)_{10}$ + 00101100

100000110 buang carry dan hasilnya 00000110 = 6

B. Tambahkan —38 ke —44 menggunakan komplemen dua bertanda 8 bit

$$-38 = 11011010$$
$$-44 = 11010100$$
$$10101110 = -82$$

C. Tambahkan +100 ke +44

$$\begin{array}{rl} 100 & = 01100100 \\ 44 & = 00101100 \\ \hline & 10010000 \end{array}$$ tanda hasil negatif maka hasil menghasilkan overflow

Penambahan Overflow Kasus berikut menghasilkan overflow untuk menambahkan dua nomor yang ditandatangani jika:
(a) Kedua bilangan tersebut negatif, dan hasil penjumlahan menjadi positif:

$$(-A) + (-B) = +C$$

(b) Kedua bilangan itu positif, dan hasil penjumlahannya menjadi negatif:

$$(+A) + (+B) = -C$$

1.6 REPRESENTASI TITIK MENGAMBANG

Unit pemrosesan pusat (CPU) biasanya terdiri dari unit logika aritmatika (ALU), unit titik mengambang (FLU/FPU), register, unit kontrol, dan memori cache. Unit logika aritmatika melakukan operasi aritmatika bilangan bulat seperti penjumlahan, pengurangan, dan operasi logika seperti AND, OR, XOR, dll. Bilangan bulat adalah bilangan bulat tanpa komponen pecahan. 1, 2, dan 3 adalah bilangan bulat, sedangkan 0,1, 2.2, dan 3.0001 semuanya memiliki komponen pecahan yang disebut bilangan floating point.

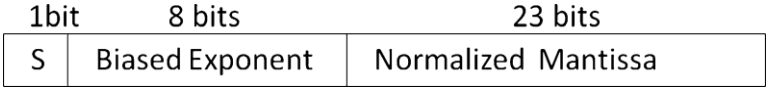
Unit floating point melakukan operasi floating point. Angka floating point memiliki tanda, mantissa, dan eksponen. Institute of Electrical and Electronics Engineers (IEEE) mengembangkan standar untuk mewakili angka floating point, yang disebut sebagai IEEE 754. Standar ini mendefinisikan format untuk angka floating point presisi tunggal (32-bit) dan ganda (64-bit). Titik apung desimal diwakili oleh M 10E, di mana M adalah mantissa bertanda dan E adalah eksponen.

Representasi Presisi Tunggal

Angka floating point dalam presisi tunggal diwakili oleh 32 bit seperti yang ditunjukkan pada Gambar. 1.8

Eksponen Bias Eksponen bias adalah eksponen + 127 $(01111111)_2$; oleh karena itu, eksponen diwakili oleh bilangan positif.

Mantissa yang dinormalisasi Mantissa diwakili oleh 1. M, di mana M disebut mantissa yang dinormalisasi; jika M = 00101, maka mantissa adalah 1,00101.



Gambar 1.8 IEEE 745 floating point presisi tunggal (S = mewakili tanda mantissa. S = 0 berarti mantissa positif, dan S = 1 berarti mantissa negatif)

Contoh 1.19 Temukan mantissa ternormalisasi dan eksponen bias dari $(111.0000111)_2$.

111.0000111 dapat ditulis dalam bentuk $1.110000111 * 2^{10}$

Dimana

M = 110000111

Eksponen bias = 10 + 01111111 = 10000001

Representasi dari 111.0000111 dalam presisi tunggal adalah

1bit	8 bits	23 bits
0	10000001	11000011100000000000000

Contoh 1.20 Ubahlah titik apung presisi tunggal berikut menjadi bilangan desimal.

101111101 11001000000000000000000

S = 1 berarti mantissa negatif.

Eksponen bias = 01111101.

Eksponen = 01111101–01111111 —00000010.

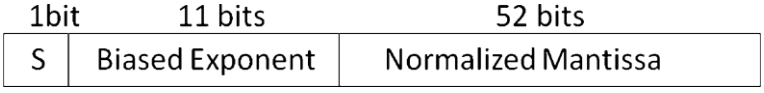
Mantissa yang dinormalisasi = 11001000000000000000000.

Mantissa = 1. 11001000000000000000000.

Bilangan desimal = 1.11001000000000000000000 $*2^{-10}$ = 0,01110011.

Presisi Ganda. Untuk meningkatkan akurasi angka floating point, IEEE 745 menawarkan presisi ganda yang diwakili oleh 64 bit seperti yang ditunjukkan pada Gambar 1.9.

Eksponen bias = eksponen + 1023



Gambar 1.9 IEEE 745 format floating point presisi ganda

Contoh 1.21 Mewakili 5,75 dalam presisi tunggal IEEE 745.

$-15.625 = (1111.101)_2$

$-1111.101 = -1.11101101 * 2^{11}$

S = 1

Mantissa yang dinormalisasi = 0.11101101.

Eksponen bias = 11 + 01111111 = 10000010.

IEEE745 presisi tunggal adalah

1 10000010 11101101000000000000000.

Desimal Berkode Biner (Bcd)

Dalam kehidupan sehari-hari, kita menggunakan bilangan desimal di mana angka terbesarnya adalah 9, yang diwakili oleh 1001 dalam biner. Tabel 1.4 menunjukkan angka desimal dan kode BCD yang sesuai.

Tabel 1.4 Desimal berkode biner (BCD)

Decimal	BCD
0	0000
1	0001

2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Contoh 1.21 Mengubah 345 ke BCD

Menggunakan tabel: 0011 0100 0101

Contoh 1.22 Konversi $(10100010010)_{BCD}$ ke desimal, pisahkan masing-masing 4 bit dari kanan ke kiri, dan substitusikan bilangan desimal yang sesuai dengan BCD hasilnya 512.

1.7 SKEMA PENGKODEAN

Kode ASCII

Setiap karakter dalam kode ASCII memiliki representasi menggunakan 8 bit, dimana bit yang paling signifikan digunakan untuk bit paritas. Tabel 1.5 menunjukkan kode ASCII dan ekuivalen heksadesimalnya. Karakter dari heksadesimal 00 hingga 1F dan 7F adalah karakter kontrol yang merupakan karakter yang tidak dapat dicetak, seperti NUL, SOH, STX, ETX, ESC, dan DLE (data link escape).

Contoh 1.23 Ubah kata "jaringan" menjadi biner dan tunjukkan hasilnya dalam heksadesimal. Dengan menggunakan Tabel 1.4, setiap karakter diwakili oleh 7 bit dan menghasilkan:

1001110	1100101	1110100	1110111	1101111	1110010	1101011
N	e	t	w	o	r	k

Atau dalam heksadesimal

4E	**65**	**74**	**77**	**6F**	**72**	**6B**

Kode Universal atau Unicode

Unicode adalah standar pengkodean karakter 16-bit baru untuk mewakili karakter dan angka dalam sebagian besar bahasa seperti Yunani, Arab, Cina, dan Jepang. Kode ASCII menggunakan 8 bit untuk mewakili setiap karakter dalam bahasa Latin, dan dapat mewakili 256 karakter. Kode ASCII tidak mendukung simbol matematika dan simbol ilmiah. Karena Unicode menggunakan 16 bit, ia dapat mewakili 65.536 karakter atau simbol. Karakter dalam Unicode diwakili oleh biner 16-bit, setara dengan 4 digit dalam heksadesimal. Misalnya, karakter B dalam Unicode adalah U0042H (U mewakili Unicode). Kode ASCII diwakili antara $(00)_{16}$ dan $(FF)_{16}$. Untuk mengonversi kode ASCII ke Unicode, dua angka nol ditambahkan ke sisi kiri kode ASCII; oleh karena itu, Unicode untuk mewakili karakter ASCII adalah antara $(0000)_{16}$ dan $(00FF)_{16}$. Tabel 1.6 menunjukkan beberapa Unicode untuk karakter Latin dan

Yunani. Unicode dibagi menjadi blok kode, dengan setiap blok ditugaskan ke bahasa tertentu. Tabel 1.7 menunjukkan setiap blok Unicode untuk beberapa bahasa yang berbeda (Gbr. 1.10).

Tabel 1.5 Kode Standar Amerika untuk Pertukaran Informasi (ASCII)

Biner	Hex	Char	Biner	Hex	Char	Biner	Hex	Char	Biner	Hex	Char
0000000	00	*NUL*	0100000	20	SP	1000000	40	@	1100000	60	,
0000001	01	*SOH*	0100001	21	!	1000001	41	A	1100001	61	a
0000010	02	*STX*	0100010	22	”	1000010	42	B	1100010	62	b
0000011	03	*ETX*	0100011	23	#	1000011	43	C	1100011	63	c
0000100	04	*EOT*	0100100	24	$	1000100	44	D	1100100	64	d
0000101	05	*ENQ*	0100101	25	%	1000101	45	E	1100101	65	e
0000110	06	*ACK*	0100110	26	&	1000110	46	F	1100110	66	f
0000111	07	*BEL*	0100111	27	’	1000111	47	G	1100111	67	g
0001000	08	*BS*	0101000	28	(	1001000	8	H	1101000	68	h
0001001	09	*HT*	0101001	29	)	1001001	49	I	1101001	69	i
0001010	0A	*LF*	0101010	2A	*	1001010	4A	J	1101010	6A	j
0001011	0B	*VT*	0101011	2B	þ	1001011	4B	K	1101011	6B	k
0001100	0C	*FF*	0101100	2C	,	1001100	4C	L	1101100	6C	l
0001101	0D	CR	0101101	2D	-	1001101	4D	M	1101101	6D	m
0001110	0E	*SO*	0101110	2E	.	1001110	4E	N	1101110	6E	n
0001111	0F	SI	0101111	2F	/	1001111	4F	O	1101111	6F	o
0010000	10	DLE	0110000	30	0	1010000	50	P	1110000	70	P
0010001	11	DC1	0110001	31	1	1010001	51	Q	1110001	71	q
0010010	12	DC2	0110010	32	2	1010010	52	R	1110010	72	r
0010011	13	DC3	0110011	33	3	1010011	53	S	1110011	73	s
0010100	14	DC4	0110100	34	4	1010100	54	T	1110100	74	t
0010101	15	NACK	0110101	35	5	1010101	55	U	1110101	75	u
0010110	16	SYN	0110110	36	6	1010110	56	V	1110110	76	v
0010111	17	ETB	0110111	37	7	1010111	57	W	1110111	77	w

0011000	18	CAN	0111000	38	8	1011000	58	X	1111000	78	x
0011001	19	EM	0111001	39	9	1011001	59	Y	1111001	79	y
0011010	1A	SUB	0111010	3A	:	1011010	5A	Z	1111010	7A	z
0011011	1B	ESC	0111011	3B	;	1011011	5B	[	1111011	7B	[
0011100	1C	FS	0111100	3C	<	1011100	5C	\	1111100	7C	\
0011101	1D	GS	0111101	3D	¼	1011101	5D	]	1111101	7D	}
0011110	1E	RS	0111110	3E	<	1011110	5E	^	1111110	7E	~
0011111	1F	US	0111111	3F	?	1011111	5F	-	1111111	7F	DEL

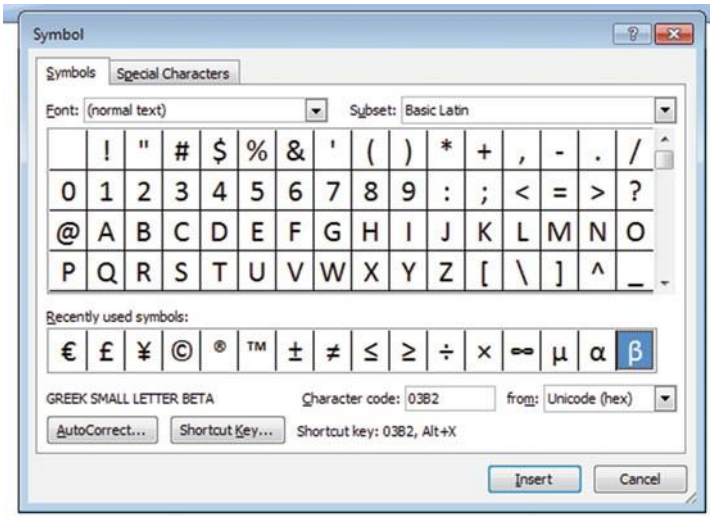
Tabel 1.6 Nilai Unicode untuk beberapa karakter Latin dan Yunani

Latin		Orang yunani	
Karakter	Kode (heksa)	Karakter	Kode (heksa)
A	U0041	ϕ	U03C6
B	U0042	α	U03B1
C	U0043	γ	U03B3
0	U0030	μ	U03 BC
8	U0038	β	U03B2

Tabel 1.7 Alokasi blok Unicode

Kode awal (heksadesimal)	Kode akhir (heksa)	Nama blok
U0000	U007F	Latin dasar
U0080	U00FF	suplemen latin
U0370	U03FF	Orang yunani
U0530	U058F	orang armenia

U0590	U05FF	Ibrani
U0600	U06FF	Arab
U01A0	U10FF	bahasa Georgia



Gambar 1.10 Contoh Unicode

Contoh Unicode: buka Microsoft Word dan klik insert maka akan muncul simbol Gambar 1.10. Klik karakter mana saja untuk menampilkan nilai Unicode karakter tersebut, misalnya Unicode untuk β adalah 03B2 dalam hex

1.8 BIT PARITAS

Bit paritas digunakan untuk pendeteksian kesalahan informasi, karena bit atau bit dapat diubah selama transmisi informasi dari sumber ke tujuan, bit paritas adalah bit tambahan yang ditambahkan ke informasi. Ini mewakili apakah jumlah satu atau nol adalah genap atau ganjil dalam transmisi asli dan dapat mengingatkan tujuan akan hilangnya informasi.

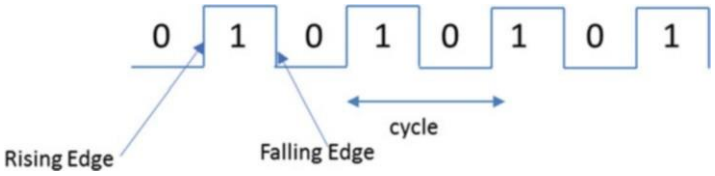
Bahkan Paritas. Bit ekstra (0 atau 1) dipilih sedemikian rupa sehingga jumlah yang menjadi genap.

Contoh 1.24 Pesan kami adalah $(00111)_2$. Dengan menambahkan satu ke sisi kiri pesan, kita buat (100111)2. Bit paritas genap kami telah membuat jumlah total menjadi genap (dari 3 menjadi 4). Pesan kami adalah $(10111)_2$. Dengan menambahkan nol ke sisi kiri pesan, kita buat $(010111)_2$. Bit paritas genap kami telah meninggalkan jumlah total yang genap (4 orang).

Paritas Ganjil. Bit ekstra (0 atau 1) dipilih sedemikian rupa sehingga jumlah yang menjadi ganjil. Pesan kami adalah $(10111)_2$. Dengan menambahkan satu ke sisi kiri pesan, kita buat $(110111)_2$. Bit paritas ganjil kami telah membuat jumlah total menjadi genap (dari 4 hingga 5).

Clock

0 dan 1 yang terus berulang disebut clock seperti yang ditunjukkan pada Gambar 1.11, ketika jam berubah dari 0 ke 1 disebut tepi naik dari jam dan ketika jam berubah dari 1 ke 0 disebut tepi jatuh dari jam. Setiap siklus jam terdiri dari 1 dan 0 atau 0 dan 1; diukur dengan waktu (sekon). Jika satu siklus diwakili oleh T dan satuan T adalah detik, maka F (frekuensi) 1/ T dimana satuan frekuensi adalah hertz (Hz) dan satuan T adalah detik.



Gambar 1.11 Sinyal clock

Contoh 1.25 Berapa frekuensi jam jika satu siklus jam sama dengan 0,5 ms?

$$F = 1/T = 1/0.5 \times 10^{-3} = 2000 \text{ Hz}$$

1000 Hz	Kilohertz (KHz)
10^6 Hz	Megahertz (MHz)
10^9 Hz	Gigahertz (GHz)

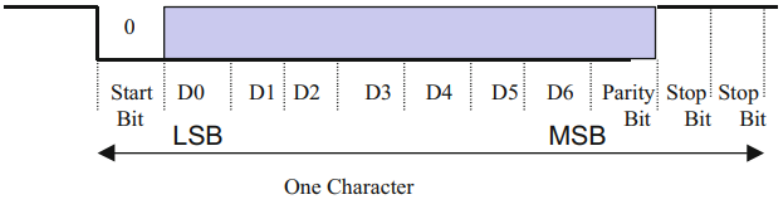
1.9 MODE DAN METODE TRANSMISI

Ketika data ditransfer dari satu komputer ke komputer lain dengan sinyal digital, komputer penerima harus membedakan ukuran setiap sinyal untuk menentukan kapan sinyal berakhir dan sinyal berikutnya dimulai. Misalnya, ketika komputer mengirim sinyal seperti yang ditunjukkan pada Gambar 1.12, komputer penerima harus mengenali berapa banyak satu dan nol dalam sinyal. Metode sinkronisasi antara perangkat sumber dan tujuan umumnya dikelompokkan menjadi dua kategori: asinkron dan sinkron.

Transmisi Asinkron

Transmisi asinkron terjadi karakter demi karakter dan digunakan untuk komunikasi serial, seperti oleh modem atau printer serial. Dalam transmisi asinkron, setiap karakter data memiliki bit awal yang mengidentifikasi awal karakter dan 1 atau 2 bit yang mengidentifikasi akhir karakter, seperti yang ditunjukkan pada Gambar 1.13. Karakter data adalah 7 bit. Mengikuti bit data mungkin merupakan bit paritas, yang digunakan oleh penerima untuk deteksi kesalahan. Setelah bit paritas dikirim, sinyal harus kembali ke tinggi setidaknya selama 1 bit untuk mengidentifikasi akhir karakter. Bit awal yang baru berfungsi sebagai indikator ke perangkat penerima bahwa karakter data akan datang dan memungkinkan pihak penerima untuk menyinkronkan jamnya. Karena jam penerima dan pemancar tidak disinkronkan secara terus-menerus, pemancar menggunakan bit awal untuk mengatur ulang jam penerima agar sesuai dengan jam pemancar. Juga, penerima sudah diprogram untuk jumlah bit dalam setiap karakter yang dikirim oleh pemancar.

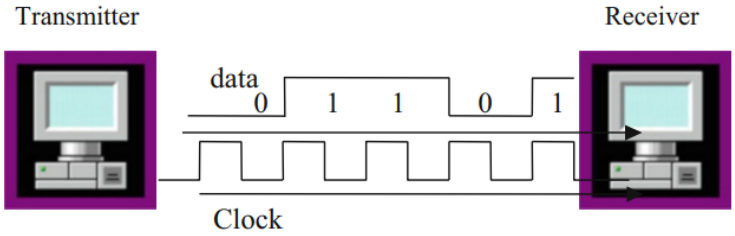
Gambar 1.12 Sinyal digital



Gambar 1.13 Transmisi asinkron

Transmisi Sinkron

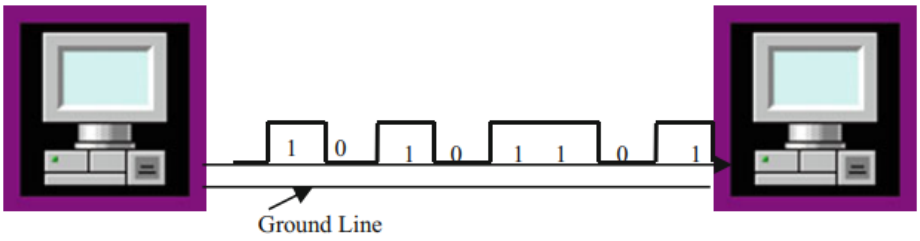
Beberapa aplikasi memerlukan transfer blok data yang besar, seperti file dari disk atau mentransfer informasi dari komputer ke printer. Transmisi sinkron adalah metode yang efisien untuk mentransfer blok data yang besar dengan menggunakan interval waktu untuk sinkronisasi. Salah satu metode sinkronisasi pemancar dan penerima adalah melalui penggunaan koneksi eksternal yang membawa pulsa clock. Pulsa clock mewakili kecepatan data sinyal, seperti yang ditunjukkan pada Gambar 1.14, dan digunakan untuk menentukan kecepatan transmisi data. Penerima Gambar 1.14 membaca data sebagai 01111, setiap lebar bit diwakili oleh satu jam. Gambar 1.14 menunjukkan koneksi tambahan yang diperlukan untuk membawa pulsa clock untuk transmisi sinkron. Dalam jaringan, satu media digunakan untuk transmisi informasi dan pulsa clock. Kedua sinyal dikodekan sedemikian rupa sehingga sinyal sinkronisasi disematkan ke dalam data. Ini dapat dilakukan dengan pengkodean Manchester atau pengkodean Diferensial Manchester.



Gambar 1.14 Transmisi sinkron

Metode Transmisi

Ada dua jenis metode transmisi yang digunakan untuk mengirim sinyal digital dari satu stasiun ke stasiun lain melalui saluran komunikasi: transmisi serial dan transmisi paralel.



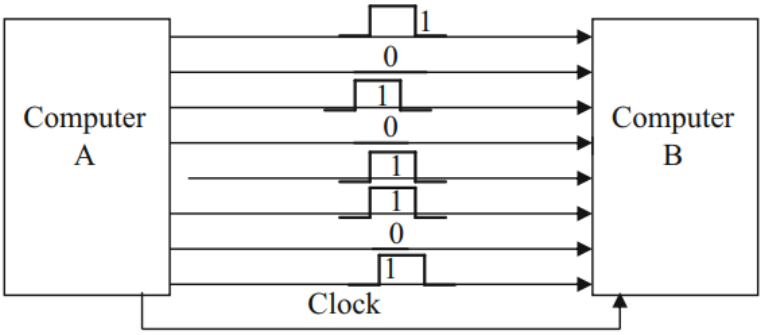
Gambar 1.15 Transmisi serial

Transmisi Serial

Dalam transmisi serial, informasi ditransmisikan 1 bit pada satu waktu melalui satu kabel seperti yang ditunjukkan pada Gambar 1.15.

Transmisi Paralel

Dalam transmisi paralel, beberapa bit dikirim secara bersamaan, 1 byte atau lebih pada suatu waktu, bukan bit demi bit seperti pada transmisi serial. Gambar 1.16 menunjukkan bagaimana komputer A mengirimkan 8 bit informasi ke komputer B secara bersamaan dengan menggunakan delapan kabel yang berbeda. Transmisi paralel lebih cepat dari transmisi serial, pada kecepatan clock yang sama.



Gambar 1.16 Transmisi paralel

1.10 RINGKASAN

Pertukaran informasi antara dua sistem kelistrikan dalam bentuk sinyal digital atau analog. Komputer modern bekerja dengan sinyal digital:

- Sinyal digital diwakili oleh dua tegangan.
- Biner adalah representasi dari bilangan berbasis 2.
- Frekuensi sinyal adalah kebalikan dari waktu siklusnya.
- Satu digit dalam biner disebut bit, 8 bit disebut 1 byte, dan 4 byte disebut satu kata.
- Informasi direpresentasikan di dalam komputer dengan biner atau basis 2.
- Angka negatif di dalam komputer dilambangkan dengan komplemen keduanya.
- Bit paling signifikan dalam bilangan bertanda dilambangkan dengan tanda bilangan tersebut.
- Positif dilambangkan dengan nol dan negatif dilambangkan dengan satu.
- Komplemen satu bilangan biner adalah komplemen dari setiap bit bilangan tersebut.
- Komplemen dua dari bilangan biner adalah komplemen dari bilangan ditambah satu.
- Desimal berkode biner (BCD) digunakan untuk mewakili angka desimal dari 0 hingga 9.
- Sistem bilangan heksadesimal digunakan dalam sistem digital dan komputer sebagai cara yang efisien untuk merepresentasikan besaran biner.
- Bit paritas digunakan untuk pendeteksian kesalahan kesalahan satu bit.
- Standar IEEE 757 digunakan untuk merepresentasikan bilangan floating point.
- Informasi diwakili oleh kode ASCII di dalam komputer; Kode ASCII terbuat dari 7 bit.
- Informasi antar komponen komputer dapat ditransmisikan dalam bentuk serial atau paralel.
- Dalam transmisi serial, informasi ditransmisikan 1 bit pada satu waktu
- Dalam transmisi paralel, informasi ditransmisikan dalam beberapa bit sekaligus.

- Bab selanjutnya akan membahas logika Boolean, teorema aljabar Boolean, dan gerbang logika. Gerbang logika merupakan komponen dasar dari rangkaian terintegrasi (IC)

Masalah dan Pertanyaan

1. Sebutkan tiga perangkat input komputer.
2. Sebutkan tiga perangkat keluaran komputer.
3. Menampilkan sinyal analog.
4. Tampilkan sinyal digital.
5. Berapa bitnya:
   (a) Byte
   (b) Setengah kata
   (c) Kata
6. Ubahlah bilangan desimal berikut menjadi biner:
   (a) 35
   (b) 85
   (c) 23,25
7. Ubahlah bilangan biner berikut ke desimal:
   (a) 1111101
   (b) 1010111.1011
   (c) 11111111
   (d) 10000000
8. Ubahlah bilangan biner berikut menjadi heksadesimal:
   (a) 1110011010
   (b) 1000100111
   (c) 101111.101
9. Temukan frekuensi sinyal digital dengan siklus clock berikut:
   (a) 1 detik
   (b) 0,1 detik
   (c) 0,02 s
   (d) 0,02 ms
10. Frekuensi sinyal digital berikut diberikan, temukan siklus jam sinyal digital:
    (a) 10 Hz
    (b) 200 Hz
    (c) 10.000 Hz
    (d) 4 MHz
11. Ubahlah setiap bilangan berikut menjadi basis 10:
    (a) $(34A)_{16}$
    (b) $(FAC)_{16}$
12. Ubahlah bilangan desimal berikut menjadi basis 16:
    (a) $(234)_{10}$
    (b) $(75)_{10}$

13. Ubahlah bilangan berikut menjadi biner:
    (a) $(3FDA)_{16}$
    (b) $(FDA.5F)_{16}$
14. Lakukan penambahan berikut:

$$\begin{array}{rr} 1101010 & 1100101 \\ 1011011 & +1010111 \end{array}$$

15. Tentukan komplemen dua bilangan berikut:
    (a) 11111111
    (b) 10110000
    (c) 10000000
    (d) 00000000
16. Kata "LOGIKA" diberikan.
    (a) Mewakili dalam ASCII.
    (b) Tambahkan bit paritas genap ke setiap karakter dan nyatakan setiap karakter dalam heksa.
17. Mewakili (465)10 dalam BCD.
18. Nyatakan (100101100111)BCD dalam desimal.
19. Mewakili (110010000100)BCD dalam desimal.
20. Ubahlah bilangan komplemen dua berikut menjadi desimal:
    (a) 1011
    (b) 11111001
    (c) 10011111
21. Kurangi bilangan tak bertanda berikut menggunakan komplemen dua:
    (a) 11110011–11000011
    (b) 10001101-11111000
22. Lakukan penjumlahan bilangan bertanda berikut; asumsikan setiap angka diwakili oleh 6 bit dan nyatakan jika hasil setiap penambahan menghasilkan overflow:
    (a) (12) + (+7)
    (b) (+25) + (+30)
    (c) (—5) + (+9)
    (d) (—6) + (—7)
    (e) (—36) + (—12)
23. Berapa nilai biner 16-bit terbesar yang dapat diwakili oleh:
    (a) Nomor tidak bertanda tangan
    (b) Besaran bertanda
    (c) Komplemen dua bertanda tangan
24. Nyatakan bilangan desimal berikut dalam presisi tunggal IEEE 745:
    (a) 34.375
    (b) —0,045
25. Ubah presisi tunggal IEEE 745 berikut ke bilangan desimal:
    (a) 1 10000100 01110000000000000000000
    (b) 0 01111100 11100000000000000000000

26. Sebutkan jenis-jenis mode transmisi.
27. Ubahlah setiap bilangan besaran bertanda berikut ke desimal:
    (a) 11000011
    (b) 10001111
28. Nyatakan masing-masing bilangan berikut dalam komplemen dua bertanda 8-bit:
    (a) —15
    (b) —24
    (c) —8
29. Lakukan penambahan berikut:

$$\begin{array}{r} 0F4A \\ +420B \\ \hline \end{array}$$

BAB 2
LOGIKA BOOLEAN DAN GERBANG LOGIKA

Tujuan: Setelah menyelesaikan bab ini, Anda diharapkan dapat:

- Memahami operasi dasar teorema Boolean.
- Menjelaskan pengoperasian gerbang logika yang berbeda seperti gerbang AND, OR, NOT, XOR, dan NAND.
- Tunjukkan tabel kebenaran dari gerbang logika yang berbeda.
- Membedakan berbagai jenis sirkuit terpadu (IC).
- Terapkan teorema Boolean untuk menyederhanakan fungsi Boolean.
- Gambarlah rangkaian logika untuk fungsi Boolean.
- Tunjukkan tabel kebenaran fungsi Boolean.
- Temukan fungsi keluaran dari rangkaian logika digital.
- Bedakan antara SSI, MSI, LSI, dan VLSI.

2.1 PENDAHULUAN

Gerbang logika terbuat dari transistor, dan merupakan komponen dasar dari rangkaian terintegrasi (IC). Gerbang logika digunakan untuk merancang sistem digital; ada tiga operasi logika dasar dan mereka disebut AND, OR, dan NOT. Karakteristik sistem digital dapat direpresentasikan dengan fungsi atau tabel kebenaran. Teorema Boolean digunakan untuk menyederhanakan fungsi Boolean agar gerbang logika yang digunakan lebih sedikit. Sirkuit terpadu diklasifikasikan berdasarkan jumlah gerbang yang dikandungnya, dan mereka disebut SSI, MSI, LSI, dan VLSI.

2.2 LOGIKA BOOLEAN DAN GERBANG LOGIKA

Logika AND Logika AND dilambangkan dengan titik "." tetapi sebagian besar waktu, titik tersebut ditinggalkan. X.Y atau XY diucapkan sebagai X DAN Y.

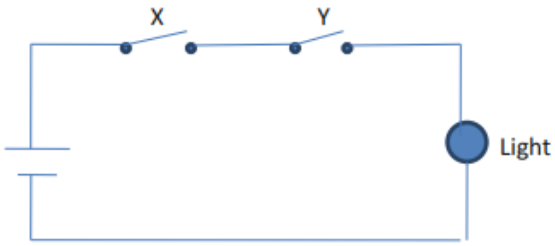
X AND Y = Z, Z = 1 jika dan hanya jika X = 1 dan Y = 1 jika tidak Z = 0.

Operasi logika AND dapat direpresentasikan oleh rangkaian listrik pada Gambar 2.1. Asumsikan X dan Y adalah sakelar dan Z adalah lampu; X 0, Y 0 berarti sakelar terbuka dan lampu mati berarti nol dan lampu menyala berarti satu. Kemudian kita bisa membuat tabel; Tabel 2.1 menunjukkan pengoperasian Gambar 2.1. Gambar 2.2 menunjukkan gerbang AND 2-input dan Tabel 2.2 menunjukkan tabel kebenaran untuk gerbang AND. Output dari gerbang AND adalah satu ketika kedua input adalah satu.

OR Logika. Operasi OR diwakili oleh tanda plus, "+" atau V, di mana "+" adalah simbol yang paling populer digunakan. X Y diucapkan X OR Y.

X + Y = Z, Z = 1 jika X = 1 OR Y = 1 atau keduanya X = 1 dan Y = 1.

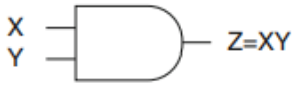
Operasi OR dapat diwakili oleh rangkaian listrik pada Gambar. 2.3. Pada Gambar 2.3, lampu mati ketika kedua sakelar mati, dan lampu menyala ketika setidaknya satu sakelar ditutup. Gambar 2.4 menunjukkan gerbang OR 2 masukan dan Tabel 2.3 menunjukkan tabel kebenaran untuk gerbang OR 2 masukan.



Gambar 2.1 Representasi dari operasi AND

Tabel 2.1 Operasi Gambar 2.1

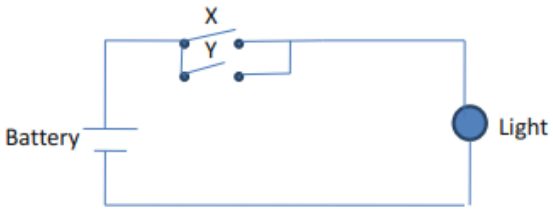
X	kamu	Lampu
Mati	Mati	Mati
Mati	Nyala	Mati
Nyala	Mati	Mati
Nyala	Nyala	Nyala



Gambar 2.2 Gerbang AND 2-masukan

Tabel 2.2 Tabel kebenaran gerbang AND

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1



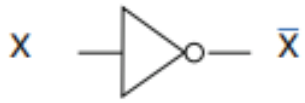
Gambar 2.3 Representasi rangkaian listrik dari operasi OR

Gambar 2.4 Gerbang OR 2-masukan

Tabel 2.3 Tabel kebenaran gerbang OR 2-masukan

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

Logika NOT Logika NOT melakukan komplemen, yang berarti mengubah 1 ke 0 dan 0 ke 1. Juga disebut inverter, NOT X diwakili oleh X0 atau X¯. Gambar 2.5 menunjukkan gerbang NOT dan Tabel 2.4 menunjukkan tabel kebenaran gerbang NOT (inverter).



Gambar 2.5 Gerbang NOT

Tabel 2.4 Tabel kebenaran gerbang NOT

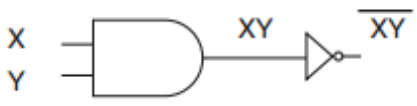
X	X′
0	1
1	0

Gerbang NAND. Gambar 2.6 menunjukkan gerbang NAND 2-input. Gerbang NAND dapat dibuat dari gerbang AND dan gerbang NOT seperti yang ditunjukkan pada Gambar 2.7. Tabel 2.5 menunjukkan tabel kebenaran gerbang NAND 2-masukan.

Gambar 2.6 Gerbang NAND 2-masukan

Tabel 2.5 Tabel kebenaran dari 2-input NAND

X	Y	XY
0	0	1
0	1	1
1	0	1
1	1	0



Gambar 2.7 Gerbang AND-NOT digunakan bersama-sama untuk bertindak sebagai NAND

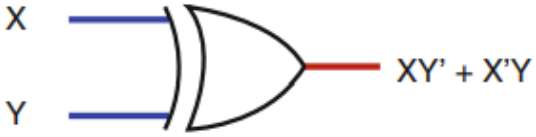
Gerbang NOR. Gambar 2.8 menunjukkan gerbang logika NOR. Gerbang NOR terbuat dari gerbang OR dan NOT. Tabel 2.6 menunjukkan tabel kebenaran gerbang NOR dengan 2 masukan.

Gambar 2.8 Gerbang NOR

Tabel 2.6 Tabel kebenaran untuk gerbang NOR 2-masukan

X	Y	X + Y
0	0	1
0	1	0
1	0	0
1	1	0

Gerbang OR Eksklusif Gambar 2.9 menunjukkan gerbang OR eksklusif. Exclusive OR diwakili oleh ⊕ dan diberi label XOR. Tabel 2.7 menunjukkan tabel kebenaran untuk gerbang XOR.



Gambar 2.9 2-input XOR

Tabel 2.7 Tabel Kebenaran Gerbang XOR

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Gerbang NOR Eksklusif Gambar 2.10 menunjukkan gerbang NOR eksklusif. Eksklusif NOR diwakili oleh ⊙ dan diberi label XNOR. Tabel 2.8 menunjukkan tabel kebenaran untuk gerbang NOR eksklusif.

$$X \oplus Y = X'Y + XY'$$

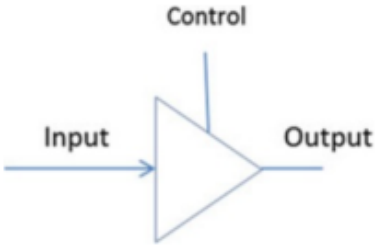
Gambar 2.10 Gerbang NOR Eksklusif

Tabel 2.8 Tabel kebenaran untuk gerbang NOR eksklusif

X	Y	X ʘ Y
0	0	1
0	1	0
1	0	0
1	1	1

Perangkat Tri-State

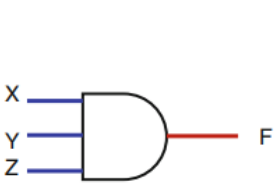
Gambar 2.11 menunjukkan diagram perangkat tri-state; garis kontrol mengontrol pengoperasian perangkat tri-state. Pada Gambar 2.11, jika garis kontrol diset ke nol, maka tidak ada hubungan antara input dan output (output berimpedansi tinggi). Jika garis kontrol diatur ke satu, maka nilai keluaran sama dengan nilai masukan.



Gambar 2.11 Perangkat tri-state

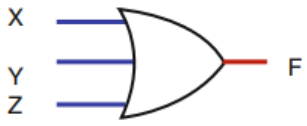
Gerbang Logika Masukan Ganda

Gambar 2.12 menunjukkan gerbang AND 3 masukan dengan tabel kebenarannya, Gambar 2.13 menunjukkan gerbang OR 3 masukan dengan tabel kebenarannya, Gambar 2.14 menunjukkan gerbang NOR 3 masukan dengan tabel kebenarannya, dan Gambar 2.15 menunjukkan NAD 3 masukan gerbang dengan tabel yang sesuai.



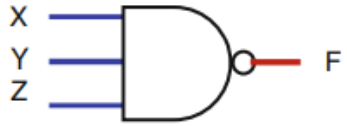
X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Gambar 2.12 Gerbang AND 3-masukan dan tabel kebenarannya



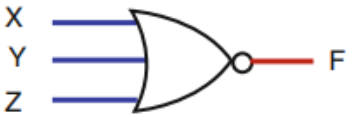
X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Gambar 2.13 Gerbang OR 3-input dan tabel kebenarannya



X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Gambar 2.14 Gerbang NAND 3-input dan tabel kebenarannya



X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

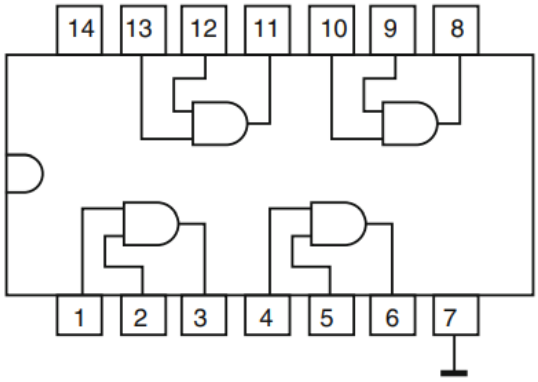
Gambar 2.15 Gerbang NOR 3-input dan tabel kebenarannya

2.3 KLASIFIKASI SIRKUIT TERPADU (IC)

Transistor adalah komponen dasar dari sirkuit terpadu (IC). Gambar 2.16 menunjukkan transistor dengan IC. Transistor bertindak seperti saklar di sirkuit terpadu. Sirkuit terpadu dibuat dari ratusan hingga jutaan transistor. Sirkuit terpadu diklasifikasikan berdasarkan jumlah gerbang di dalam IC seperti SSI, MSI, LSI, dan VLSI.

Small-Scale Integration (SSI) SSI mengacu pada IC yang memiliki kurang dari 10 gerbang. Gambar 2.17 menunjukkan bagian dalam IC 74HC08 (7408), dan Gambar 2.18 menunjukkan gambar IC 74HC08.

Gambar 2.16 Transistor (kiri), IC (kanan)



Gambar 2.17 TTL 7408 2 input AND gerbang

Gambar 2.18 74LS08 pin

Penomoran Pin Sirkuit Terpadu Gambar 2.18 menunjukkan IC TTL 7408; chip IC harus dibaca dengan takik di sisi kiri. Konvensi pelabelan dimulai dengan pin kiri bawah di bawah takik dan bergerak berlawanan arah jarum jam.

Pin kiri bawah adalah pin #1.

Pin kanan bawah adalah pin #7.

Pin kanan atas adalah pin #8.

Pin kiri atas adalah pin #14.

Seperti yang ditunjukkan pada Gambar 2.18, nomor IC adalah 74LS08 di mana LS mewakili bahan yang digunakan untuk membuat IC. Pada IC tersebut juga terdapat huruf M yang mewakili Motorola yang merupakan produsen dari IC tersebut, Intel menggunakan karakter "i", dan Texas Instruments menggunakan peta Texas.

Integrasi Skala Menengah (MSI). Mengacu pada IC yang berisi antara 10 dan 100 gerbang seperti decoder dan multiplexer.

Integrasi Skala Besar (LSI). Mengacu pada IC yang berisi antara 100 dan 1000 gerbang.

Integrasi Skala Sangat Besar (VLSI). Mengacu pada IC yang berisi lebih dari 1000 gerbang.

2.4 TEOREMA ALJABAR BOOLEAN

Teorema Boolean digunakan untuk menyederhanakan fungsi Boolean agar gerbang yang digunakan lebih sedikit. Variabel apa pun seperti X dalam biner dapat memiliki nilai satu atau nol.

Teorema #1

$$X + X = X$$

Bukti: Pilih X sebagai 0 lalu 0 + 0 = 0; pilih X = 1 lalu 1 + 1 = 1 hasil: X + X = X.

Teorema #2

$$X + 1 = 1$$

Bukti: Pilih X = 0 lalu 0 + 1 = 1; pilih X = 1 lalu 1 + 1 = 1; kedua kasus hasilkan 1 maka X + 1 = 1.

Teorema #3

$$X + 0 = X$$

Bukti: Pilih x = 0 lalu 0 + 0 = 0; pilih X = 1 lalu 1 + 0 = 1; hasilnya adalah berapapun nilai X.

Teorema #4

$$X + X' = 1$$

Bukti: Pilih X = 0 lalu 0 + 1 = 1; pilih X = 1 lalu 1 + 0 = 1; dalam kedua kasus hasilnya adalah 1.

Teorema #5

$$X.X = X$$

Bukti: Pilih X = 1 lalu 1.1 = 1; pilih X = 0 lalu 0,0 = 0; jadi XX = X.

Teorema #6

$$X.1 = X$$

Bukti: Pilih X = 1 lalu 1.1 = 1; pilih X = 0 lalu 0,1 = 0; oleh karena itu X.1 = X.

Teorema #7

$$X . X' = 0$$

Bukti: Pilih X = 0 lalu 0,1 = 0; pilih X = 1 lalu 1,0 = 0; kedua nilai X menghasilkan 0.

Teorema #8

$$(X')' = X$$

$(0')' = (1)' = 0$, $(1')' = (0)' = 1$ Berapa pun nilai X yang dimiliki.

Teorema distributif

$$X(Y + Z) = XY + XZ$$

Untuk membuktikan teorema di atas, tabel kebenaran dari kedua sisi teorema dihasilkan seperti yang ditunjukkan pada Tabel 2.9 dan menunjukkan kedua sisi menghasilkan tabel kebenaran yang sama.

Tabel 2.9 Tabel kebenaran X(Y + Z) = XY + XZ

X	Y	Z	Y+Z	X(Y+Z)	XY	XZ	XY + XZ
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Teorema I De Morgan

$$(X + Y)' = X'\,Y'$$

Dalam teorema ini OR antara X dan Y dinegasikan dan mengubah operasi OR menjadi operasi AND.

Bukti: Dengan membuat tabel kebenaran untuk kedua sisi teorema, menunjukkan bahwa kedua sisi teorema menghasilkan tabel kebenaran yang sama (Tabel 2.10).

Tabel 2.10 Tabel kebenaran yang menunjukkan Hukum De Morgan

X	Y	X+Y	(X +Y)´	X´	Y´	X´Y´
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Teorema II De Morgan

$$(XY)' = X' + Y'$$

Dalam teorema ini, XY dilengkapi dan mengubahnya dari operasi AND menjadi operasi OR dengan masing-masing komponen dilengkapi.

Contoh:

$$(WXYZ)' = W' + X' + Y' + Z'$$

Jika tabel kebenaran kedua ruas dibangkitkan, maka akan terlihat bahwa kedua ruas memiliki hasil tabel kebenaran yang sama.

Hukum komutatif

$$X + Y = Y + X$$
$$XY = YX$$

Hukum Asosiatif

$$X(YZ) = (XY)Z$$
$$X + (Y + Z) = (X + Y) + Z$$

Teorema lainnya

Berikut ini adalah teorema yang berguna:

(a) $X + X'Y = X + Y$
(b) $X' + XY = X' + Y$
(c) $X + X'Y' = X + Y'$
(d) $X' + XY' = X' + Y'$

Contoh: Sederhanakan Fungsi-Fungsi Berikut

(a) $F(X, Y, Z) = XY'Z + XY'Z' + XY$
$F(X, Y, Z) = XY'(Z + Z') + YZ$ where $Z + Z' = 1$ then
$F(X, Y, Z) = XY' + XY = X(Y + Y') = X$

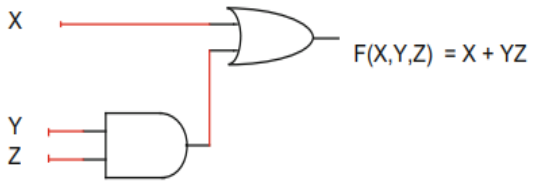
(b) $F(X, Y, Z) = (X' + Y)(X + Y') = X'X + X'Y' + XY + YY'$ where $X'X$ and YY' are zero then $F(X, Y, Z) = X'Y' + XY$

2.5 FUNGSI BOOLEAN

Fungsi Boolean diwakili oleh ekspresi aljabar yang dibuat dari variabel biner seperti X, Y, dan Z dan operasi logika antar variabel seperti AND, OR, dan NOT.

F(X, Y, Z) = X + YZ adalah fungsi Boolean.

Gambar 2.19 menunjukkan rangkaian logika untuk fungsi F di mana X, Y, dan Z adalah input dan F adalah output; Tabel 2.11 menunjukkan tabel kebenaran fungsi F. Tabel kebenaran menunjukkan karakteristik fungsi F; fungsi F 1 ketika input ke rangkaian adalah 100 atau 101 atau 110 atau 111.



Gambar 2.19 Rangkaian logika untuk fungsi F(X, Y, Z) = X + YZ

Tabel 2.11 Tabel kebenaran fungsi F(X, Y, Z) = X + YZ

X	Y	Z	YZ	X + YZ
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1

1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Komplemen dari Fungsi

Untuk melengkapi suatu fungsi, kedua sisi fungsi harus dilengkapi.

Contoh: Lengkapi fungsi berikut:

$$F(X, Y, Z) = XY + Y'Z$$

Melengkapi kedua sisi fungsi F(X, Y, Z) menghasilkan sebagai berikut: F0ðX; Y; ZÞ XY Y0Z 0 menggunakan teorema De Morgan:

$$F'(X, Y, Z) = (XY + Y'Z)'$$
$$F'(X, Y, Z) = (XY)'(Y'Z)'$$
$$F'(X, Y, Z) = (X' + Y')(Y + Z')$$

Contoh: Carilah komplemen dari fungsi berikut:

$$F(X, Y, Z) = (X' + Y')(Y + Z')$$

Lengkapi kedua sisi fungsi:

$$F'(X, Y, Z) = [(X' + Y')(Y + Z')]'$$

Menerapkan hasil teorema De Morgan

$$F'(X, Y, Z) = [(X' + Y')]' + [(Y + Z')]'$$
$$F'(X, Y, Z) = [(XY)] + [(Y'Z)]$$
$$F'(X, Y, Z) = (XY) + (Y'Z)$$

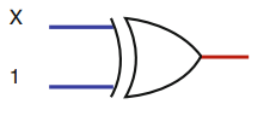
2.6 RINGKASAN

- Logika Boolean terdiri dari logika AND, OR, dan NOT.
- Keluaran dari gerbang AND 2 masukan adalah satu jika kedua masukan adalah satu; jika tidak, outputnya nol.
- Output dari gerbang OR 2-input adalah satu ketika setidaknya salah satu inputnya adalah satu; jika tidak, outputnya nol.
- Gerbang NOT melakukan komplemen seseorang.
- Sirkuit terpadu (IC) diklasifikasikan berdasarkan SSI (integrasi skala kecil), MSI (integrasi skala menengah), LSI (integrasi skala besar), dan integrasi skala sangat besar.
- Gerbang NAND ekuivalen dengan AND-NOT.
- Sebuah gerbang NOR setara dengan NOR-NOT.
- XOR 2-input memiliki output satu ketika inputnya tidak sama; jika tidak, outputnya nol.
- NOR eksklusif 2-input (XNOR) setara dengan XOR-NOT.
- Bab 3 akan membahas minterm dan maxterm, menerapkan K-map untuk menyederhanakan suatu fungsi, dan menggunakan gerbang universal untuk menggambar rangkaian logika.

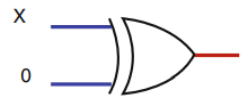
Masalah

1. Tunjukkan tabel kebenaran untuk gerbang AND, OR, NOR, dan NAD 4 masukan.

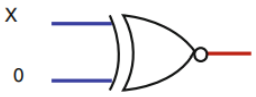
2. Jika A = 11001011 dan B = 10101110, maka berapa nilai dari operasi berikut:
   (a) A DAN B
   (b) A ATAU B
3. Jika A = 11001011 dan B = 10101110, berapakah nilai dari operasi berikut (F dalam hex 1111):
   (a) TIDAK
   (b) A XOR B
   (c) A DAN 0F
   (d) A DAN F0
4. Gambarlah rangkaian logika untuk fungsi-fungsi berikut:
   (a) $F(X, Y, Z) = XY' + XZ' + YZ$
   (b) $F(X, Y, Z) = (X + Y')\,(X' + Z')(Y + Z)$
5. Gunakan teorema Boolean untuk menyederhanakan ekspresi berikut:
   (a) $X + X + X$
   (b) $XY + XY$
   (c) YYY
   (d) $X + XY$
   (e) $XY' + Y'$
   (f) $(X + Y)Y'$
   (g) $(XY) + (XY)'$
   (h) $X'Y' + XY$
6. Sederhanakan fungsi berikut:
   (a) $F(X, Y, Z) = XY + X'Y + XZ$
   (b) $F(X, Y, Z) = (X + Y)\,(X' + Y + Z)$
   (c) $F(X, Y, Z) = XY'Z + XYZ + Y'Z$
   (d) $F(X, Y, Z) = XY + X'YZ$
   (e) $F(X, Y, Z) = X'Y + XYZ'$
   (f) $F(X, Y, Z) = (XY) + (X + Y + Z)'X + YZ$
   (g) $F(X, Y, Z) = (XY)' + (X + Y + Z)'$
7. Carilah tabel kebenaran untuk fungsi-fungsi berikut:
   (a) $F(X, Y, Z) = XY' + YZ + XZ'$
   (b) $F(X, Y, Z) = (X + Y')(Y + Z)(X' + Z')$
8. Jika A = 10110110 dan B = 10110011, maka tentukan
   (a) A NAND B
   (b) A NOR B
   (c) A XOR B
9. Tentukan keluaran dari gerbang berikut:
   (a)



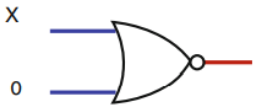
(b)



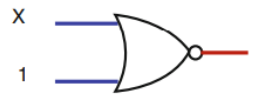
(c)



(d)

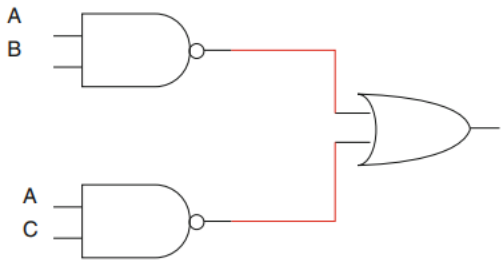


(e)

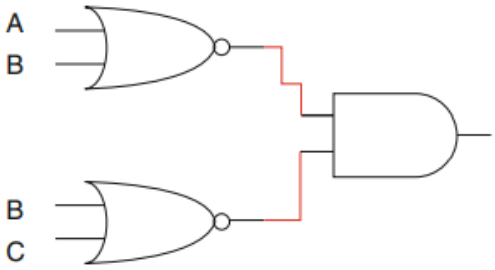


10. Tunjukkan output dari rangkaian logika berikut:

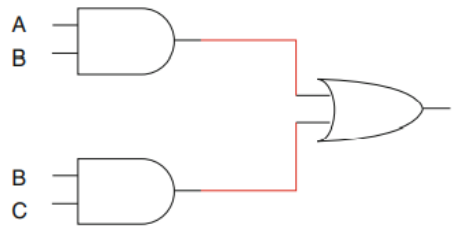
(a)



(b)

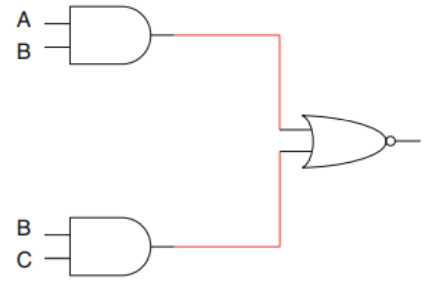


(c)

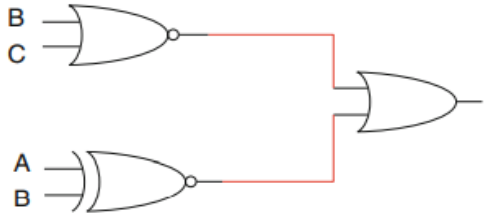


11. Carilah fungsi keluaran dari rangkaian logika berikut:

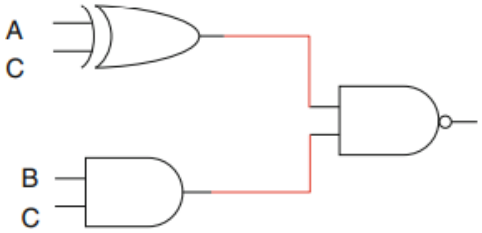
(a)



(b)



(c)



12. Carilah fungsi keluaran dari rangkaian logika berikut:

13. Gambarlah rangkaian logika dan tunjukkan tabel kebenaran untuk fungsi-fungsi berikut:
    (a) $F(X, Y) = (XY)' + X(X + Y')$
    (b) $F(X, Y, Z) = (X + Y + Z')' (X' + Y')$
    (c) $F(X, Y, Z) = (X \text{ XOR } Y) (X \text{ NOR } Y')$
    (d) $F(X, Y, Z) = (X' + Y' + Z) (X + Y)$
14. Tunjukkan tabel kebenaran untuk setiap fungsi berikut:
    (a) $F(X, Y, Z) = XY' + XZ' + YZ$
    (b) $F(X, Y, Z) = (X + Y) (X + Z')$
    (c) $F(X, Y, Z) = XY (Y + Z')$
15. Sederhanakan fungsi berikut:
    (a) $F(X, Y, Z) = YZ + (X + Y)' + (XYZ)'$
    (b) $F(X, Y, Z) = (X + Y + Z)' (X + Y)$
16. Gambarlah rangkaian Logika untuk fungsi-fungsi berikut:
    (a) $F(X, Y, Z) = (X + Y)' + YZ$
    (b) $F(X, Y, Z) = (XYZ)' + XZ + YZ$

BAB 3
MINTERMS, MAXTERMS, KARNAUGH MAP (K-MAP) DAN GERBANG UNIVERSAL

Tujuan: Setelah menyelesaikan bab ini, Anda diharapkan dapat:

- Mewakili fungsi Boolean dalam bentuk jumlah minter dan produk maxterm.
- Menghasilkan tabel kebenaran dari fungsi yang diwakili oleh jumlah minterms.
- Menghasilkan tabel kebenaran dari fungsi yang diwakili oleh produk minterms.
- Mengembangkan fungsi dari tabel kebenaran.
- Gunakan K-map untuk menyederhanakan suatu fungsi.
- Terapkan kondisi tidak peduli di K-map.
- Gambarlah rangkaian logika hanya dengan menggunakan gerbang NAND atau NOR.

3.1 PENDAHULUAN

Rangkaian digital dapat direpresentasikan dengan tabel kebenaran atau fungsi Boolean. Dalam rangkaian digital dengan beberapa input digital dan beberapa output digital, output bergantung pada nilai input saat ini. Sebuah fungsi Boolean dapat direpresentasikan dalam bentuk jumlah minterms atau produk dari maxterms, yang memungkinkan desainer untuk membuat tabel kebenaran lebih mudah. Selain itu, fungsi Boolean dapat disederhanakan menggunakan peta Karnaugh (K-map) tanpa menggunakan teorema Boolean, dengan mentransfer fungsi ke K-map dan membaca fungsi yang disederhanakan dari K-map. Kebanyakan sistem digital dirancang dengan menggunakan gerbang universal (NAND atau NOR).

3.2 MINTERM

Sebuah minterm dikaitkan dengan setiap kombinasi variabel dalam suatu fungsi. Jika suatu fungsi memiliki n variabel, maka fungsi tersebut memiliki 2^n minterm. Pertimbangkan dua variabel Boolean, X dan Y. Ada empat kemungkinan kombinasi berbeda yang dapat dihasilkan dari X DAN Y, dan mereka adalah $\bar{X}\bar{Y}$, $\bar{X}$, Y, X$\bar{Y}$, dan XY. Keempat kombinasi ini disebut minterm untuk X DAN Y. Tabel 3.1 menunjukkan minterm dan sebutannya untuk F(X,Y) (X DAN Y).

Pada Tabel 3.1, $\bar{X}\bar{Y}$ = 1, jika X 0 dan Y 0 maka $\bar{X}\bar{Y}$ diwakili oleh m_0 (nomor decimal 00); $\bar{X}$ Y = 1 jika X = 0 dan Y = 1 maka $\bar{X}$ Y dilambangkan dengan m_1;X $\bar{Y}$ = 1 jika X = 1 dan Y = 0 dan X $\bar{Y}$ dilambangkan dengan m_2; XY = 1 jika X = 1 dan Y = 1 maka XY diwakili oleh m_3.

Tabel 3.1 Minterm dari F(X,Y)

X Y	Minterm	Designation
0 0	$\bar{X}\,\bar{Y}$	m_0
0 1	$\bar{X}$ Y	m_1
1 0	X $\bar{Y}$	m_2
1 1	X Y	m_3

Aplikasi Minterm. Sangat mudah untuk menghasilkan tabel kebenaran dari minterms dan sebaliknya. Perhatikan fungsi F(X,Y) = X $\bar{Y}$+ $\bar{X}$ Y dan tabel kebenarannya (Tabel 3.2); fungsi ini dapat direpresentasikan sebagai F(X,Y) = m_1 + m_2 atau setiap minterm yang mewakili satu dalam tabel kebenaran. Ini juga dapat ditulis ulang sebagai F (X, Y) = $\sum(1,2)$.

Tabel 3.2 Tabel kebenaran fungsi F(X,Y) = X $\bar{Y}$+ $\bar{X}$ Y dengan minterms

X	Y	F	
0	0	0	~~m0~~
0	1	1	m_1
1	0	1	m_2
1	1	0	~~m3~~

Tiga Variabel Minterm Ketiga variabel X, Y, dan Z menghasilkan delapan minterm seperti yang ditunjukkan pada Tabel 3.3.

Contoh 3.1 Temukan tabel kebenaran untuk fungsi berikut:

$$F(X, Y, Z) = X'Y'Z + X'YZ + XYZ$$

Fungsi F dapat diwakili oleh jumlah minterms (atau di mana F 1):

$$F(X, Y, Z) = m_1 + m_3 + m_7$$

Atau

$$F(X, Y, Z) = \sum (1, 3, 7)$$

Tabel 3.3 Minterm tiga variable

X Y Z	Minterms	Designation
0 0 0	X′Y′Z′	m_0
0 0 1	X′Y′Z	m_1
0 1 0	X′YZ′	m_2
0 1 1	X′YZ	m_3
1 0 0	XY′Z′	m_4
1 0 1	XY′Z	m_5
1 1 0	XYZ′	m_6
1 1 1	XYZ	m_7

Tabel kebenaran untuk fungsi ini berisi satu baris 1, baris 3, dan baris 7. Sisa baris adalah nol seperti yang ditunjukkan pada Tabel 3.4. Fungsi tabel kebenaran juga dapat ditentukan dari jumlah mintermnya.

Tabel 3.4 Tabel Kebenaran Fungsi F(X,Y,Z) $X'Y'Z + X'YZ + XYZ$

X Y Z	F
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	0
1 1 0	0
1 1 1	1

Contoh 3.2 Diberikan tabel kebenaran berikut; cari fungsi F

Pada tabel berikut, keluaran dari fungsi F adalah satu bila masukannya adalah 001 = m_1, 011 = m_3, 101 = m_5, dan 111 = m_7; oleh karena itu F(X,Y,Z) = $m_1 + m_3 + m_5 + m_7$ atau

$$F(X, Y, Z) = \sum (1, 3, 5, 7)$$

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Mengganti setiap sebutan minterm dengan istilah produk yang sebenarnya (misalnya, m_0 = $X'Y'Z'$) menghasilkan fungsi berikut:

$$F(X, Y, Z) = X'Y'Z + X'YZ + XY'Z + XYZ$$

Contoh 3.3 Untuk tabel kebenaran berikut:

(a) Tentukan fungsi F.
(b) Sederhanakan fungsi tersebut.
(c) Gambarkan rangkaian logika untuk fungsi yang disederhanakan.

X	Y	Z	F
0	0	0	1
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

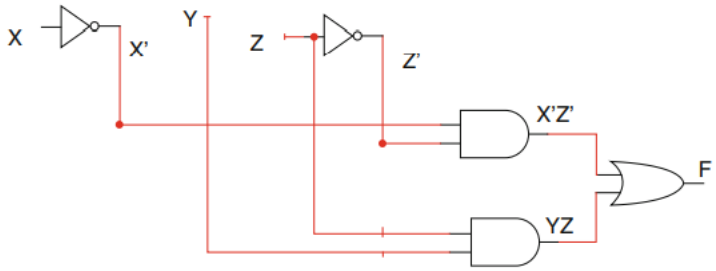
Dari tabel kebenaran, di mana F = 1, kami memilih minterm m_0, m_2, m_3, dan m_7:

$$F(X, Y, Z) = m_0 + m_2 + m_3 + m_7$$

Atau

$$F(X, Y, Z) = \bar{X}\bar{Y}\bar{Z} + \bar{X}Y\bar{Z} + \bar{X}YZ + XYZ$$
$$\bar{X}\bar{Z} \qquad\qquad \bar{Z}$$
$$F(X, Y, Z) = \bar{X}\bar{Z}(\bar{Y} + Y) + YZ(\bar{X} + X)$$
$$F(X, Y, Z) = \bar{X}\bar{Z} + YZ$$

Rangkaian logika untuk fungsi F yang disederhanakan diberikan pada Gambar 3.1.



Gambar 3.1 Rangkaian logika untuk F(X,Y,Z) = $\bar{X}\bar{Z}$ = YZ

3.3 MAKSTERM

Maxterm adalah pelengkap dari minterm. Jika minterm m_0 adalah = $\bar{X}\ \bar{Y}\ \bar{Z}$, maka maxterm M0 adalah

$$(\bar{X}\bar{Y}\bar{Z}) = X + Y + Z$$

Tabel 3.5 menunjukkan maxterm untuk tiga variabel.

Tabel 3.5 Suku Maks Fungsi F(X,Y,Z)

X Y Z	Maxterm	Designation
0 0 0	$X + Y + Z$	M_0
0 0 1	$X + Y + Z'$	M_1
0 1 0	$X + Y' + Z$	M_2
0 1 1	$X + Y' + Z'$	M_3
1 0 0	$X' + Y + Z$	M_4
1 0 1	$X' + Y + Z'$	M_5
1 1 0	$X' + Y' + Z$	M_6
1 1 1	$X' + Y' + Z'$	M_7

Dalam tabel kebenaran, output satu mewakili minterms, dan output nol mewakili maxterms. Perhatikan kebenaran Tabel 3.6, di mana fungsi F dapat dinyatakan sebagai produk dari maxterms. (Harap dicatat: produk dari maxterms, sebagai lawan dari jumlah minterms.)
F(X,Y,Z) = $M_0M_2M_4M_5M_6$, atau dapat dilambangkan dengan

$$F(X, Y, Z) = \pi(0, 2, 4, 5, 6)$$

Mengganti setiap maxterm yang ditentukan dengan maxterm yang sesuai menghasilkan:

$$F(X, Y, Z) = (X + Y + Z)(X + Y' + Z)(X' + Y + Z)(X' + Y + Z')$$

Tabel 3.6 Tabel Kebenaran F(X,Y,Z) = $M_0M_2M_4M_5M_6$

X Y Z	F
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	0
1 1 0	0
1 1 1	1

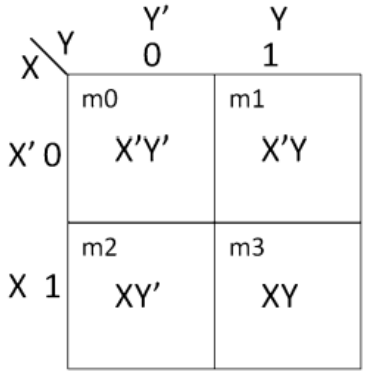
3.4 PETA KARNAUGH (K-MAP)

Peta Karnaugh digunakan untuk menyederhanakan fungsi Boolean tanpa menggunakan teorema aljabar Boolean. K-map juga merupakan cara lain untuk merepresentasikan tabel kebenaran suatu fungsi. K-maps terbuat dari sel-sel di mana setiap sel mewakili minterm. Sel yang ditandai dengan satu akan menjadi minterm yang digunakan untuk jumlah representasi minterm dari suatu fungsi. Sebaliknya, sel yang ditandai dengan nol akan digunakan untuk produk dari representasi maxterms.

Dua variabel X dan Y dapat memiliki empat minterm seperti yang ditunjukkan pada Tabel 3.7. Setiap minterm diwakili oleh sebuah sel di K-map, jadi K-map dua variabel berisi empat sel seperti yang ditunjukkan pada Gambar 3.2. Pada Gambar 3.2 setiap sel mewakili minterm. Sel yang terletak di baris 0 dan kolom 0 mewakili m_0 (minterm nol) atau $X'Y'$. Sel yang terletak pada baris 1 dan kolom 1 diwakili oleh m3 atau XY. Seperti ditunjukkan pada Gambar. 3.2, kedua sel di baris nol berisi X', jadi baris ini diberi label baris X'. Kedua sel di baris 1 berisi X sehingga baris diberi label sebagai baris X. K-map dari suatu fungsi adalah cara lain untuk merepresentasikan tabel kebenaran dari fungsi tersebut, seperti terlihat pada Gambar 3.3. Pertimbangkan fungsi F(X,Y) X Y' + $X'Y'$ = m_2 + m_0. Tabel kebenaran untuk fungsi tersebut diberikan pada Tabel 3.8, dan fungsi tersebut juga dipetakan ke K-map seperti yang ditunjukkan pada Gambar 3.3.

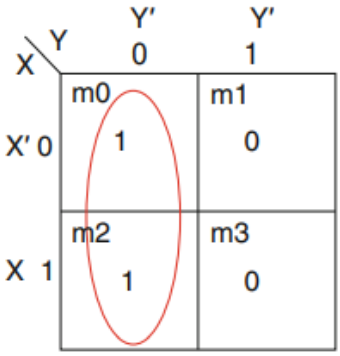
Tabel 3.7 Minterm untuk dua variable

X Y	Minterms	Designation
0 0	X′Y′	m_0
0 1	X′Y	m_1
1 0	XY′	m_2
1 1	XY	m_3



Gambar 3.2 K-map dua variable

Sel Berdekatan Dua sel dikatakan bertetangga jika hanya berbeda pada satu variabel. Sel $X'Y'$ dan $X'Y$ berdekatan karena satu-satunya perbedaan mereka adalah Y' dan Y. Sel yang berdekatan dapat digabungkan untuk menyederhanakan fungsi K-map.



Gambar 3.3 Fungsi dalam K-map

Tabel 3.8 Tabel kebenaran fungsi F(X,Y) = $XY' + X'Y'$

X Y	F
0 0	1
0 1	0
1 0	1
1 1	0

Seperti yang ditunjukkan pada K-map, sel m0 dan m2 berisi satu dan sel lainnya berisi nol. Sel m_0 dan m_2 saling berdekatan. Perhatikan bahwa sel-sel yang berdekatan mengambil seluruh kolom Y', dan semua sel lainnya adalah nol. Kami disederhanakan karena itu fungsinya adalah F(X,Y) Y'.

Contoh 3.4 Sederhanakan fungsi berikut:

$$F(X, Y) = X'Y + XY' + XY$$

atau

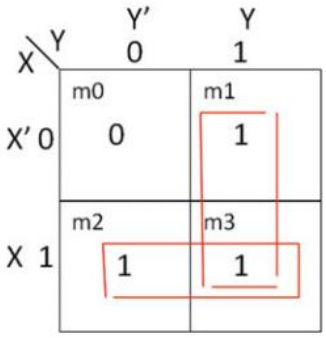
$$F(X, Y) = m_1 + m_2 + m_3$$

Mentransfer minterm ke dalam K-map menghasilkan pada Gambar 3.4.
Seperti ditunjukkan pada Gambar 3.4, sel m_2 dan m_3 berdekatan, sehingga dapat digabungkan. Demikian juga, sel m_1 dan m_3 dapat digabungkan. Dengan membaca peta, Anda akan memiliki fungsi yang disederhanakan. Sel m_2 dan m_3 adalah seluruh baris X, dan sel m_1 dan m_3 adalah seluruh kolom Y, dengan sel lainnya menjadi nol. Karena itu,

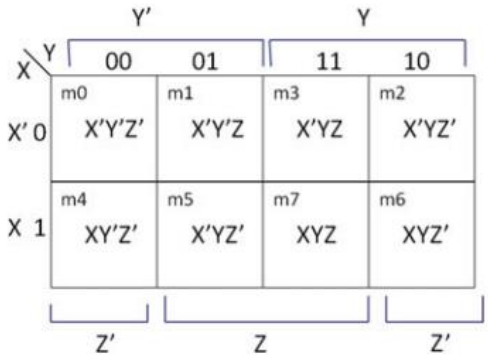
$$F(X, Y) = X + Y$$

Peta Tiga Variabel

K-map tiga variabel berisi delapan sel, dan setiap sel mewakili minterm seperti yang ditunjukkan pada Gambar 3.5.



Gambar 3.4 K-map untuk F(X, Y) = $X'Y + XY' + XY$



Gambar 3.5 Peta Tiga Variabel

Perhatikan hal berikut tentang K-map pada Gambar 3.5:

(a) Pada baris 0, keempat sel berisi X'; oleh karena itu baris ini diberi label X'.
(b) Pada baris 1, keempat sel berisi X; oleh karena itu baris ini diberi label X.
(c) Pada kolom 11 dan 10, keempat sel berisi Y; oleh karena itu kolom-kolom ini diberi label Y.
(d) Pada kolom 00 dan 01, keempat sel berisi Y'; oleh karena itu kolom ini diberi label Y'.
(e) Pada kolom 01 dan 11, keempat sel berisi Z; oleh karena itu kolom-kolom ini adalah berlabel Z
(f) Pada kolom 00 dan 10, keempat sel berisi Z'; oleh karena itu kolom ini diberi label Z'.

Sel-sel yang berdekatan dapat dikelompokkan bersama dalam peta-K; dalam K-map dapat menggabungkan 2 sel, 4 sel, 8 sel, dan 16 sel. Gambar 3.6 menunjukkan bagaimana yang dapat dikelompokkan dalam K-map. Pada Gambar 3.6, keempat sel dapat digabungkan. Melipat K-map secara horizontal dua kali akan menghasilkan semua peta yang tumpang tindih, dan baris X0 mencakup keempat peta tersebut. Untuk Gambar 3.7, pertimbangkan untuk melipat K-map sekali lagi. Keempatnya akan tumpang tindih jika peta dilipat sekali secara horizontal, lalu vertikal. Perhatikan juga bahwa Z0 mencakup keempatnya.

Contoh 3.5 Sederhanakan fungsi berikut:

$$F(X, Y, Z) = X'Y + XZ + XZ'$$

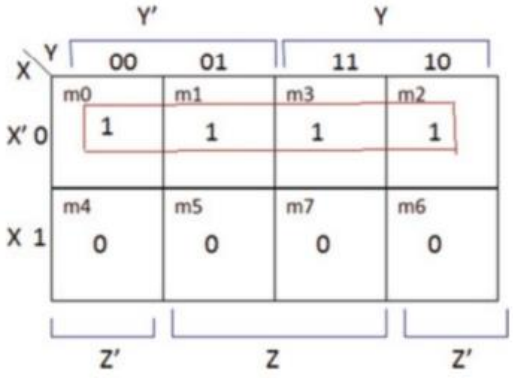
Pertama, setiap istilah fungsi harus ditransfer ke K-map.

(a) Suku pertama adalah $X'Y$, tempatkan satu pada setiap sel yang terletak di persimpangan baris X' dan kolom Y seperti yang ditunjukkan pada Gambar 3.8. (m_3 dan m_2)

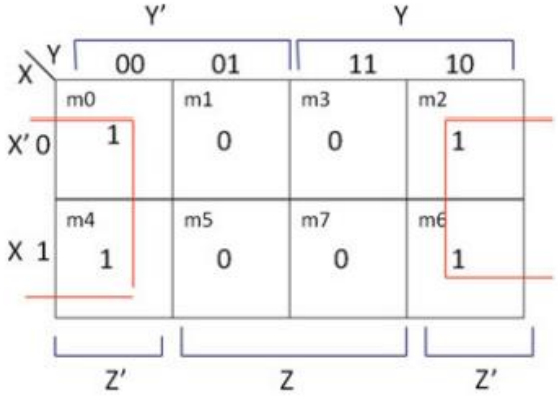
(b) Suku kedua adalah XZ, jadi tempatkan satu pada setiap sel yang terletak di perpotongan kolom Y' dan baris X. (m_5 dan m_7)

(c) Suku ketiga adalah XZ', tempatkan satu pada setiap sel di perpotongan baris X dan kolom Z. (m_4 dan m_6)

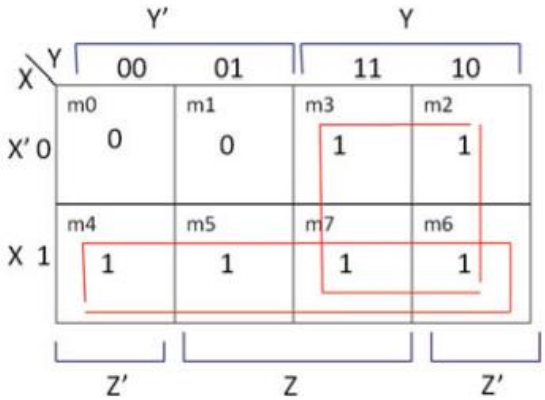
Karena sel-sel yang berdekatan mencakup seluruh baris X dan setiap sel dalam kolom Y, bentuk sederhana dari fungsi ini adalah F(X,Y,Z) = X + Y.



Gambar 3.6 Pengelompokan empat sel dalam peta-K di mana F(X,Y,Z) = X' yang disederhanakan



Gambar 3.7 Menyisir keempat sel Z' bersama-sama dalam K-peta



Gambar 3.8 Kombinasi sel-sel dari fungsi contoh dalam peta-K

Contoh 3.6 Baca K-maps pada Gambar 3.9a–d untuk menentukan fungsi yang disederhanakan.

(a) Semua sel yang berdekatan di kolom Z' dan kolom Y adalah satu.

$$F(X, Y, Z) = Z' + Y$$

(b) Sel-sel pada baris X', kolom Y' berdekatan, seperti sel-sel pada baris X, kolom Y.

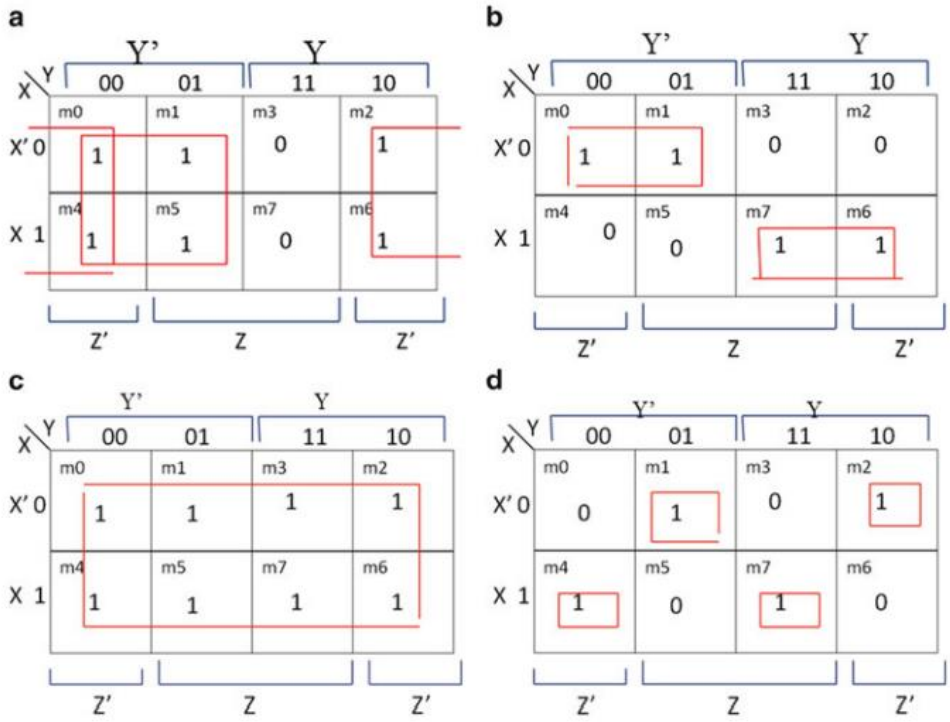
$$F(X, Y, Z) = X'Y' + XY$$

(c) Semua sel adalah satu, jadi fungsinya selalu sama dengan satu.

$$F(X, Y, Z) = 1$$

(d) Tanpa sel yang berdekatan untuk menyederhanakan istilah, fungsinya sama dengan yang:

$$F(X, Y, Z) = X'Y'Z + X'YZ' + XY'Z' + XYZ$$

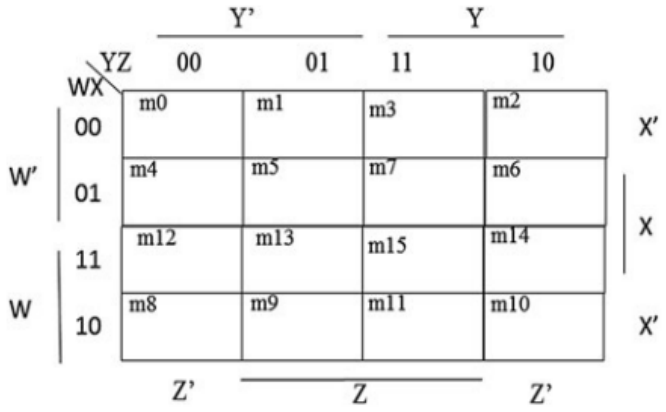


Gambar 3.9 K-Maps Contoh 3.6

K-Map Empat-Variabel

K-map empat variabel berisi 16 sel seperti yang ditunjukkan pada Gambar 3.10. Harap perhatikan tata letak spesifik peta. Berikut ini gambaran cakupan masing-masing variabel berdasarkan K-map:

- W mencakup baris 11 dan 10.
- W' mencakup baris 00 dan 01.
- X mencakup baris 01 dan 11.
- X' mencakup baris 00 dan 10.
- Y meliputi kolom 11 dan 10.
- Y' meliputi kolom 00 dan 01.
- Z meliputi kolom 01 dan 11.
- Z' meliputi kolom 00 dan 10.



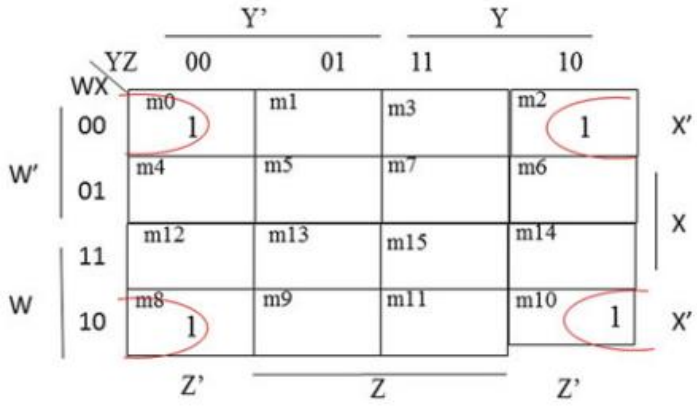
Gambar 3.10 K-map empat variable

Contoh 3.7 Sederhanakan fungsi berikut:

$$F(W, X, Y, Z) = m_0 + m_2 + m_8 + m_{10}.$$

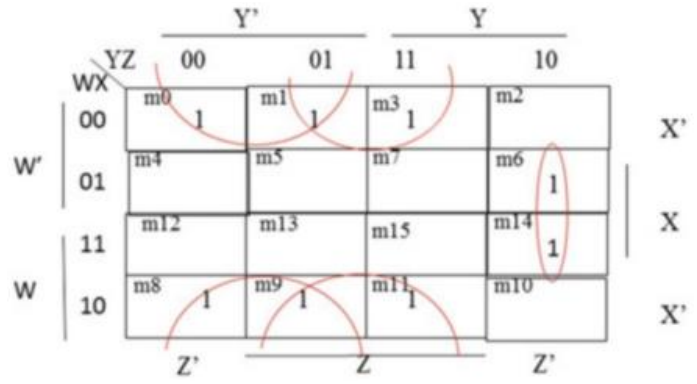
Fungsi tersebut ditransfer ke K-map seperti yang ditunjukkan pada Gambar 3.11. Jika K-map dilipat satu kali secara vertikal dan horizontal dari tengah, maka keempat sel yang berisi satu saling tumpang tindih. Perhatikan bahwa masing-masing sel ini membentuk semua persimpangan X' dan Z'.

Fungsi yang disederhanakan adalah F(W,X,Y,Z) = $X'\,Z'$.



Gambar 3.11 K-map untuk F(W, X,Y,Z) = $m_0 + m_2 + m_8 + m_{10}$

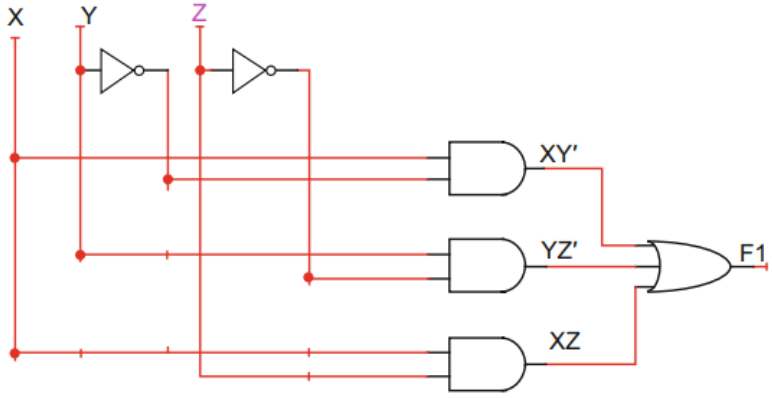
Contoh 3.8 Baca K-map berikut:



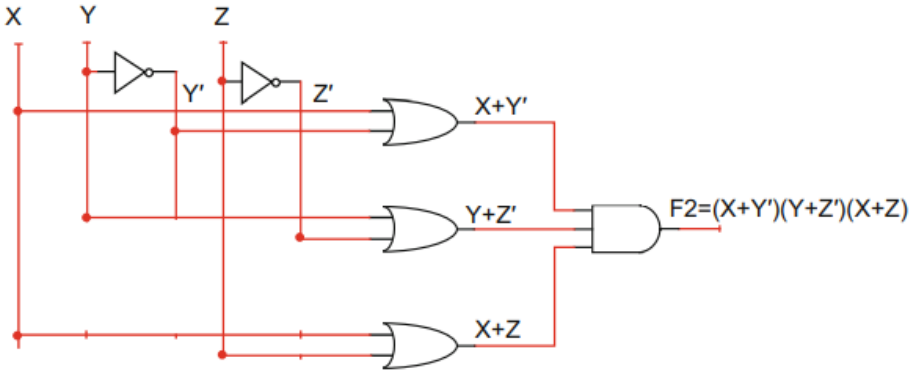
$$F(W, X, Y, Z) = X'Y' + X'Z + XYZ'$$

3.5 JUMLAH PRODUK (SOP) DAN PRODUK JUMLAH (POS)

Jumlah produk dari suatu fungsi adalah jumlah minterm yang disederhanakan. Perhatikan fungsi F(X,Y) = XY' + XY dalam bentuk SOP dimana tanda penjumlahan digunakan untuk logika OR, dan XY disebut hasil kali. Perhatikan fungsi F1 (X,Y,Z) = XY' + YZ' + XZ, dan Gambar 3.12 menunjukkan rangkaian logika untuk fungsi F1 yang terbuat dari AND-OR.



Gambar 3.12 Rangkaian logika F1(X,Y,Z) = XY' + YZ' + XZ terbuat dari gerbang AND-OR



Gambar 3.13 Rangkaian logika untuk F2(X,Y,Z) = (X + Y')(Y + Z')(X + Z)

Perhatikan fungsi F2(X,Y,Z) = (X + Y')(Y + Z')(X + Z) yang direpresentasikan oleh hasil kali jumlah, dan Gambar 3.13 menunjukkan rangkaian logika untuk fungsi F2 yang berbentuk OR -DAN.

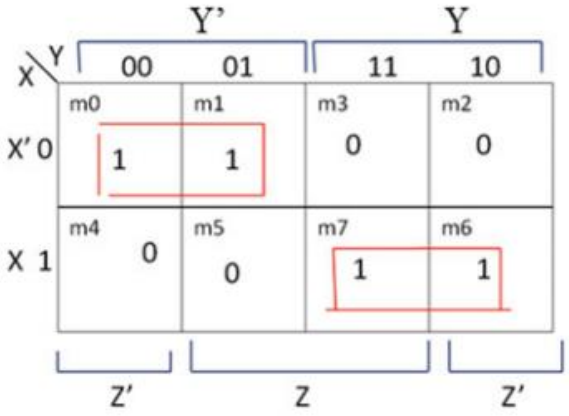
Contoh 3.9 Sederhanakan fungsi berikut dalam bentuk SOP dan POS.

- F(X,Y,Z) = $\sum(0,1,6,7)$

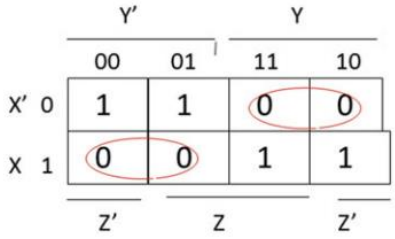
Menggabungkan satu (minterm m_0, m_1, m_6, dan m_7) dalam K-map menghasilkan pada Gambar 3.14.

Oleh karena itu, jumlah produknya adalah: F(X,Y,Z) = $X'Y'$ + XY. Menggabungkan nol dalam K-map mengembalikan maxterms signifikan seperti pada Gambar. 3.15. Jika F sama dengan $\sum(0,1,6,7)$, maka F' sama dengan $\pi(M_2M_3M_4M_5)$. Karena produk maxterms sama dengan komplemen dari F, untuk mencari F, kedua sisi fungsi akan dilengkapi:

$$F(X, Y, Z) = (XY' + X'Y)'$$



Gambar 3.14 K-map untuk F(X, Y,Z) = $\sum(0,1,6,7)$



Gambar 3.15 K-map untuk F'(X, Y,Z) = π($M_2M_3M_4M_5$)

Menggunakan teorema DeMorgan

$$F(X, Y, Z) = (XY')'(X'Y)'$$

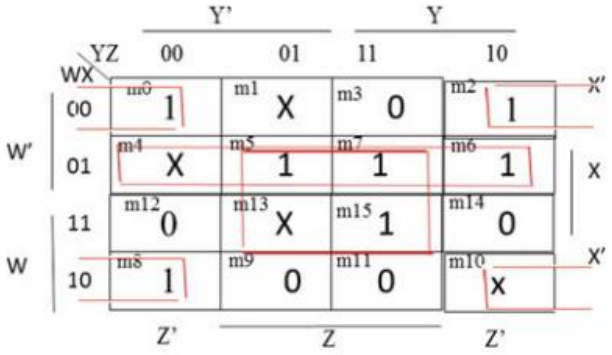
atau

$$F(X,Y,Z) = (X' + Y)(X + Y')$$

pada akhirnya, hasil kali bentuk penjumlahan.

Kondisi Jangan Peduli

Dalam tabel kebenaran, jika kombinasi tertentu dari variabel input tidak mungkin, mereka dianggap tidak peduli kondisi. Kondisi ini adalah di mana output dari fungsi tidak menjadi masalah. Misalnya, desimal berkode biner (BCD) adalah 4 bit dan hanya 0000 hingga 1001 yang digunakan, jadi dari 1010 hingga 1111 bukan BCD; tabel kebenaran atau nilai K-map tidak peduli. Tabel kebenaran atau sel K-map yang ditandai dengan "X" atau "d" adalah istilah yang tidak peduli, dan output tidak akan terpengaruh apakah itu satu atau nol. Not care dapat digunakan untuk memperluas kedekatan sel dalam K-map untuk lebih menyederhanakan fungsi, karena outputnya tidak penting.

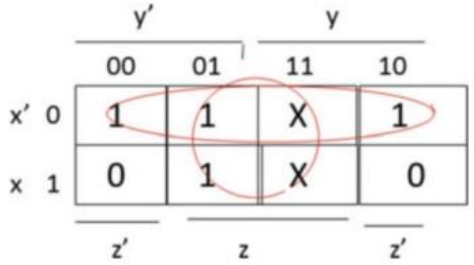


Gambar 3.16 K-map dengan not care minterm

Contoh 3.10 Gambar 3.16 menunjukkan K-map dengan not care minterms pada m_1, m_{10}, dan m_{13}. Karena don't care dapat menghasilkan nol atau satu, kita dapat menganggapnya sebagai satu untuk memperluas pengelompokan sel yang berdekatan.

Dari Gambar 3.16, fungsinya adalah ***F(W,X,Y,Z) = XZ + $X'Z'$ + XW'.***

Ketika minterm untuk fungsi F adalah suku tidak peduli, fungsi tidak peduli D sama dengan jumlah dari minterm tidak peduli. Jika m_7 adalah satu-satunya tidak peduli, maka fungsi tidak peduli diwakili oleh D(X,Y,Z) = m_7.



Gambar 3.17 K-map untuk F(X, Y,Z) = $m_0 + m_1 + m_2 + m_5$ dan D(X,Y,Z) = $m_3 + m_7$

Contoh 3.11 Sederhanakan fungsi berikut di mana D adalah fungsi tidak peduli:

$$F(X, Y, Z) = m_0 + m_1 + m_2 + m_5$$
$$D(X, Y, Z) = m_3 + m_7$$

Menggunakan nilai-nilai ini menghasilkan K-map pada Gambar 3.17. Dengan mengelompokkan sel yang berdekatan dan menggunakan istilah tidak peduli, F(X,Y,Z)= X' + Z.

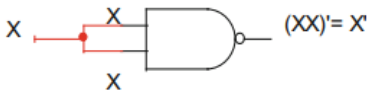
3.6 GERBANG UNIVERSAL

Gerbang NAND dan NOR disebut gerbang universal. Dengan NAND atau NOR, desainer dapat membangun gerbang logika lain seperti gerbang OR, AND, dan NOT.

Menggunakan Gerbang NAND

(a) NOT dari NAND

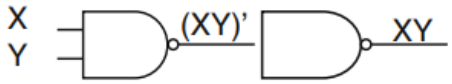
Gerbang NOT dihasilkan dengan menghubungkan input gerbang NAND bersama-sama seperti yang ditunjukkan pada gambar berikut.



Atau, dapat diwakili oleh:

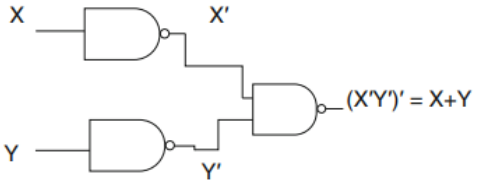
(b) AND dari NAND

Gerbang AND dibangun dengan menghubungkan input ke gerbang NAND dan menempatkan NAND lain pada output (untuk bertindak sebagai gerbang NOT).



(c) OR dari NAND

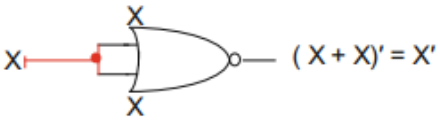
Gerbang OR dibangun dengan menghubungkan setiap input ke gerbang NAND individu dan menempatkan setiap output ke dalam satu NAND yang bertindak sebagai gerbang NOT.



Menggunakan Gerbang NOR

(a) NOT dari NOR

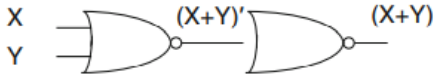
Gerbang NOT dihasilkan dengan menghubungkan input gerbang NOR bersama-sama seperti yang ditunjukkan pada gambar berikut.



Atau, dapat diwakili oleh:

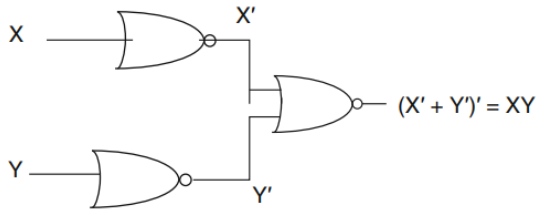
(b) OR dari NOR

Gerbang OR dibangun dengan menghubungkan input ke gerbang NOR dan menempatkan NOR lain pada output (untuk bertindak sebagai gerbang NOT).



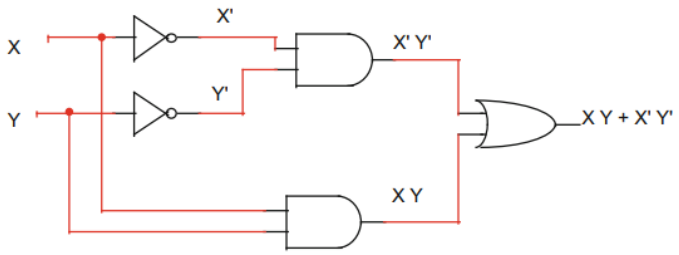
(c) AND dari NOR

Gerbang AND dibangun dengan menghubungkan setiap input ke gerbang NOR individu dan menempatkan setiap output ke dalam satu NOR yang bertindak sebagai gerbang NOT.



Implementasi Fungsi Logika Menggunakan Gerbang NAND atau Gerbang NOR Saja

Banyak fungsi logika diimplementasikan hanya dengan menggunakan gerbang NAND atau NOR, daripada kombinasi dari berbagai gerbang. Sebagian besar IC gerbang logika berisi beberapa gerbang dari satu jenis, seperti IC yang berisi delapan gerbang AND. Menggunakan satu jenis gerbang dapat mengurangi jumlah IC yang dibutuhkan. Perhatikan fungsi F(X,Y) = $X'Y'$ + XY dan diagram rangkaian logikanya pada Gambar 3.18. Diagram ini akan membutuhkan satu IC untuk gerbang AND, satu lagi untuk gerbang NOT, dan satu gerbang OR, atau total tiga IC yang terpisah. Dengan melengkapi fungsi dua kali, ruas kanan persamaan mungkin lebih mudah digunakan dengan gerbang NAND dan NOR.



Gambar 3.18 Rangkaian logika untuk F(X,Y) = XY + $X'Z'$

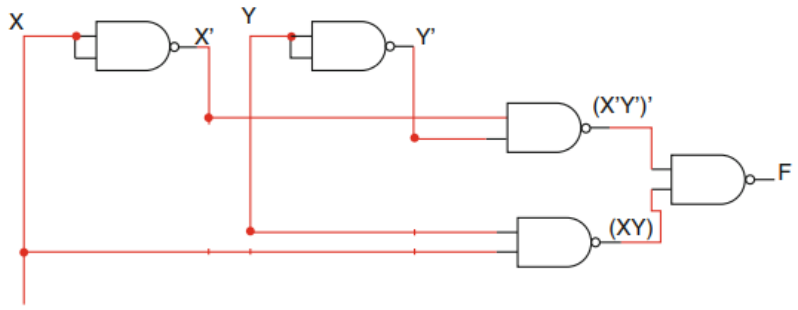
Contoh 3.12 Buat rangkaian logika hanya menggunakan NAND dan hanya NOR untuk fungsi ***F(X,Y) = $X'Y'$ + XY.***

Menggunakan Gerbang NAND

- Lengkapi ruas kanan persamaan dua kali.
- F(X,Y) = $X'Y'$ + XY → F(X, Y) = [($X'Y'$ + XY) ′]′.
- Gunakan teorema Boolean untuk membuatnya ramah NAND:
  – F(X, Y) = [($X'Y'$ + XY) ′]′

Pertimbangkan fungsi terakhir sekali lagi: F(X, Y) [(X0Y0)0(XY)0]0, dan gantikan placeholder untuk suku dalam (Gambar. 3.19).

- F = [(X′Y′)′(XY)′]′ = [AB]′ (A *NAND* B)
  - A = (X′Y′)′ = X′ *NAND* Y′
  - B = (XY)′ = X *NAND* Y
  - X′ = X *NAND* X
  - Y′ = Y *NAND* Y



Gambar 3.19 Rangkaian logika F(X,Y) = $X'Y'$ + XY hanya menggunakan gerbang NAND

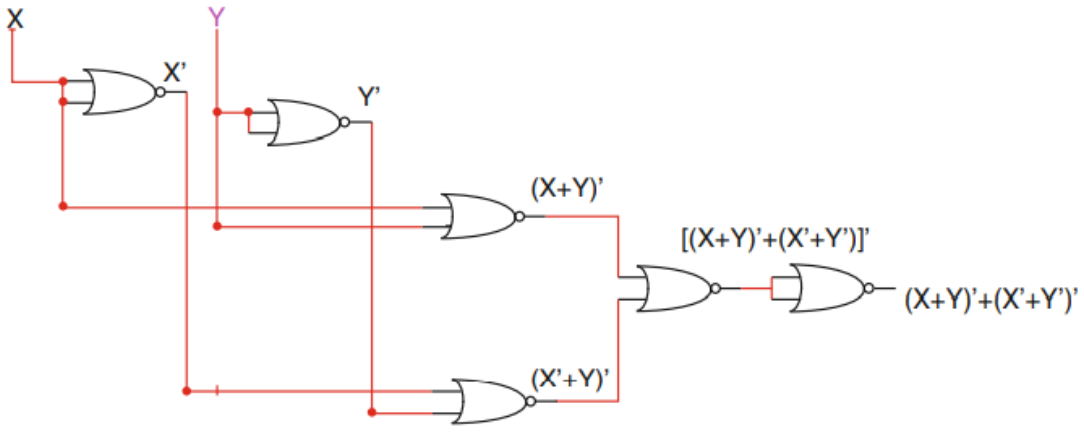
Menggunakan Gerbang NOR

- Lengkapi ruas kanan persamaan dua kali:
  - $F(X,Y) = X'Y' + XY \rightarrow F(X, Y) = [(X'Y' + XY)']'$

- Gunakan teorema Boolean untuk membuatnya NOR friendly:
  - $F(X,Y) = [(X'Y')'(XY)']'$
  - $F(X,Y) = [(X+Y)(X'+Y')]'$
  - $F(X,Y) = [(X+Y)' + (X'+Y')']'$

Pertimbangkan fungsi terakhir sekali lagi:
$F(X, Y) = [(X+Y)' + (X'+Y')']'$,
dan gantikan placeholder untuk suku dalam (Gambar. 3.20).

- $F = [(X'Y')'(XY)']' = [A+B]'$ (A *NOR* B)
  - $A = (X+Y)' = X$ *NOR* Y
  - $B = (X'+Y')' = X'$ *NOR* Y'
  - $X' = X$ *NOR* X
  - $Y' = Y$ *NOR* Y



Gambar 3.20 Rangkaian logika F(X,Y) = $X'Y'$ + XY hanya menggunakan gerbang NOR

3.8 RINGKASAN

- Sebuah sirkuit digital terbuat dari kombinasi gerbang yang berbeda dengan beberapa input digital dan beberapa output digital; output hanya bergantung pada nilai input saat ini.

- Rangkaian logika kombinasional dapat direpresentasikan dengan fungsi Boolean atau tabel kebenaran.
- Teorema Boolean atau K-map dapat digunakan untuk menyederhanakan fungsi Boolean.
- Fungsi Boolean dapat direpresentasikan dengan jumlah hasil kali (SOP) atau hasil kali jumlah (POS).
- Gerbang NAND dan NOR disebut gerbang universal. Itu dapat menghasilkan gerbang lain dengan menggunakan gerbang NAND atau NOR.
- Kondisi Don't Care adalah nilai input yang tidak pernah diterapkan pada rangkaian kombinasional yang menghasilkan output dengan kondisi Don't Care (0 atau 1).
- Bab 1, 2, dan 3 mencakup topik dasar untuk dapat merancang sistem digital; Bab 4 menyajikan bagaimana merancang sistem digital dan mencakup komponen digital yang digunakan untuk merancang sistem digital seperti decoder, multiplexer, binary adder, binary subtractor, dan arithmetic logic unit (ALU).

Masalah

1. Temukan fungsi keluaran dari setiap tabel kebenaran:
   (a) Sebagai jumlah dari minterms
   (b) Sebagai produk dari maxterms

X Y Z	F
0 0 0	1
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	1

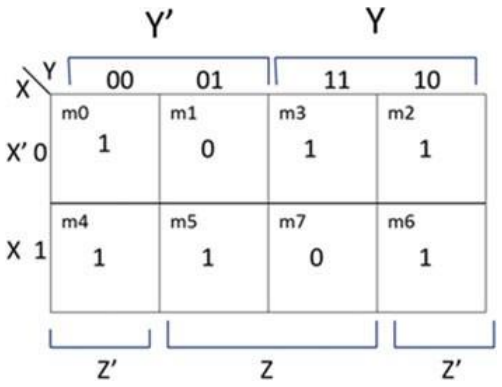
A B C D	F
0 0 0 0	1
0 0 0 1	0
0 0 1 0	1
0 0 1 1	1
0 1 0 0	0
0 1 0 1	1
0 1 1 0	1
0 1 1 1	1
1 0 0 0	1
1 0 0 1	1

1 0 1 0	0
1 0 1 1	0
1 1 0 0	0
1 1 0 1	1
1 1 1 0	1
1 1 1 1	1

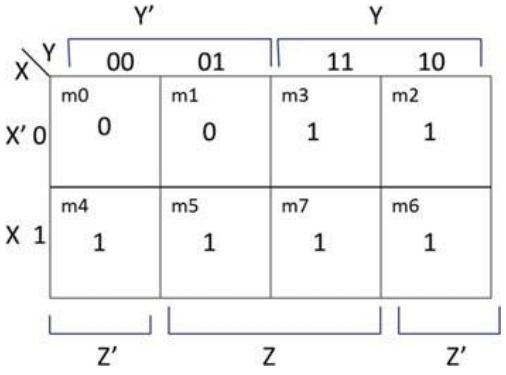
2. Buatlah tabel kebenaran untuk fungsi-fungsi berikut:
   (a) $F(X,Y,Z) = \sum(1,3,6,7)$
   (b) $F(X,Y,Z) = \pi(1,3,4)$
   (c) $F(W,X,Y,Z) = \sum(1,4,7,10,12,15)$
   (d) $F(W,X,Y,Z) = \pi(2,3,4,7,10,11,12,13)$

3. Hasilkan fungsi F untuk K-maps berikut:

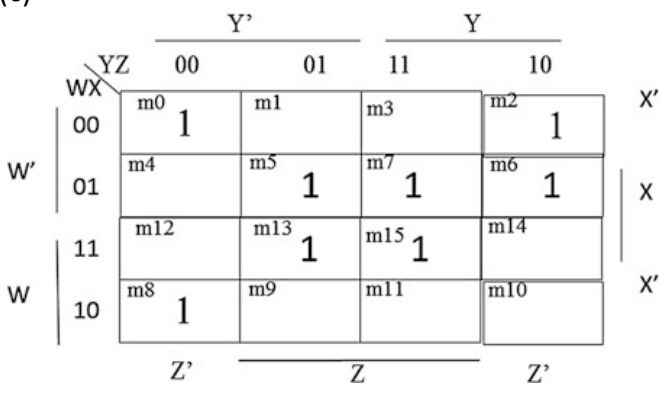
(a)



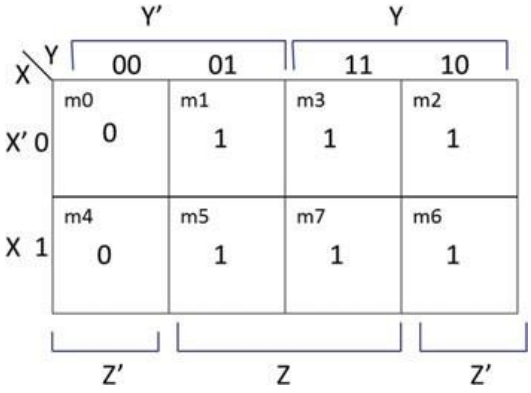
(b)



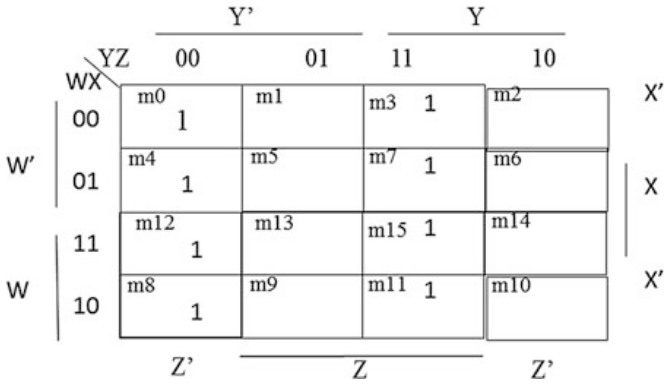
(c)



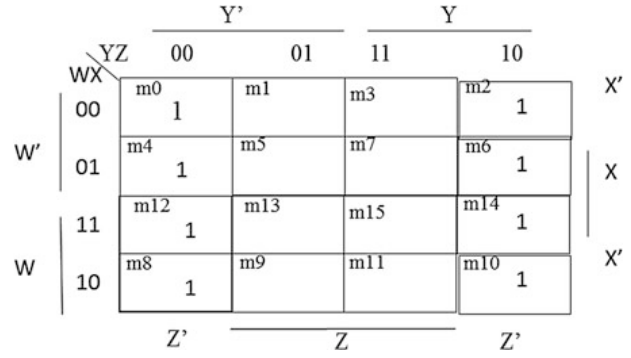
(d)



(e)



(f)



4. Temukan fungsi output yang disederhanakan untuk tabel kebenaran berikut menggunakan K-map:

(a)

X	Y	F
0	0	1
0	1	1
1	0	1
1	1	0

(b)

X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

(c)

A B C D	F
0 0 0 0	1
0 0 0 1	0
0 0 1 0	1
0 0 1 1	1
0 1 0 0	0
0 1 0 1	1
0 1 1 0	1
0 1 1 1	1
1 0 0 0	1
1 0 0 1	1
1 0 1 0	0
1 0 1 1	0
1 1 0 0	0
1 1 0 1	1
1 1 1 0	1
1 1 1 1	1

5. Sederhanakan fungsi berikut menggunakan K-map:

(a) $F(X,Y) = m_2 + m_3$
(b) $F(X,Y) = X + X'Y$
(c) $F(X,Y) = X' + XY'$
(d) $F(X,Y,Z) = m_0 + m_2 + m_5 + m_7$
(e) $F(X,Y,Z) = X'Y'Z' + X'YZ + XY'Z + XYZ$
(f) $F(X,Y,Z) = \pi(0, 2, 5, 7)$
(g) $F(X,Y,Z) = XY'Z + X' + Z + Y'Z'$
(h) $F(W,X,Y,Z) = X'Y'Z' + XYZ' + WXY + W'X'Y' + WZ$
(i) $F(W,X,Y,Z) = X' + XZ' + WX'Y + W'Y' + WZ$

6. Sederhanakan fungsi berikut di mana D adalah fungsi tidak peduli:
   (a) $F(X,Y,Z) = \sum(0,3,4)$
   $D(X,Y,Z) = \sum(2,6)$
   (b) $F(W,X,Y,Z) = \sum(0,1,3,5,9,11)$
   $D(W,X,Y,Z) = \sum(2,4,8,10)$
7. Sederhanakan fungsi-fungsi berikut dalam bentuk SOP dan POS, dan gambarkan rangkaian logikanya:
   (a) $F(X,Y,Z) = \sum(0,2,5,7)$
   (b) $F(W,X,Y,Z) = \sum(0,1,4,6,9,11,13,15)$
8. Gambarkan sirkuit logika untuk fungsi yang disederhanakan dari Soal 6:
   1. Menggunakan gerbang NAND
   2. Menggunakan gerbang NOR
9. Sederhanakan fungsi berikut dan gambar rangkaian logika menggunakan:
   (a) gerbang NAND
   (b) gerbang NOR

$$F(W, X, Y, Z) = W'X'Z' + XY'Z' + WX + WY + WX''Y'Z'$$

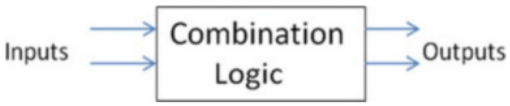
BAB 4
LOGIKA KOMBINASI

Tujuan: Setelah menyelesaikan bab ini, Anda diharapkan dapat:

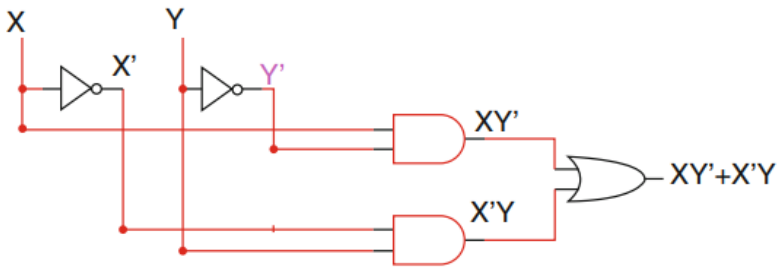
- Temukan fungsi keluaran dari rangkaian digital yang diberikan.
- Rancang rangkaian logika kombinasional menggunakan deskripsi masalah.
- Pelajari pengoperasian decoder dan aplikasinya.
- Pelajari aplikasi encoder.
- Merancang dan mempelajari fungsi multiplexer.
- Kembangkan half adder, full adder dari gerbang logika.
- Gunakan penambah penuh untuk merancang penambah dan pengurang biner.
- Pelajari cara mendesain ALU (unit logika aritmatika).
- Gunakan dekoder BCD ke tujuh segmen untuk menampilkan angka dalam desimal.

4.1 PENDAHULUAN

Rangkaian digital diklasifikasikan sebagai logika kombinasional atau sekuensial. Logika kombinasional adalah sirkuit digital dengan input atau input digital dan output atau output digital, sirkuit digital ini melakukan fungsi tertentu, output logika kombinasional tergantung pada nilai input saat ini, dan ini adalah sirkuit tanpa memori, tetapi logika sekuensial mengandung elemen memori.



Gambar 4.1 Blok diagram logika kombinasional



Gambar 4.2 Rangkaian logika kombinasional

Gambar 4.1 menunjukkan diagram blok logika kombinasi nasional dengan input dan output; dalam logika kombinasional, output adalah fungsi dari input. Perhatikan F(X,Y) = XY' = $X'Y$ yang diberikan oleh Gambar 4.2; rangkaian logika terbuat dari gerbang NOT, AND, dan OR; output logika kombinasional akan berubah dengan mengubah input. Tabel 4.1 menunjukkan tabel kebenaran dari Gambar 4.2.

Tabel 4.1 Tabel Kebenaran Gambar 4.2

X	Y	XY′	X′Y	F
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

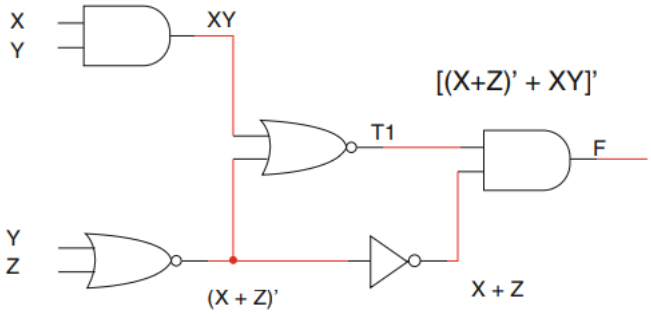
4.2 ANALISIS LOGIKA KOMBINASI

Tujuan dari analisis logika kombinasional adalah untuk menemukan fungsi keluaran dan tabel kebenaran dari rangkaian logika kombinasional, Gambar 4.3 menunjukkan logika kombinasional, output logika kombinasional diberikan oleh fungsi F, dan Tabel 4.2 menunjukkan tabel kebenaran dari fungsi F.

$$T1 = \left[(X+Z)' + XY \right]' = (XY)'\,(X+Z)(XY)' = (X+Z)\,(X'+Y')$$
$$= XY' + X'Z + Y'Z$$

$$F\,(X,Y,Z) = (X+Z)\,T1 = (X+Z)\,(XY' + X'Z + Y'Z)$$
$$= XY' + XY'Z + XY'Z + X'Z + Y'Z$$

$$F\,(X,Y,Z) = XY' + XY'Z + X'Z + Y'Z = XY' + Y'Z(X+1) + X'Z$$
$$= XY' + Y'Z + X'Z$$



Gambar 4.3 Logika kombinasional

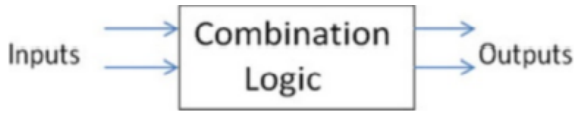
Tabel 4.2 Tabel Kebenaran Gambar 4.3

X Y Z	XY′	Y′Z	XZ′	F
0 0 0	0	0	0	0
0 0 1	0	1	0	1
0 1 0	0	0	0	0
0 1 1	0	0	0	0
1 0 0	1	0	1	1
1 0 1	1	1	0	1
1 1 0	0	0	1	1
1 1 1	0	0	0	0

Desain Logika Kombinasi

Gambar 4.4 menunjukkan diagram blok logika kombinasional; Langkah-langkah berikut menunjukkan bagaimana merancang logika kombinasional:

1. Pernyataan masalah yang menggambarkan fungsi logika kombinasional.
2. Tentukan jumlah input dan output atau mungkin diberikan oleh pernyataan masalah.
3. Tetapkan variabel ke input dan output.
4. Kembangkan tabel kebenaran dengan menuliskan semua kombinasi untuk input. Output akan ditentukan oleh pernyataan masalah.
5. Tulis fungsi output menggunakan K-map.
6. Gambarlah rangkaian logika.



Gambar 4.4 Blok diagram logika kombinasional

Contoh 4.1 Rancang rangkaian logika kombinasional dengan tiga masukan dan satu keluaran; output adalah satu ketika nilai biner dari input lebih besar dari atau sama dengan tiga; jika tidak, output akan menjadi nol.

Solusi Gambar 4.4 menunjukkan diagram blok logika kombinasional dengan tiga input dan satu output: variabel X, Y, dan Z ditetapkan ke input, dan variabel F ditetapkan ke output. Tabel 4.3 menunjukkan tabel kebenaran untuk masalah (Gambar. 4.5).

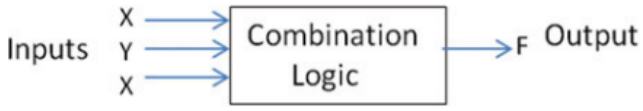
Pada Tabel 4.3, semua kombinasi input untuk X, Y, dan Z terdaftar; maka menurut rumusan masalah, keluaran F adalah satu bila masukannya tiga atau lebih; jika tidak, outputnya nol. Fungsi keluaran F dapat direpresentasikan dengan jumlah minterms:

$$F(X, Y, Z) = m_3 + m_4 + m_5 + m_6 + m_7$$

(jumlah minterms)

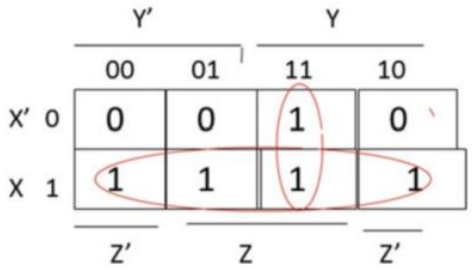
Tabel 4.3 Tabel kebenaran contoh 1

X Y Z	F
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	1
1 0 1	1
1 1 0	1
1 1 1	1

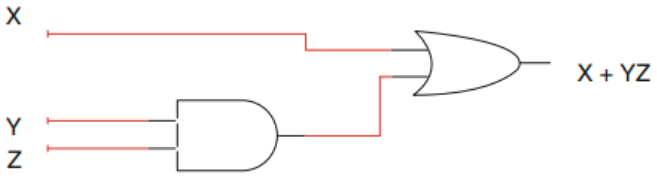


Gambar 4.5 Diagram blok logika kombinasional contoh 4

Dengan mentransfer minterm dari F(X,Y,Z) ke dalam K-map seperti yang ditunjukkan pada Gambar. 4.6 dan membaca fungsi yang disederhanakan dari hasil K-map F (X,Y,Z) X YZ. Gambar 4.7 menunjukkan rangkaian logika untuk fungsi F (Gambar. 4.7).



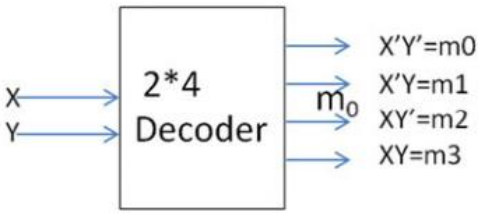
Gambar 4.6 K-peta misalnya 4



Gambar 4.7 Diagram rangkaian misalnya 4

4.3 DEKODER DAN ENKODER

Decoder adalah logika MSI yang menghasilkan minterm dari sekumpulan input; kedua variabel X dan Y menghasilkan empat minterm, yaitu $X'Y'$ = m_0, $X'Y$ = m_1, XY' = m_2, dan XY = m_3. Gambar 4.8 menunjukkan diagram blok dari decoder 2*4 (2 input dan 4 output), dan Tabel 4.4 menunjukkan tabel kebenaran untuk decoder 2*4. Dari tabel kebenaran decoder, fungsi-fungsi berikut adalah output dari decoder:



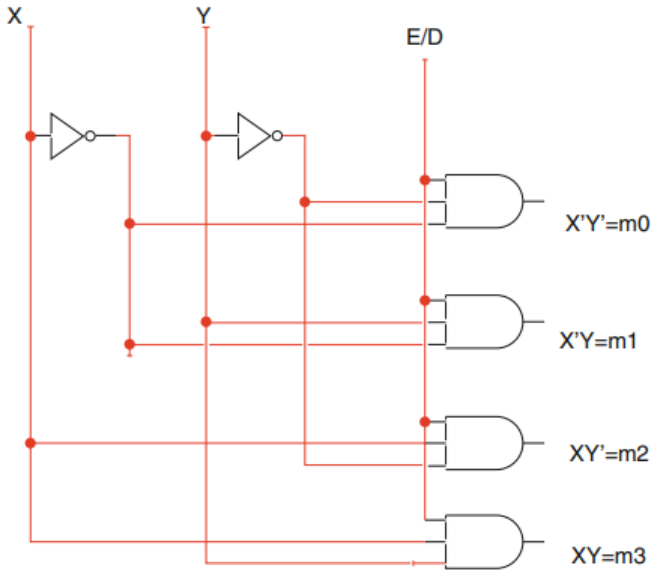
Gambar 4.8 Diagram blok dekoder 2*4

Tabel 4.4 Tabel kebenaran untuk dekoder 2*4

X Y	m_0	m_1	m_2	m_3
0 0	1	0	0	0
0 1	0	1	0	0
1 0	0	0	1	0
1 1	0	0	0	1

$$m_0 = X'Y', m_1 = X'Y, m_2 = XY', \text{and } m_3 = XY$$

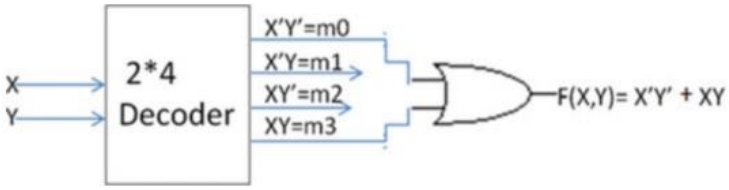
Gambar 4.9 menunjukkan rangkaian logika dekoder 2*4. Kebanyakan IC MSI memiliki input tambahan yang disebut enable/disable (E/D); fungsi input E/D adalah untuk mengaktifkan atau menonaktifkan IC seperti yang ditunjukkan pada Gambar 4.9. Ketika E/D = 0, semua output dari decoder akan menjadi nol (artinya decoder dinonaktifkan); decoder hanya menghasilkan minterms ketika E/D disetel ke satu.



Gambar. 4.9 Rangkaian logika dekoder 2*4 dengan E/D

Menerapkan Fungsi Menggunakan Decoder

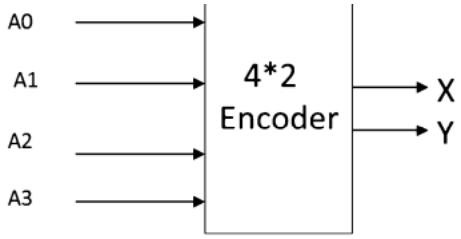
Decoder dapat digunakan untuk merancang sirkuit kombinasional. Pertimbangkan F(X, Y) = XY + $X'Y'$ yang dapat diimplementasikan menggunakan dekoder. Fungsi tersebut dapat direpresentasikan dengan F(X,Y) = m3 + m_0. Fungsi tersebut berisi dua variabel yang merupakan input ke decoder; oleh karena itu, decoder 2*4 diperlukan, dan output F adalah jumlah minterm m3 dan m0 yang ditunjukkan pada Gambar 4.10.



Gambar 4.10 Mengimplementasikan fungsi menggunakan decoder

Enkoder

Encoder adalah kebalikan dari decoder; memiliki 2^n input dan n output, untuk n = 2 berarti encoder memiliki 2^2 = 4 input dan 2 output; outputnya adalah nilai biner dari input yang dipilih. Gambar 4.11 menunjukkan diagram blok encoder 4*2, dan Tabel 4.5 menunjukkan tabel kebenaran untuk encoder 4*2. Pada Tabel 4.5, jika A3 = 1, maka output XY = 11, atau jika A2 = 1, maka output XY = 10. Gambar 4.12 menunjukkan K-map untuk fungsi X dan Y; kombinasi input yang tidak tercantum dalam tabel kebenaran dekoder 2*4 tidak peduli, ditandai dengan "X", di kedua K-map. Gambar 4.13 menunjukkan diagram rangkaian logika dari dekoder 2*4.

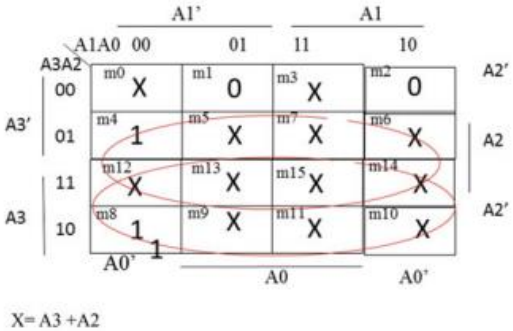


Gambar 4.11 Diagram blok dari encoder 4*2

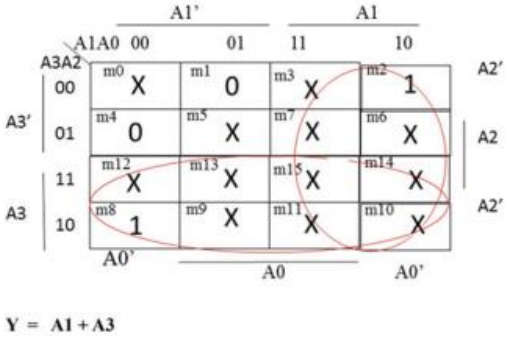
Tabel 4.5 Tabel kebenaran untuk encoder 4*2

A3	A2	A1	A0	X	Y
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

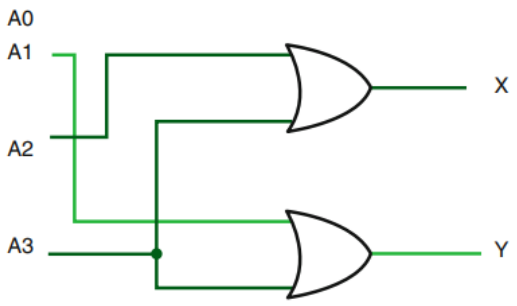
X



Y



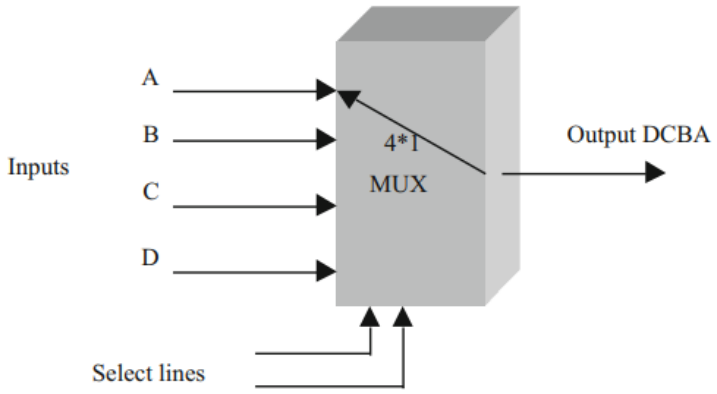
Gambar 4.12 K-maps untuk fungsi X dan Y



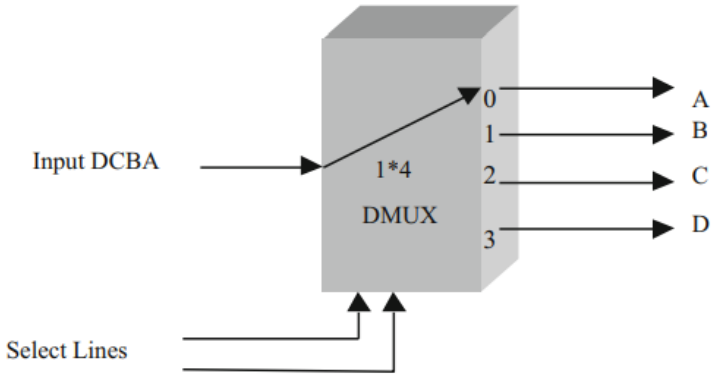
Gambar 4.13 Diagram rangkaian logika encoder 4*2

4.4 MULTIPLEKSER (MUX)

MUX adalah rangkaian logika kombinasional dengan input N dan satu output; fungsi MUX adalah untuk memilih salah satu input dari banyak input dan mengarahkan input ke output. Gambar 4.14 menunjukkan arsitektur dasar multiplexer. Multiplexer yang memiliki N input dan satu output disebut multiplexer N-ke-1. Saklar internal memilih satu jalur input pada satu waktu dan mentransfer input itu ke output. Ketika sakelar berada di posisi A, ia mentransfer input A ke output; ketika sakelar bergerak ke posisi B, ia mentransfer input B ke output. Metode ini berlanjut sampai sakelar bergerak ke posisi D dan mentransfer input D ke output.



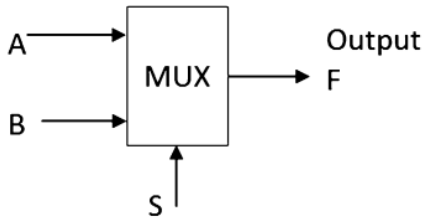
Gambar 4.14 Multiplexer 4-ke-1



Gambar 4.15 Demultiplexer 1-ke-4 (DMUX)

Kebalikan dari multiplexer adalah demultiplexer (DMUX), ditunjukkan pada Gambar 4.15. Saklar bergerak untuk mengirim setiap input ke output yang sesuai. DMUX memiliki satu input dan N output – ini disebut demultiplexer 1-ke-N. Ketika sakelar berada di posisi 0, ia mentransfer A ke port output 0 dan kemudian pindah ke port output 1 dan mentransfer B ke port ini. Proses ini berlanjut hingga switch pindah ke port output 3 dan mentransfer D ke port 3.

Gambar 4.16 menunjukkan MUX 2*1 di mana A dan B adalah input dan S adalah garis pilih; ketika S = 0, keluaran multiplexer adalah nilai A; ketika S = 1, output MUX adalah nilai B. Tabel 4.6 menunjukkan tabel kebenaran MUX untuk Gambar 4.16.



Gambar 4.16 Diagram blok 2*1 MX

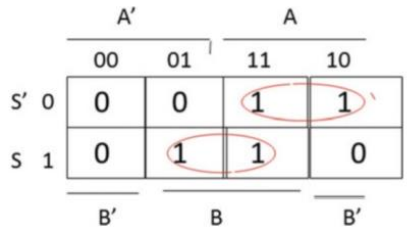
Tabel 4.6 Tabel Kebenaran 2*1 MUX

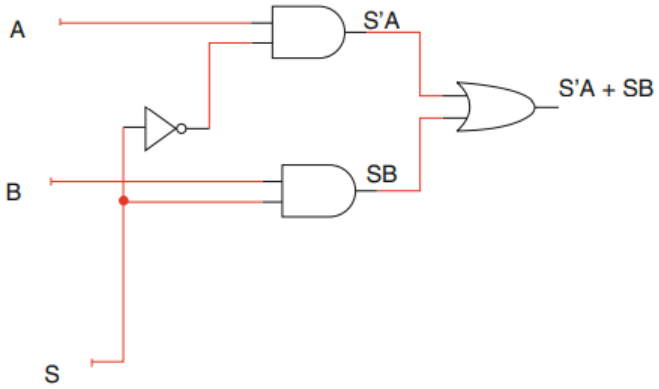
S	A	B	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Membaca hasil K-map F(S,A,B) = $S'A$ + SB. Gambar 4.17 menunjukkan rangkaian logika untuk 2*1 MUX. Gambar 4.18 menunjukkan MUX 4*1 di mana I_0, I_1, I_2, dan I_3 adalah input, Y adalah output, dan S_0 dan S_1 adalah jalur pilih. Tabel 4.11 menunjukkan pengoperasian MUX. Fungsi Y dapat dibangkitkan dari Tabel 4.7; ketika input S_0S_1 = 00, maka Y = I_0; ketika S_0S_1 = 01, maka Y I_1; ketika S_0S_1 = 10, maka Y = I_2; Kapan S_0S_1 = 11, lalu Y = I_3; oleh karena itu, keluaran Y adalah

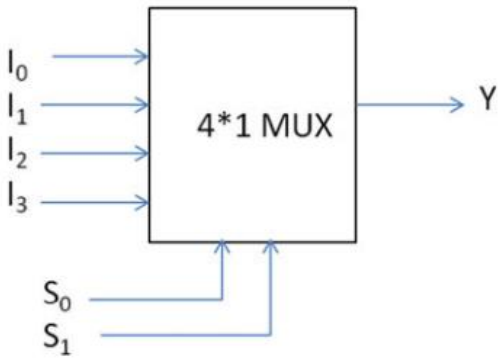
$$Y = S_0' S_1' I_0 + S_0' S_1 I_1 + S_0 S_1' I_2 + S_0 S_1 I_0$$

$$F(S, A, B) = m_3 + m_5 + m_7$$





Gambar 4.17 Rangkaian logika untuk 2*1 MUX

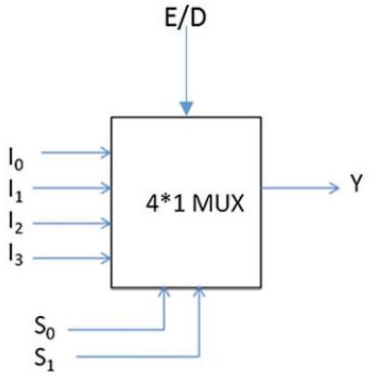


Gambar 4.18 Diagram blok 4*1 MUX

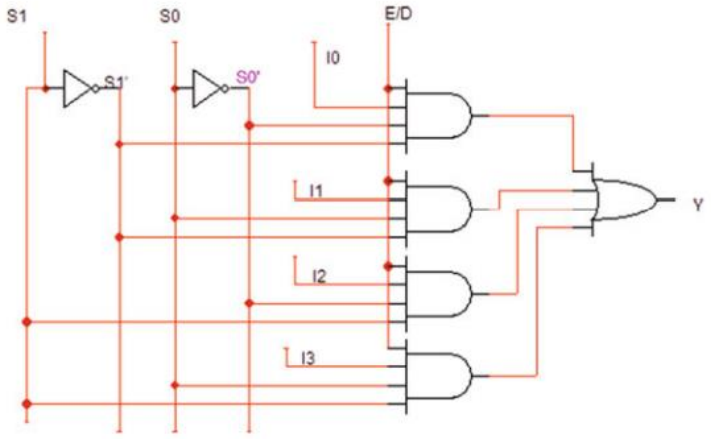
Tabel 4.7 Operasi MUX

S_0	S_1	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Gambar 4.19 menunjukkan diagram blok untuk MUX 4*1 dengan E/D. Gambar 4.20 menunjukkan rangkaian logika 4*1 MUX; seperti yang ditunjukkan pada gambar ini, input E/D ditambahkan ke diagram logika MUX; ketika E/D = 0, output Y = 0, dan MUX dinonaktifkan.



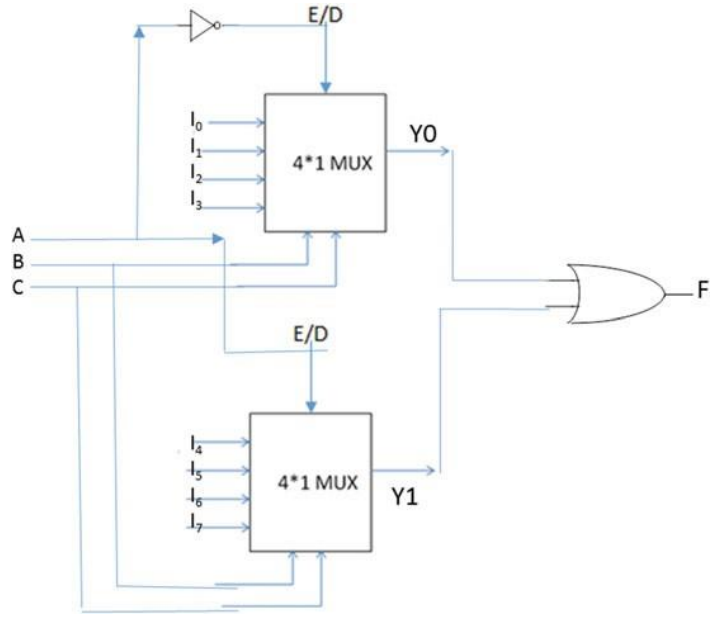
Gambar 4.19 Menunjukkan diagram blok 4*1 MUX dengan E/D



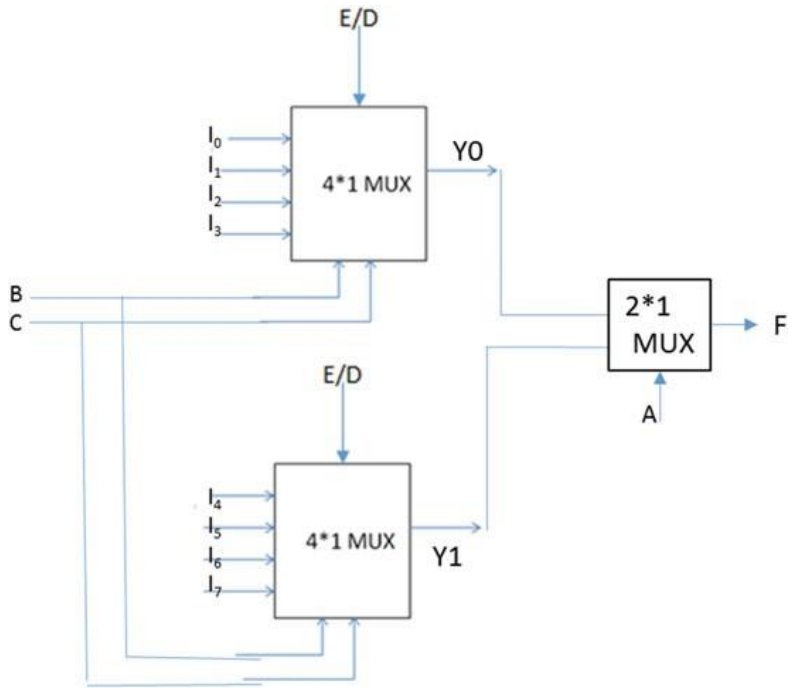
Gambar 4.20 Diagram logika rangkaian untuk 4*1 MUX

Merancang Multiplexer Besar Menggunakan Multiplexer Kecil

Dengan menggabungkan MUX kecil, maka mampu menghasilkan MUX yang lebih besar; sebuah MUX 8*1 dapat dibangun oleh dua multiplexer 4*1 dan satu gerbang OR seperti yang ditunjukkan pada Gambar 4.21. Pada gambar ini, A, B, dan C adalah garis pilih. Saat A = 0, MUX atas diaktifkan, dan saat A = 1, MUX bawah diaktifkan. Juga, 8*1 MUX dapat diimplementasikan dengan menggunakan dua 4*1 dan satu MUX 2*1 seperti yang ditunjukkan pada Gambar 4.22. Pada Gambar 4.22, ketika A= 0, output F = Y0, dan ketika A = 1, output F = Y1.



Gambar 4.21 8*1 MUX menggunakan 4*1 MUX dan gerbang OR



Gambar. 4.22 8*1 MUX dibangun dengan dua MUX 4*1 dan 2*1 MUX

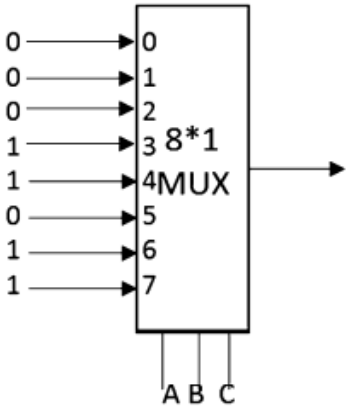
Menerapkan Fungsi Menggunakan Multiplexer

(a) Menerapkan tabel kebenaran tiga variabel menggunakan 8*1 MUX

MUX dapat digunakan untuk mengimplementasikan fungsi digital; pertimbangkan kebenaran Tabel 4.8; tabel kebenaran terbuat dari tiga variabel, dan dapat menggunakan 8*1 MUX untuk menanamkan tabel kebenaran (Tabel 4.8). Tabel kebenaran dibuat dari tiga variabel; oleh karena itu, MUX 8*1 diperlukan; variabel A, B, dan C terhubung ke garis pilih MUX, dan input MUX sesuai dengan output tabel kebenaran (Gambar. 4.23).

Tabel 4.8 Tabel kebenaran MUX dengan tiga variabel

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

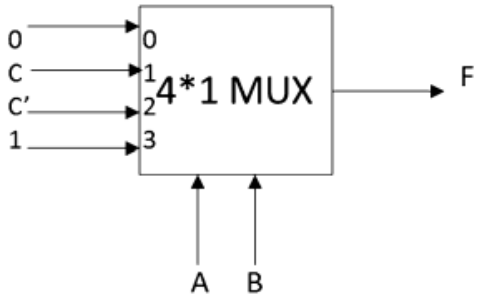


Gambar 4.23 Diagram blok MUX 8*1

(b) Menerapkan Tabel 4.8 menggunakan MUX 4*1 Jumlah variabel – 1 = jumlah baris terpilih. Tiga variabel – 1 = 2 jumlah baris pilihan untuk MUX. Jika A dan B dihubungkan untuk memilih jalur, ketika AB = 00, ada dua baris pada Tabel 4.9 dengan AB = 00, maka keluaran F bergantung pada nilai C; dalam hal ini C = 0. Pertimbangkan baris dengan AB = 01, F = 0 untuk C = 0, dan F = 1 untuk C = 1; oleh karena itu, F = C. Gambar 4.24 menunjukkan implementasi Tabel 4.9 menggunakan 4*1 MUX.

Tabel 4.9 Tabel Kebenaran untuk Gambar 4.24

A	B	C	F	
0	0	0	0	0
0	0	1	0	
0	1	0	0	C
0	1	1	1	
1	0	0	1	C′
1	0	1	0	
1	1	0	1	1
1	1	1	1	



Gambar 4.24 Multiplexer untuk Tabel 4.9

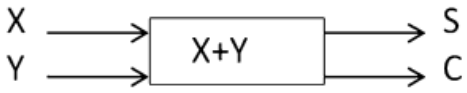
4.5 HALF ADDER, FULL ADDER, BINARY ADDER, DAN PENGURANG

Half adder (HA) adalah rangkaian logika yang menjumlahkan bit X dan Y; Gambar 4.25 menunjukkan diagram blok HA; input ke HA adalah X dan Y; output dari HA adalah S (sum) dan

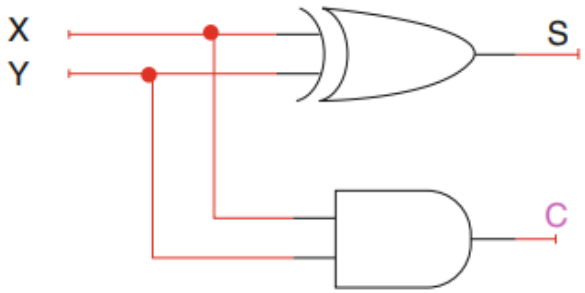
C (carry). Tabel 4.10 menunjukkan tabel kebenaran untuk HA. Pada tabel kebenaran ini, ketika X Y = 1, maka X ditambah Y = 10 menghasilkan S = 0 dan C = 1.
Fungsi S dan C adalah

$$S = m_1 + m_2 = X'Y + XY' = X \text{ XOR } Y$$
$$C = XY = X \text{ AND } Y$$

Gambar 4.26 menunjukkan rangkaian logika half adder (HA).



Gambar 4.25 Diagram blok dari setengah penambah



Gambar 4.26 Rangkaian logika untuk HA

Full Adder (FA) Gambar 4.27 menunjukkan diagram blok full adder (FA). FA akan menambahkan X + Y + Cin (masing-masing hanya 1 bit), output dari FA disebut S dan Cout, dan Tabel 4.11 menunjukkan tabel kebenaran FA; dalam tabel ini, X, Y, dan Cin ditambahkan, dan hasilnya menghasilkan jumlah (S) dan carry (Cout).

Fungsi S dapat diwakili oleh jumlah minterms seperti yang ditunjukkan pada Persamaan. 4.1:

$$S(X, Y, Cin) = X'Y'Cin + X'YCin' + X'YCin + XY'Cin' + XYcin \qquad (4.1)$$

ATAU

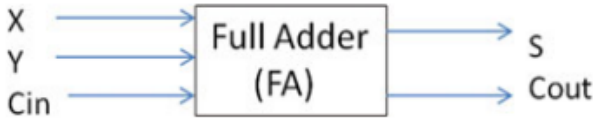
$$S(X, Y, Cin) = Cin\ (X'Y' + XY) + Cin'\ (X'Y + X'Y) \qquad (4.2)$$
$$X'Y + X'Y = X \text{ XOR } Y = A$$
$$X'Y' + XY = (X'Y + X'Y)' = A'$$

Oleh karena itu, Persamaan. 4.2 dapat ditulis dalam bentuk Persamaan. 4.3:

$$S(X, Y, Cin) = CinA' + Cin'A = Cin \text{ XOR } A \qquad (4.3)$$

Tabel 4.10 Tabel kebenaran untuk setengah penjumlah

X Y	C	S
0 0	0	0
0 1	0	1
1 0	0	1
1 1	1	0



Gambar 4.27 Diagram blok dari setengah penambah

Tabel 4.11 Tabel kebenaran untuk setengah penjumlah

X Y Cin	Cout	s
0 0 0	0	0
0 0 1	0	1
0 1 0	0	1
0 1 1	1	0
1 0 0	0	1
1 0 1	1	0
1 1 0	1	0
1 1 1	1	1

Cout dapat ditulis dalam bentuk jumlah minterms:

$$\text{Cout}\ (\text{X}, \text{Y}, \text{Cin}) = \text{X}'\text{YCin}, + \text{XY}'\text{Cin} + \text{XYCin}' + \text{XYCin}$$

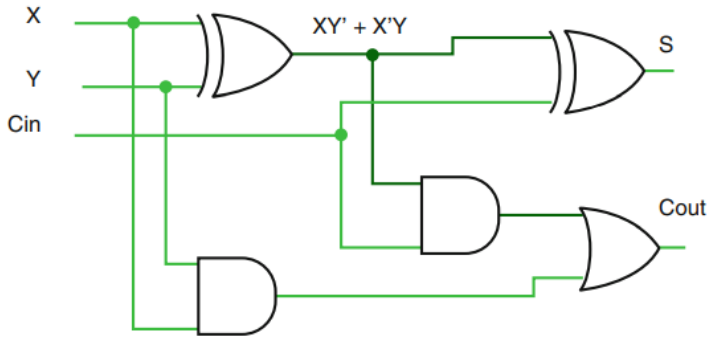
ATAU

$$\text{Cout}\ (\text{X}, \text{Y}, \text{Cin}) = \text{Cin}\ (\text{X}'\text{Y}, + \text{XY}') + \text{XY}\ (\text{Cin}' + \text{Cin})$$

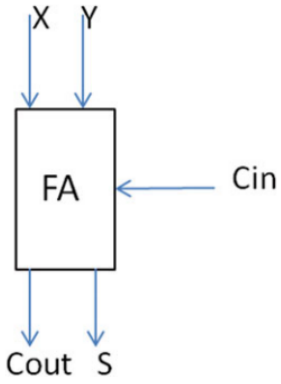
ATAU

$$\text{Cout}\ (\text{X}, \text{Y}, \text{Cin}) = \text{Cin}\ (\text{X}'\text{Y}, + \text{XY}') + \text{XY}$$

Gambar 4.28 menunjukkan rangkaian logika untuk penambah penuh.
Gambar 4.29 menunjukkan diagram blok dari full adder (FA), di mana X, Y, dan Cin adalah inputnya dan S dan Cout adalah outputnya.



Gambar 4.28 Rangkaian logika penambah penuh



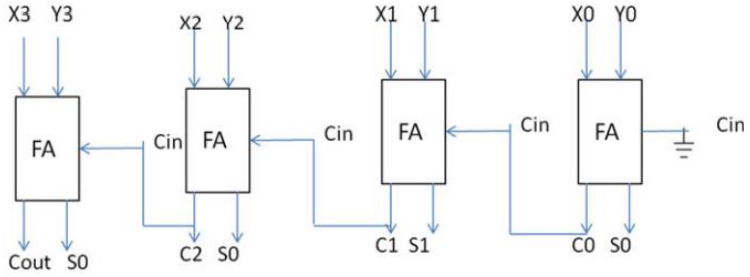
Gambar 4.29 Blok diagram FA

Penambah Biner 4-Bit

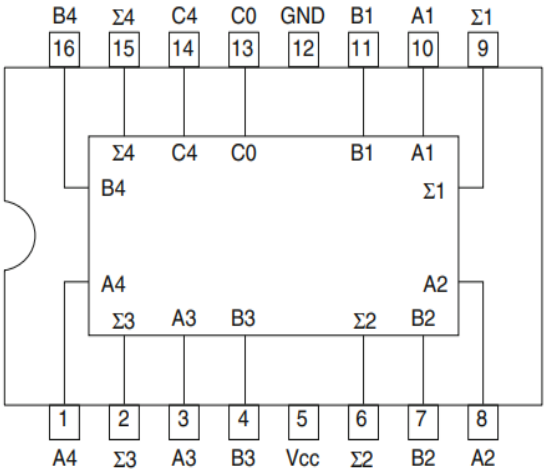
Fungsi dari 4-bit binary adder adalah menjumlahkan dua bilangan 4-bit seperti:

$$X_3X_2X_1X_0 + Y_3Y_2Y_1Y_0$$

Saat menambahkan X_0 dengan Y_0, itu menghasilkan jumlah (S0) dan carry (C0); C0 kemudian ditambahkan ke X_1 dan Y1 yang menghasilkan S1 dan C1. Gambar 4.30 menunjukkan penambah biner 4-bit; Cin terhubung ke ground untuk mewakili nol. Diagram pabrikan dari penambah biner 4-bit ditunjukkan pada Gambar 4.31 sebagai satu IC, dan nomor IC adalah 7483.



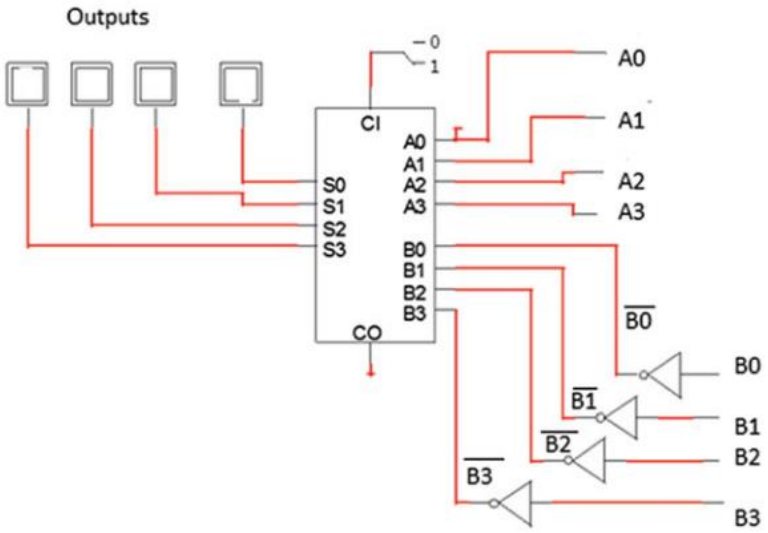
Gambar 4.30 Penjumlah biner 4-bit



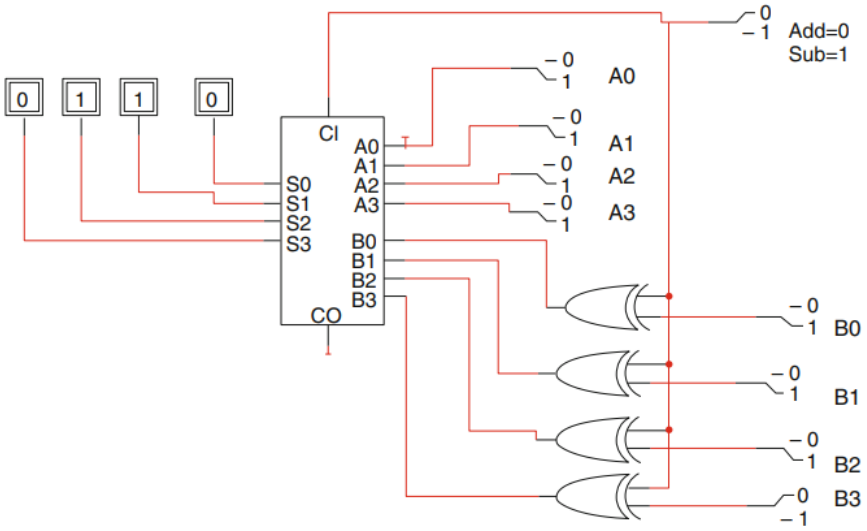
Gambar 4.31 7483 Penambah binar 4-bit

Pengurang

Sebuah subtraktor melakukan pengurangan dari A - B atau A + B' + 1. Gambar 4.32 menunjukkan diagram sebuah subtractor menggunakan sebuah penambah biner 4-bit. CI diatur ke satu, dan input B_0, B_1, B_2, dan B_3 dilengkapi. Gambar 4.33 adalah modifikasi dari Gambar 4.32 yang dapat melakukan penjumlahan dan pengurangan. CI adalah carry in dan CO adalah carry out dari penambah biner 4-bit. Dengan menyetel sakelar Tambah/Sub ke nol, ia melakukan penambahan, dan dengan menyetel sakelar Tambah/Sub ke satu, ia melakukan pengurangan.



Gambar 4.32 Pengurang 4 bit

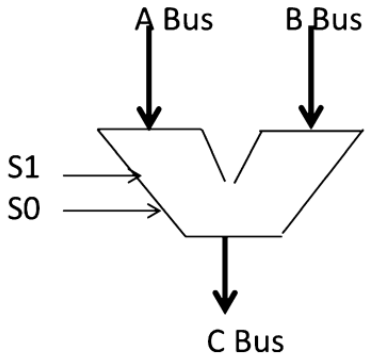


Gambar 4.33 Rangkaian logika untuk penambah dan pengurang 4-bit

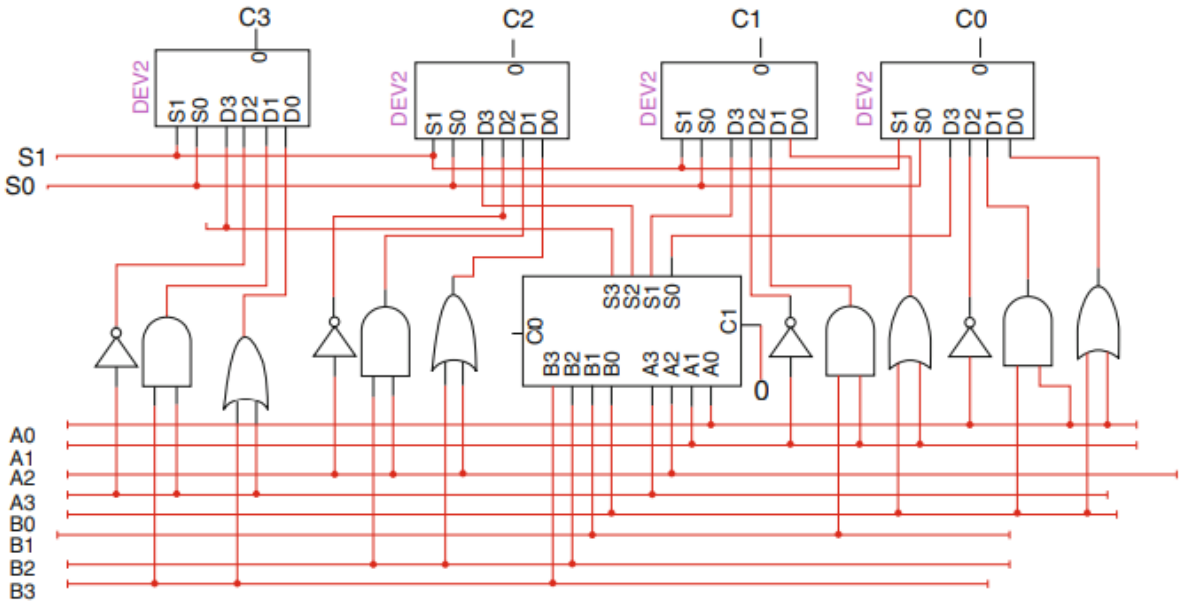
4.6 ARITHMETIC LOGIC UNIT (ALU)

Fungsi dari unit logika aritmatika (ALU) adalah untuk melakukan operasi aritmatika seperti penambahan dan pengurangan dan operasi logika bit-bijaksana seperti AND, OR, dan NOT. Gambar 4.34 menunjukkan diagram blok dari ALU. Pada Gambar 4.34, bus A dan B adalah input, dan bus C adalah output dari ALU; S1 dan S0 adalah jalur pilih yang memilih

fungsi ALU; Tabel 4.12 menunjukkan fungsi dari sebuah ALU; asumsikan A dan B adalah 4 bit dan diwakili oleh A3, A2, A1, dan A0 dan B3, B2, B1, dan B0. Garis pilih ALU menentukan ukuran multiplexer, karena ada 2 garis pilih; oleh karena itu, ukuran MUX adalah 4*1 (4 input dan 1 output); jumlah bit menentukan jumlah multiplexer. A dan B adalah 4 bit; oleh karena itu, empat multiplexer 4*1 diperlukan. Gambar 4.35 menunjukkan diagram ALU.



Gambar 4.34 Diagram blok dari ALU



Gambar 4.35 Diagram rangkaian logika dari ALU

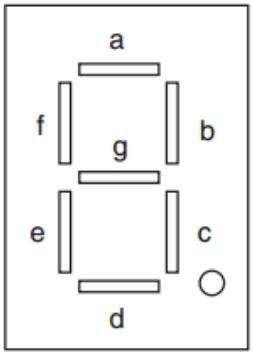
Tabel 4.12 Fungsi ALU

S1	S0	ALU
0	0	A OR B
0	1	A + B
1	0	A AND B
1	1	A′

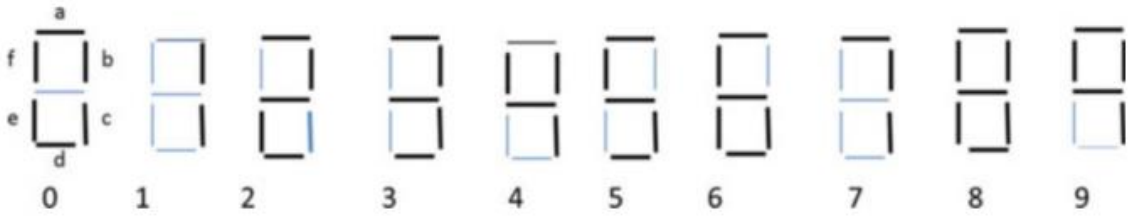
4.7 TAMPILAN TUJUH SEGMEN

Tampilan tujuh segmen terbuat dari tujuh LED (light-emitting diode) seperti yang ditunjukkan pada Gambar 4.36; tampilan tujuh segmen dapat menampilkan satu digit dari nol hingga sembilan; Gambar 4.37 menunjukkan segmen yang harus aktif untuk menampilkan

angka dari 0 sampai 9. Untuk menampilkan 0, semua segmen harus aktif kecuali untuk g; untuk menampilkan 8, semua segmen harus aktif. Ini membutuhkan dekoder khusus yang disebut dekoder BCD ke tujuh segmen untuk mengubah desimal berkode biner (BCD) menjadi tampilan tujuh segmen.

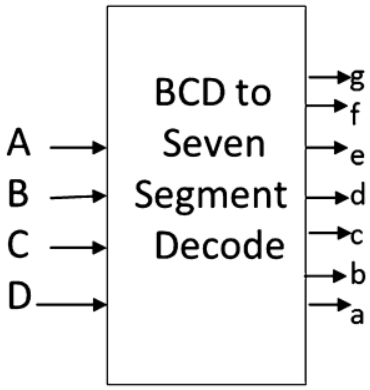


Gambar 4.36 Tampilan tujuh segmen



Gambar 4.37 Tampilan tujuh segmen dari 0 hingga 9

Gambar 4.38 menunjukkan diagram blok dekoder BCD ke tujuh segmen, dan Tabel 4.13 menunjukkan tabel kebenaran dekoder BCD ke tujuh segmen. Decoder memiliki 4 input dan 7 output; masukan ke dekoder adalah BCD (desimal berkode biner yaitu dari 0000 sampai 1001), seperti yang ditunjukkan pada Tabel 4.13; jika nilai input lebih besar dari 1001, maka output tidak peduli.



Gambar 4.38 Diagram blok dari dekoder BCD ke tujuh segmen

Ada tujuh output, dan setiap output membutuhkan K-map untuk menemukan fungsi output. Gambar 4.39 menunjukkan K-map untuk output a.

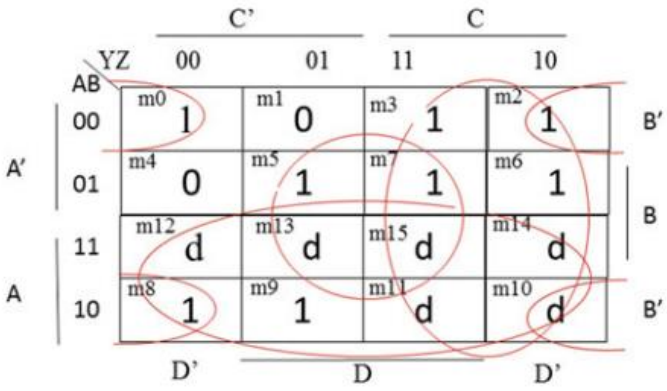
Tabel 4.13 Tabel kebenaran untuk dekoder BCD ke tujuh segmen

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	1	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	d	d	d	d	d	d	d
1	0	1	1	d	d	d	d	d	d	d
1	1	0	0	d	d	d	d	d	d	d
1	1	0	1	d	d	d	d	d	d	d
1	1	1	0	d	d	d	d	d	d	d
1	1	1	1	d	d	d	d	d	d	d

Dengan membaca K-map pada Gambar 4.39, hasilnya adalah fungsi a:

$$a = B'D' + A + C + BD$$

Dengan menggunakan prosedur di atas, seseorang dapat menemukan fungsi output lainnya.



Gambar 4.39 K-map untuk keluaran a

4.8 RINGKASAN

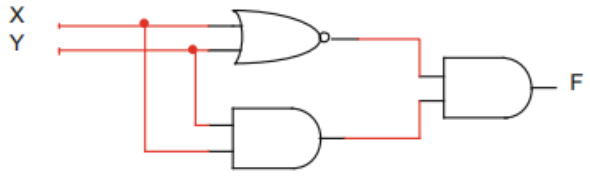
- Rangkaian kombinasional adalah rangkaian digital dengan satu atau lebih input digital dan output digital.
- Keluaran dari rangkaian kombinasional bergantung pada nilai arus masukan.
- Rangkaian logika kombinasional dapat direpresentasikan dengan fungsi atau tabel kebenaran.
- Decoder adalah logika kombinasional dengan n input dan 2n output; jika n 2 maka decoder adalah 2*4.
- Decoder menghasilkan minterms input; decoder dengan dua input menghasilkan empat minterms.

- Multiplexer adalah rangkaian kombinasional dengan 2^n input dan satu output, di mana n adalah jumlah jalur yang dipilih. Jika n = 2, maka multiplexer akan menjadi 4*1.
- Fungsi half adder (HA) adalah untuk menjumlahkan 2 bit dan menghasilkan sum dan carry.
- Fungsi full adder (FA) adalah untuk menambahkan 3 bit.
- Fungsi penambah biner 4-bit adalah menjumlahkan dua bilangan 4-bit.
- Unit logika aritmatika (ALU) adalah logika kombinasional yang melakukan operasi aritmatika dan operasi logika.
- Tampilan tujuh segmen digunakan untuk menampilkan satu digit angka desimal.
- Dekoder BCD ke tujuh segmen mengubah BCD 4-bit menjadi 7 bit untuk antarmuka ke tampilan 7-segmen.
- Bab 5 mencakup rangkaian logika sekuensial, dan topik yang disajikan dalam bab ini adalah D, J-K, dan T flip-flop, register, register geser, diagram status, tabel status, dan perancangan pencacah.

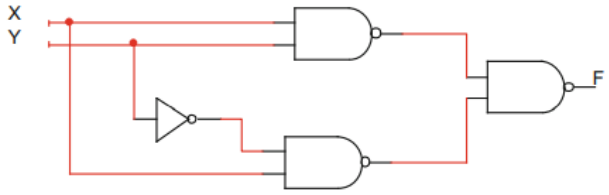
Masalah

1. Diagram logika rangkaian kombinasional berikut diberikan; temukan fungsi keluaran dan tabel kebenaran untuk setiap fungsi.

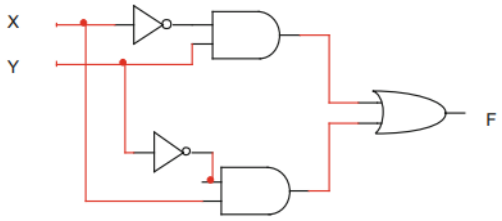
   (a)



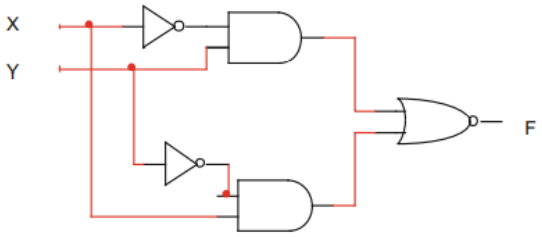
   (b)



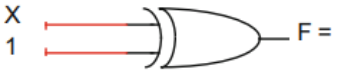
   (c)



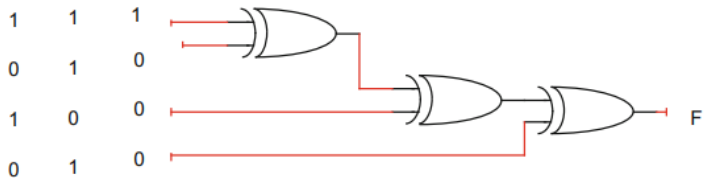
(d)



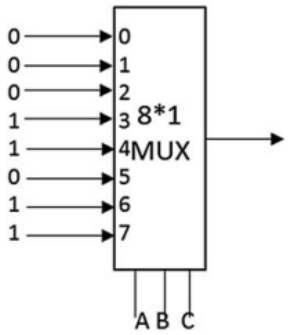
2. Tentukan keluaran dari gerbang berikut.



3. Temukan output F untuk setiap set input.



4. Rancang rangkaian logika dengan tiga input dan satu output; output menghasilkan bit paritas genap dari input.
   (a) Tunjukkan tabel kebenarannya.
   (b) Temukan fungsi keluaran.
   (c) Gambarlah rangkaian logika.
5. Implementasikan fungsi F(X,Y,Z) = $XY' + XZ'$ menggunakan:
   (a) Dekoder
   (b) Multiplekser
6. Implementasikan fungsi berikut hanya dengan menggunakan satu dekoder dan gerbang eksternal:
   F1(X,Y,Z) (0,3,4)
   F2(X,Y,Z) (2,3,5)
7. Implementasikan full adder menggunakan decoder.
8. Multiplexer berikut diberikan; melengkapi tabelnya.



A	B	C	F
0	0	0	
0	1	1	
0	1	0	
1	1	1	
1	0	0	
1	0	1	
1	1	0	
0	0	1	

9. Implementasikan fungsi F(W,X,Y,Z) = $\sum(0,1,3,4,7,8,9,11,12,15)$ menggunakan MUX.
10. Rancang sebuah penambah biner 8-bit menggunakan penambah biner 4-bit.
11. Rancang penambah biner 16-bit menggunakan penambah biner 4-bit.
12. Merancang logika kombinasional dengan tiga input dan tiga output; jika input 0, 1, 2, atau 3 maka output 3 lebih banyak dari input, jika input 4, 5, 6, atau 7 maka output 3 lebih kecil dari input.
13. Kereta api dengan 7 gerbong mengangkut penumpang dan bernomor 1 sampai 7; setiap gerbong berisi sakelar biner untuk keadaan darurat, dan ketika salah satu sakelar dihidupkan, maka tampilkan nomor gerbong di kabin konduktor dalam desimal. Verifikasi desain Anda menggunakan Logisim.
14. Rancang rangkaian kombinasional dengan empat input dan satu output; input ke rangkaian kombinasional adalah BCD, dan output menghasilkan paritas genap untuk input.
15. Desain MUX 16*1 menggunakan 4 *1 MUX.
16. Rancang ALU 4-bit untuk melakukan fungsi-fungsi berikut:

$$A + B,\ A - B,\ A + 1,\ A',\ B',\ A \text{ OR } B,\ A \text{ XOR } B,\ A \text{ AND } B$$

17. Rancanglah logika kombinasional yang membandingkan X dan Y, dimana X X1X0 dan Y Y1Y0; output dari logika kombinasional adalah 1, ketika X < Y; jika tidak, outputnya adalah 0.
    (a) Tunjukkan tabel kebenaran.
    (b) Cari fungsi keluaran menggunakan K-map.

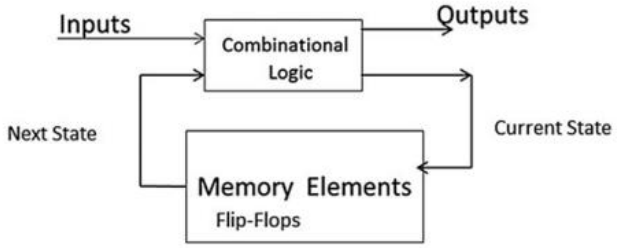
BAB 5
LOGIKA SEKUENSIAL SINKRON

Tujuan: Setelah menyelesaikan bab ini, Anda diharapkan dapat:

- Menganalisis logika sekuensial.
- Pelajari pengoperasian kait S-R.
- Desain flip-flop D dari kait S-R.
- Pelajari aplikasi flip-flop D.
- Pelajari pengoperasian flip-flop J-K dan T.
- Rancang register dan register geser menggunakan D flip-flop.
- Kembangkan tabel status untuk rangkaian sekuensial.
- Mengembangkan diagram keadaan dari tabel keadaan.
- Mengembangkan tabel eksitasi untuk setiap jenis flip-flop.
- Desain penghitung digital.

5.1 PENDAHULUAN

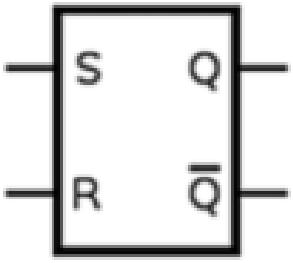
Rangkaian logika sekuensial berisi elemen memori, dan outputnya bergantung pada nilai input saat ini dan kondisi level input sebelumnya. Gambar 5.1 menunjukkan diagram blok logika sekuensial seperti yang ditunjukkan pada gambar ini; output bergantung pada input dan status elemen memori saat ini; dalam gambar ini, output dari logika kombinasional adalah input ke elemen memori, dan output dari elemen memori adalah input ke logika kombinasional; elemen dasar elemen memori adalah flip-flop yang dapat menyimpan nilai biner selama perangkat diberi daya. Output dari logika sekuensial sinkron tergantung pada output elemen memori dan input. Aplikasi logika sekuensial sinkron adalah merancang register, counter, dan memori. Logika sekuensial sinkron beroperasi dengan pulsa clock.



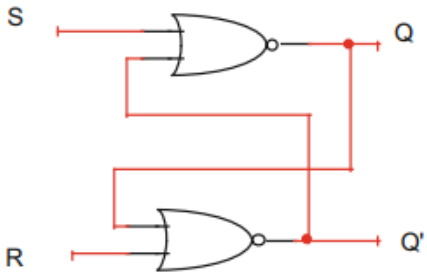
Gambar 5.1 Blok diagram logika sekuensial sinkron

5.2 S-R LATCH

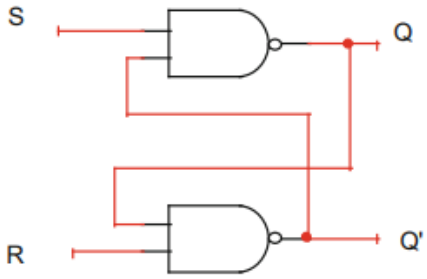
S-R latch (Kait S-R) adalah jenis memori dengan dua input S (set) dan R (reset), dua output Q, dan Q0, dan outputnya saling melengkapi. Gambar 5.2 menunjukkan diagram blok gerendel S-R. Kait S-R dapat dibuat dengan gerbang NOR atau NAND: Gambar 5.3 menunjukkan kait S-R menggunakan gerbang NOR dan Gambar 5.4 menunjukkan kait S-R menggunakan gerbang NAND.



Gambar 5.2 Blok diagram gerendel S-R



Gambar 5.3 Rangkaian logika gerendel SR menggunakan gerbang NOR



Gambar 5.4 Rangkaian logika gerendel SR menggunakan gerbang NAND

Tabel 5.1 Tabel karakteristik gerendel S-R

S	R	Q	Q^0
0	0	1	1 Dilarang
0	1	1	0
1	1	1	0 Tidak ada perubahan
1	0	0	1
1	1	0	1 Tidak ada perubahan

Operasi Latch S-R Pertimbangkan latch S-R pada Gambar 5.4 yang dibangun dengan gerbang NAND; langkah-langkah berikut menjelaskan pengoperasian gerendel S-R, dan Tabel 5.1 menunjukkan tabel karakteristiknya:

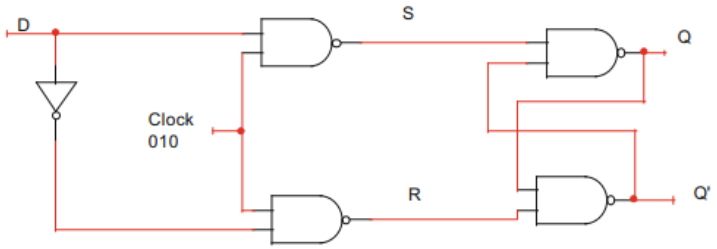
1. Dengan setting S = 0 dan R = 0, menghasilkan output Q = Q' = 1 yang tidak diperbolehkan karena Q dan Q = 0 harus saling melengkapi; oleh karena itu, S = R = 0 dilarang.

2. Dengan setting S = 0 dan R = 1, hasil Q = 1 dan Q' = 0 seperti terlihat pada Tabel 5.1, jika S berubah dari 0 menjadi 1 maka Q tidak berubah.
3. Dengan setting S = 1 dan R = 0, menghasilkan Q = 0 dan Q' = 1, jika R berubah dari 0 menjadi 1 maka Q tidak berubah. Dapat disimpulkan ketika S R 1 output Q tidak berubah (jika Q = 0 tetap 0 atau Q = 1 tetap 1). Latch S-R adalah rangkaian logika dasar untuk D, J-K, dan T flip-flop.

5.3 D FLIP-FLOP, J-K FLIP-FLOP, DAN T FLIP-FLOP

D Flip-Flop,

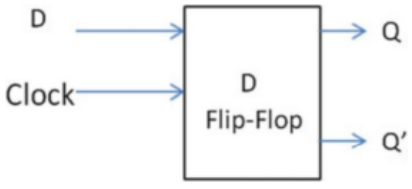
D flip-flop adalah memori 1-bit, dan digunakan untuk merancang SRAM (RAM Statis) dan mendaftar; Gambar 5.5 menunjukkan diagram logika D flip-flop. Input ke flip-flop adalah D dan clock. Pada saat clock 0, maka S = R = 1, dan sesuai Tabel 5.2 keluaran flip-flop tidak berubah, dengan setting D menjadi 0, dan merubah clock dari 0 menjadi 1 menghasilkan S = 0 dan R = 1 kemudian Q = 0 dan Q' = 1; ini berarti ketika input D 0 dan menerapkan jam, output Q berubah menjadi 0. Mengatur D ke 1 dan mengubah jam dari 0 ke 1 menghasilkan S = 0 dan R = 1, dan kemudian menurut Tabel 5.2, Q menjadi satu; ini berarti ketika input D = 1 dan menerapkan jam, output Q berubah menjadi 1. Gambar 5.6 menunjukkan diagram blok D flip-flop, dan tepi naik jam diwakili oleh " " dan Tabel 5.2 menunjukkan tabel karakteristik D flip-kegagalan.



Gambar 5.5 Rangkaian logika D flip-flop

Tabel 5.2 Tabel karakteristik flip-flop D

Clock	D	Q
↑	0	0
↑	1	1



Gambar 5.6 Diagram blok D flip-flop

J-K Flip-Flop

Gambar 5.7 menunjukkan diagram blok dari flip-flop J-K di mana J, K, dan clock adalah input ke flip-flop J-K. Aplikasi J-K flip-flop adalah counter dan pembagi frekuensi. Tabel 5.3

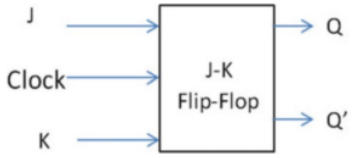
menunjukkan tabel karakteristik flip-flop J-K, dan langkah-langkah berikut menggambarkan operasi flip-flop J-K:

(a) Dengan mengatur J = K = 0 dan menerapkan pulsa clock ke flip-flop, output Q tidak berubah, jika Q = 0 kemudian tetap 0, atau jika Q = 1 kemudian tetap 1.

(b) Dengan mengatur J = 0, K = 1 dan menerapkan pulsa clock ke flip-flop, maka output Q berubah menjadi 0.

(c) Dengan menyetel J = 1, K = 0 dan menerapkan clock ke flip-flop, output Q berubah menjadi 1.

(d) Dengan menyetel J = K = 1 dan menerapkan pulsa clock, keluaran flip-flop adalah komplemen dari keluaran saat ini; ini berarti jika Q = 0 dan menerapkan clock, maka output berubah menjadi 1 dan jika Q = 1 dan menerapkan pulsa clock maka output akan berubah menjadi 0.



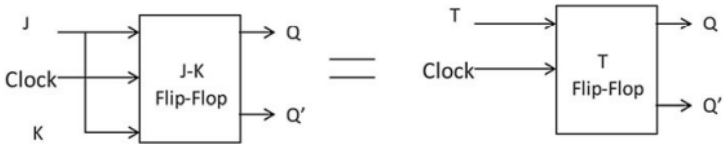
Gambar 5.7 J-K flip-flop

Tabel 5.3 Tabel Karakteristik Flip-Flop J-K

Clock	J	K	Q
↑	0	0	No change
↑	0	1	0
↑	1	0	1
↑	1	1	Complement

T Flip-Flop

Flip-flop T adalah kasus khusus dari flip-flop JK, dan dengan menghubungkan input J dan K dari flip-flop JK bersama-sama menghasilkan flip-flop T; Gambar 5.8 menunjukkan diagram blok T flip-flop, dan Tabel 5.4 menunjukkan tabel karakteristik T flip-flop; seperti terlihat pada Tabel 5.4 jika T = 0 dan menerapkan pulsa clock, maka keluaran T flip-flop tidak berubah, dan jika T = 1 dan menerapkan clock, maka keluaran flip-flop menjadi komplemen dari keluaran yang ada.



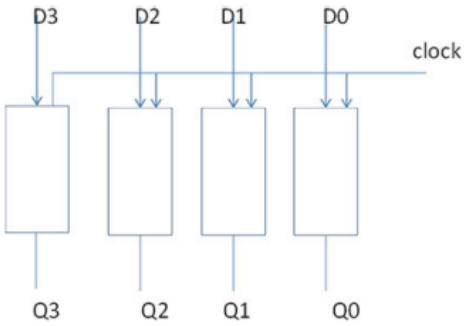
Gambar 5.8 Blok diagram T flip-flop

Tabel 5.4 Tabel Karakteristik T flip-flop

Clock	T	Q
↑	0	No change
↑	1	Complement

Daftar

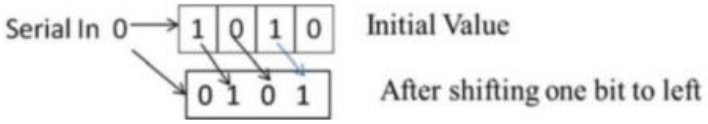
D flip-flop adalah memori 1-bit atau register 1-bit. Jika sekelompok D flip-flop berbagi jam yang sama, itu disebut register; register N-bit dibuat dengan flip-flop ND, dan jika flip-flop 32D menggunakan clock yang sama, maka disebut register 32-bit. Gambar 5.9 menunjukkan register 4-bit, dan pada gambar ini dengan menempatkan 1101 pada input dan menerapkan pulsa clock, maka outputnya akan menjadi 1101.



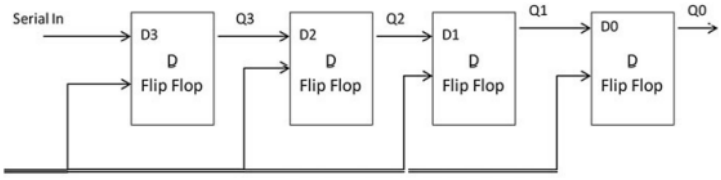
Gambar 5.9 Register 4-bit

Daftar Shift

Register geser memiliki satu input serial, dan satu bit dimuat dari input serial ke register oleh setiap pulsa clock, dan kemudian setiap bit register digeser ke posisi bit berikutnya. Gambar 5.10 menunjukkan operasi register geser kanan 4-bit, dan setelah menggeser satu bit ke kanan, isi register akan menjadi 0101. Gambar 5.11 menunjukkan register kanan geser serial 4-bit.

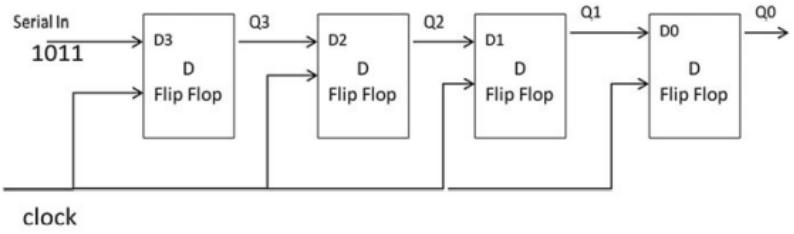


Gambar 5.10 Operasi register geser kanan 4-bit



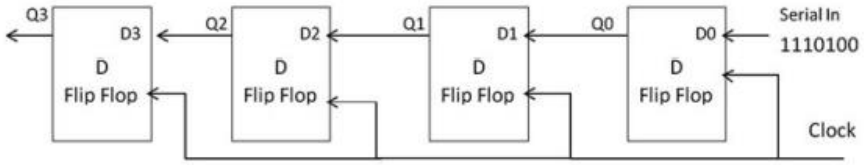
Gambar 5.11 Register kanan geser serial 4-bit

Contoh Gambar 5.12 menunjukkan register kanan geser 4-bit; tunjukkan isi register setelah menerapkan empat pulsa clock, dan anggap output awal setiap D flip-flop adalah nol.
Gambar 5.13 menunjukkan register geser kiri dengan input serial 1110100, dan Tabel 5.5 menunjukkan isi register setelah menerapkan lima pulsa clock; asumsikan output awal setiap flip-flop adalah 0.



Clock#	Q3	Q2	Q1	Q0	
	0	0	0	0	Initial values
1	1	0	0	0	
2	1	1	0	0	
3	0	1	1	0	
4	1	0	1	1	

Gambar 5.12 Register geser kanan 4-bit



Q3	Q2	Q1	Q0	Clock #	
0	0	0	0	0	initial values
0	0	0	1	1	
0	0	1	1	2	
0	1	1	1	3	
1	1	1	0	4	
1	1	0	1	5	

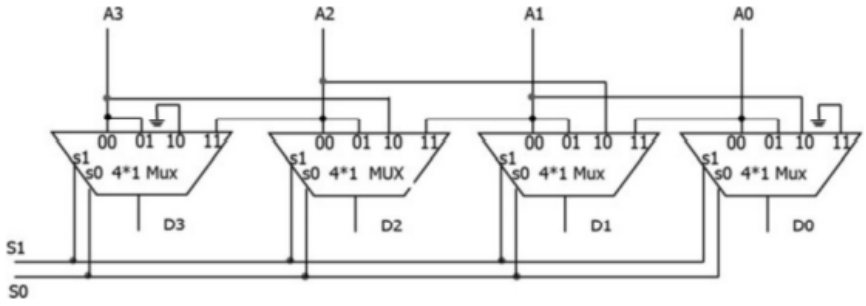
Gambar 5.13 Register geser kiri 4-bit

Pemindah barel

Barrel shifter digunakan untuk menggeser data ke kiri dan ke kanan; shifter barel menggunakan logika kombinasi daripada register geser; logika kombinasional tidak memerlukan jam dan ini adalah pemindah tercepat; Gambar 5.14 menunjukkan shifter barel 4-bit dan Tabel 5.5 menunjukkan tabel operasi untuk shifter barel.

Tabel 5.5 Operasi pemindah barel

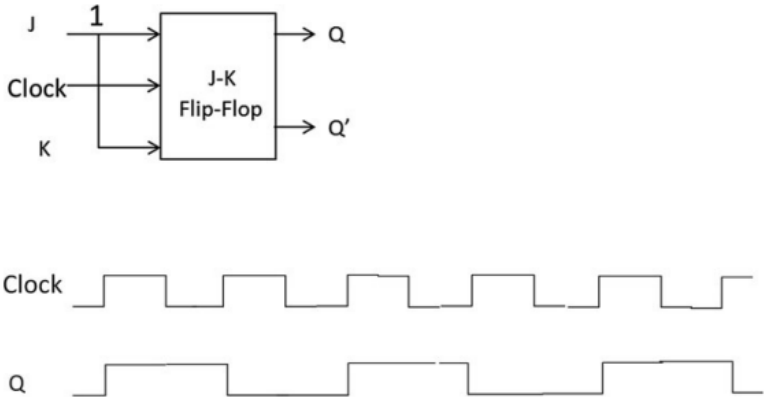
S1	S0	D3	D2	D1	D0
0	X	A3	A2	A1	A0
1	0	0	A3	A2	A1
1	1	A2	A1	A0	0



Gambar 5.14 Pemindah barel 4-bit

Pembagi Frekuensi Menggunakan J-K Flip-Flop

Gambar 5.15 menunjukkan JK flip-flop sebagai pembagi frekuensi; input J dan K diatur ke 1 dan mengasumsikan nilai awal Q = 0; seperti yang ditunjukkan pada gambar ini, untuk setiap dua pulsa clock yang diterapkan pada flip-flop, maka Q membangkitkan satu pulsa clock seperti yang ditunjukkan pada Gambar 5.15; ini berarti rangkaian membagi frekuensi dengan 2.

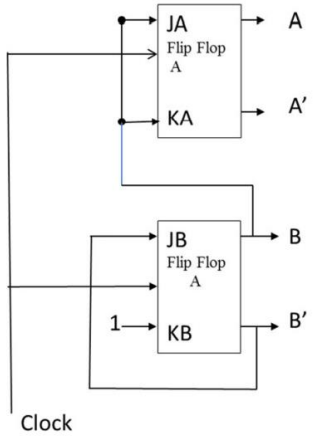


Gambar 5.15 Pembagi frekuensi menggunakan J-K flip-flop

5.4 ANALISIS LOGIKA SEKUENSIAL

Dalam logika kombinasional, tabel kebenaran mewakili karakteristik suatu fungsi, tetapi karakteristik logika sekuensial diwakili oleh tabel keadaan. Tabel keadaan diwakili oleh keadaan sekarang, keadaan selanjutnya, dan keluaran. Keadaan sekarang adalah keadaan saat ini dari flip-flop (keluaran saat ini), dan keadaan berikutnya adalah keluaran dari flip-flop setelah menerapkan jam ke logika sekuensial. Gambar 5.16 menunjukkan rangkaian sekuensial dengan dua JK flip-flop dengan dua keadaan A dan B; tabel state terdiri dari dua kolom present state dan next state; keadaan sekarang mewakili keluaran arus dari flip-flop dengan semua nilai yang mungkin untuk A dan B (00, 01, 10, dan 11) seperti yang ditunjukkan pada Tabel 5.6.

Pertimbangkan baris pertama; keadaan sekarang adalah 00 (berarti A = 0 dan B = 0); itu tertarik untuk menemukan output dari flip-flop (keadaan berikutnya) dengan menerapkan pulsa clock. Jika A = 0 dan B = 0 menghasilkan JA = KA = 0, JB = KB = 1, maka penerapan pulsa clock ke flip-flop menghasilkan A = 0 dan B = 1 (keadaan selanjutnya AB = 01).



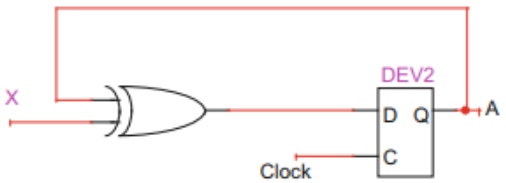
Gambar 5.16 Logika sekuensial

Perhatikan baris kedua, untuk keadaan sekarang A = 0 dan B = 1; oleh karena itu, JA = KA = 1, JB = 0, dan KB = 1; menerapkan pulsa clock ke hasil flip-flop ke keadaan berikutnya dengan A = 1 dan B = 0; prosedur yang sama digunakan untuk baris 10 dan 11 untuk menemukan keadaan berikutnya. Tabel 5.6 menunjukkan tabel keadaan untuk logika sekuensial Gambar 5.16.

Tabel 5.6 Tabel Status untuk Gambar 5.16

Keadaan sekarang		keadaan selanjutnya	
A	B	A	B
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Contoh 5.1 Temukan tabel keadaan pada Gambar 5.17.
Gambar 5.17 berisi input eksternal X, dan Tabel 5.7 menunjukkan stabil dengan dua kolom untuk status berikutnya: satu untuk X = 0 dan satu lagi untuk X = 1.



Gambar 5.17 Rangkaian sekuensial

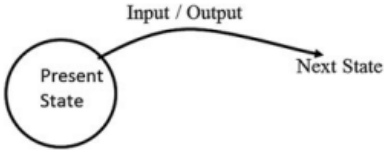
Tabel 5.7 Tabel keadaan untuk Gambar 5.1

Keadaan sekarang	Keadaan selanjutnya untuk X = 0	Keadaan selanjutnya untuk X = 1
A	A	A
0	0	1
1	1	0

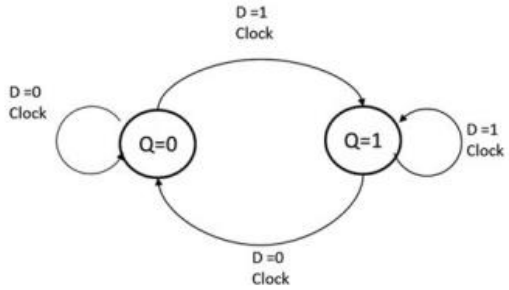
Diagram Keadaan

Cara lain untuk merepresentasikan karakteristik logika sekuensial adalah dengan diagram keadaan, seperti yang ditunjukkan pada Gambar 5.18. Keadaan sekarang adalah nilai flip-flop saat ini, dan menerapkan jam keadaan sekarang berubah ke keadaan berikutnya. Diagram Keadaan D Flip-Flop Gambar 5.19 menunjukkan diagram keadaan D flip-flop, dan langkah-langkah berikut menggambarkan diagram keadaan:

1. Jika Q = 0 (keadaan sekarang), dengan mengatur D = 1 dan menerapkan jam, maka Q berubah dari 0 ke 1 (keadaan berikutnya).
2. Jika Q = 1, dengan mengatur D = 1 dan menerapkan jam, maka output tetap 1.
3. Jika Q = 1, dengan mengatur D = 0 dan menerapkan jam, maka output berubah menjadi 0.
4. Jika Q = 0, dengan mengatur D = 0 dan menerapkan clock, maka output tetap 0.

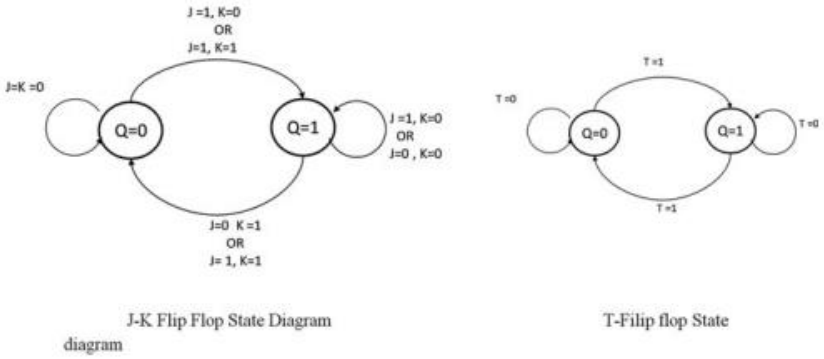


Gambar 5.18 Diagram keadaan



Gambar 5.19 Diagram keadaan untuk D flip-flop

Gambar 5.20 menunjukkan diagram keadaan JK dan T flip-flop.



diagram

Gambar 5.20 Diagram keadaan flip-flop J-K dan T

5.5 TABEL EKSITASI FLIP-FLOP

Penerapan tabel eksitasi adalah untuk menentukan masukan atau masukan dari suatu flip-flop agar mendapatkan keluaran yang telah ditentukan.

D Tabel Eksitasi Flip-Flop

Tabel 5.8 menunjukkan tabel eksitasi untuk D flip-flop: Q(t) adalah keluaran saat ini (keadaan sekarang) dan Q(t + 1) adalah keadaan berikutnya.

Tabel 5.8 Tabel eksitasi untuk D flip-flop

Q(*t*)	Q(*t* + 1)	D
0	0	0
0	1	1
1	0	0
1	1	1

Operasi Tabel Eksitasi Perhatikan baris pertama dari Tabel 5.8, jika Q(t) 0 (keadaan sekarang) dan diinginkan setelah menerapkan pulsa clock, output Q(t + 1) tetap 0 dan kemudian D harus diatur ke 0.

Pertimbangkan baris kedua, keluaran saat ini dari D flip-flop (keadaan sekarang) adalah 0, dan diinginkan untuk mengubah keluaran Q (t + 1) menjadi satu; oleh karena itu, input D harus disetel ke 1.

Pertimbangkan baris ketiga, keluaran saat ini adalah 1, dan diinginkan untuk mengubah keluaran (keadaan berikutnya) menjadi 0; oleh karena itu input D harus disetel ke 0. Pertimbangkan baris keempat, keadaan sekarang adalah 1, dan diinginkan untuk tetap 1; oleh karena itu, D harus disetel ke satu.

Tabel Eksitasi Flip-Flop J-K

Tabel 5.9 menunjukkan tabel eksitasi flip-flop J-K, dan langkah-langkah berikut menjelaskan bagaimana tabel ini dibuat:

1. Pertimbangkan baris pertama tabel eksitasi, keadaan flip-flop saat ini adalah nol, dan diinginkan untuk tetap 0 dengan menerapkan pulsa clock; oleh karena itu, J harus diset ke nol dan K tidak peduli (0 atau 1).
2. Pertimbangkan baris kedua, keadaan flip-flop saat ini adalah 0, dan diinginkan untuk mengubah output menjadi 1 dengan menerapkan pulsa clock; oleh karena itu, J harus disetel ke 1 dan K tidak peduli.
3. Pertimbangkan baris ketiga, keadaan sekarang Q(t) adalah 1, dan diinginkan untuk mengubahnya menjadi 0; oleh karena itu, J tidak peduli dan k = 1.
4. Pertimbangkan baris keempat, keadaan sekarang adalah 1, dan diinginkan untuk tetap 1; oleh karena itu, J tidak peduli dan K = 0.

Tabel 5.9 Tabel eksitasi flip-flop J-K

Q(*t*)	Q(*t* + 1)	J	K
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

Tabel Eksitasi Flip-Flop T. Tabel 5.10 menunjukkan tabel eksitasi T flip-flop.

Tabel 5.10 Tabel eksitasi flip-flop T

Q(*t*)	Q(*t* + 1)	T
0	0	0
0	1	1
1	0	1
1	1	0

5.6 PENGHITUNG

Pencacah adalah logika sekuensial yang digunakan untuk menghitung jumlah pulsa yang diterapkan padanya atau membagi frekuensi jam jika suatu sistem memiliki jam 16 Hz, dan dimungkinkan untuk menggunakan pencacah untuk mengubah jam 16 Hz menjadi 4 Hz. Langkah-langkah berikut menjelaskan cara mendesain counter:

(a) Tentukan urutan hitungan yang merupakan urutan yang akan dihitung oleh penghitung.
(b) Gunakan urutan hitungan untuk menentukan jumlah flip-flop.
(c) Memilih jenis flip-flop.
(d) Gunakan urutan hitungan untuk mengembangkan tabel keadaan.
(e) Gunakan tabel keadaan dan tabel eksitasi flip-flop untuk mengembangkan tabel eksitasi untuk pencacah.
(f) Gunakan K-map untuk menemukan fungsi atau fungsi input untuk setiap flip-flop.
(g) Gambarkan logika sekuensial untuk penghitung.

Contoh: Rancang pencacah untuk menghitung 0—1—2—3 dan ulangi menggunakan JK flip-flop. Angka terbesar dalam urutan hitungan adalah 3 yang diwakili dalam biner oleh 11; oleh karena itu, diperlukan dua buah flip-flop dan disebut A dan B seperti terlihat pada Gambar 5.22, dan Tabel 5.11 menunjukkan tabel state untuk counter.

Tabel 5.11 Status tabel penghitung

Present state		Next state	
A	B	A	B
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

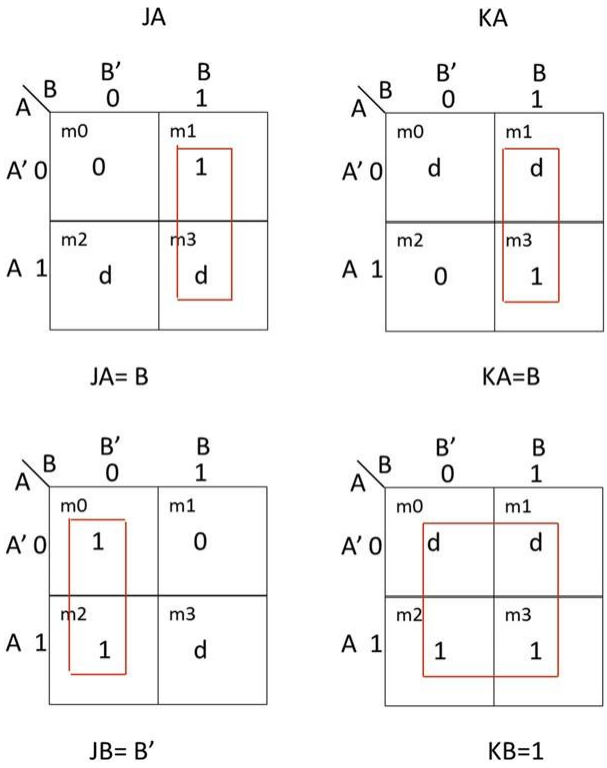
Status saat ini mendefinisikan keluaran saat ini dari flip-flop, dan keadaan berikutnya adalah keluaran dari flip-flop setelah menerapkan pulsa clock. Tabel 5.12 menunjukkan tabel eksitasi untuk pencacah yang dikembangkan dengan menggunakan tabel eksitasi flip-flop JK.

Tabel 5.12 Tabel eksitasi untuk penghitung

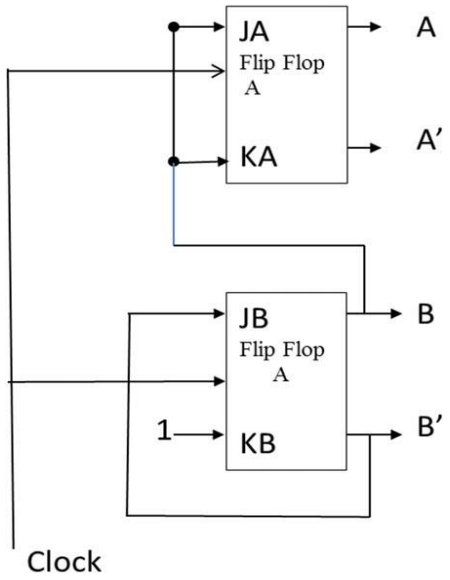
Present state		Next state					
A	B	A	B	JA	KA	JB	KB
0	0	0	1	0	d	1	d
0	1	1	0	1	d	d	1
1	0	1	1	d	0	1	d
1	1	0	0	d	1	d	1

Perhatikan baris pertama, keluaran saat ini dari sandal jepit JK adalah 00 (A = 0, B = 0), dan diinginkan keluarannya berubah menjadi 01 (A = 0 dan B = 1); oleh karena itu harus di set JA 0, KA d (tidak peduli) agar A tetap 0 dan set JB = 1, KB d agar B berubah dari 0 menja di 1. Pertimbangkan baris kedua, keadaan sekarang adalah 01 (A 0 dan B 1), dan diinginkan output berubah menjadi 10 (A = 1 dan B = 0); oleh karena itu harus diset JA = 1, KA = d, dan JB = 0, KB = d.

Diinginkan untuk menemukan fungsi input ke flip-flop, keadaan sekarang adalah input, dan JA, KA, JB, dan KB adalah output dari Tabel 5.12, dengan mentransfer output ke K-maps, dan membaca K-maps menghasilkan fungsi input ke flip-flop; Gambar 5.21 menunjukkan K-maps untuk JA, KA, JB, dan KB. Fungsi input ke flip-flop adalah JA = JB = B, JB = B' dan KB = 1, dan Gambar 5.22 menunjukkan rangkaian pencacah 2 bit.



Gambar 5.21 K-maps untuk pencacah 2-bit



Gambar 5.22 Logika sekuensial pencacah 2-bit

5.7 RINGKASAN

- Rangkaian logika sekuensial membutuhkan jam untuk beroperasi.
- Kait S-R adalah komponen dasar untuk flip-flop.
- S-R latch dapat dibangun dengan gerbang NAND atau NOR.
- Elemen dasar logika sekuensial adalah flip-flop.
- Flip-flop adalah elemen memori dengan dua keluaran Q dan Q'.
- Dengan menerapkan clock ke flip-flop D, nilai input D menyalin pada output Q.
- D flip-flop digunakan untuk mendesain register.
- Register adalah grup D flip-flop yang berbagi jam yang sama.
- J-K flip-flop digunakan untuk merancang pencacah.
- Menghubungkan input flip-flop JK bersama-sama menghasilkan flip-flop T.
- Tabel keadaan dan diagram keadaan menunjukkan operasi rangkaian sekuensial.
- Bab 6 adalah pengantar arsitektur komputer yang mencakup komponen dasar komputer mikro serta teknologi CPU, arsitektur CPU, prosesor multicore, langkah eksekusi instruksi, pipelining, dan bus mikrokomputer.

Soal

1. Lengkapi tabel berikut untuk D flip-flop.

D	Q(t) present output	Q(t + 1) next output
0	0	
0	1	
1	0	
1	1	

2. Lengkapi tabel berikut untuk flip-flop J-K.

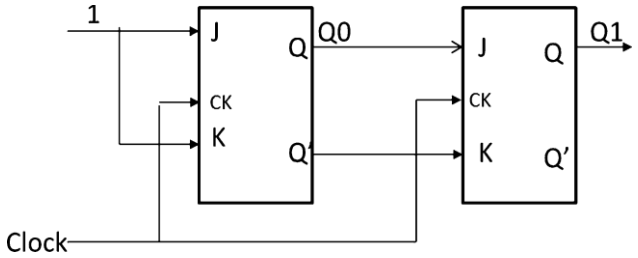
J	K	Q(t) present output	Q(t + 1) next output
0	0	0	

0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

3. Lengkapi tabel berikut untuk T flip-flop.

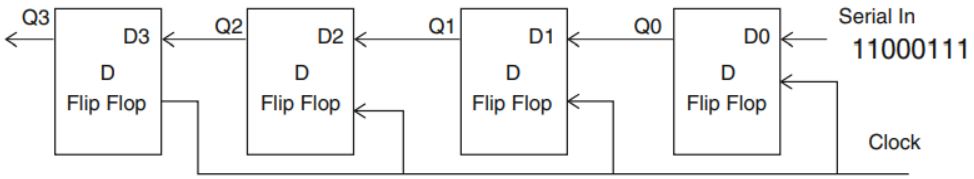
T	Q(*t*) present output	Q(*t* + 1) next output
0	0	
0	1	
1	0	
1	1	

4. Gambar berikut menunjukkan logika sekuensial; lengkapi tabel berikut dengan asumsi nilai awal Q1 = 0 dan Q2 = 0. Gunakan logistikim untuk memverifikasi jawaban Anda.

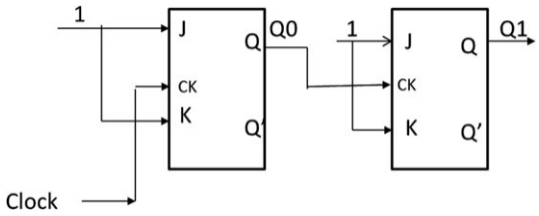


Clock	Q0	Q1
Initial value	0	0
Clock #1		
Clock #2		
Clock #3		

5. Tunjukkan register 8-bit menggunakan D flip-flop.
6. Register geser berikut diberikan, cari output setelah lima pulsa clock.



7. Dengan logika sekuensial berikut yang diberikan, asumsikan nilai awal untuk Q0 = 0 dan Q1 = 0, dan flip-flop berubah status di tepi naik pulsa clock; lengkapi tabel berikut dan kemudian gunakan logistikim untuk memverifikasi hasil Anda.



Clock	Q0	Q1
Initial value	0	0
Clock #1		
Clock #2		
Clock #4		
Clock #4		

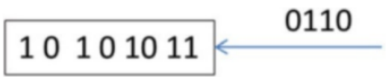
8. Lengkapi tabel eksitasi berikut untuk JK flip-flop.

Q(t)	Q(t + 1)	J	K
0	0		
0	1		
1	0		
1	1		

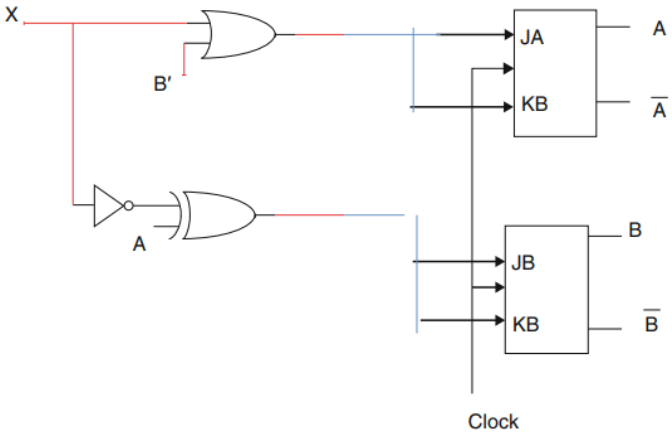
9. Rancang penghitung untuk menghitung 0—1—2—3—4—5—6—7 dan ulangi.
   (a) Gunakan sandal jepit JK.
   (b) Gunakan sandal jepit T.
   (c) Verifikasi desain Anda menggunakan logistikim.
10. Temukan diagram keadaan untuk tabel keadaan berikut.

	AB	AB
AB	X = 0	X = 1
00	01	10
01	10	00
10	11	01
11	00	10

11. Apa isi register berikut setelah digeser lima kali ke kiri?



12. Tunjukkan tabel keadaan dan diagram keadaan untuk rangkaian berikut.


X
B'
A
JA
KB
A
$\overline{A}$
JB
KB
B
$\overline{B}$
Clock

BAB 6
PENGANTAR ARSITEKTUR KOMPUTER

Tujuan: Setelah menyelesaikan bab ini, Anda diharapkan dapat:

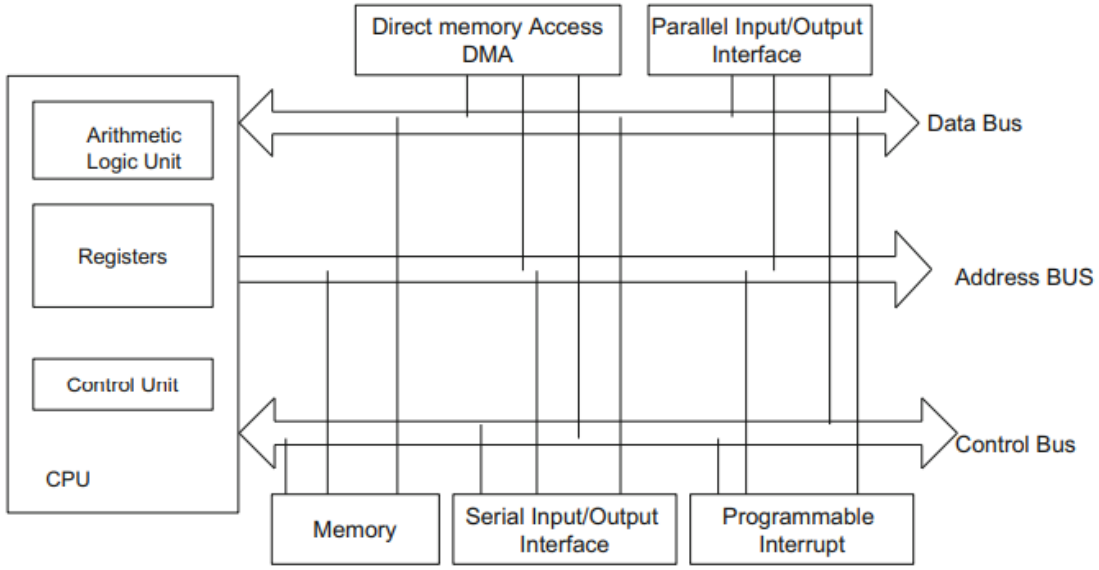
- Sebutkan komponen-komponen komputer mikro.
- Daftar komponen CPU.
- Bedakan teknologi CPU.
- Pelajari arsitektur prosesor multicore.
- Bandingkan prosesor RISC dengan prosesor CISC.
- Jelaskan perbedaan antara arsitektur von Neumann dan Harvard.
- Bedakan antara prosesor 32-bit dan prosesor 64-bit.
- Menjelaskan langkah-langkah eksekusi instruksi.
- Tunjukkan keuntungan dari pipelining instruksi.
- Membedakan berbagai jenis bus mikrokomputer.
- Jelaskan pengoperasian bus USB.

6.1 PENDAHULUAN

Sama seperti arsitektur bangunan mendefinisikan desain dan fungsinya secara keseluruhan, demikian juga arsitektur komputer mendefinisikan desain dan fungsionalitas sistem komputer. Komponen komputer mikro dirancang untuk berinteraksi satu sama lain, dan interaksi ini memainkan peran penting dalam operasi sistem secara keseluruhan.

6.2 KOMPONEN KOMPUTER MIKRO

Komputer mikro standar terdiri dari mikroprosesor (CPU), bus, memori, input/output paralel, input/output serial, interupsi I/O yang dapat diprogram, dan DMA akses memori langsung. Gambar 6.1 menunjukkan komponen-komponen mikrokomputer.



Gambar 6.1 Komponen mikrokomputer

Unit Pemrosesan Pusat (CPU)

Unit pemrosesan pusat (CPU) adalah "otak" komputer dan bertanggung jawab untuk menerima data dari perangkat input, memproses data menjadi informasi, dan mentransfer informasi ke memori dan perangkat output. CPU diatur ke dalam tiga bagian utama berikut:

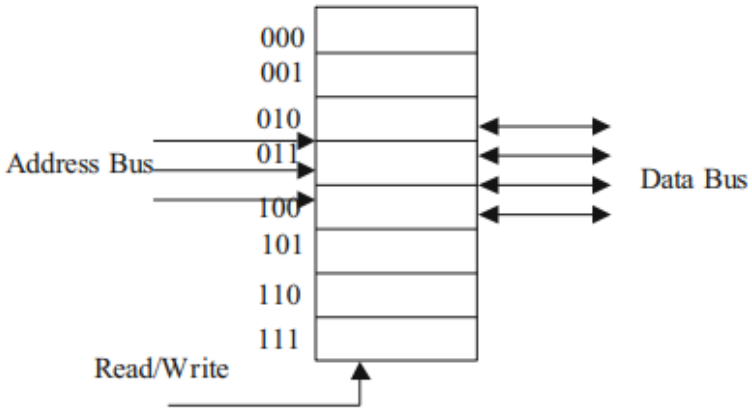
1. Unit logika aritmatika (ALU)
2. Unit control
3. Mendaftar

Fungsi dari unit logika aritmatika (ALU) adalah untuk melakukan operasi aritmatika seperti penambahan, pengurangan, pembagian, dan perkalian dan operasi logika seperti AND, OR, dan NOT. Fungsi unit kontrol adalah untuk mengontrol perangkat input/output, membangkitkan sinyal kontrol ke komponen komputer lainnya seperti sinyal baca dan tulis, dan melakukan eksekusi instruksi. Informasi dipindahkan dari memori ke register; register kemudian meneruskan informasi ke ALU untuk operasi logika dan aritmatika. Register Register adalah memori tercepat di komputer yang menyimpan informasi.

Bus CPU

Ketika lebih dari satu kabel membawa jenis informasi yang sama, itu disebut bus. Bus yang paling umum di dalam komputer mikro adalah bus alamat, bus data, dan bus kontrol.

Alamat Bus Alamat Bus mendefinisikan jumlah lokasi yang dapat dialamatkan dalam IC memori dengan menggunakan rumus 2^n, di mana n menyatakan jumlah baris alamat. Jika bus alamat terdiri dari tiga jalur, maka ada $2^3 = 8$ lokasi memori yang dapat dialamatkan, seperti yang ditunjukkan pada Gambar 6.2. Ukuran bus alamat secara langsung menentukan jumlah maksimum lokasi memori yang dapat diakses oleh CPU. Misalnya, CPU dengan 32 bus alamat dapat memiliki 232 lokasi memori yang dapat dialamatkan.

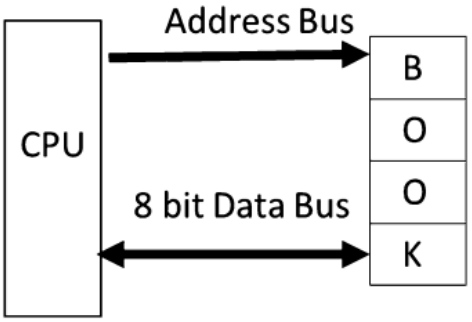


Gambar 6.2 Memori dengan tiga jalur alamat dan empat jalur data

Bus Data Bus data digunakan untuk membawa data ke dan dari memori. Pada Gambar 6.2, setiap lokasi hanya dapat menampung empat bit. Ukuran IC memori diwakili oleh $2^n \times m$ di mana n adalah jumlah baris alamat dan m adalah ukuran setiap lokasi. Di Gambar 6.2, di mana n 3 dan m 4, ukuran memorinya adalah

$2^3 * 4$ 32 bit:

Ukuran bus data memainkan faktor penting pada kinerja CPU, bus data CPU saat ini adalah 32 bit atau 64 bit, dan CPU dengan bus data 32 bit berarti dapat membaca atau menulis 32 bit data dari dan ke memori. Generasi awal CPU berisi bus data 8-bit, dan setiap lokasi memori menampung satu byte, untuk membaca kata "buku" seperti yang ditunjukkan pada Gambar 6.3. CPU perlu mengakses memori empat kali. Dengan meningkatkan bus data dari 8 bit menjadi 32 bit, maka CPU dapat mengakses memori dan membaca seluruh kata “buku”. Kebanyakan CPU menawarkan instruksi untuk membaca 1 byte, 2 byte, atau 4 byte dari memori.



Gambar 6.3 CPU dengan Bus Data 8 bit

Bus Kontrol Bus kontrol membawa sinyal kontrol dari unit kontrol ke komponen komputer untuk mengontrol operasi setiap komponen. Selain itu, unit kontrol menerima sinyal kontrol dari komponen komputer. Beberapa sinyal kontrol adalah sebagai berikut:

Sinyal baca:	Jalur baca diatur ke tinggi untuk dibaca dari lokasi memori atau perangkat input/output (I/O).
Sinyal tulis:	Garis tulis digunakan untuk menulis data ke dalam memori.
Interupsi:	Menunjukkan permintaan interupsi.
Permintaan bus:	Perangkat meminta untuk menggunakan bus komputer.
Hibah Bus:	Memberikan izin kepada perangkat yang meminta untuk menggunakan bus komputer.
I/O Membaca dan Menulis:	Baca dan tulis I/O digunakan untuk membaca dari atau menulis ke perangkat I/O.

CPU 32-Bit Versus 64-Bit

Ukuran register memainkan peran penting dalam kinerja CPU. Prosesor 32-bit berarti dapat melakukan operasi pada data 32-bit; oleh karena itu, ukuran register adalah 32 bit dan ALU juga melakukan operasi 32-bit. CPU 64-bit melakukan operasi dalam data 64-bit; oleh karena itu, ia berisi register 64-bit dan ALU 64-bit. Sebagian besar komputer desktop dan server menggunakan prosesor AMD dan Intel; mereka mungkin menggunakan 32 bit atau 64 bit. Prosesor Intel dan AMD menggunakan arsitektur yang sama; ini berarti program di komputer dengan prosesor Intel dapat berjalan di komputer dengan prosesor AMD.

Teknologi CPU

Ada dua jenis teknologi yang digunakan untuk merancang CPU dan mereka disebut CISC dan RIS.

CISC (Complex Instruction Set Computer) Pada tahun 1978, Intel mengembangkan chip mikroprosesor 8086. 8086 dirancang untuk memproses kata data 16-bit; itu tidak memiliki instruksi untuk operasi floating point. Saat ini, Pentium memproses kata 32-bit dan 64-bit, dan dapat memproses instruksi floating point.

Intel merancang prosesor Pentium sedemikian rupa sehingga dapat menjalankan program yang ditulis untuk prosesor 80 86 sebelumnya. Karakteristik 80 86 disebut komputer set instruksi kompleks (CISC), yang mencakup instruksi untuk prosesor Intel sebelumnya. Prosesor CISC lainnya adalah VAX 11/780, yang dapat menjalankan program untuk komputer PDP-11. Prosesor CISC berisi banyak instruksi dengan mode pengalamatan yang berbeda, misalnya, VAX 11/780 memiliki lebih dari 300 instruksi dengan 16 mode alamat yang berbeda. Karakteristik utama dari prosesor CISC adalah sebagai berikut:

1. Sejumlah besar instruksi.
2. Banyak mode pengalamatan.
3. Variabel panjang instruksi.
4. Kebanyakan instruksi dapat memanipulasi operand dalam memori.
5. Unit kontrol diprogram mikro.

RISC Sampai pertengahan 1990-an, produsen komputer sedang merancang CPU yang kompleks dengan set instruksi yang besar. Pada saat itu, sejumlah produsen komputer memutuskan untuk merancang CPU yang hanya mampu mengeksekusi serangkaian instruksi yang sangat terbatas. Salah satu keuntungan dari komputer set instruksi yang dikurangi adalah mereka dapat mengeksekusi instruksi mereka dengan sangat cepat karena instruksinya sederhana. Selain itu, chip RISC membutuhkan transistor yang lebih sedikit daripada chip CISC. Beberapa prosesor RISC adalah PowerPC, prosesor MIPS, IBM RISC System/6000, ARM, dan SPARC.

Karakteristik utama prosesor RISC adalah sebagai berikut:

1. Memerlukan sedikit instruksi.
2. Semua instruksi memiliki panjang yang sama (dapat dengan mudah didekodekan).
3. Sebagian besar instruksi dieksekusi dalam satu siklus jam mesin.
4. Unit kontrol sudah terpasang.
5. Beberapa mode alamat.
6. Sejumlah besar register.

Tabel 6.1 Perbandingan prosesor RISC dan CISC

CISC	RISC
Panjang instruksi variabel	Panjang instruksi tetap
Panjang opcode variabel	Panjang opcode tetap
Operan memori	Memuat/menyimpan instruksi
Contoh: Pentium	ARM, MIPS

Prosesor RISC menggunakan perangkat keras dan mikroprogram prosesor CISC untuk unit kontrol, dan unit kontrol dengan perangkat keras menggunakan lebih sedikit ruang di CPU; oleh karena itu, perancang CPU dapat menambahkan lebih banyak register ke prosesor

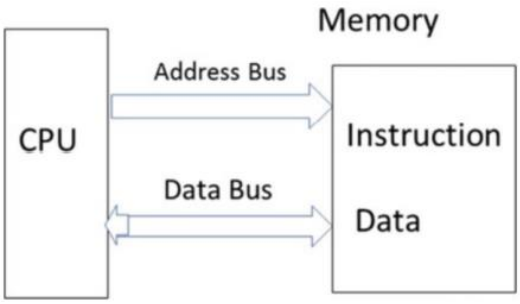
RISC dibandingkan dengan CISC. Keunggulan prosesor CISC adalah desainer dapat menambahkan instruksi baru tanpa mengubah arsitektur prosesor. Tabel 6.1 menunjukkan perbandingan CISC dan RISC.

Arsitektur CPU

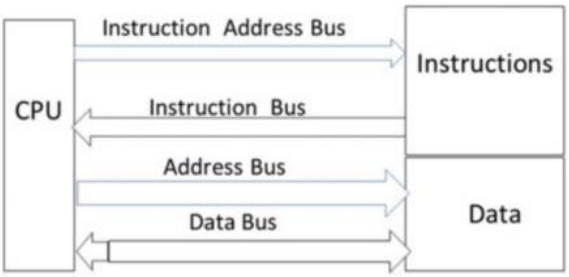
Ada dua jenis arsitektur CPU dan mereka adalah von Neumann dan arsitektur Harvard.

Arsitektur Von Neumann Merupakan program yang terdiri dari kode (instruksi) dan data. Gambar 6.4 menunjukkan diagram blok arsitektur von Neumann. Von Neumann menggunakan bus data untuk mentransfer data dan instruksi dari memori ke CPU.

Arsitektur Harvard Arsitektur Harvard menggunakan bus terpisah untuk instruksi dan data seperti yang ditunjukkan pada Gambar 6.5. Bus alamat instruksi dan bus instruksi digunakan untuk membaca instruksi dari memori. Bus alamat dan bus data digunakan untuk menulis dan membaca data ke dan dari memori.



Gambar 6.4 Arsitektur Von Neumann



Gambar 6.5 Arsitektur Harvard

6.3 KELUARGA MIKROPROSESOR INTEL

Intel mendesain dan memproduksi mikroprosesor untuk mikrokomputer. Setiap prosesor memiliki nomor atau nama, yang digunakan oleh perancang komputer untuk mengakses informasi yang diberikan oleh produsen prosesor.

Nomor dan nama IC mikroprosesor Intel adalah 8088, 80.286, 80.386, 80.486, Pentium, Pentium II, Pentium III, dan Pentium IV yang mereka sebut sebagai IA-86 (Arsitektur Intel-86). Baru-baru ini Intel dan HP mengembangkan prosesor baru yang disebut Itanium yang merupakan prosesor 64-bit, yang akan dijelaskan nanti dalam bab ini. Berikut ini adalah daftar karakteristik mikroprosesor:

1. *Ukuran register:* Register digunakan untuk menyimpan informasi di dalam prosesor. Ukuran register dapat bervariasi dari 8- hingga 16- hingga 32- hingga 64-bit.
2. *Jumlah register:* Sebuah prosesor dengan beberapa register dapat menyimpan lebih banyak informasi dalam CPU untuk diproses.

3. *Ukuran bus data:* Ukuran bus data menentukan berapa banyak bit data yang dapat ditransfer secara paralel ke atau dari memori atau port input/output.
4. *Ukuran bus alamat:* Ukuran alamat tipikal adalah 16, 32, dan 64 bit. Ukuran bus alamat menentukan jumlah lokasi memori yang dapat diakses oleh mikroprosesor.
5. *Kecepatan jam:* Kecepatan jam menentukan kecepatan prosesor mengeksekusi instruksi.
6. *Koprosesor matematika:* Koprosesor matematika adalah prosesor khusus yang melakukan operasi matematika kompleks.
7. *Mode nyata:* Mode nyata memungkinkan kompatibilitas perangkat lunak dengan perangkat lunak lama. Ini memungkinkan prosesor untuk meniru prosesor Intel 8088 terendah dan hanya menggunakan memori 1 MB pertama.
8. *Mode terproteksi:* Mode terproteksi adalah jenis penggunaan memori yang tersedia pada mikroprosesor model 80.286 dan yang lebih baru. Dalam mode terproteksi, setiap program dapat dialokasikan bagian tertentu dari memori dan program lain tidak dapat menggunakan memori ini. Mode terproteksi juga memungkinkan satu program mengakses lebih dari 1 MB memori.
9. *Ukuran cache:* Memori cache adalah sejumlah kecil memori berkecepatan tinggi yang digunakan untuk penyimpanan data sementara berdasarkan antara prosesor dan memori utama. Ukuran cache dapat membantu mempercepat waktu eksekusi suatu program.
10. *Teknologi MMX:* Teknologi MMX Intel dirancang untuk mempercepat aplikasi multimedia dan komunikasi, seperti video, animasi, dan grafik 3D. Teknologi tersebut meliputi teknik single instruction multiple data (SIMD) (artinya dengan satu instruksi, komputer dapat melakukan beberapa operasi), 57 instruksi baru, delapan register MMX 64-bit, dan empat tipe data baru.

Tabel 6.2 Memberikan referensi cepat ke daftar karakteristik sebelumnya Sebagian besar workstation atau laptop menggunakan CPU yang diproduksi oleh Intel dan AMD. Perusahaan. Prosesor Intel diklasifikasikan berdasarkan IA-16 (Prosesor Intel arsitektur 16-bit), IA-32, dan IA-64.

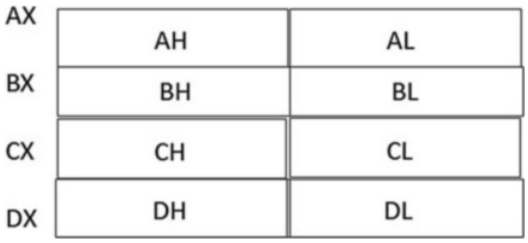
Kompatibilitas ke Atas Arsitektur Intel kompatibel ke atas yang berarti program yang ditulis untuk prosesor IA-16 dapat berjalan di IA-32. Gambar 6.6 menunjukkan register umum IA-16, di mana AH, AL, BH, BL, CH, CL, DH, dan DL adalah register 8-bit dan AX, BX, CX, dan DX adalah register 16-bit. AX, BX, CX, dan DX adalah kombinasi dari dua register.

Tabel 6.2 Karakteristik mikroprosesor Intel

	80486dx	Pentium	Pentium Pro	Pentium Pro II	Pentium II
Daftar ukuran	32 bit	32 bit	32 bit	32/64 bit	32/64 bit
Ukuran bus data	32 bit	64 bit	64 bit	64 bit	64 bit
Ukuran alamat	32 bit	32 bit	32 bit	32 bit	32 bit

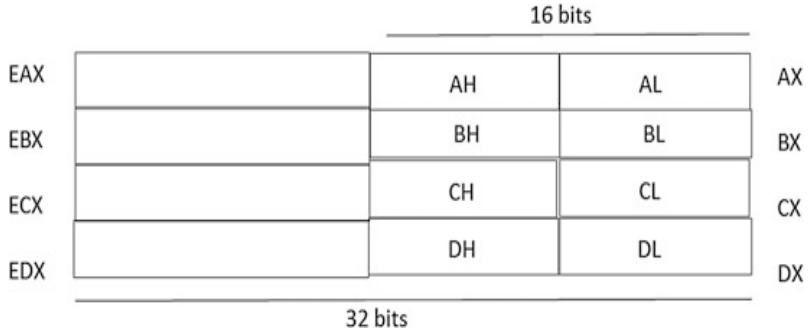
Memori maksimum	4 GB	4 GB	4 GB	4 GB	4 GB
Kecepatan jam	25, 33 MHz	60, 166 MHz	150, 200 MHz	233, 340, 400 MHz	450, 500 MHz
prosesor matematika	Built-in	Built-in	Built-in	Built-in	Built-in
L1 cache	8 KB, 16 KB	8 KB instruksi 8 KB data	Instruksi 8 KB, data 8 KB	Instruksi 16 KB, data 16 KB	Instruksi 16 KB Data 16 KB
L2 cache	Tidak	Tidak	256 KB atau 512 KB	512 KB	512 KB
teknologi MMX	Tidak	Tidak	Ya	Ya	Ya

L1 cache adalah memori cache yang dibangun di dalam mikroprosesor
L2 cache bukan bagian dari mikroprosesor; itu ada di IC yang terpisah



Gambar 6.6 Register IA-16

Gambar 6.7 menunjukkan register umum IA-32 dimana EAX, EBX, ECX, dan EDX adalah register umum 3-bit, dan juga Gambar 6.6 juga berisi register IA-16-bit; Oleh karena itu, program yang ditulis untuk IA-16 dapat dieksekusi oleh IA-32.

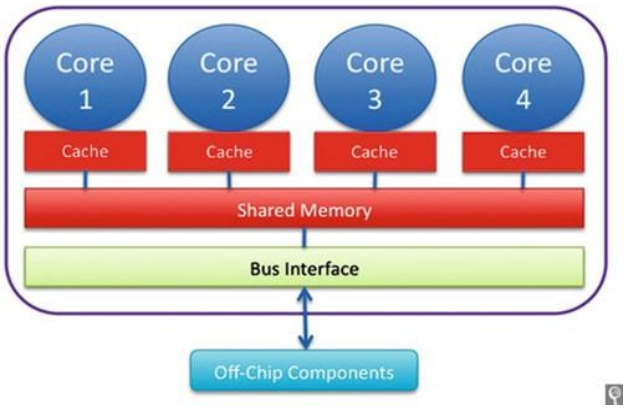


Gambar 6.7 Register Intel IA-32-bit

6.4 PROSESOR MULTICORE

Sebuah prosesor multicore adalah sirkuit terpadu (IC) dengan dua atau lebih CPU independen yang disebut inti, dan mereka mengeksekusi beberapa instruksi secara bersamaan untuk meningkatkan kinerja. Prosesor quad-core adalah chip dengan empat unit independen yang disebut core yang membaca dan mengeksekusi instruksi seperti menambah, memindahkan data, dan bercabang. Gambar 6.8 menunjukkan diagram blok prosesor quad-core yang semuanya berbagi memori. Berikut ini adalah beberapa prosesor multicore:

- Dua inti (CPU dual-core) seperti AMD Phenom II X2 dan Intel Core Duo
- Tiga core (CPU tri-core) seperti AMD Phenom II X3
- Empat inti (CPU quad-core) seperti AMD Phenom II X4, prosesor Intel i5 dan i7
- Enam core (hexa-core CPU) seperti AMD Phenom II X6 dan Intel Core i7 Extreme Edition 980X
- Delapan core (CPU octa-core) seperti Intel Core i7 5960X Extreme Edition dan AMD FX-8350
- Sepuluh core (CPU deca-core) seperti Intel Xeon E7-2850



Gambar 6.8 Arsitektur prosesor multicore

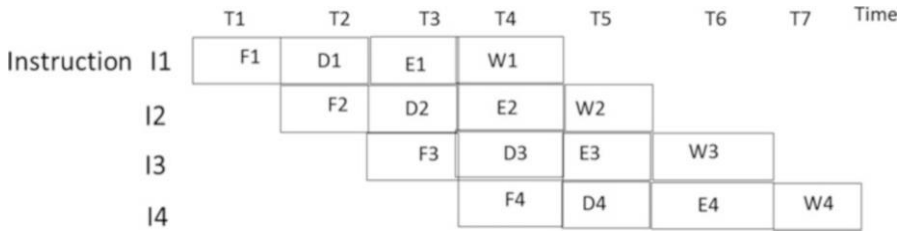
6.5 EKSEKUSI INSTRUKSI CPU

Secara umum, CPU melakukan langkah-langkah berikut untuk mengeksekusi satu instruksi:

1. Ambil instruksi (F): Memindahkan Instruksi dari memori ke CPU.
2. Decode instruksi (D): Tentukan jenis instruksi seperti ADD, AND, OR, dan Store operand ke dalam register jika diperlukan.
3. Eksekusi instruksi (E): Jalankan instruksi seperti penambahan.
4. Write result (R): Menyimpan hasil eksekusi ke dalam register atau memori.

Pipe. Pipelining akan meningkatkan kinerja CPU yang berarti mengeksekusi lebih banyak instruksi dalam waktu yang lebih singkat. Gambar 6.9 menunjukkan eksekusi empat instruksi tanpa pipelining; dalam gambar ini, CPU mengeksekusi satu instruksi pada satu waktu, dan setiap tahap membutuhkan waktu T detik; maka total waktu eksekusi adalah 16 T. Gambar 6.10 menunjukkan eksekusi empat instruksi menggunakan pipeline; CPU mengambil instruksi I1 dan memindahkannya ke unit decode; CPU saat mendekode I1 akan mengambil instruksi I2 dan proses ini akan berlanjut.

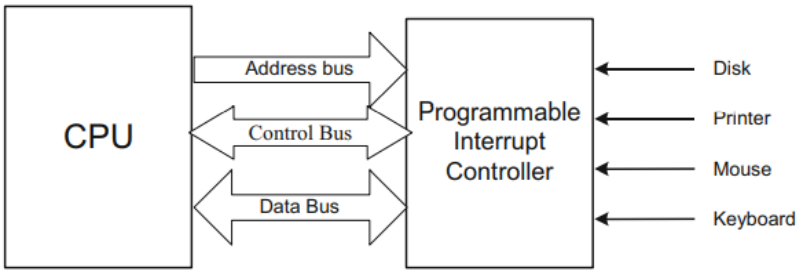
Gambar 6.9 Eksekusi instruksi tanpa pipeline



Gambar 6.10 Eksekusi instruksi menggunakan pipeline

Seperti ditunjukkan pada Gambar 6.10, dibutuhkan 7 T untuk menyelesaikan eksekusi empat instruksi. Gambar 6.11 menunjukkan bahwa pada saat T4 CPU menulis hasil eksekusi I1 ke dalam memori dan pada saat yang sama mengambil instruksi I4 dari memori, tetapi tidak mungkin untuk membaca dan menulis pada saat yang sama ke atau dari memori; oleh karena itu, memiliki dua cache terpisah (cache instruksi dan cache data) akan mengatasi konflik ini, dan jenis arsitektur ini disebut arsitektur Harvard.

Akses memori langsung: Akses memori langsung (DMA) memungkinkan transfer blok data dari memori ke perangkat I/O atau sebaliknya. Tanpa DMA, CPU membaca data dari memori dan menulisnya ke perangkat I/O. Mentransfer blok data dari memori ke perangkat I/O memerlukan CPU untuk melakukan satu baca dan satu tulis untuk setiap operasi. Metode transfer data ini membutuhkan banyak waktu. Fungsi DMA adalah mentransfer data dari memori ke perangkat I/O secara langsung, tanpa menggunakan CPU, sehingga CPU bebas melakukan fungsi lainnya.



Gambar 6.11 Pengontrol interupsi yang dapat diprogram

DMA melakukan fungsi berikut untuk menggunakan bus komputer:

- DMA mengirimkan sinyal permintaan ke CPU.
- CPU merespon DMA dengan permintaan hibah, mengizinkan DMA untuk menggunakan bus.
- DMA mengontrol bus dan perangkat I/O dapat membaca atau menulis secara langsung ke atau dari memori.
- DMA dapat memuat file dari disk eksternal ke memori utama ketika blok data yang besar perlu ditransfer ke rentang memori berurutan. DMA jauh lebih cepat dan lebih efisien daripada CPU.

Interupsi I/O yang Dapat Diprogram Ketika beberapa perangkat I/O seperti drive eksternal, hard disk, printer, monitor, dan modem terhubung ke komputer seperti yang ditunjukkan pada Gambar 6.11; mekanisme diperlukan untuk menyinkronkan semua permintaan perangkat. Fungsi interupsi yang dapat diprogram adalah untuk memeriksa status setiap

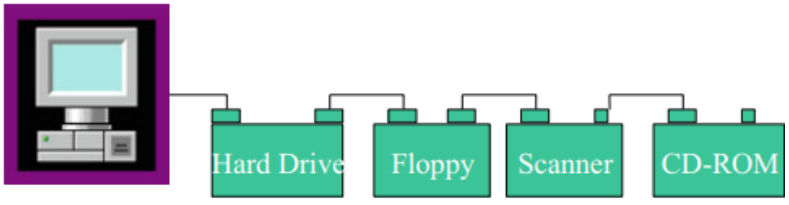
perangkat dan memberi tahu CPU tentang status masing-masing; misalnya, printer tidak siap, disk dilindungi dari penulisan, ini adalah disk yang tidak diformat, dan ada koneksi yang hilang ke modem. Setiap perangkat mengirimkan sinyal ke pengontrol interupsi I/O yang dapat diprogram untuk memperbarui statusnya. Gambar 6.11 menunjukkan pengontrol interupsi I/O yang dapat diprogram.

6.6 PENGONTROL DISK

Pengontrol disk menggerakkan kepala drive disk, membaca, dan/atau menulis data. Saat ini, ada dua jenis pengontrol disk: IDE (elektronik disk terintegrasi) dan SCSI (antarmuka sistem komputer kecil).

Elektronik Disk Terintegrasi (IDE). Disk drive IDE terhubung ke bus ISA dengan kabel pita datar. Pengontrol disk IDE mendukung dua hard disk, masing-masing dengan kapasitas 528 megabita. Pada tahun 1994, vendor hard disk drive memperkenalkan EIDE (extended IDE) yang mendukung empat perangkat, seperti hard disk, tape drive, perangkat CD-ROM, dan hard disk drive yang lebih besar. EIDE memiliki dua konektor. Setiap kabel terhubung ke pengontrol EIDE dan dapat mendukung dua hard disk drive dengan kapasitas hingga 250 GB. EIDE digunakan di komputer yang kompatibel dengan IBM.

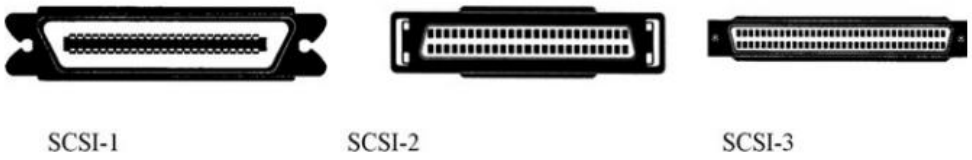
Small Computer System Interface (SCSI) Standar Small Computer System Interface (SCSI) ditetapkan oleh American National Standards Institute (ANSI) untuk menghubungkan daisy chaining beberapa perangkat I/O, seperti pemindai, hard disk, dan CD-ROM, ke mikrokomputer, seperti yang ditunjukkan pada Gambar 6.12. SCSI adalah antarmuka standar untuk semua jenis mikrokomputer. Ini digunakan di Macintosh, workstation RISC, dan komputer mini, serta komputer yang kompatibel dengan IBM kelas atas. Bus SCSI dilengkapi dengan berbagai jenis pengontrol. Tabel 6.3 menunjukkan karakteristik beberapa jenis pengontrol SCSI, dan Gambar 6.13 menunjukkan konektor SCSI-1, SCSI-2, dan SCSI-3. SATA (Serial Advanced Technology Attachment) SATA adalah serial bus untuk menghubungkan mess storage seperti hard disk, optical drive, dan solid-state drive ke komputer.



Gambar 6.12 Bus SCSI

Tabel 6.3 Karakteristik Beberapa Jenis Kontroler SCSI

	Bandwidth	Kecepatan data MB/dtk
SCSI-1	8 bit	5
SCSI-2	16 bit	10–20
Ultra SCSI	8 bit	20
SCSI-3	16 bit	40



Gambar 6.13 Konektor SCSI-1, SCSI-2, dan SCSI-3

6.7 BUS MIKROKOMPUTER

Saat ini ada sejumlah bus komputer yang berbeda di pasaran yang dirancang untuk mikrokomputer. Beberapa BUS komputer tersebut adalah ISA, MCA, EISA, VESA PCI, FireWire, USB, dan PCI Express. Universal serial bus (USB) dan PC Express dibahas lebih rinci karena lebih maju daripada bus lainnya.

Bus ISA

Bus arsitektur standar industri (ISA) diperkenalkan oleh IBM untuk PC IBM menggunakan mikroprosesor 8088. Bus ISA memiliki bus data 8-bit dan 20 saluran alamat pada kecepatan clock 8 MHz. Jenis PC AT menggunakan prosesor 80.286 yang memiliki bus data 16-bit dan saluran alamat 24-bit dan kompatibel dengan PC.

Bus Arsitektur Microchannel

Bus arsitektur saluran mikro (MCA) diperkenalkan oleh IBM pada tahun 1987 untuk komputer mikro PS/2-nya. Bus MCA adalah bus 32-bit yang dapat mentransfer empat byte data sekaligus dan berjalan pada kecepatan clock 10 MHz. Ini juga mendukung transfer data 16-bit dan memiliki saluran alamat 32-bit. Arsitektur saluran mikro sangat mahal sehingga vendor non-IBM mengembangkan solusi yang sebanding tetapi lebih murah yang disebut bus EISA.

Bus EISA

Extended ISA (EISA) bus adalah bus 32-bit yang juga mendukung arsitektur bus transfer data 8- dan 16-bit. EISA berjalan pada kecepatan clock 8-MHz dan memiliki saluran alamat 32-bit.

Bus VESA

Video Electronic Standard Association (VESA) bus, yang juga disebut video local bus (VL-BUS), adalah antarmuka standar antara komputer dan perluasannya. Ketika aplikasi menjadi lebih intensif secara grafis, bus VESA diperkenalkan untuk memaksimalkan throughput memori grafis video. Bus VESA menyediakan aliran data yang cepat antar stasiun dan dapat mentransfer hingga 132 Mbps.

Bus PCI

Bus interkoneksi komponen periferal (PCI) dikembangkan oleh Intel Corporation. Teknologi bus PCI mencakup bus 32-/64-bit yang berjalan pada kecepatan clock 33/66 MHz. PCI menawarkan banyak keuntungan untuk koneksi ke hub, router, dan kartu antarmuka jaringan (NIC). Secara khusus, PCI menyediakan lebih banyak bandwidth: hingga 1 gigabit per detik sesuai kebutuhan komponen perangkat keras ini. Bus PCI dirancang untuk meningkatkan bandwidth dan mengurangi latensi dalam sistem komputer. Versi bus PCI saat ini mendukung kecepatan data 1056 Mbps dan dapat ditingkatkan hingga 4224 Mbps. Bus PCI dapat

mendukung hingga 16 slot atau perangkat di motherboard. Sebagian besar pemasok ATM (mode transfer asinkron) dan NIC 100BaseT menawarkan antarmuka PCI untuk produk mereka. Bus PCI dapat diperluas untuk mendukung bus data 64-bit. Tabel 6.4 membandingkan arsitektur bus yang berbeda yang menunjukkan karakteristik bus ISA, EISA, MCA, VESA, dan PCI. Gambar 6.14 menunjukkan bus PCI.

Tabel 6.4 Karakteristik berbagai bus

Jenis bus	ISA	EISA	MCA	VESA	PCI	PCI-64
Kecepatan (MHz)	8	8.3	10	33	33	64
Bandwidth bus data (bit)	16	32	32	32	32	66
Maks. kecepatan data (MB/s)a	8	32	40	132	132	508
Plug and play mampu	Tidak	Tidak	Ya	Ya	Ya	Ya

Gambar 6.14 kartu PCI

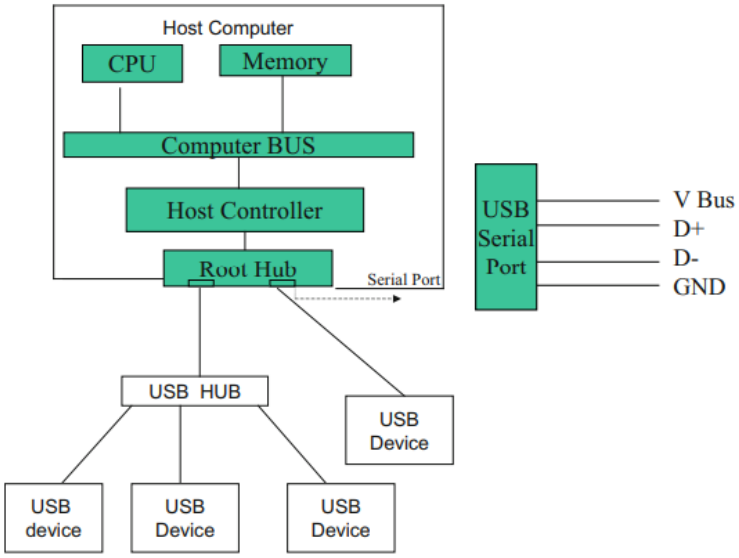
Universal Serial BUS (USB)

Universal serial bus (USB) adalah bus serial komputer yang memungkinkan pengguna untuk menghubungkan periferal seperti mouse, keyboard, modem, CD-ROM, pemindai, dan printer, ke luar komputer tanpa konfigurasi apa pun. Komputer pribadi yang dilengkapi dengan USB akan memungkinkan pengguna untuk menghubungkan periferal ke komputer, dan komputer akan secara otomatis dikonfigurasikan saat perangkat terpasang padanya. Ini berarti bahwa USB memiliki kemampuan untuk mendeteksi ketika perangkat telah ditambahkan atau dihapus dari PC. USB adalah bus plug-and-play sejati. Hingga 127 periferal dapat dihubungkan ke PC dengan USB. USB versi 1.1 dirilis pada tahun 1998 yang mendukung kecepatan data 12 Mbps (kecepatan penuh) dan 1,5 Mbps (kecepatan rendah); kecepatan rendah digunakan untuk perangkat seperti mouse, keyboard, dan joystick. USB versi 2 berkecepatan tinggi (480 Mbps) yang kompatibel dengan USB 1.1 Spesifikasi USB 2.0 dikembangkan oleh tujuh produsen komputer terkemuka dan diumumkan pada tahun 1999. Panjang kabel maksimum untuk USB adalah 5 meter.

Arsitektur USB

Gambar 6.15 menunjukkan arsitektur USB; sistem USB secara logis adalah topologi pohon tetapi fisik adalah topologi bintang karena setiap perangkat USB berkomunikasi langsung dengan Hub Root. Hanya ada satu Host Controller di sistem USB mana pun. Sistem USB terdiri dari pengontrol host USB, hub root USB, hub USB, kabel USB, perangkat USB, perangkat lunak klien, dan perangkat lunak pengontrol host. Host Controller Host controller memulai semua transfer data, dan root hub menyediakan koneksi antara perangkat dan host controller. Hub root menerima transaksi yang dihasilkan oleh pengontrol host dan mengirimkan ke perangkat USB. Pengontrol host menggunakan polling untuk mendeteksi perangkat baru dan terhubung ke bus atau terputus. Selain itu, pengontrol host USB melakukan fungsi berikut:

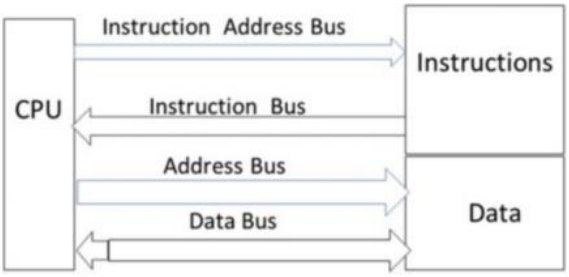
(a) Pengontrol host menyiapkan perangkat untuk operasi (konfigurasi perangkat).
(b) Pembuatan paket.
(c) Serializer/deserializer.
(d) Memproses permintaan dari perangkat dan host.
(e) Kelola protokol USB.
(f) Mengelola aliran antara perangkat host dan USB.
(g) Tetapkan alamat ke perangkat.
(h) Jalankan perangkat lunak klien.
(i) Mengumpulkan bit status dari port USB.



Gambar 6.15 Arsitektur USB

Root Hub Root hub melakukan distribusi daya ke perangkat, mengaktifkan dan menonaktifkan port, dan melaporkan status setiap port ke pengontrol host. Hub root menyediakan koneksi antara pengontrol host dan port USB.

Hub Hub digunakan untuk menambah jumlah perangkat yang terhubung ke sistem USB. Hub dapat mendeteksi saat perangkat dipasang atau dilepas dari port. Gambar 6.16 menunjukkan arsitektur hub. Port upstream terhubung ke host, dan perangkat USB terhubung ke port downstream.



Gambar 6.5 Arsitektur Harvard

Dalam transmisi downstream, semua perangkat yang terhubung ke hub akan menerima paket, tetapi hanya perangkat yang menerima paket yang alamat perangkatnya cocok dengan alamat di token. Dalam transmisi upstream, perangkat mengirimkan paket ke hub, dan hub mentransmisikan paket ke port upstream saja. USB meningkatkan kecepatannya dan setiap beberapa tahun versi baru dikembangkan. Tabel 6.5 menunjukkan versi USB yang berbeda.

Tabel 6.5 Versi USB dan kecepatan datanya

versi USB	rilis data	Kecepatan data	Penunjukan kecepatan data
USB 1.0	1996	1,5 Mbps	Kecepatan rendah
USB 1.1	1998	12 Mbps	Kecepatan penuh
USB 2.0	2000	480 Mbps	Kecepatan tinggi
USB 3.0	2008	5 Gbps	kecepatan super
USB 3.1	2013	10 Gbps	Kecepatan super +

Kabel USB. Gambar 6.15 menunjukkan port USB dengan empat pin, yang terdiri dari empat kabel, dengan bus V yang digunakan untuk memberi daya pada perangkat. D dan D digunakan untuk transmisi sinyal.

Perangkat USB Perangkat USB dibagi menjadi beberapa kelas seperti hub, printer, atau penyimpanan massal. Perangkat USB memiliki informasi tentang konfigurasinya seperti kelas, jenis, ID pembuatan, dan kecepatan data. Pengontrol host menggunakan informasi ini untuk memuat perangkat lunak perangkat dari hard disk. Perangkat USB mungkin memiliki beberapa fungsi seperti volume di speaker. Setiap fungsi dalam perangkat USB ditentukan oleh alamat titik akhir.

Bus PCI Express

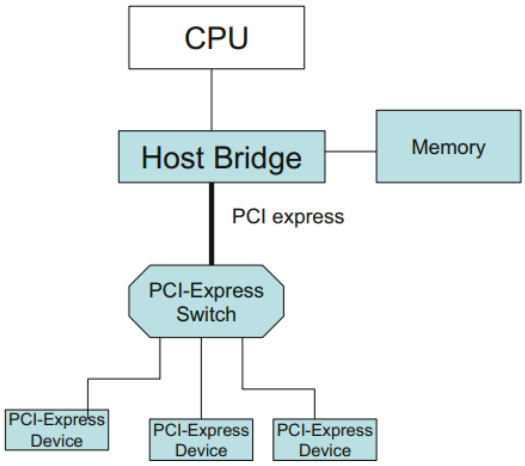
PCI express diperkenalkan pada pertengahan 1990 dengan frekuensi 33 MHz, dan selama ini kecepatan BUS ditingkatkan menjadi 66 MHz. Karena perkembangan baru dalam teknologi jaringan seperti Gigabit Ethernet dan perangkat I/O yang menuntut lebih banyak bandwidth, maka dibutuhkan teknologi bus baru dengan bandwidth yang lebih tinggi. PCI express telah disetujui oleh Special Interest Group pada tahun 2002, dan chipset mulai dikirimkan pada tahun 2004. PCI express memiliki beberapa fitur berikut:

- PCI express adalah koneksi point-to-point antar perangkat.
- PCI express adalah bus serial.

- PCI express menggunakan arsitektur pocket dan layer.
- Kompatibel dengan bus PCI melalui perangkat lunak.
- Integritas data tautan ujung ke ujung (deteksi kesalahan).
- Transfer data isokron.
- Bandwidth yang dapat dipilih.

Arsitektur PCI Express

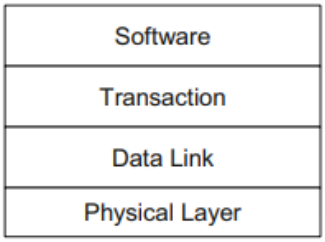
Gambar 6.17 menunjukkan arsitektur PCI express, dan fungsi host bridge adalah untuk menghubungkan bus CPU dengan memori dan sakelar PCI express. Sakelar digunakan untuk menambah jumlah port ekspres PCI.



Gambar 6.17 Arsitektur PCI express

Arsitektur Protokol PCI Express

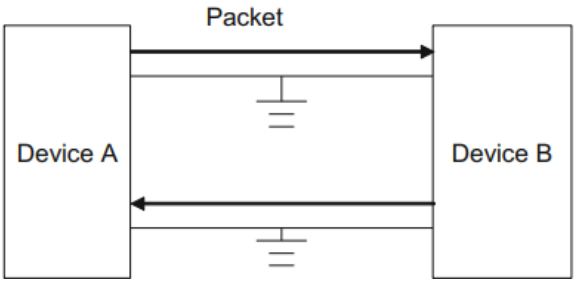
Gambar 6.18 menunjukkan arsitektur protokol PCI express. Protokol terdiri dari perangkat lunak PCI, transaksi, tautan data, dan lapisan fisik.



Gambar 6.18 Arsitektur protokol PCI express

Software Layer Lapisan software digunakan untuk kompatibilitas dengan PCI, inisialisasi, dan enumerasi perangkat yang terhubung ke PCI express.

PCI Express Physical Layer Gambar 6.19 menunjukkan dua perangkat terhubung melalui PCI express link (jalur); setiap jalur terbuat dari empat kabel, dan setiap jalur PCI express terdiri dari dua koneksi simpleks, satu untuk mentransmisikan paket dan satu lagi untuk menerima paket. Jalur PCI express mendukung transfer 2,5 giga/detik setiap arah.



Gambar 6.19 Koneksi PCI express

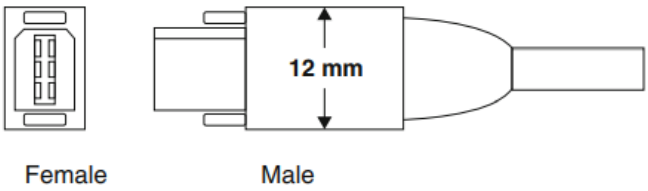
Link PCI express dapat mengkonfigurasi jalur X1, X2, X4, X4, X16, dan X32, di mana X1 berarti 1 jalur dengan 4 kabel dan X4 berarti 4 jalur dengan 16 kabel dan terakhir X32 berarti 32 jalur dengan 128 kabel. PCI-32 berarti PCI express dengan 32 jalur. Jam untuk tautan serial PCI express disematkan ke dalam data dengan menggunakan pengkodean 8B/10B.

6.8 FIREWIRE

FireWire atau IEEE 1394 adalah bus serial berkecepatan tinggi yang digunakan untuk menghubungkan perangkat digital seperti video digital atau camcorder. Bus mampu mentransfer data dengan kecepatan 100, 200, atau 400 Mpbs. Kabel IEEE 1394 terdiri dari enam kabel tembaga; dua kabel membawa daya dan empat kabel membawa sinyal seperti yang diilustrasikan pada Tabel 6.6. Beberapa konektor FireWire dilengkapi dengan empat pin, tanpa memiliki pin daya. Gambar 6.20 menunjukkan konektor laki-laki dan perempuan FireWire.

Tabel 6.6 IEEE 1394 pin

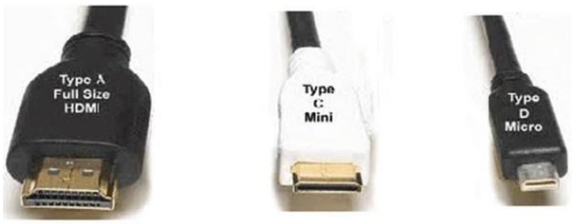
Pin	Nama sinyal	Keterangan
1	Kekuatan	DC yang tidak diatur; 17–24 V tanpa beban
2	Tanah	Pengembalian tanah untuk daya dan pelindung kabel dalam
3	TPB—	Twisted-pair B, sinyal diferensial
4	TPB+	Twisted-pair B, sinyal diferensial
5	TPA—	Twisted-pair A, sinyal diferensial
6	TPA+	Twisted-pair A, sinyal diferensial



Gambar 6.20 Konektor laki-laki dan perempuan FireWire

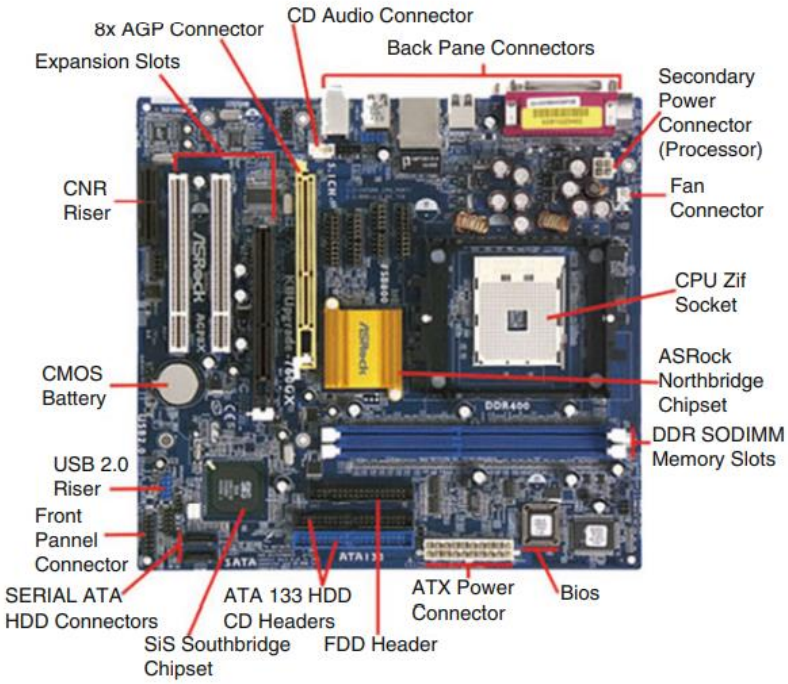
HDMI (Antarmuka Multimedia Definisi Tinggi)

HDMI adalah antarmuka antara dua perangkat untuk mentransfer data video terkompresi dan data audio digital terkompresi atau tidak terkompresi. Beberapa aplikasi HDMI adalah monitor komputer, TV digital, dan proyektor video. Gambar 6.21 menunjukkan berbagai jenis konektor HDMI.



Gambar 6.21 Konektor HDMI

Motherboard Motherboard adalah papan sirkuit tercetak (PCB) yang berisi sebagian besar komponen komputer seperti CPU, RAM, ROM, slot ekspansi, dan USB. Gambar 6.22 menunjukkan gambar motherboard.



Gambar 6.22 Gambar motherboard

6.9 RINGKASAN

- Komponen mikrokomputer adalah CPU, memori, I/O paralel, I/O serial, interupsi yang dapat diprogram, dan DMA.
- Fungsi CPU adalah mengeksekusi instruksi.
- Komponen CPU adalah arithmetic logic unit (ALU), control unit, dan register.
- Sebagian besar komputer menggunakan tiga jenis memori: memori cache (SRAM), memori utama (DRAM atau SDRAM), dan memori sekunder (hard disk, tape drive, dan floppy disk).
- Jenis memori semikonduktor adalah DRAM, SDRAM, EDORAM, DDR, SDRAM, RDRAM, ROM, dan EPROM.
- SRAM digunakan dalam memori cache; DRAM dan SDRAM digunakan di memori utama.
- SATA, SCSI-1, SCSI-2, dan SCSI-3 adalah pengontrol periferal komputer.

- Bus ISA, EISA, MCA, dan PCI adalah bus mikrokomputer, dan FireWire adalah bus serial berkecepatan tinggi dengan kecepatan data hingga 400 Mbps.
- USB adalah bus serial.
- PCI express adalah bus serial.
- Bab 7 mencakup memori semikonduktor, hard disk, solid-state drive, metode pemetaan memori cache, dan memori virtual.

Tinjau Pertanyaan

Soal pilihan ganda

1. Fungsi dari adalah untuk melakukan operasi aritmatika.
   (a) bus
   (b) Port serial
   (c) ALU
   (d) Unit kontrol

2. Saat Anda membandingkan fungsi CPU dan mikroprosesor, .
   (a) Mereka adalah sama
   (b) Mereka tidak sama
   (c) CPU lebih cepat dari mikroprosesor
   (d) Mikroprosesor lebih cepat dari CPU

3. Prosesor RISC menggunakan .
   (a) Set instruksi yang kompleks
   (b) Set instruksi yang dikurangi
   (c) a dan b
   (d) Tidak satu pun di atas

4. Unit kontrol prosesor CISC adalah .
   (a) Perangkat Keras
   (b) Mikrokode
   (c) a dan b
   (d) Tidak satu pun di atas

5. Akses memori langsung memungkinkan transfer blok data dari memori ke perangkat I/O (atau sebaliknya) tanpa menggunakan .
   (a) CPU
   (b) Bus data
   (c) Bus kendali
   (d) pengontrol DMA

6. Manakah dari bus berikut yang 32-bit?
   (a) ISA

(b) PCI dan EISA
(c) EISA dan ISA
(d) MCA dan ISA

7. Manakah dari sistem operasi berikut yang mendukung plug and play?
   (a) Jendela NT dan Jendela 95
   (b) Jendela 98 dan Jendela NT
   (c) Jendela XP dan Jendela 2000
   (d) DOS dan Windows NT

1. Sebutkan komponen-komponen komputer mikro.
2. Jelaskan fungsi CPU.
3. Sebutkan fungsi dari ALU.
4. Apa fungsi dari unit kontrol?
5. Sebutkan komponen CPU?
6. Berapa bit setengah kata?
7. Berapa banyak bit sebuah kata?
8. Bedakan antara CPU dan mikroprosesor.
9. Jelaskan fungsi bus alamat dan bus data.
10. Jelaskan fungsi DMA.
11. Apa aplikasi dari port paralel?
12. Apa aplikasi port serial?
13. Apa itu interupsi?
14. Berapa memori maksimum untuk CPU dengan 16 jalur alamat dan 8 jalur data?
15. Sebutkan karakteristik prosesor CISC.
16. Sebutkan karakteristik prosesor RISC.
17. Bedakan antara arsitektur von Neumann dan arsitektur Harvard.
18. Apa keunggulan prosesor multicore dibandingkan single core?
19. Daftar langkah-langkah eksekusi instruksi CPU.
20. Jelaskan instruksi pengambilan
21. Berapa lama waktu yang dibutuhkan CPU untuk mengeksekusi lima instruksi menggunakan pipelining jika setiap tahap pipeline membutuhkan waktu 20 menit.
22. Hitung waktu eksekusi untuk masalah 21 menggunakan prosesor non-pipa.
23. Apa karakteristik prosesor 64-bit?
24. Daftar jenis pengontrol disk.
25. Daftar bus komputer.
26. Sebutkan dua serial bus.
27. Berapa jumlah maksimum port USB yang dapat dimiliki komputer?
28. Tampilkan koneksi pin port USB.
29. Tunjukkan diagram jalur PCIe.
30. Apa aplikasi FireWire?
31. Apa itu aplikasi HDMI?

BAB 7
PENYIMPANAN

Tujuan: Setelah menyelesaikan bab ini, Anda diharapkan dapat:

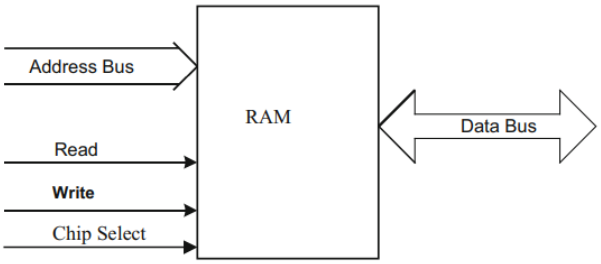
- Membedakan berbagai jenis memori semikonduktor.
- Jelaskan sektor, track pada hard disk.
- Untuk menghitung kapasitas disk.
- Pelajari hierarki memori.
- Jelaskan cache miss, cache hit, dan cache hit ratio.
- Menjelaskan jenis penggunaan memori di komputer.
- Jelaskan metode pemetaan yang berbeda yang digunakan dalam memori cache.
- Terjemahkan alamat virtual ke alamat fisik.
- Menjelaskan fungsi tabel halaman.
- Menghasilkan alamat fisik dari alamat virtual.

7.1 PENDAHULUAN

Dalam sebuah komputer, memori menyimpan instruksi (kode) dan data, memori memainkan bagian penting dari kinerja komputer, dan register adalah jenis memori dengan kapasitas kecil. Ada dua jenis memori yang digunakan di komputer, dan mereka diklasifikasikan sebagai memori semikonduktor dan hard disk. Memori semikonduktor dapat berupa memori volatil atau non-volatil. Memori volatil kehilangan isinya ketika daya dicabut, sedangkan memori nonvolatil akan menyimpan isinya tanpa daya.

7.2 MEMORI SEMIKONDUKTOR

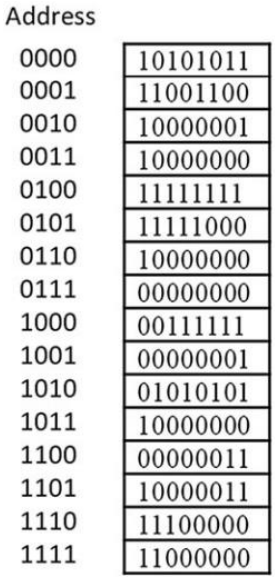
Ada dua jenis memori semikonduktor: memori akses acak (RAM) dan memori hanya baca (ROM).



Gambar 7.1 diagram blok RAM

RAM Data dapat dibaca dari atau ditulis ke dalam memori akses acak (RAM). RAM dapat menyimpan data selama daya disuplai ke sana dan ini disebut memori volatil. Gambar 7.1 menunjukkan diagram blok umum RAM yang terdiri dari bus data, bus alamat, dan sinyal baca/tulis. Bus data membawa data keluar dari atau ke dalam RAM. Bus alamat digunakan untuk memilih lokasi memori. Sinyal baca menjadi aktif saat membaca data dari RAM, dan

jalur tulis menjadi aktif saat menulis ke RAM. Ingat, RAM hanya dapat menyimpan informasi jika memiliki daya. Gambar 7.2 menunjukkan RAM 16 * 8 bit atau RAM 2^4 * 8 bit atau 16 B.



Gambar 7.2 16 byte RAM

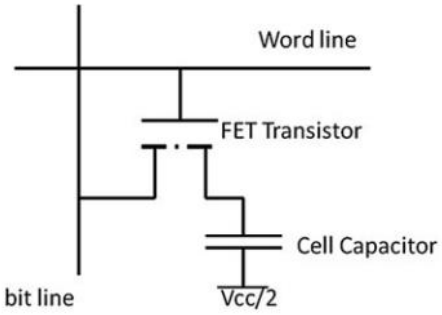
Tabel 7.1 Jumlah alamat dan lokasi Memori

Jumlah alamat	Jumlah lokasi memori	Perwakilan
10	2^{10} = 1024	1 K
11	2^{11} = 2048	2 K
12	2^{12} = 4096	4 K
13	2^{13} = 8192	8 K
14	2^{14} = 16,384	16 K
16	2^{16} = 65,536	64 K
20	2^{20} = 1,048,576	1 M
24	2^{24} = 16,777,261	16 M
32	2^{32} = 4,294,967,296	4 G

Pada Gambar 7.2, alamatnya adalah 4 bit; oleh karena itu, ada 2^4 = 16 lokasi memori, dan jika setiap lokasi menampung 1 B, maka ada 16 B memori, memori dengan m baris alamat, maka ada 2m lokasi memori. Tabel 7.1 menunjukkan jumlah baris alamat dan jumlah desimal yang setara dari lokasi memori.

Ada banyak jenis RAM, seperti RAM dinamis (DRAM), DRAM sinkron (SDRAM), RAM EDO, DDR SDRAM, RDRAM, dan RAM statis (SRAM).

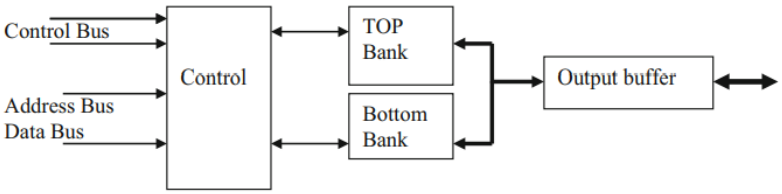
- *Dynamic* RAM (DRAM) digunakan di memori utama. DRAM menggunakan lebih sedikit komponen untuk membuat satu bit; oleh karena itu, dapat merancang sirkuit terpadu (IC) DRAM dengan kapasitas besar 4 GB per IC; Gambar 7.3 menunjukkan DRAM satu bit.



Gambar 7.3 DRAM 1-bit

Kapasitor sel dapat diisi dengan logika satu atau nol, tetapi perlu disegarkan (diisi ulang) setiap 1 ms. CPU tidak dapat membaca dari atau menulis ke memori saat DRAM sedang di-refresh; ini membuat DRAM menjadi memori yang berjalan paling lambat.

- *Synchronous* DRAM (SDRAM): Teknologi SDRAM menggunakan DRAM dan menambahkan antarmuka khusus untuk sinkronisasi. Ini dapat berjalan pada kecepatan clock yang jauh lebih tinggi daripada DRAM. SDRAM menggunakan dua bank memori independen. Saat satu bank sedang diisi ulang, CPU dapat membaca dan menulis ke bank lain. Gambar 7.4 menunjukkan diagram blok SDRAM.



Gambar 7.4 Diagram blok SDRAM

- Extended Data Out RAM (EDORAM) mentransfer blok data ke atau dari memori.
- Double Data Rate SDRAM (DDR SDRAM) adalah jenis SDRAM yang mentransfer data pada tepi naik dan tepi turun jam. Itu dapat memindahkan data dua kali lebih cepat dari SDRM; oleh karena itu, memori dapat berjalan pada kecepatan clock . DDR2 dan DDR3 merupakan kemajuan teknologi DDR dan semakin meningkatkan jumlah transfer data per siklus clock. DDR2 RAM menyediakan 4 transfer data per siklus, dan DDR3 mentransfer 8 data per siklus clock. Untuk kecepatan clock 100 MHz dan bus data 64 bit, kecepatan transfer untuk DDR adalah:

  DDR = 100*2*8 = 1600 MB/dtk (MB/dtk)
  DDR2 = 100*4*8 = 3200 MB/dtk
  DDR3 = 100*8*8 = 6400 MB/dtk

- Rambus DRAM (RDRAM) dikembangkan oleh Rambus Corporation. Ini menggunakan beberapa bank DRAM dengan antarmuka baru yang memungkinkan bank DRAM untuk mentransfer banyak kata dan juga mentransfer data di tepi naik dan tepi turun jam. Penyegaran RDRAM dilakukan oleh antarmuka. RDAM generasi kedua disebut

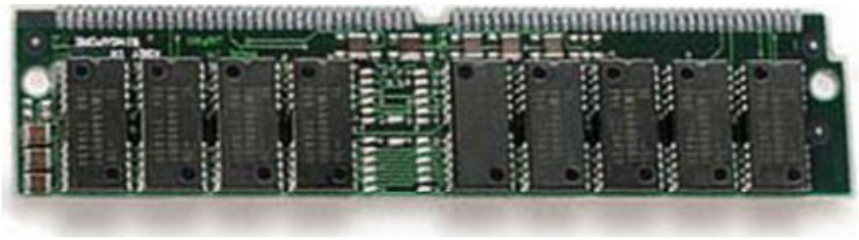
DRDRAM (Direct RDRAM), dan dapat mentransfer data dengan kecepatan 1,6 Gbps. Gambar 7.5 menunjukkan modul RDRAM.

Gambar 7.5 Modul memori Rambus (Courtesy Samsung Corp.)

Kemasan DRAM

DRAM hadir dalam berbagai jenis kemasan seperti SIMM (modul memori in-line tunggal) dan DIMM (modul memori in-line ganda).

Gambar 7.6 menunjukkan SIMM, yang merupakan papan sirkuit kecil yang satu sisi papannya menampung beberapa chip. Ini memiliki bus data 32 bit.



Gambar 7.6 DRAM SIMM

DIMM adalah papan sirkuit yang kedua sisi papannya menampung beberapa chip memori tetapi memiliki bus data 64 bit.

- *RAM Statis (SRAM)* digunakan dalam memori cache. SRAM hampir 20 kali lebih cepat dari DRAM dan juga jauh lebih mahal. D Flip Flop adalah satu bit RAM statis.

ROM (Memori Hanya-Baca)

Seperti namanya, informasi hanya dapat disiapkan dari read-only memory (ROM). ROM menyimpan informasi secara permanen, bahkan saat tidak ada daya ke ROM; jenis memori ini disebut memori nonvolatile. Dua jenis ROM tercantum di bawah ini:

- *Memori Hanya Baca yang Dapat Diprogram yang Dapat Dihapus (EPROM):* EPROM dapat dihapus dengan sinar ultraviolet dan diprogram ulang dengan perangkat yang disebut pemrogram EPROM. Flash ROM adalah jenis EEPROM.
- *PROM yang Dapat Dihapus Secara Elektrik (EEPROM):* EEPROM dapat dihapus dengan menerapkan tegangan tertentu ke salah satu pinnya dan dapat diprogram ulang dengan pemrogram EPROM.
- *Memori Flash:* memori flash adalah memori non-volatil yang memiliki berbagai aplikasi seperti flash drive, solid-state drive, kartu memori, dan sistem tertanam. Memori flash adalah jenis EEPROM yang memungkinkan beberapa lokasi memori untuk ditulis atau dihapus dalam satu operasi. Ada dua jenis teknologi yang digunakan untuk memori flash, yaitu memori flash NAND dan NOR; Memori flash NAND memiliki waktu akses

yang lebih kecil daripada memori flash NOR; sebagian besar memori flash menggunakan teknologi NAND.

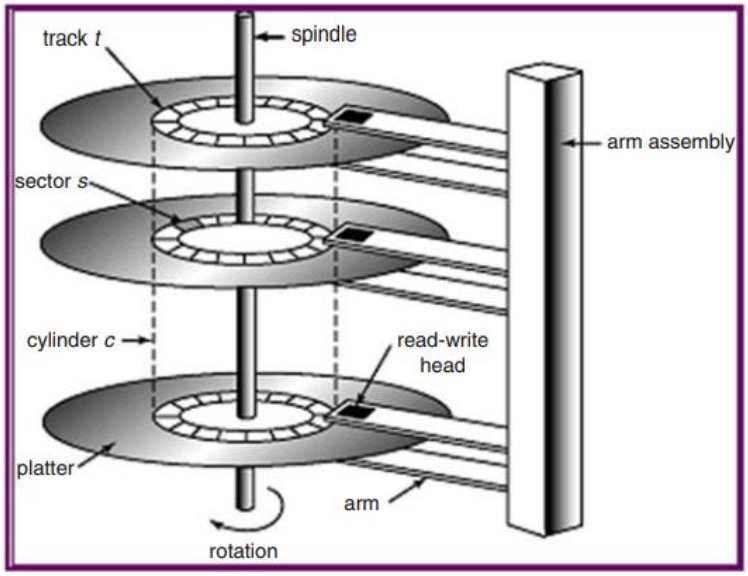
Memory Access Time Waktu CPU menempatkan alamat pada bus alamat dan data muncul pada bus data atau menulis data ke dalam memori. Tabel 7.2 menunjukkan waktu akses untuk berbagai jenis memori.

Tabel 7.2 Waktu akses memori

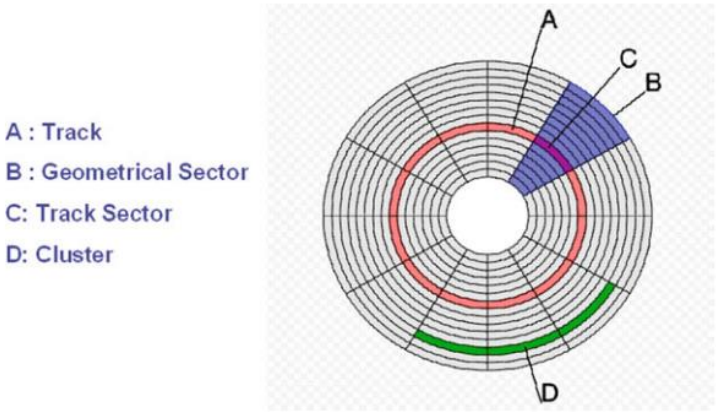
Teknologi memori	Access time
SRAM	0.5–2.5 ns
DRAM	50–70 ns
Kilatan	$5 * 10^3$–$5 * 10^5$

7.3 HHD dan SSD / (Hard Disk) dan (Solid-State Drive)

Gambar 7.7 menunjukkan arsitektur internal hard disk yang terbuat dari beberapa platter, dan platter menyimpan informasi; fungsi kepala adalah untuk membaca atau menulis informasi dari permukaan disk. Permukaan platter terbuat dari beberapa track, dan setiap track dibagi menjadi sektor-sektor seperti yang ditunjukkan pada Gambar 7.8.



Gambar 7.7 Arsitektur internal hard disk



Gambar 7.8 Permukaan piring

Karakteristik Disk

Access Time: waktu yang diperlukan untuk memulai transfer data, dan merupakan penjumlahan dari seek time dan rotational delay.

Seek Time: waktu yang membuat kepala bergerak ke jalur yang benar.

Rotational Delay: waktu yang dibutuhkan sektor untuk diposisikan di bawah read/write head dan bergantung pada kecepatan rotasi. Kecepatan rotasi yang diwakili oleh putaran per menit (RPM) mengasumsikan sektor ini jauh dari setengah bagian depan trek; oleh karena itu penundaan rotasi dihitung dengan

$$\textit{Penundaan Rotasi} = \textit{Waktu untuk setengah putaran} = 60s/RPM * 2$$

Kapasitas Disk: kapasitas disk yang dihitung dengan

$$\textit{Kapasitas Disk} = \textit{Jumlah permukaan} * \textit{Jumlah trek/Permukaan} * \textit{Jumlah sector/Track} * \textit{Jumlah byte/Sektor}$$

Gugus

Setiap sektor dari sebuah disk berukuran 512 byte (B), dan cluster terdiri dari satu atau lebih sektor, jika sebuah cluster berukuran 1 kB, maka dibuat dari dua sektor. Cluster 2 kB terdiri dari 4 sektor. Tabel 7.3 menunjukkan nilai default ukuran cluster.

Tabel 7.3 Ukuran klaster default

ukuran DISK	Ukuran cluster NTFS
512–1024 MB	1 kB
1024 MB–2 GB	2 kB
2 GB–2 TB	4 kB

Contoh 7.1 Sebuah disk drive memiliki 8 permukaan, setiap permukaan memiliki 1024 trek, setiap trek memiliki 64 sektor, dan setiap sektor dapat menampung 512 B dan kecepatan putaran 6000RPM.

(a) Berapa kapasitas disk?

(b) Berapakah penundaan rotasi?

Kapasitas disk = 8 * 1024 * 64 * 512 = 268.435.456 B

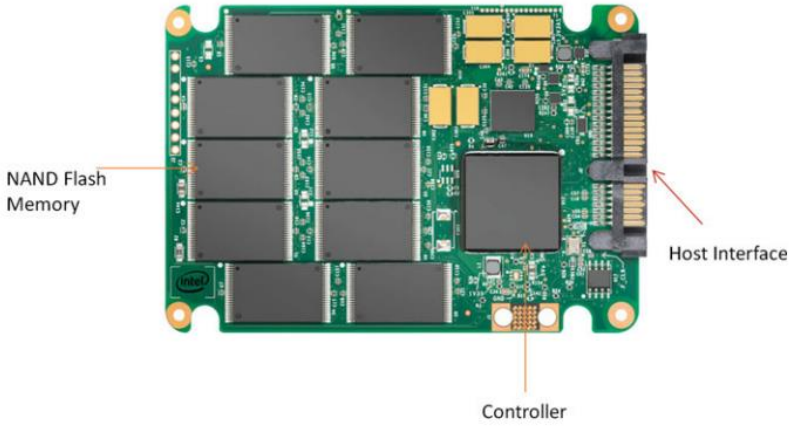
Penundaan rotasi = 60/6000 * 2 = 0,005 s

Sistem File Disk

Sistem file mendefinisikan organisasi informasi yang disimpan dalam hard disk; OS windows (sistem operasi) menawarkan FAT16 (tabel alokasi file) dan FAT32 yang digunakan untuk OS windows awal; saat ini sebagian besar OS windows menggunakan NTFS (New Technology File System). NTFS menawarkan keamanan yang lebih baik seperti izin untuk membatasi akses dan enkripsi.

Solid-State Drive (SSD)

Hard drive adalah perangkat yang lambat, dan dapat diganti dengan SSD. SSD terbuat dari memori flash NAND yang tidak mudah menguap. Gambar 7.9 menunjukkan arsitektur SSD, dan Tabel 7.4 membandingkan SSD dengan HDD.



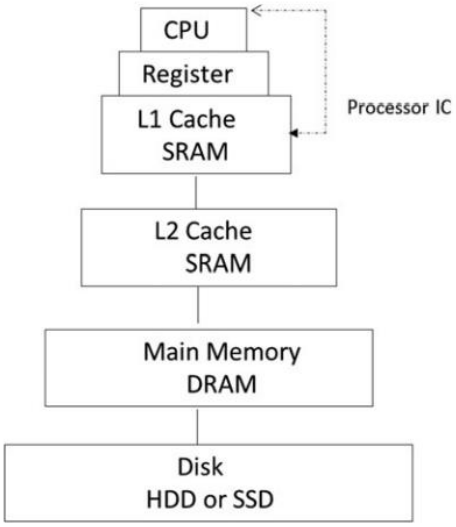
Gambar 7.9 Arsitektur SSD

Tabel 7.4 Membandingkan SSD dengan HDD

Karakteristik	SSD	HDD
Waktu akses	100 kali lebih cepat dari HDD	5000–10,000 μs
Harga	Mahal	Lebih murah
Keandalan	Lebih dapat diandalkan karena tidak memiliki bagian mekanis apa pun	Kurang dapat diandalkan
Kapasitas	Gigabytes	Terabyte
Kekuatan	Less power than HDD	More power than SSD

7.4 HIRARKI MEMORI

Komputer datang dengan tiga jenis memori, yang disusun secara hierarkis, seperti yang ditunjukkan pada Gambar 7.10:



Gambar 7.10 Hirarki memori komputer mikro

Tabel 7.5 Tampilkan harga berbagai jenis memori

Jenis memori	SRAM	DRAM	SSD	HDD
Biaya	$8.00/MB	$0.16/MB	$0.20/GB	0.05/GB
Waktu akses	0.5–2.5 ns	50–70 ns	70–150 ns	5–20 ms

1. *Memori cache:* Memori cache adalah jenis memori tercepat dan paling sering digunakan di dalam CPU yang disebut cache L1, dan lebih cepat daripada memori utama dan, oleh karena itu, lebih mahal daripada memori utama.
2. *Memori utama:* Memori utama menggunakan DRAM dan SDRAM. Program yang akan dieksekusi berpindah dari memori sekunder (disk atau tape) ke memori utama.
3. *Memori sekunder:* Memori kedua mengacu pada memori seperti hard disk, SSD, dan CD-ROM (Tabel 7.5).

Memori Cache

Setiap lokasi memori cache disebut jalur cache yang dapat menampung satu blok data dari memori utama. Untuk sebagian besar prosesor, memori cache terletak di dalam CPU dan disebut cache L1; Ada dua jenis cache dalam CPU:

1. Data cache (D-cache): Data cache menyimpan data dan dapat dibaca atau ditulis oleh CPU.
2. Cache instruksi (I-Cache): Cache instruksi menyimpan instruksi dan CPU hanya membaca dari I-cache.

Terminologi Cache

Miss: ketika CPU mengakses cache dan data tidak ada di cache, disebut cache miss

Hit: ketika CPU mengakses cache dan data ada di cache, maka disebut cache hit

Rasio hit: jumlah hit/jumlah miss number hit (jumlah total pembacaan) Blok: beberapa lokasi memori utama disebut blok

Alamat fisik: alamat yang dihasilkan oleh CPU untuk mengakses memori utama

Alamat virtual: alamat yang dihasilkan oleh CPU untuk mengakses memori virtual atau memori sekunder

Baris cache atau blok cache: masing-masing baris atau blok dapat menampung beberapa bye atau kata; ukuran garis cache adalah blok yang sama di memori utama

Lokalitas temporal: setelah lokasi memori direferensikan, maka ada kemungkinan tinggi untuk direferensikan lagi dalam waktu dekat

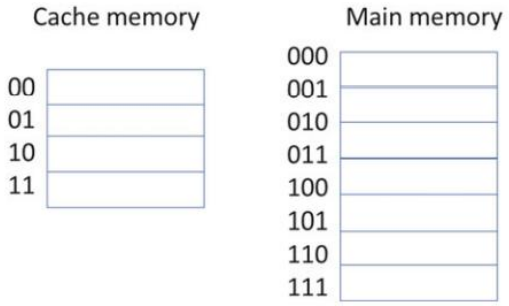
Lokalitas spasial: ketika suatu lokasi memori diakses, maka sangat mungkin lokasi terdekat akan segera diakses.

Metode Pemetaan Memori Cache

Gambar 7.11 menunjukkan cache dengan 4 lokasi dan memori utama dengan 8 lokasi memori, dan cache hanya dapat menampung 4 lokasi memori dari memori utama; CPU pertama-tama mengakses cache jika data tidak ada dalam cache dan kemudian mengakses memori utama dan memindahkan data blok ke dalam cache; pertanyaannya adalah di mana data akan disimpan dalam cache; ini membawa subjek metode pemetaan.

Metode pemetaan digunakan untuk memetakan blok memori utama ke dalam baris cache (blok cache), dan berikut adalah jenis metode yang digunakan untuk pemetaan:

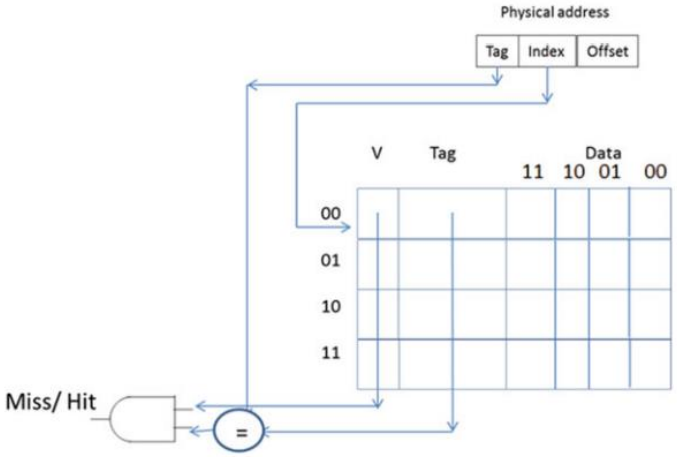
1. Pemetaan langsung
2. Pemetaan asosiatif
3. Tetapkan asosiatif



Gambar 7.11 Cache dan memori utama

Pemetaan Langsung

Gambar 7.12 menunjukkan memori cache dengan empat baris cache, dan setiap baris menampung 4 B; alamat fisik yang dilihat oleh cache dibagi menjadi tiga bidang dan mereka adalah:



Gambar 7.12 Memori cache dengan 4 baris cache

Offset: menentukan ukuran saluran cache (jumlah byte atau kata) karena setiap saluran cache dapat menampung 4 lokasi memori dan kemudian offsetnya adalah 2 bit, dan setiap blok adalah 4 lokasi *memori;* juga offset menentukan mana dari empat data yang akan ditransfer ke CPU.

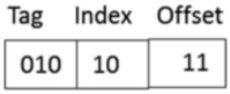
Indeks: indeks adalah alamat ke cache; pada gambar ini ada 4 baris cache dan kemudian indeks adalah 2 bit.

V-bit (valid bit): V-bit disetel ke satu untuk menyatakan bahwa data dalam baris cache valid.

Tag: Ukuran Tag = Ukuran Alamat Fisik — (Ukuran Indeks + Ukuran Offset).

Jika alamat fisik 7 bit, maka tag = 7 - (2 + 2) = 3 bit.

Jika bit yang valid adalah satu dan tag di bidang alamat cocok dengan tag yang disimpan dalam cache, maka hasilnya akan tercapai; sebaliknya adalah rindu.



Gambar 7.13 Format alamat fisik untuk 0101011

Ketika CPU menerima miss dari cache, maka akses memori utama dan transfer satu blok data dari memori utama ke jalur cache menggunakan persamaan berikut untuk pemetaan langsung:

Alamat baris cache = (Nomor Blok Memori Utama) Modulo N di mana N adalah jumlah baris cache.

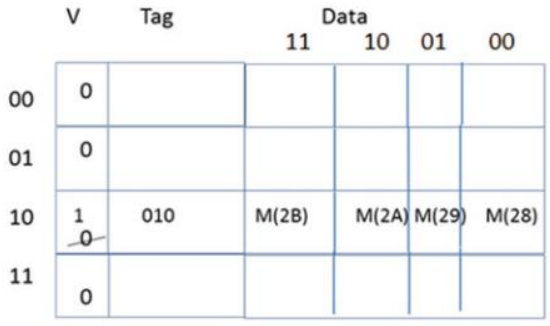
Perhatikan Gambar 7.13 cache dan asumsikan CPU menghasilkan alamat 010 10 11, Gambar 7.13 menunjukkan format alamat fisik yang dilihat oleh cache. CPU menggunakan nilai indeks untuk mengakses cache line 10 dan jika bit yang valid adalah nol menghasilkan miss, CPU mengakses memori utama dan mentransfer blok 01010 ke cache line 10 sesuai dengan persamaan berikut:

$$(01010)_2 = (10)_{10}$$

$$\text{Nomor baris cache} = 10 \bmod 4 = 2 \text{ atau } (10)_2$$

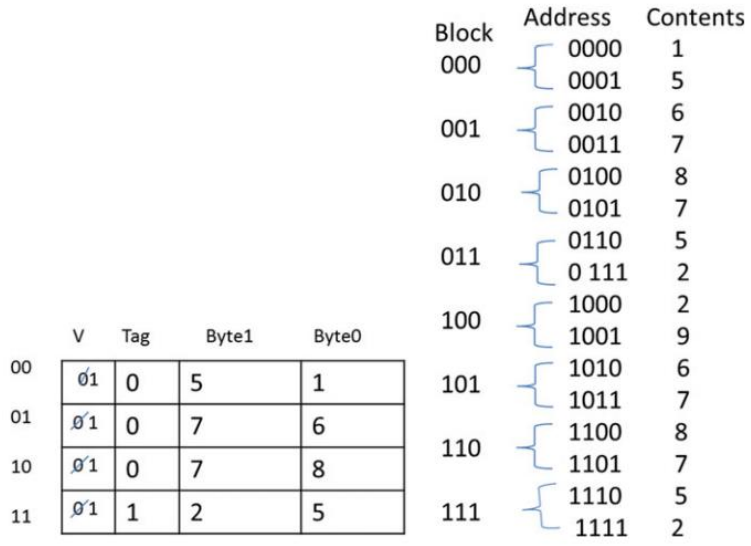
Dalam hal ini setiap blok terdiri dari 4 lokasi memori. Karena offset adalah dua bit, ia dapat merujuk salah satu dari 4 (2^2) potongan data dalam satu blok memori. Blok pada alamat 01010 di memori utama dengan offset 2 bit terdiri dari lokasi memori 0101000 (M28), 0101001 (M29), 0101010 (M2A), dan 0101011 (M2B).

Dalam hal ini, M28, M29, M2A, dan M2B ditransfer ke cache baris 10, dan V-bit diatur ke satu. Tag disimpan dalam bidang tag cache seperti yang ditunjukkan pada Gambar 7.14. Sekarang, jika CPU menghasilkan alamat 0101010, di mana tag = 010, indeks = 10, dan offset = 10, CPU menggunakan indeks untuk mengakses cache baris 10. Jika pada cache baris 10 V-bit sama dengan 1 dan tag alamat cocok dengan tag di baris cache, maka hasilnya adalah hit, dan CPU menggunakan offset 10 untuk memindahkan data (M2A) ke dalam CPU. (Di mana offset 00 = M28, 01 = M29, 10 = M2A, 11 = M2B.)



Gambar 7.14 Memori cache dengan empat baris (empat blok)

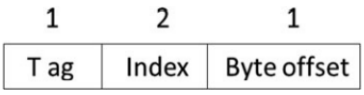
Contoh 7.2 Gambar 7.15 menunjukkan memori utama dan memori cache komputer. CPU menghasilkan (dalam hex) alamat 0 x 0, 0 x 2, 0 x 5, dan 0 x 2. Dengan asumsi cache kosong di awal, tampilkan isi cache.



Gambar 7.15 Memori cache dan memori utama Contoh 7.2

Dalam contoh ini setiap blok dan jalur cache adalah 2 B dan memori utama terdiri dari 8 blok. Gambar 7.16 menunjukkan alamat fisik seperti yang terlihat oleh cache.

- Offset byte adalah satu bit. (2 B per blok)
- Indeks adalah dua bit. (cache terdiri dari 4 = baris 2^2)
- Tag adalah satu bit. (Bit dalam blok - bit dalam indeks = 3 - 2 = 1)
- CPU menghasilkan alamat 0 x 0 atau 0000. Oleh karena itu, tagnya adalah 0, indeksnya adalah 00, dan offsetnya adalah 0. CPU mengakses cache line 00 dan V-bit adalah nol yang mengakibatkan miss. CPU mengakses alamat memori utama 0000, mentransfer blok 000 ke saluran cache 00, dan menetapkan bit tag ke nol dan V-bit ke satu. Oleh karena itu, baris cache 00 berisi: V = 1, Tag = 0, Byte1 = 0 x 5, Byte0 = 0 x 1.
- Selanjutnya CPU menghasilkan alamat 0 x 2 atau 0010. Indeksnya adalah 01 sehingga CPU mengakses cache line 01, dimana bit yang valid adalah nol yang mengakibatkan miss. Kemudian CPU mengakses lokasi memori utama 0010 dan mentransfer blok 001 ke baris cache 01, mengubah bit yang valid menjadi satu, dan menyimpan bagian tag dari alamat ke dalam tag baris cache. Oleh karena itu, baris cache 01 sekarang: V = 1, Tag = 0, Byte1 = 0 x 7, Byte0 = 0 x 6.
- CPU menghasilkan alamat 0 x 5 atau 0101. Indeksnya 10, sehingga CPU mengakses cache baris 10 dan hasilnya miss. CPU mengakses lokasi memori utama 0101 dan mentransfer blok 010 ke cache, menetapkan V ke satu, dan menyimpan tag alamat ke dalam tag cache. Cache baris 10 sekarang: V = 1, Tag = 0, Byte1 = 0 x 7, Byte0 = 0 x 8.
- CPU menghasilkan alamat 0 x 2 atau 0010, sehingga mengakses baris cache 01. Bit yang valid adalah 1 dan tag baris cache cocok dengan tag alamat, yang menghasilkan hit. Offsetnya adalah 0, sehingga CPU membaca byte 0 dari cache line 01. Proses ini akan dilanjutkan untuk alamat lain.



Gambar 7.16 Format alamat fisik untuk Contoh 7.1

Menggunakan cache dan memori utama dengan ukuran yang sama seperti di atas, blok 000 dan 100 keduanya akan dipetakan ke baris cache 00. Jika CPU menghasilkan alamat 0000, 1000, 0001, dan 1001 secara berurutan, maka hasilnya akan hilang untuk keempat alamat tersebut. Untuk mengurangi misses, maka cache dapat dibagi menjadi set, dan metode pemetaannya disebut set associative mapping.

Tetapkan Pemetaan Asosiatif

Dalam pemetaan asosiatif set, memori cache dibagi menjadi set. Ukuran himpunan bisa bermacam-macam: contohnya termasuk himpunan dua arah, himpunan empat arah, dan seterusnya. Gambar 7.17 menunjukkan cache asosiatif set dua arah, dan Gambar 7.18 menunjukkan format alamat fisik yang dilihat oleh cache. Menggunakan contoh sebelumnya sebagai basis, byte offset akan tetap satu bit karena mengacu pada salah satu dari dua byte yang disimpan pada baris cache. Pengenal yang ditetapkan akan menjadi satu bit yang mendefinisikan alamat yang ditetapkan; ukuran tag dihitung sebagai berikut:

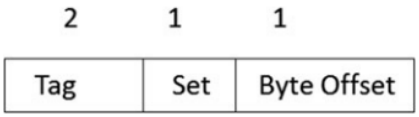
Ukuran Tag = Ukuran alamat fisik — Setel alamat — Offset

Asumsikan alamat fisik adalah 4 bit, maka Gambar 7.18 menunjukkan alamat fisik yang dilihat oleh cache.

	V	Tag	Byte1	Byte0
0	0			
1	0			

V	Tag	Byte1	Byte0
0			
0			

Gambar. 7.17 Cache asosiatif set dua arah



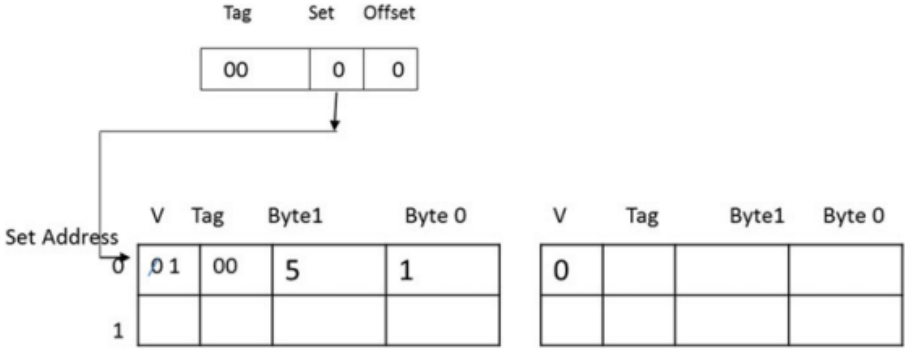
Gambar 7.18 Alamat fisik dilihat oleh cache untuk pemetaan asosiatif yang ditetapkan

Contoh 7.3 Pertimbangkan memori utama dari Gambar 7.15, di mana cache kosong dan dibagi menjadi dua set seperti yang ditunjukkan pada Gambar 7.19. Tampilkan isi cache jika CPU menghasilkan alamat 0 × 0, 0 × 8, 0 × 0, dan 0 × 8.

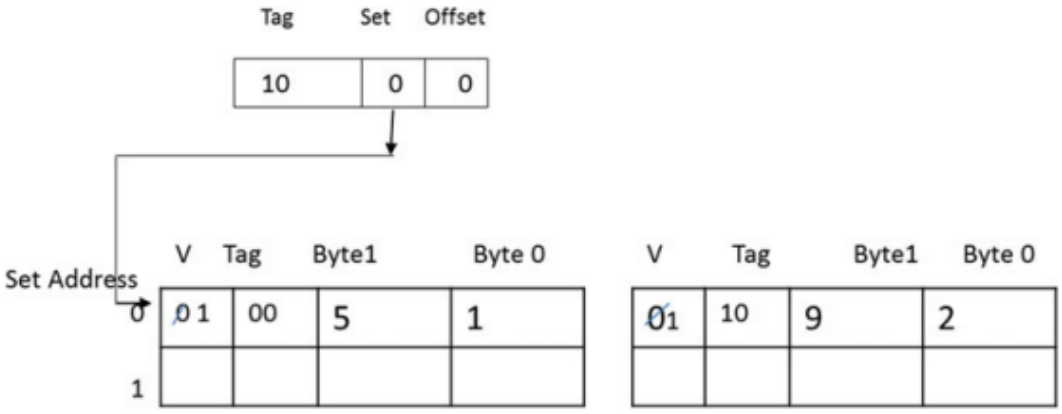
- CPU menghasilkan alamat 0 x 0 atau 0000 dan mengakses cache set 0. Kedua baris cache di set 0 memiliki bit valid 0, sehingga hasilnya miss. CPU kemudian mengakses memori utama, mentransfer isi lokasi memori 0000 dan 0001 ke dalam cache, kemudian mengubah bit yang valid menjadi satu, dan menyimpan bagian tag dari alamat ke dalam tag cache.

Selanjutnya, CPU menghasilkan alamat 0 x 8 atau 1000 dan mengakses set cache 0. Baris pertama cache memiliki bit 1 yang valid tetapi tagnya tidak cocok (10 vs. 00). CPU kemudian

mengakses memori utama, mentransfer isi lokasi memori 1000 dan 1001 ke baris kedua cache di set 0, dan mengubah bit yang valid menjadi satu dan menyimpan bagian tag dari alamat ke dalam tag cache. Sekarang, ketika CPU menghasilkan alamat 0 × 0 dan 0 × 8 lagi hasilnya hits.



Gambar 7.19 Isi cache untuk alamat 0000

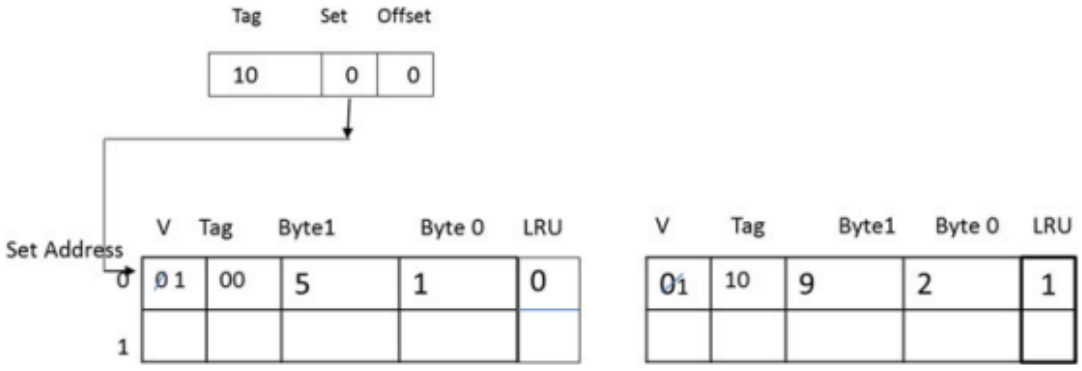


Gambar 7.20 Isi cache untuk alamat 0000 dan 1000

Metode Penggantian

Dalam himpunan asosiatif, ketika CPU membawa blok baru ke dalam cache, maka salah satu baris cache harus diganti dengan blok baru; perhatikan Gambar 7.20. Jika CPU menghasilkan alamat 0100 (4) (tag = 01, set alamat = 0, dan offset adalah 0), ia mengakses set 0, dan kedua jalur cache memiliki bit satu yang valid, tetapi tag pada baris cache tidak cocok dengan tag alamat hasil miss, dan kemudian CPU mengakses memori utama dan harus memindahkan isi lokasi memori 0100 dan 0101 ke dalam cache. CPU menggunakan metode least recent used (LRU) yang memindahkan blok baru dari memori utama dan menggantinya dengan blok yang lebih lama di cache. Metode ini diselesaikan dengan menambahkan bit LRU ke setiap baris cache dari cache seperti yang ditunjukkan pada Gambar 7.21.

Pada Gambar 7.21, asumsikan kedua baris cache di set 0 kosong, blok baru pindah ke baris cache pertama di set 0, dan LRU berubah dari 0 ke 1; blok kedua pindah ke baris cache kedua set 0, dan LRU berubah dari 0 ke 1, tetapi pada saat yang sama, LRU baris cache pertama akan berubah dari 1 ke 0; oleh karena itu, baris cache dengan LRU = 0 berisi blok yang lebih panjang di baris cache.



Gambar 7.21 Dua himpunan asosiatif dengan LUR

Pemetaan Asosiatif Sepenuhnya

Dalam pemetaan asosiatif penuh, seluruh alamat disimpan dalam cache dengan datanya. Gambar 7.22 menunjukkan cache asosiatif penuh dengan empat baris setelah CPU mengakses lokasi memori utama 0 x 0 dan 0 x 8. Jika CPU selanjutnya menghasilkan alamat 0 x 0 atau 0000, maka CPU akan membandingkan alamat dengan setiap alamat dalam cache, dan jika cocok, maka ia akan membaca data dari cache.

Valid	Address	Byte1	Byte0
1	*000*	5	1
1	*100*	9	2
0			
0			

Gambar 7.22 Pemetaan asosiatif

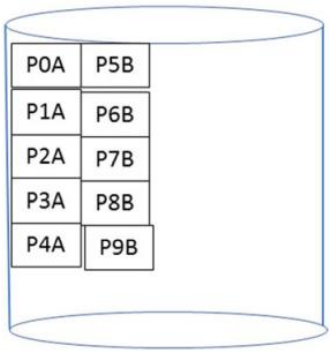
Metode Pembaruan Cache

1. *Write Through*: Ketika informasi baru ditulis ke cache, memori utama juga diperbarui.
2. *Buffered Write Through:* Ada buffer antara cache dan memori utama, ketika informasi baru ditulis ke cache, informasi ini ditulis ke buffer, dan CPU dapat mengakses memori ini sebelum informasi baru dapat ditulis ke memori utama.
3. *Write Back (Copy Back):* Dalam metode ini, hanya cache yang diperbarui, dan memori utama akan diperbarui ketika baris cache yang sesuai ditimpa. Dalam metode ini setiap baris cache memiliki bit kotor untuk menunjukkan apakah baris cache telah dimodifikasi atau tidak.

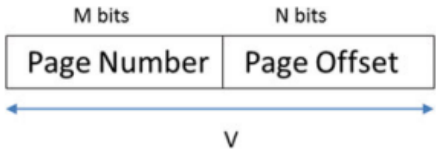
Memori Virtual

Memori virtual adalah HDD atau SSD; digunakan untuk menyimpan data aplikasi dan instruksi yang saat ini tidak diperlukan untuk diproses oleh CPU. Memori virtual memungkinkan sistem untuk menjalankan aplikasi yang lebih besar dari memori utama. Disk dilihat oleh CPU sebagai memori virtual, jika CPU memiliki 16 baris alamat, maka ukuran memori virtual menjadi 216 B. Aplikasi berada di disk dan disebut proses. Ketika pengguna menjalankan program, sistem operasi memindahkan halaman proses ke memori utama. Memori virtual dibagi menjadi halaman-halaman seperti yang ditunjukkan pada Gambar 7.23; proses A menempati halaman P0 hingga P4, dan proses B menempati halaman P5–P9.

CPU menghasilkan alamat virtual (untuk mengakses alamat di disk) dari V-bit. Bit-bit ini dibagi menjadi dua pengidentifikasi: nomor halaman virtual dari M bit dan offset halaman dari N bit seperti yang ditunjukkan pada Gambar 7.24. Jumlah total halaman dalam sistem sama dengan 2^M, dan jumlah byte (dalam sistem byte-addressable) dalam satu halaman adalah sama dengan 2^N. Jumlah total alamat dalam suatu sistem sama dengan $2^{(N+M)}$ atau 2^V.



Gambar 7.23 Memori virtual



Gambar 7.24 Format alamat virtual

Contoh 7.4 Kapasitas disk virtual adalah 2 MB (megabyte), dan setiap halaman adalah 2 kB (kilobyte) dalam sistem byte-addressable.

(a) Berapakah nilai N dan M?

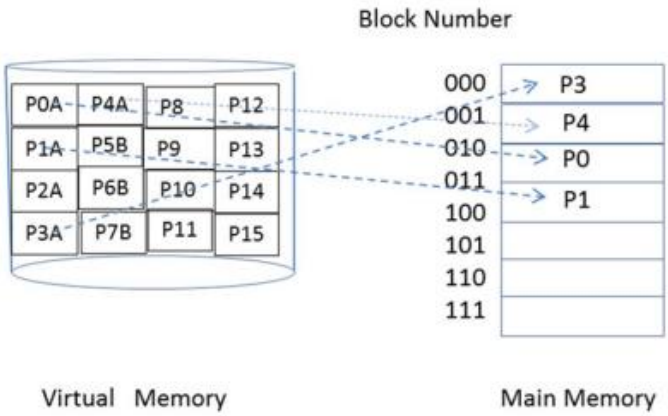
(b) Berapa banyak halaman dalam disk?

Karena setiap halaman 2 kB, 2^N = 2048 B, artinya N sama dengan 11.

Kapasitas disk adalah 2 MB. 2^V = 2 M sama dengan 2^{21} jadi V = 21 bit.

Jadi, jumlah halaman sama dengan $2^{(21-11)}$. Disk berisi 210 atau 1024 halaman, dan ukuran setiap halaman adalah 2 kB.

Tabel Halaman Dengan memori utama yang dibagi menjadi beberapa blok, ukuran setiap blok (atau bingkai) sama dengan ukuran halaman. Ketika CPU mentransfer halaman ke memori utama, itu mencatat nomor halaman dan blok yang sesuai di tabel halaman. Baris alamat tabel halaman adalah nomor halaman. Setiap baris berisi nomor bingkai atau blok dari lokasi yang cocok di memori utama dan bit valid yang menunjukkan apakah jalur tersebut valid atau tidak. Gambar 7.25 menunjukkan tabel halaman dimana halaman P0, P1, P3, dan P4 dipindahkan ke blok 2, 3, 0, dan 1 secara berurutan.



Page Number V Block Number

0000	1	010
0001	1	011
0010	0	
0011	1	000
0100	1	001
0101	0	
0110	0	
0111	0	
1000	0	
1001	0	
1010	0	
1011	0	
1100	0	
1101	0	
1110	0	
1111	0	

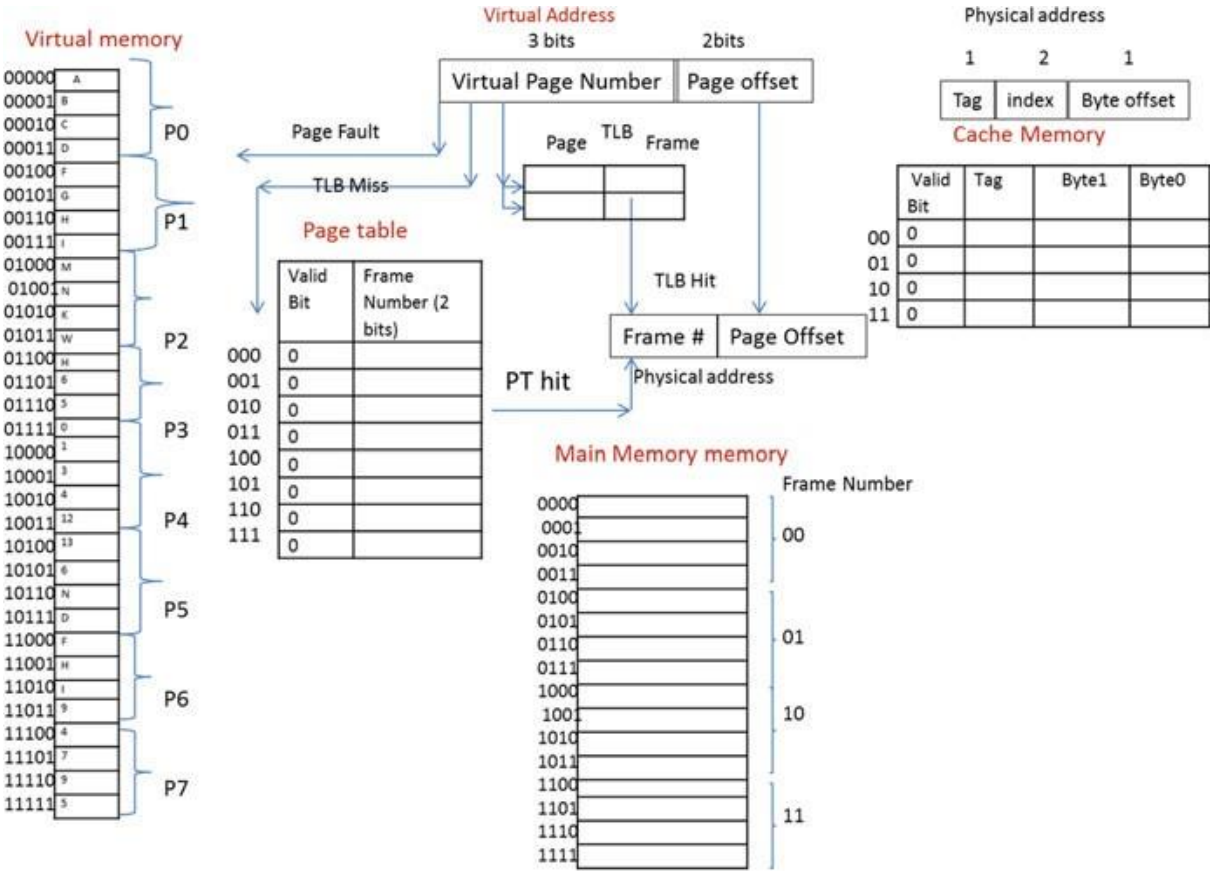
Gambar 7.25 Memori virtual, memori, dan tabel halaman

Setiap proses memiliki tabel halaman sendiri yang disimpan di memori utama. Karena memori cache lebih cepat, bagian dari tabel halaman yang disebut sebagai translation lookaside buffer (TLB) disimpan dalam cache. TLB menggunakan pemetaan asosiatif.

Organisasi Memori Komputer

Gambar 7.26 menunjukkan organisasi memori komputer yang dalam contoh ini terdiri dari:

A. Memori virtual (hard disk atau solid-state drive)
B. Memori utama (Jenis DRAM)
C. Memori cache (SRAM)
D. Tabel halaman yang melacak halaman di memori utama
E. TLB yang menampung bagian dari tabel halaman



Gambar 7.26 Organisasi memori komputer

Operasi Memori

Langkah-langkah berikut menjelaskan pengoperasian memori komputer. Pertama, CPU menghasilkan alamat virtual dan memeriksa TLB untuk melihat apakah halaman yang sesuai sudah ada di memori utama atau belum.

A. Jika TLB menunjukkan bahwa halaman yang sesuai ada di memori, buat alamat fisik dan periksa apakah data ada di cache.
   (a) Jika ada di cache, maka ini disebut hit dan membaca data dari cache
   (b) Jika tidak ada dalam cache, maka disebut miss, dan CPU mengakses memori dan memindahkan blok data ke dalam cache kemudian membacanya.

B. Jika halaman yang sesuai tidak ada di TLB, maka CPU memeriksa tabel halaman.
   1. Jika halaman yang sesuai ada di memori utama, perbarui TLB dan ulangi dari langkah 1.
   2. Jika halaman terkait tidak ada di memori utama, pindahkan halaman dari memori virtual ke memori utama, perbarui tabel halaman, perbarui TLB, dan ulangi dari langkah 1.

C. Jika halaman yang sesuai tidak ada dalam tabel halaman, maka CPU memindahkan halaman dari memori virtual ke memori utama, memperbarui tabel halaman, memperbarui TLB, dan mengulangi dari langkah 1.

Pertanyaan dan Masalah

1. Bedakan antara memori yang mudah menguap dan yang tidak mudah menguap.

2. Apa singkatan dari RAM?
3. Sebutkan tiga jenis RAM yang berbeda.
4. Manakah dari jenis memori berikut yang digunakan untuk memori utama?
   (a) ROM dan SDRAM
   (b) SRAM dan DRAM
   (c) SDRAM dan DRAM
   (d) DRAM dan EPROM
5. _____ menyimpan informasi secara permanen, bahkan ketika tidak ada daya.
   (a) ROM
   (b) DRAM
   (c) RAM
   (d) SRAM
6. Apa singkatan dari ROM?
7. Apakah memori flash sejenis RAM atau ROM?
8. Apa perbedaan antara EEPROM dan EPROM?
9. Berapa kapasitas memori dengan 10 baris alamat dan menampung satu byte per lokasi memori?
10. Apa aplikasi utama SRAM?
11. Apa aplikasi utama DRAM?
12. Definisikan istilah-istilah berikut:
    (a) jalur
    (b) Sektor
    (c) Gugus
13. Sebuah hard disk terdiri dari 4 permukaan, setiap permukaan terdiri dari 80 track, dan setiap track terdiri dari 32 sektor. Setiap sektor menampung 512 B. Berapa kapasitas disk ini?
14. Apa fungsi dari tabel alokasi file (FAT)?
15. Sebutkan jenis-jenis memori di komputer dari yang tercepat hingga yang paling lambat.
16. Apa saja jenis-jenis cache?
17. Jenis memori apa yang digunakan untuk memori cache?
18. Apa itu memori virtual?
19. Bedakan antara alamat virtual dan alamat fisik.
20. Alamat fisik menentukan ukuran
    (a) Memori virtual
    (b) Memori fisik
    (c) Memori cache
21. Menampilkan format alamat virtual.
22. Apa itu rasio hit?
23. Jelaskan lokalitas temporal.
24. Menjelaskan lokalitas spasial.
25. Daftar metode pemetaan cache.

26. Tampilkan format alamat yang dilihat oleh cache untuk pemetaan langsung.
27. Daftar metode pemetaan cache.
28. Tampilkan format alamat yang dilihat oleh cache untuk pemetaan asosiatif yang ditetapkan.
29. Apa fungsi nomor halaman pada alamat virtual?
30. Berapa bit offset halaman jika setiap halaman menampung 8 kB?
31. Apa fungsi dari tabel halaman?
32. Informasi apa yang disimpan di TLB? Di mana TLB disimpan?
33. Daftar metode pemetaan cache.
34. Apa keuntungan dari pemetaan himpunan asosiatif versus pemetaan cache langsung?
35. Apa tiga kebijakan tulis yang digunakan untuk memori?
36. _____ adalah jenis memori tercepat.
    (a) Memori cache
    (b) Memori utama
    (c) Memori sekunder
    (d) Harddisk

Masalah

1. Memori utama dan cache berikut diberikan. CPU menghasilkan alamat 0 × 0, 0 × 2, 0 × 3, 0 × 4, 0 × 5, 0 × 3, 0 × 6, 0 × 6, 0 × 7, 0 × B, 0 × D, dan 0 × F. Tampilkan isi cache dan temukan rasio hit.

ADDRESS	Contents
0000	5
0001	0
0010	1
0011	11
0100	15
0101	09
0110	16
0111	23
1000	65
1001	01
1010	8
1011	9
1100	15
1101	0
1110	2
1111	5

v	Tag	Data

2. Memori berikut dan memori cache diberikan. CPU menghasilkan alamat 0 × 1, 0 × 2, 0 × 1, 0 × 8, 0 × 9, 0 × 1C, 0 × 1D, 0 × 3, dan 0 × 4.
   (a) Tampilkan isi cache menggunakan pemetaan asosiatif set dua arah; mengambil kebijakan penggantian LRU.
   (b) Berapa hit ratenya?

Set Address	V	Tag	B1	B0	LRU	V	Tag	B1	B0	LRU
00	0				0	0				0
01	0				0	0				0
10	0				0	0				0
11	0				0	0				0

Address	Content	Address	Content
00000	5	10000	5
00001	3	10001	0
00010	11	10010	1
00011	6	10011	11
00100	7	10100	15
00101	8	10101	09
00110	9	10110	12
00111	12	10111	23
01000	0	11000	65
01001	0	11001	21
01010	8	11010	8
01011	7	11011	7
01100	9	11100	9
01101	0	11101	0
01110	2	11110	2
01111	5	11111	5

3. Komputer memiliki alamat fisik 24 bit dan setiap lokasi memori menampung satu byte. Komputer ini memiliki 64 baris cache dan setiap baris menampung 16 B. Tunjukkan format alamat (tag, indeks, dan offset byte) menggunakan
   (a) Pemetaan langsung
   (b) himpunan 4-arah asosiatif
   (c) himpunan asosiatif 8 arah
4. Sebuah komputer memiliki memori virtual 32 kB dan memori utama 8 kB dengan ukuran halaman 512 B.
   (a) Berapa banyak bit dalam alamat virtual?
   (b) Berapa banyak halaman dalam memori virtual?
   (c) Berapa banyak bit yang diperlukan untuk alamat fisik?
   (d) Berapa banyak frame atau blok dalam memori utama?
5. Komputer dengan memori virtual 256 MB, memori utama 4 MB, dan memori cache 8 kB. Asumsikan ukuran halaman 2 kB.
   (a) Berapa ukuran alamat virtual?

(b) Berapa ukuran alamat fisik?
(c) Berapa banyak halaman dalam memori virtual?
(d) Berapa banyak blok yang ada di memori utama?
(e) Berapa ukuran tabel halaman? (Sertakan jumlah lokasi dan ukuran total setiap lokasi termasuk semua informasi.)

6. Sebuah komputer memiliki 20 bit memori virtual dan setiap halaman adalah 2 kB.
   (a) Berapa ukuran memori virtual?
   (b) Berapa banyak halaman dalam memori virtual?
7. Komputer dengan 4 kata per blok memiliki 4 K blok cache dan 1 M blok memori utama.
   (a) Berapa ukuran alamat fisik?
   (b) Tentukan ukuran tag, indeks, dan offset kata dari alamat fisik menggunakan pemetaan langsung.
   (c) Tentukan ukuran tag, himpunan, dan offset kata dari alamat fisik menggunakan pemetaan asosiatif himpunan dua arah.
8. CPU pada Gambar 7.26 menghasilkan alamat 0 00 dan 0 0b; asumsikan halaman 0 dipetakan ke blok 1 dan halaman 2 dipetakan di blok 0, tampilkan isi tabel halaman.

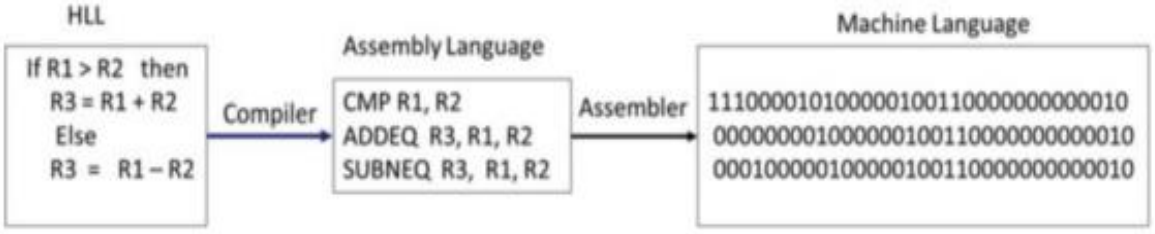
BAB 8
BAHASA ASSEMBLY DAN INSTRUKSI ARM BAGIAN I

Tujuan: Setelah menyelesaikan bab ini, Anda diharapkan dapat:

- Menjelaskan fungsi compiler dan assembler.
- Konversikan HLL ke bahasa mesin.
- Tampilkan arsitektur prosesor ARM.
- Mendeskripsikan fungsi processor state register (PSR).
- Daftar klasifikasi instruksi berdasarkan jumlah operan.
- Pelajari berbagai jenis instruksi ARM.
- Jelaskan pengoperasian instruksi bersyarat.
- Konversikan HLL ke bahasa rakitan.
- Jelaskan instruksi shift dan rotasi.
- Menjelaskan pengoperasian instruksi stack.
- Menjelaskan penerapan instruksi Cabang.

8.1 PENDAHULUAN

Pemrogram menggunakan bahasa tingkat tinggi untuk mengembangkan program aplikasi; agar program menjadi bentuk yang dapat dieksekusi, itu harus dikonversi dalam kode mesin (biner). Gambar 8.1 menunjukkan bahasa tingkat tinggi (HLL) yang dikonversi ke kode mesin, kompiler mengubah HLL menjadi bahasa rakitan, dan kemudian assembler mengubah bahasa rakitan ke bahasa mesin (bit) oleh assembler. Setiap CPU memiliki seperangkat instruksi yang mewakili jenis operasi yang dapat dilakukan CPU, dan instruksi ini direpresentasikan dalam bentuk mnemonic atau singkatan, misalnya, instruksi penambahan diwakili oleh "ADD", dan instruksi pengurangan diwakili oleh "SUB .” ADD R1, R2, R3 berarti menambahkan isi R2 dengan register R3 dan menyimpan hasilnya di register R1. R1, R2, dan R3 disebut operand.



Gambar 8.1 Mengonversi HLL ke bahasa mesin

HLL berikut dikonversi ke bahasa rakitan:
Bahasa Majelis HLL

```
R3=R1+R2 ADD R3, R1, R2

R3=R1-R2 SUB R3, R1, R2
```

Programmer menggunakan instruksi untuk menulis bahasa assembly. Aplikasi dari bahasa assembly adalah:

- Bahasa assembly digunakan untuk menulis kode tercepat.
- Ini membantu untuk lebih memahami HLL.
- Menulis compiler untuk HLL membutuhkan pengetahuan tentang bahasa assembly.
- Digunakan dalam sistem dan driver tertanam.
- HLL mungkin tidak menyediakan akses ke perangkat keras maka bahasa rakitan dapat digunakan.

8.2 INSTRUCTION SET ARCHITECTURE (ISA)

Produsen CPU menerbitkan dokumen yang berisi informasi tentang prosesor seperti daftar register, fungsi setiap register, ukuran bus data, ukuran bus alamat, dan daftar instruksi yang dapat dieksekusi oleh CPU. Setiap CPU memiliki set instruksi yang diketahui yang dapat digunakan programmer untuk menulis program bahasa assembly. Set instruksi khusus untuk setiap jenis prosesor. Prosesor Pentium menggunakan set instruksi yang berbeda dari prosesor ARM. Instruksi diklasifikasikan berdasarkan jumlah operan atau jenis operasi.

Klasifikasi Instruksi Berdasarkan Jumlah Operand

Tidak ada Instruksi Operand. Berikut ini adalah beberapa instruksi yang tidak memerlukan operan:

HLT **Hentikan CPU**
NOP **Tidak ada operasi**

PUSH *operan*: Dorong operan ke atas tumpukan
POP *operan*: Hapus operan dari atas tumpukan

Instruksi One-Operand Berikut ini adalah beberapa instruksi yang membutuhkan satu operand.

INC	operan	Contoh: INC R1 – Kenaikan register R1 sebesar 1
DEC	operan	Contoh: DEC R1 – Turunkan register R1 sebanyak 1
J	target	Lompat ke lokasi memori yang diberi label oleh target
ADD	operan	Tambahkan operan ke akumulator (ACC) ACC → ACC + operan

Instruksi Dua Operand Berikut ini adalah beberapa instruksi yang membutuhkan dua operan.

```
ADD Rd, Rn Example: ADD R1, R2 - R1←R1+R2
```

Arsitektur Set Instruksi Intel menggunakan dua operan.

```
MOV EAX, EBX ; EAX ← EBX
```

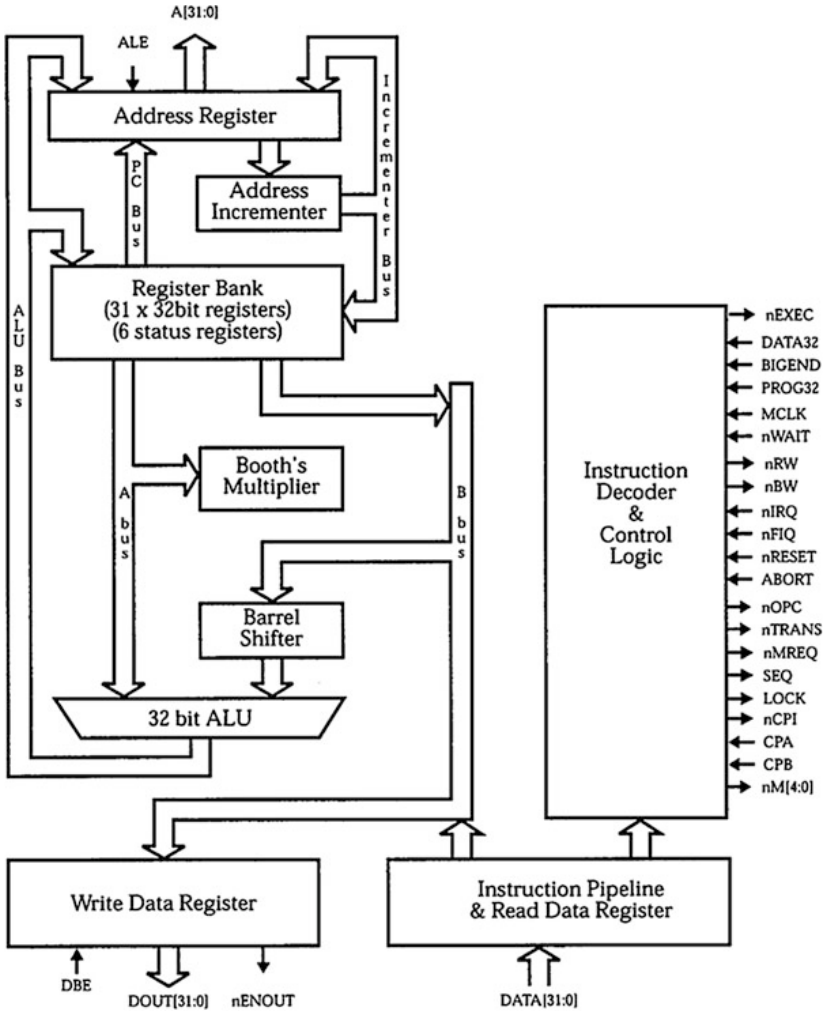
Instruksi Tiga Operasi Kebanyakan prosesor modern menggunakan instruksi dengan tiga operan, seperti ARM, MIPS, dan Itanium.

```
ADD R1, R2, R3 ; R1← R2 +R3
```

8.3 ARSITEKTUR PROSESOR, REGISTER DAN INTRUKSI ARM

Advanced RISC Machine (ARM) dikembangkan oleh Perusahaan Acorn. ARM adalah pemasok pemimpin mikroprosesor di dunia, ARM mengembangkan CPU inti, dan ribuan pemasok menambahkan lebih banyak unit fungsional ke inti. ARM menggunakan dua jenis instruksi yang disebut Thumb dan Thumb-2. Instruksi jempol adalah 16 bit dan instruksi jempol-2 adalah 32 bit; saat ini sebagian besar prosesor ARM menggunakan instruksi 32-bit. ARM berisi 15 register yang disebut R0 hingga R15, R0 hingga R12 yang disebut register usulan umum. ARM mampu mengeksekusi instruksi Thumb (instruksi 16-bit) dan instruksi Thumb-2 32 bit. Instruksi ibu jari menggunakan register R0 sampai R7.

ARM ditujukan untuk aplikasi yang membutuhkan prosesor hemat daya, seperti telekomunikasi, komunikasi data (protocol converter), instrumen portabel, komputer portabel, dan kartu pintar. ARM pada dasarnya adalah prosesor RISC 32-bit (bus data 32-bit dan bus alamat) dengan respons interupsi yang cepat untuk digunakan dalam aplikasi waktu nyata. Diagram blok prosesor ARM7 ditunjukkan pada Gambar 8.2.



Gambar 8.2 Blok diagram arsitektur ARM7

- **Dekoder Instruksi dan Kontrol Logika** Fungsi dekoder instruksi dan kontrol logika adalah untuk mendekode instruksi dan membangkitkan sinyal kontrol ke bagian lain dari prosesor untuk mengeksekusi instruksi.

- **Address Register** Untuk menyimpan alamat 32-bit untuk bus alamat
- **Address Increment** Digunakan untuk menambah alamat menjadi empat dan menempatkannya di address register.
- **Register Bank** Register bank berisi tiga puluh satu register 32-bit dan 6 register status.
- **Barrel Shifter** Digunakan untuk operasi shift cepat.
- **ALU** 32-bit ALU digunakan untuk operasi aritmatika dan logika.
- **Write Data Register** Prosesor memasukkan data ke Write Data Register untuk operasi tulis.
- **Baca Register Data** Ketika prosesor membaca dari memori, ia menempatkan hasilnya dalam register ini.

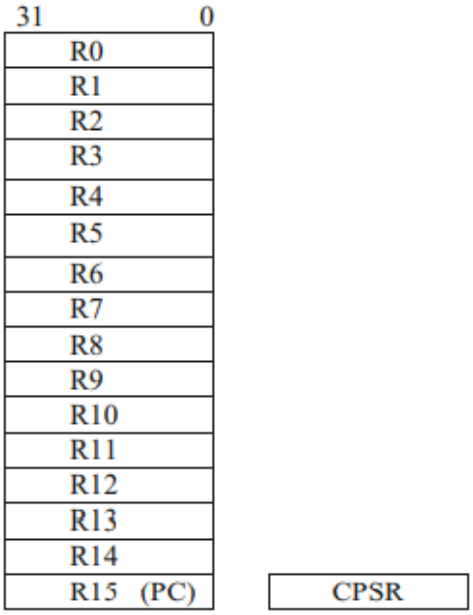
Mode Operasi ARM

ARM dapat beroperasi dalam salah satu mode berikut:

1. *Mode pengguna:* Gunakan untuk operasi normal.
2. *Mode IRQ:* Mode interupsi ini dirancang untuk menangani operasi interupsi.
3. *Mode pengawasan:* Digunakan oleh sistem operasi.
4. *Mode FIQ:* Mode interupsi cepat.
5. *Mode tidak terdefinisi:* Ketika instruksi yang tidak terdefinisi dieksekusi.
6. *Abort mode:* Mode ini menunjukkan bahwa akses memori saat ini tidak dapat diselesaikan, seperti ketika data tidak ada dalam memori dan prosesor memerlukan lebih banyak waktu untuk mengakses disk dan mentransfer blok data ke memori.

Register ARM

ARM7 memiliki 31 register umum dan 6 register status. Pada mode pengguna, hanya 16 register dan 1 Program Status Register (PSR) yang tersedia untuk pemrogram. Register diberi label R0 sampai R15. R15 digunakan untuk program counter (PC), R14 digunakan untuk link register, dan R13 digunakan untuk stack pointer (SP). Gambar 8.3 menunjukkan register mode pengguna.



Gambar 8.3 Register mode pengguna

Daftar Status Program Saat Ini (CPSR). Gambar 8.4 menunjukkan format PSR. Register ini digunakan untuk menyimpan bit kontrol dan bit flag. Bit flagnya adalah N, Z, C, dan V, dan bit kontrolnya adalah I, F, dan M0 sampai M4. Bit bendera dapat diubah selama operasi logika, aritmatika, dan perbandingan.

31	30	29	28	27	7	6	5	4	3	2	1	0
N	Z	C	V	Unused	I	F	T	M4	M3	M2	M1	M0

Gambar. 8.4 Format penyimpanan untuk CPSR

Tandai Bit

N (negatif): N = 1 berarti hasil operasi negatif, dan N 0 berarti hasil operasi positif.

Z (nol): Z = 1 berarti hasil operasi adalah nol, dan Z 0 hasil operasi tidak nol.

C (carry): C = 1 berarti hasil operasi menghasilkan carry, dan C 0 berarti hasil operasi tidak menghasilkan carry.

V (overflow): V = 1 berarti hasil operasi menghasilkan overflow, dan V 0 berarti hasil operasi tidak menghasilkan overflow.

Kontrol Bit

I (interrupt bit): Ketika bit ini disetel ke satu, itu akan menonaktifkan interupsi, dan ini berarti prosesor tidak menerima interupsi perangkat lunak apa pun.

F-bit digunakan untuk menonaktifkan dan mengaktifkan mode fast interrupt request mode (FIQ).

M4, M3, M2, M1, dan M0 adalah bit mode, dan sama dengan 10000 untuk mode pengguna.

T (State bit): T = 1 Prosesor mengeksekusi instruksi Thumb, prosesor T = 0 mengeksekusi instruksi ARM.

Instruksi ARM

Arsitektur ARM mendukung set instruksi Thumb 16-bit dan Thumb-2 32-bit. Sebagian besar instruksi ARM menggunakan tiga operand. Instruksi ini diklasifikasikan berdasarkan format instruksi dan operasinya yang didaftar sebagai berikut:

A. Instruksi pemrosesan data
B. Pertukaran data tunggal
C. Menggeser dan memutar instruksi
D. Instruksi tanpa syarat dan instruksi bersyarat
E. Operasi tumpukan
F. Cabang
G. Kalikan instruksi
H. Transfer data

Instruksi Pemrosesan Data

Petunjuk pengolahan data adalah sebagai berikut: AND, EOR, SUB, RSB, ADD, ADC, SBC, RSB TST, TEQ, CMP, CMN, ORR, MOV, BIC, dan MNW. Data instruksi pemrosesan menggunakan operan register dan operan langsung. Format umum instruksi pemrosesan data adalah:

Mnemonic {S}{Condition} Rd, Rn, operand2

Mnemonic: Mnemonic is abbreviation of an operation such as ADD for addition

{}: Commands inside the { } is optional such as S and condition

S: When an instruction contains S mean update the Processor Status Register (PSR) flag bits

Condition: Condition define the instruction will executed if meet the condition

Rd: Rd is destination register

Rn: Rn is operand1

Operand2: Operand2 can be register or immediate value

A. Register Operand Operand berada di register. Register pertama adalah register tujuan, register kedua adalah operand1 dan register ketiga adalah operand2.
Berikut ini adalah instruksi operasi aritmatika dan logika dengan operand register.

ADD R0, R1, R2 ;R0=R1+R2 Add contents of register R1 with register R2 and place the result in register R0.

ADC R0, R1, R2; ;R0 = R1+R2 +C Add with carry C is carry bit.

SUB R0, R2, R3 ;R0=R2-R3 where R2 is first operand and R3 is second operand

SBC R0, R2, R3; ;R0=R2-R3+C-1 SUB with carry.

RSB R0, R2, R5 ;R0= R5-R2 Reverse SUB.

RSC R0, R2, R5 ;R0=R5-R2+C-1 Reverse sub with carry.

AND R0, R3, R5 ;R0= R3 AND R5.

ORR R7, R3, R5; ;R7=R3 OR R5.

EOR R0, R1, R2 ;R0 = R1 Exclusive OR with R2.

BIC R0, R1, R2 ;Bit clear. The one in second operand clears corresponding bit in first operand and stores the results in destination register.

Contoh 8.1 Asumsikan isi R1 adalah 1111111111011111, dan R2 adalah 1000 0100 1110 0011 setelah eksekusi BIC R0,R1, R2 R0 berisi 0111 101100011100.

A. Operand Segera Dalam operan langsung, operan2 adalah nilai langsung dan maksimum dapat 12 bit

```
ADD R1, R2, #&25 ;R1=R2+&amp;amp;amp;amp;amp;amp;25, # means
immediate and &amp;amp;amp;amp;amp;amp; means the immediate
value is in hexadecimal.

AND R2, R3, #&45 ;R2 = R3 AND &45.

EOR R2, R3, #&45 ;R2= R3 Exclusive OR &45.
```

Contoh 8.2 Apa isi R1 setelah menjalankan instruksi berikut? Asumsikan R2 berisi 0x12345678

```
ADD R1, R2, #0x345
```

Instruksi ADD akan menambahkan berisi R2 dengan 0x2345 dan menyimpan hasilnya di R1, kemudian R1 = 0x123459BD.

Setting Flag Bits PSR Instruksi di atas tidak mempengaruhi flag bit PSR karena instruksi tidak memiliki opsi S. Dengan menambahkan suffix S pada instruksi, instruksi akan mempengaruhi flag bit.

```
ADDS R1, R2, R3 ;The suffix S means set appropriate flag bit.

SUBS R1, R2, R2; ;This will set zero flag to 1.
```

Bandingkan dan Uji Instruksi

Prosesor ARM menggunakan instruksi perbandingan dan pengujian untuk mengatur bit bendera PSR dan berikut adalah instruksi perbandingan dan pengujian.

CMP, CMN, TST, dan TEQ, instruksi ini menggunakan dua operan untuk membandingkan dan menguji, hasil operasinya tidak ditulis ke register mana pun

Instruksi CMP (Bandingkan Instruksi) Instruksi CMP memiliki format berikut:

```
CMP Operand1, Operand2
```

Instruksi CMP membandingkan Operand1 dengan Operand2; instruksi ini mengurangi Operand2 dari Operand1 dan menetapkan flag yang sesuai. Bit bendera ditetapkan berdasarkan hasil operasi sebagai berikut:

Bendera **Z** disetel jika Operand2 sama dengan operand1

Bendera **N** disetel jika operand1 kurang dari operand2

Bendera **C** diatur jika hasil operasi menghasilkan carry

Contoh 8.4 Asumsikan R1 berisi 0x00000024, dan R2 berisi 0x00000078; operasi CMP R1, R2 akan menyetel N flag ke 1.

CMP Rd, nilai langsung, nilai langsung dapat berupa 8 bit seperti:

```
CMP R1, #0xFF
```

CMN Bandingkan Meniadakan CMN memiliki format berikut:

```
CMN Operand1, Operand2
```

Instruksi akan menambahkan operand1 dengan operand 2 dan mengatur bit flag yang sesuai.

Contoh 8.5 Asumsikan R1 berisi 0x00000024 dan R2 berisi 0x13458978: operasi CMN R1, R2 dengan hasil carry dan set flag C ke 1.

TST (Instruksi Tes) Instruksi tes memiliki format berikut:

```
TST Oprand1, Operand2
```

Instruksi pengujian melakukan operasi AND antara operand1 dan Operand2 dan menetapkan bit flag yang sesuai. Operand dapat berupa nilai langsung atau register seperti

```
TST R1, R2 ;This instruction performs R1 AND R2 operation and sets
the appropriate flag.
```

ATAU

```
TST R1, immediate, the immediate value can be 8 bits such as
TST R1, 0xFF
TEQ R1, R2 ;This instruction performs R1 Exclusive OR R2.
```

Jika R1 sama dengan R2, maka Z-flag disetel ke 1.

Mendaftar Instruksi Swap (MOV dan MVN)

Instruksi swap register memiliki format umum berikut.

```
A. MOV{S}{condition} Rd, Rm

Move the contents of Rm to Rd
```

Contoh 8.6 Apa isi R1 setelah eksekusi instruksi berikut?

Asumsikan R2 berisi 0X0000FFFF.

```
a. MOV R1, R2 ;R1 ← R2
 R2=0x0000FFFF

b. MVN R1, R2 ; R1 ← NOT R2
 R2= 0xFFFF0000
```

```
A. MOV{S}{condition} Rd, immediate value
```

Nilai langsung adalah 16 bit, kisaran nilai langsung jika dari 0x00000000 hingga 0x0000FFFF.

Contoh 8.7

```
MOV R2, # 0x45 , the contents of R2 will be 0x00000045

B. MOV Rn, Rm, lsl # n ; shift Rm n times to the left and store the
result Rn

C. Conditional MOV

MOVEQ R2, 0x56 ; if zero bit is set then executes MOVEQ
```

Instruksi Shift dan Rotate

ARM menggabungkan operasi putar dan geser dengan instruksi lain; prosesor ARM melakukan operasi shift berikut.

LSL	Logical Shift Left
LSR	Logical Shift Right
ASR	Arithmetic Shift Right
ROR	Rotate Right

Pergeseran Logis ke Kiri (LSL). Dalam operasi logika shift kiri, setiap bit register digeser ke kiri seperti yang ditunjukkan pada Gambar 8.5 dan nol akan ditempatkan pada bit paling tidak signifikan, pergeseran logis ke kiri mengalikan isi register dengan dua.

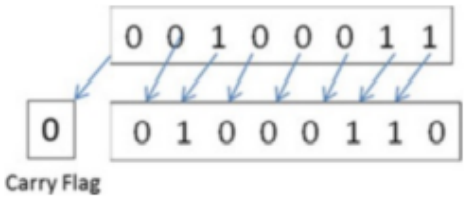
```
LSL R1, R1, n , shift to left R1 n times and store result in R1
```

Contoh 8.8 Apa isi R1 setelah mengeksekusi instruksi berikut? Asumsikan R1 berisi 0x000000500.

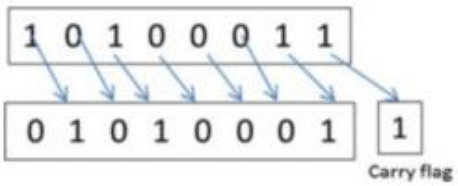
```
LSL R1, R1, 8
R1= 0x00050000
```

Pergeseran Logis Kanan (LSR). Dalam operasi logika shift kanan, setiap bit register digeser ke kanan seperti ditunjukkan pada Gambar 8.6, dan angka nol akan ditempatkan pada bit yang paling signifikan; hak logis membagi isi register dengan dua.

```
LSR R1, R1, n ,shift to right R1 n times and store result in R1
```



Gambar 8.5 Pergeseran logis ke kiri



Gambar 8.6 Pergeseran logika ke kanan

Contoh 8.9 Apa isi R1 setelah menjalankan instruksi berikut: asumsikan R1 berisi 0x000000500.

```
LSR R1, R1, 4
R1= 0x00000050
```

Pergeseran Aritmatika Kanan (ASR). Dalam pergeseran aritmatika ke kanan, bit yang paling signifikan tidak berubah dan setiap bit digeser ke kanan seperti yang ditunjukkan pada Gambar 8.7.

Rotate Right Gambar 8.8 menunjukkan register 8-bit dan Gambar 8.9 menunjukkan register setelah berputar satu kali.

Contoh 8.10 Berapakah isi R1 setelah berputar 16 kali? Asumsikan R1 berisi 0X0000FFFF

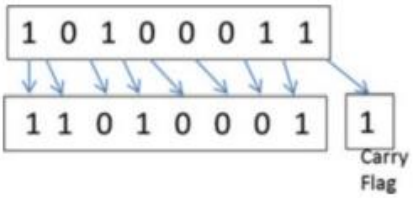
```
ROR R1, R1 , #16
R1= 0xFFFF0000
```

ARM menggabungkan instruksi pemrosesan data dan operasi shift; operasi shift diterapkan pada operan kedua dari instruksi.

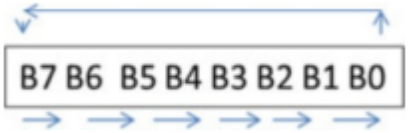
Contoh 8.11 Register R2 berisi 0XEEEEFFFF, dengan mengeksekusi.

```
MOV R1, R2, ROR # 16 ;the R2 rotate 16 times and store results in
R1
```

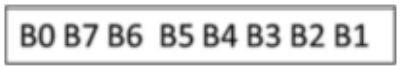
dengan memutar 16 kali isi R1 akan menjadi Xffffeee



Gambar 8.7 Pergeseran aritmatika ke kanan



Gambar 8.8 Operasi putar kanan



Gambar 8.9 Operasi putar kanan satu bit

```
ADD R1, R2, R3, LSL #4 ;R1= R2 + R3 x 2^4, R3 is shifted 4 times to
the left and result is added to R3 and placed in R1.
```

Juga register dapat menampung berapa kali operand2 harus digeser.

ADD R1, R2, R3, LSL R4	;R1 = R2 + R3 X 2^{R4} , Berapa kali R3 digeser dalam R4.
MOV R0, R1, LSL #3	; Geser R1 ke kiri tiga kali dan pindahkan hasilnya ke R0.

Instruksi Tanpa Syarat ARM dan Instruksi Bersyarat

Gambar 8.10 menunjukkan format umum dari instruksi ARM. Instruksi ARM mendefinisikan dua jenis instruksi, yaitu:

1. Instruksi tanpa syarat
2. Instruksi bersyarat

Kode kondisi mendefinisikan jenis instruksi. Jika bidang ini diatur ke 1110, maka instruksi adalah instruksi tanpa syarat, jika tidak, instruksi adalah instruksi bersyarat. Untuk menggunakan instruksi sebagai instruksi bersyarat, kondisi akan berakhiran instruksi. Sufiksnya adalah:

```
Condition Code Condition
 0000 EQ equal
 0001 NE not equal
 0010 CS carry set
 0111 CC carry is clear
 0100 MI negative (N flag is set)
 0101 PL positive (N flag is zero)
 0110 VS overflow set
 0111 VC overflow is clear
 1000 HI higher for unsigned number
 1001 LS less than for unsigned number
 1010 GT greater for signed number
 1011 LT signed less than
 1100 GT Greater Than
 1101 LE less than or equal
 1110 AL unconditional instructions
 1111 Unused code
```

Gambar 8.10 Format umum dari instruksi ARM

```
1100 GT Greater Than
1101 LE less than or equal
1110 AL unconditional instructions
1111 Unused code
```

Prosesor memeriksa flag kondisi sebelum mengeksekusi instruksi kondisional. Jika cocok dengan instruksi kondisi, maka prosesor mengeksekusi instruksi, jika tidak melewatkan instruksi.

```
ADDEQ R1, R2, R3 ;If zero flag is set and it will execute this
instruction.
```

Contoh 8.10 Ubah bahasa rakitan HLL berikut ke ARM.

```
If R1=R2 then
ADD R3, R4, R5
Endif
```

Bahasa rakitan ARM untuk program di atas adalah:

```
CMP R1, R2
ADDEQ R3, R4, R5
```

Contoh 8.11 Konversikan bahasa rakitan HLL ke ARM berikut.

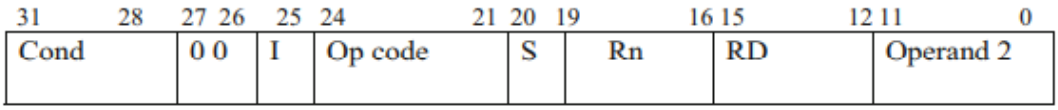
```
 If R1 = R2 Then R3= R4-R5
 Else
 If R1&amp;amp;amp;amp;amp;gt;R2 Then R3=R4+R5
```

Bahasa rakitan ARM untuk program di atas adalah:

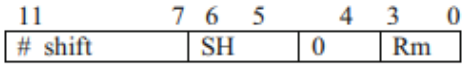
```
 CMP R1, R2
 SUBEQ R3, R4, R5
 ADDGT R3, R4, R5
```

8.4 FORMAT INSTRUKSI PEMROSESAN DATA ARM

Format instruksi digunakan oleh assembler untuk mengubah instruksi ke kode mesin; Gambar 8.11 menunjukkan format instruksi pengolahan data.



Gambar 8.11 Format instruksi pemrosesan data



Gambar 8.12 Format Operand2 ketika bit 4 sama dengan 0

Kode Kondisi Untuk menentukan apakah instruksi adalah instruksi bersyarat atau tidak bersyarat,

I bit I = 0 berarti operand2 adalah register, I = 1 berarti operand 2 adalah nilai langsung.

Kode Op Kode OP menentukan jenis instruksi dan berikut adalah kode operasi untuk instruksi pemrosesan data

Instruction	Op Code	
AND	0000	
EOR	0001	
SUB	0010	
RSB	0011	
ADD	0100	
ADC	0101	
SBC	0110	
RSC	0111	
TST	1000	
TEQ	1001	
CMP	1010	set condition by Op1-Op2
CMN	1011	set condition for Op1+ Op2
ORR	1100	
MOV	1101	Rd=operand2
BIC	1110	
MVN	1111	Rd= NOT operand2

S bit S = 0 tidak mengubah bit flag register PSR, S = 1 mengatur flag kondisi register PSR.

Rn Rn adalah operan pertama, dan dapat berupa salah satu dari 16 register, R0 hingga R15.

Rd Rd adalah register tujuan, dan dapat berupa salah satu dari 16 register, R0 hingga R15.

Operand2 Ketika I = 0 operand2 adalah register dan Gambar 8.12 menunjukkan format operand2.

 # Shift Untuk menentukan nilai langsung berapa kali Rm harus digeser

SH Untuk menentukan jenis operasi shift

Rm operan kedua

```
Operation SH value
 LSL 00 Logical Shift Left
 LSR 01 Logical Shift Right
 ASR 10 Arithmetic Shift Right
 ROR 11 Rotate Right
```

Contoh 8.12 Ubah instruksi berikut menjadi kode mesin.

```
ADD R1, R2, R3, LSL #3
```

31 28	27 26	25	24 21	20	19 16	15 12	11 7	6 5	4	3 0
Cond	0 0	I	Op code	S	Rn	RD	#Shift	SH		RM
1110		0	0100	0	0010	0001	0011	00	0	0011

Ketika bit 4 dari operand2 diset ke 1, berapa kali Rm harus digeser dalam register. Gambar 8.13 menunjukkan format operand2 dari Gambar 8.11.

I=1: The operand 2 would have following format.

11 0
Immediate Value

11 8	7	6 5	4	3 0
RS	0	SH	1	Rm

Gambar 8.13 Format Operand2 ketika bit 4 sama dengan 1

8.5 OPERASI STACK DAN INSTRUKSI

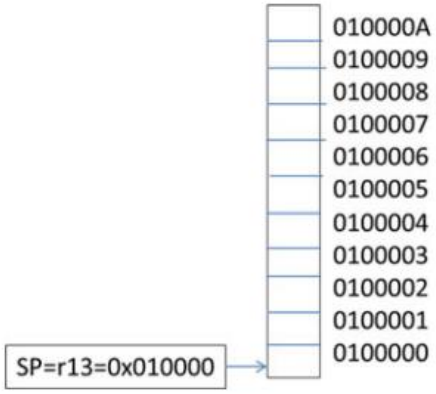
Bagian dari memori yang digunakan untuk penyimpanan sementara disebut stack; penunjuk tumpukan menyimpan alamat bagian atas tumpukan seperti yang ditunjukkan pada Gambar 8.14.

Register R13 ditetapkan sebagai penunjuk tumpukan (SP), dan tumpukan menggunakan instruksi berikut.

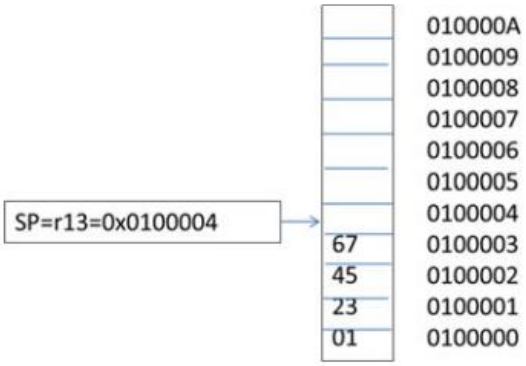
a. Push {condition} Rn : transfer the contains of Rn into stack and add 4 to the stack pointer

Contoh 8.12 Asumsikan isi R3 adalah 0x01234567; Gambar 8.15 menunjukkan isi Stack setelah menjalankan push R3.

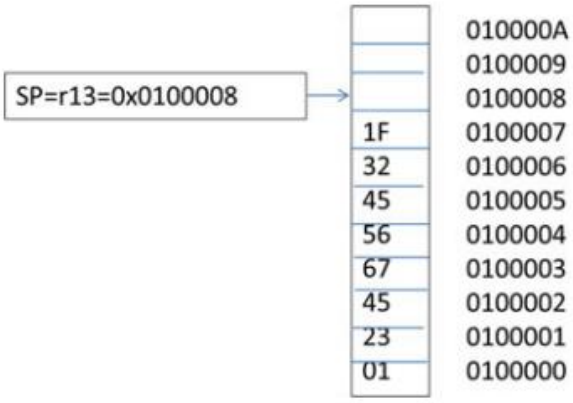
Contoh 8.14 Tunjukkan isi stack dan SP pada Gambar 8.16 setelah eksekusi Push R4; asumsikan R4 berisi 0X5645321F.



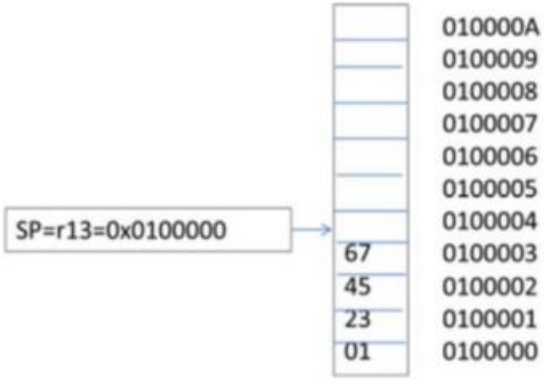
Gambar 8.14 Arsitektur tumpukan



Gambar 8.15 Menunjukkan isi tumpukan setelah eksekusi push R3



Gambar 8.16 Menampilkan tumpukan setelah operasi push



Gambar 8.17 Isi tumpukan setelah operasi POP

POP Instruction: the POP instruction has following format
POP{condition} Rn
POP Rn: the pop instruction remove the word from top the stack and store it into register rn and automatically decrement stach pointer by 4

Contoh 8.15 Tunjukkan isi stack dan SP dari Gambar 8.16 setelah eksekusi POP R0; isi dari R0 akan menjadi 0x1FAD7856 dan stack akan terlihat seperti pada Gambar 8.17.

8.6 CABANG (B) DAN CABANG DENGAN INSTRUKSI TAUTAN (BL)

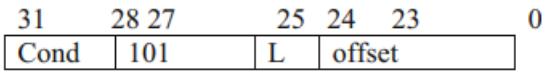
Instruksi Cabang memiliki format umum berikut.

```
B{condition} label

B label ; branch to location label.

BEQ label ; if flag bit Z=1 then execute this instruction

BL Subroutine ;it will branch to subroutine and save contents of
PC (R15) to R14 (link register) for return from subroutine.
```

Contoh 8.16 Tulislah subrutin untuk mencari nilai Y 16X 4; asumsikan R1 memegang Y dan R2 memegang X.

```
BL Funct
Funct SUB R1, R1, R1
ADD R1, R1, R2, LSL4
ADD R1, R!, #04
MOV R15, R14 ; Move return address to PC
```

Format Instruksi B dan BL



L=0 means Branch and condition for branch can be set by Cond field.
L=1 Mean Branch and Link

Instruction

```
B Branch always
BAL Branch Always
BEQ Branch if Equal
BNE Branch if Not equal
BPL Branch on positive
BMI Branch on negative
BCC Branch if carry flag is clear
BLO Branch below for unsigned number
BCS Branch carry flag is set
BHS Branch if higher for unsigned number
BVC Branch if Over flow flag is clear
BVS Branch if Over flow flag is clear
BGT Branch greater for signed number
BGE Branch greater or equal for signed number
BLT Branch Less than for signed number
BLE Branch Less than for signed number
BLS Branch less than or equal for unsigned number
```

Contoh 8.17 Tulis ulang bahasa rakitan berikut menggunakan instruksi bersyarat.

```
CMP R1,R2
BEQ Exit
ADD R1, R2, R3
Exit:
SUB R1, R5, R6
```

Dengan menggunakan instruksi bersyarat, bahasa rakitan di atas dapat diwakili oleh:

```
CMP R1,R2
SUBEQ R1,R5,R6
ADDNE R1,R2,R3
```

8.7 INSTRUKSI MULTIPLY (MUL) DAN MULTIPLY-ACCUMULATE (MLA)

```
MUL instruction
MUL Rd, Rm,Rs ;Rd= Rm*Rs

MLA Multiply and Accumulate
MLA Rd,Rm,Rs, Rn ; Rd= Rm*Rs +Rn
```

Format Instruksi Perkalian

31 28	27 22	21	20	19 16	15 12	11 8	7 4	3 0
Cond	00000	A	S	Rd	Rn	RS	1001	Rm

```
A=0 MUL instruction
A=1 MLA instruction
S=0 Do not change flag bit
S=1 Set the flag bits
Rd is destination register
Rs, Rm and Rn are the operands
```

8.8 RINGKASAN

- Fungsi compiler adalah untuk mengubah HLL ke bahasa assembly.
- Fungsi assembler adalah mengubah bahasa assembly menjadi kode mesin (biner).
- Instruksi komputer diwakili oleh bentuk mnemonic seperti “ADD”.
- Setiap instruksi mungkin memiliki satu atau dua atau tiga operand, ADD R1, R2, dan R3 dimana R1, R2, dan R3 disebut operand.
- ARM singkatan dari Advanced RISC Machine dan ARMv7 menggunakan instruksi 32-bit dan 16-bit.
- ARMv7 berisi 31 register dan hanya 16 register R0 sampai R15 yang digunakan oleh programmer.
- Register R15 digunakan untuk program counter (PC), R14 digunakan untuk link register (LR), dan R13 digunakan untuk stack pointer (SP).
- Register PSR digunakan untuk menyimpan bit kontrol (I, F, M, dan T) dan bit flag (N, Z, C, dan V).
- Prosesor ARM menawarkan dua jenis instruksi, yaitu instruksi tanpa syarat dan kondisional.
- Instruksi CMP, CMN, TST, dan TEQ akan mengatur register status prosesor.

- Bagian dari memori digunakan untuk tumpukan, dan penunjuk tumpukan menyimpan alamat bagian atas tumpukan.
- Bab 9 mencakup lebih banyak instruksi ARM seperti memuat, menyimpan, instruksi semu, instruksi bidang bit, mode pengalamatan ARM, dan representasi data dalam memori.

Masalah Dan Pertanyaan

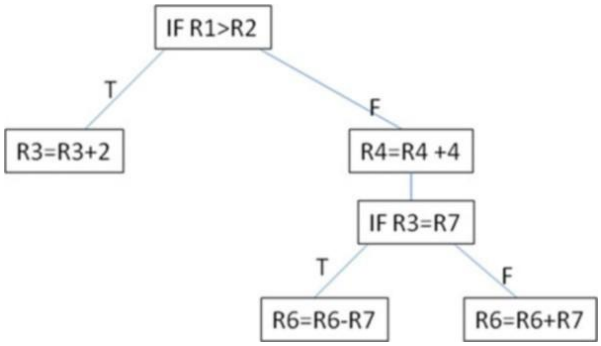
1. Jelaskan bagaimana HLL dikonversi ke kode Mesin.
2. Daftar jenis instruksi berdasarkan jumlah operan.
3. Register prosesor ARM mana yang digunakan untuk penghitung program (PC)?
4. Register prosesor ARM mana yang digunakan untuk penunjuk tumpukan (SP)?
5. Register prosesor ARM mana yang digunakan untuk register tautan?
6. Apa isi R5 setelah eksekusi instruksi berikut, anggap R2 berisi 0X34560701 dan R3 berisi 0X56745670
   (a) TAMBAHKAN R5, R2, R3
   (b) DAN R5, R3, R2
   (c) XOR R5, R2, R3
   (d) TAMBAHKAN R5, R3, #0x45
7. Apa isi R1? Asumsikan R2 0x00001234.
   (a) MOV R1, R2, LSL #4
   (b) MOV R1, R2, LSR #4\
8. Apa perbedaan antara kedua instruksi ini?
   (a) SUB R1, R2, R2
   (b) SUB R1,R2, R2
9. Ubah bahasa HLL berikut menjadi instruksi ARM.

```
IF R1>R2 AND R3>R4 then
R1= R1 +1
Else
R3=R3 +R3*8
Endif
```

10. Ubah bahasa HLL berikut menjadi instruksi ARM.

```
IF R1>R2 OR R3>R4 then
R1= R1 +1
Else
R3=R3 +R5*8
Endif
```

11. Ubah flowchart berikut ke bahasa rakitan ARM.



12. Tulis program untuk menambahkan sepuluh angka dari 0 hingga 10 atau ubah bahasa C berikut ke bahasa rakitan ARM.

```
int sum;
int i;
sum = 0;
for (i = 10 ; i > 0 ; i - - ){
sum = sum +1
}
```

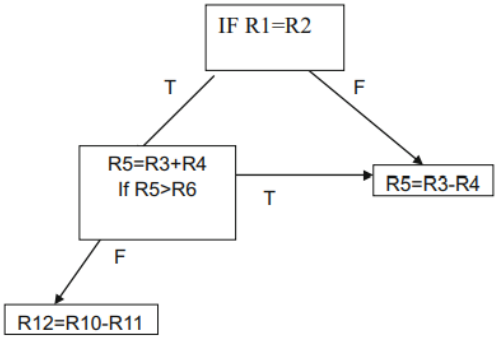
13. Tulis program untuk mengonversi rakitan HLL ke ARM berikut.

```
a= 10;
b=45;
while ( a! =b ) {
if (a < b)
a = a +5;
else
b= b+5;
}

SOLUTION:
```

14. Konversikan HLL berikut ke rakitan ARM.

```
IF R1>R2 AND R3>R4 then
R1= R1 +1
Else
R3=R3 +R5*8
Endif
```

15. Ubah flowchart berikut ke perakitan ARM.



BAB 9
INSTRUKSI ARM BAGIAN II

Tujuan: Setelah menyelesaikan bab ini, Anda diharapkan dapat:

- Jelaskan berbagai jenis instruksi beban.
- Buat daftar berbagai jenis instruksi toko.
- Membedakan berbagai jenis mode pengalamatan ARM.
- Buat daftar instruksi semu ARMv7.
- Pelajari penerapan instruksi ADR dan LDR.
- Menjelaskan operasi instruksi bidang bit.
- Pelajari bagaimana data direpresentasikan dalam memori.

9.1 PENDAHULUAN

Instruksi transfer data digunakan untuk mentransfer data dari memori ke register dan dari register ke memori. Prosesor ARM menggunakan instruksi LDR dan STR untuk mengakses memori. LDR dan STR dapat menggunakan register indirect, pengalamatan pre-index, dan pengalamatan post-index untuk mengakses memori. ARM menawarkan beberapa instruksi semu yang digunakan oleh programmer dan assembler untuk mengubahnya menjadi instruksi ARM.

9.2 INSTRUKSI TRANSFER DATA ARM

Instruksi Beban (LDR). Instruksi LDR digunakan untuk membaca data dari memori dan menyimpannya ke dalam register, dan memiliki format umum sebagai berikut.

LDR[type]{condition} Rd, Address

Di mana "ketik" mendefinisikan instruksi pemuatan berikut:

LDR	Muat 32 bit (kata)
LDRB	Muat 1 byte
LDRH	Muat 16 bit (setengah kata)
LDRS	Muat byte yang ditandatangani
LDRSB	Muat ekstensi tanda
LDRSH	Muat setengah kata yang ditandatangani
LDM	Muat banyak kata

Kondisi adalah opsional seperti LDREQ memuat data jika Z flag = 1 dan Rd adalah register tujuan.

Contoh 9.1 Asumsikan R0 menyimpan alamat 0000 dan memori berikut diberikan, tunjukkan isi R1 dan R3 setelah menjalankan instruksi berikut.

Address	Contents
0	0X85
1	0XF2
2	0X86
3	0XB6

LDRH R1, [R0] R1 = 0x0000F285
LDRSH R3, [R0] R3 = 0xFFFFF285

Instruksi ARM Pseudo

ARM mendukung beberapa instruksi semu; instruksi semu digunakan oleh programmer, dan assembler mengubah instruksi semu menjadi instruksi ARM.

ADR Pseudo Instruction ADR digunakan untuk memuat alamat lokasi memori ke dalam register dan memiliki format sebagai berikut.

ADR Rd, Address

Contoh 9.2 Instruksi berikut akan membaca alamat data dan kemudian memuat data ke register R3:

ADR R0, tabel	Pindahkan alamat yang diwakili oleh tabel
LDR R3, [R0]	R3 = 0x23456780

Alamat	Data
Meja	0x23456780

Instruksi Pseudo LDR Instruksi semu LDR digunakan untuk memuat konstanta ke dalam register. Untuk memindahkan kontestan 32 bit ke dalam register, instruksi MOV Rd, #value hanya dapat memindahkan 12 bit ke register Rd karena operand2 dalam format instruksi untuk MOV adalah 12 bit. Instruksi semu LDR memiliki format berikut.

LDR Rd, = Nilai

Contoh 9.3 Instruksi berikut akan memuat R1 dengan 0x23456789: LDR R1 , 0x23456789

Instruksi Toko (STR)

Instruksi STR digunakan untuk mentransfer isi register ke dalam memori dan memiliki format umum berikut.

STR[type]{condition} Rd, [address]

Di mana "tipe" mendefinisikan jenis instruksi berikut:

STR	Simpan 32 bit (kata)
STRB	Simpan 1 byte
STRH	Menyimpan 16 bit (setengah kata)
STM	Simpan banyak kata

Contoh 9.4

STR R5, [R3] Menyimpan konten R5 ke dalam lokasi memori dimana R3 menyimpan alamatnya; R3 adalah register dasar.

9.3 MODE PENGALAMATAN ARM

- Dukungan prosesor ARM ARM menawarkan beberapa mode pengalamatan dan mereka diindeks sebelumnya, diindeks sebelumnya dengan offset langsung, diindeks sebelumnya dengan offset register, dipra-indeks dengan register berskala, dipra-indeks dengan offset register dan ditulis kembali, pasca-indeks dengan offset langsung, pasca-indeks dengan offset register, dan pasca-indeks dengan offset register berskala; tabel berikut menunjukkan ringkasan mode pengalamatan ARM.

Mode pengalamatan	**Sintaks assembler**	**Alamat efektif (EA)**
Segera	MOV R1, #0X25	Data adalah bagian dari instruksi
Diindeks sebelumnya	[Rn]	EA =Rn
Pra-indeks dengan offset langsung	[Rn, # offset]	EA =Rn + offset
Pra-indeks dengan register offset	[Rn, ± Rm]	EA =Rn ± Rm
Per-indeks dengan register berskala	[Rn, Rm, Shifted]	EA =Rn + Rm shifted
Pra-indeks dengan offset langsung dan tulis kembali	[Rn, offset]!	EA = Rn + offset Rn = Rn + offset
Pra-indeks dengan register offset dan tulis kembali	[Rn, ± Rm,]!	EA = Rn ± Rm Rn = Rn ± Rm
Pra-indeks dengan offset register skala dan tulis kembali	[Rn, Rm, Shifted]!	EA = Rn ± Rm shifted Rn = Rn ± Rm shifted
Pasca-indeks dengan offset langsung	[Rn], offset	EA = Rn Rn = Rn + offset
Post-indeks dengan register offset	[Rn], ± Rm	EA = Rn Rn = Rn ± Rm
Post-indeks dengan offset register berskala	[Rn], ± Rm. SHL #n	EA = Rn Rn = Rn ± Rm shifted

Pengalamatan Segera

Dalam pengalamatan langsung, operan adalah bagian dari instruksi seperti

```
MOV R0, # 0x34
or
ADD R1, R2, 0x12
```

Pra-indeks

Dalam mode pengalamatan pra-indeks yang diwakili oleh [Rn], alamat efektif (EA) adalah isi dari Rn seperti:

```
LDR R2, [R3]
```

Pra-indeks dengan Offset Seger

Pra-indeks dengan offset langsung diwakili oleh [Rn, #offset] seperti

```
LDR R0, [Rn, #Offset]
```

Offset adalah nilai langsung seperti

```
LDR R1, [Rn, #0x25]
EA = R2 + 0x25
```

Diindeks sebelumnya dengan Register Offset

Offset dapat berupa register atau register dengan operasi shift:

[R5, #0x25]
EA = 0x000002345 + 0X25 = 0x0000236A

Contoh 9.5 Apa alamat efektif dari alamat berikut? Asumsikan R5 berisi 0X00002345

[R5, #0x25]
EA = 0x000002345 + 0X25 = 0x0000236A

Contoh 9.6 Apa alamat efektif dari pengalamatan pra-indeks berikut, asumsikan R5 = 0x00001542 dan R2 = 0X00001000

R5 = 0x00001542 and R2 = 0X00001000
[R5, R2]
EA = R5 + R2 = 0X00001542 + oX00001000 = 0X00002542

Pra-indeks dengan Register Berskala

Offset berisi register dengan operasi shift:

LDR R0, [Rn , R2 , LSL#2]

Contoh 9.7 Apa EA dari instruksi berikut?

LDR R0, [Rn , R2 , LSL#2]
EA = Rn + R2 * 4

R2 digeser ke kiri dua kali (kalikan dengan 4) dan ditambahkan ke Rn.

Pra-indeks dengan Write Back

Format umum untuk pengalamatan pra-indeks dengan penulisan balik adalah [Rn, Offset]! Tanda seru (!) digunakan untuk menulis kembali; offset dapat berupa nilai langsung atau register atau register geser:

EA = Rn + offset and Rn = Rn + offset

Pra-indeks dengan Offset Segera dan Tulis Kembali

LDR R0, [R1,# 4]!
EA = R1 + 4 and R1 updated by R1 = R1 + 4.

Contoh 9.8 Berapa alamat efektif dan nilai akhir R5 untuk instruksi berikut? Asumsikan isi R5 = 0x 00002456:

LDR R0, [R5, #0X4]!
EA = R5 + 0x4 = 0x000245A
R5 = R5 + 0x4 == 0x000245A

Pra-indeks dengan Register Offset dan Tulis Kembali

LDR R0, [R1, R2]!
EA = R1 + R2 R1 = R1 + R2

Contoh 9.9 Berapa alamat efektif dan nilai akhir R5 dari instruksi berikut? Asumsikan isi R5 = 0x 00002456 dan R2 = 0X00002222:

LDR R0, [R5, R2]!
EA = R5 + R2 = 0x00004678
R5 = R5 + R2 = 0x00004678

Pra-indeks dengan Offset Register Berskala dan Tulis Kembali

LDR R1, [Rn, R2, LSL#2]!
EA = Rn + R2*4
Rn = Rn + +R2*4

Pengalamatan Pasca-indeks

Format umum pengalamatan pasca-indeks adalah

LDR R0, [Rn], offset

Offset dapat berupa nilai langsung atau register atau register geser.

Pasca-indeks dengan Nilai Langsung

LDR R0, [Rn], #4
Effective address = Rn and Rn = Rn + 4

Pasca-indeks dengan Register Offset

LDR R0, [Rn], Rm
Effective address = Rn and Rn = Rn + Rm

Pasca-indeks dengan Offset Register Berskala

LDR R0, [Rn], Rm, SHL#4
Effective address = Rn and Rn = Rn + Rm*16

Tukar Memori Dan Register (SWAP)

Instruksi swap menggabungkan instruksi beban dan penyimpanan menjadi satu instruksi, dan memiliki format berikut.

SWP Rd, Rm, [Rn]

Register Rd adalah register tujuan, Rm Swap memory and register (SWAP) adalah register sumber, dan Rn adalah register basis.

Instruksi swap melakukan fungsi-fungsi berikut.

Rd ← memori [Rn] Muat Rd dari lokasi memori [Rn]

[Rn] ← Rm menyimpan isi Rm di lokasi memori [Rd]

SWPB Rd, Rm, [Rn] Tukar satu byte

9.4 INSTRUKSI BIDANG BITS

ARM menawarkan dua instruksi bidang bit dan mereka adalah bit field clear (BFC) dan bit field insertion (BFI).

BFC (Instruksi Penghapusan Bidang Bit)

BFC memiliki format umum berikut.

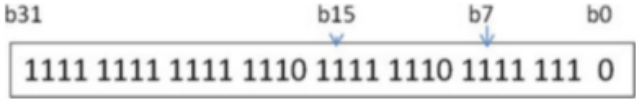
BFC {cond} Rd, # lsb, #width

Rd adalah register tujuan.

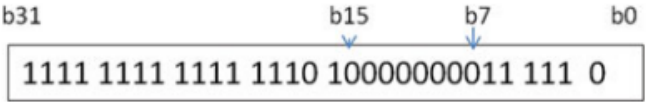
lsb menentukan awal posisi bit dalam register sumber (Rd) menjadi jelas.

Lebar menentukan jumlah bit yang akan dihapus dari lsb ke msb dari register Rd.

Contoh 9.10 Tulis instruksi untuk menghapus bit 7 sampai 15 dari register R4; asumsikan R4 berisi 0xFFFEFEFE.
BFC R4, #7, #8 menghapus bit 7 hingga bit 15 (8 bit) dari register R4. Nilai awal di R4 adalah.

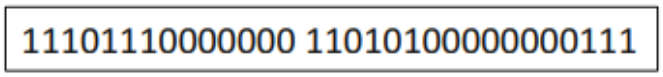


Setelah membersihkan bit 7 sampai 15 dari hasil R4.

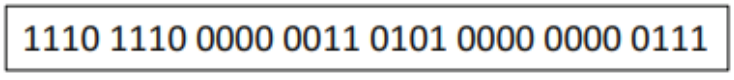


BFI (Instruksi Penyisipan Bit)
Penyisipan bit digunakan untuk menyalin satu set bit dari satu register Rn ke register Rd mulai dari lsb dari Rd; BFI memiliki format berikut.
BFI{cond} Rd, Rn, #lsb, #width
Rd adalah tujuan Reg.
Rn adalah register sumber.
#lsb mulai bit dari Rn.
#width jumlah bit mulai dari lsb dari Rn.

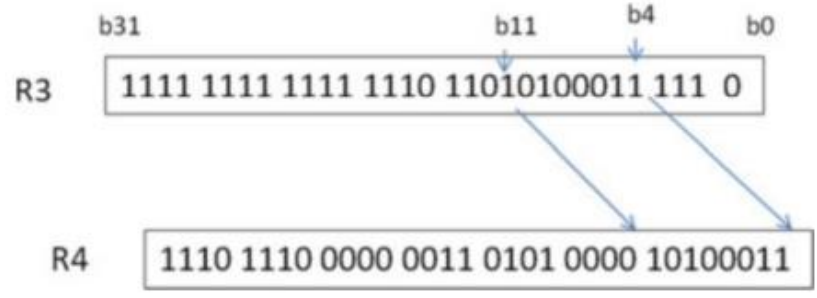
Contoh 9.11 Salin 8 bit R3 mulai dari bit 4 sampai R4; asumsikan R3 berisi 0x FFFFEBCD dan R4 berisi 0xEE035007.
BFI R4, R3, #4, #8, dan instruksi ini akan menyalin 8 bit dari B4 ke B11 dari R3 ke B0 melalui B7 dari R4, nilai awal R3 dalam biner.



Nilai awal R4 dalam biner adalah



Instruksi akan menyalin 8 bit dari bit 4 R3 ke R4 mulai dari bit 0 R4.



9.5 REPRESENTASI DATA DAN MEMORI

Prosesor ARM mendefinisikan kata sebagai 4 byte dan setengah kata sebagai 2 byte. Data dapat direpresentasikan dalam bentuk heksadesimal, desimal, dan biner.

(a) Bilangan desimal, seperti 345
(b) Bilangan heksadesimal, seperti 0x2345, di mana “x” mewakili heksadesimal
(c) Biner atau basis 2, seperti 2_10111100

Memori menyimpan data dan kode. Gambar 9.1 menunjukkan diagram blok memori. Alamat memori menentukan lokasi data, di mana setiap lokasi memori prosesor ARM menampung satu byte. Dalam bahasa rakitan, sebuah label, seperti ditunjukkan pada Gambar 9.2, mewakili alamat memori. Gambar 9.2 menunjukkan bagaimana setiap lokasi memori menampung satu byte. Menyimpan dua byte (setengah kata) data, seperti 0x4563, dapat disimpan dengan dua cara berbeda yang disebut Big Endian dan Little Endian.

Endian Besar. Di Big Endian, byte paling signifikan (MSB) data disimpan terlebih dahulu di memori. Pada Gambar 9.3, 45 adalah byte yang paling signifikan, dan 63 adalah yang paling tidak signifikan untuk angka 0x4563.

ARM 7 beroperasi di Big dan Little Endian; setiap lokasi memori ARM7 menampung satu byte dan sebuah kata (4 byte) dapat disimpan dalam memori dengan dua cara berbeda: Big Endian dan Little Endian.

Endian Besar. Di Big Endian, byte paling signifikan dari sebuah kata disimpan di alamat terendah.

Contoh 9.12 0x34569312 dapat disimpan dalam bentuk Big Endian seperti yang ditunjukkan pada Gambar 9.3.

00	23
01	4A
10	56
11	F5

Gambar 9.1 Memori dengan alamat biner

List	23
List+1	4A
List+2	56
List+3	F5

Gambar 9.2 Memori dengan alamat pelabelan

	Memory
000	34
001	56
010	93
011	12

Gambar 9.3 Representasi Big Endian dari bilangan hex 34569312

	Memory
000	12
001	93
010	56
011	34

Gambar 9.4 Representasi Little Endian dari bilangan hex 34569312

Endian kecil. Di Little Endian, byte kata yang paling tidak signifikan disimpan di alamat terendah.

Contoh 9.13 Bilangan hex 34569312 dapat disimpan dalam bentuk Little Endian seperti ditunjukkan pada Gambar 9.4.

9.6 RINGKASAN

- Instruksi ARM menggunakan LDR dan STR untuk membaca dan menulis ke memori.
- Instruksi beban dapat digunakan untuk memuat satu byte (LDRB), memuat 2 byte (LDHB), dan memuat 4 byte (LDR).
- LDRSB (load signed extension) digunakan untuk memuat satu byte dan memperpanjang tanda data.
- LDRSH (load ditandatangani ekstensi) digunakan untuk memuat dua byte dan memperpanjang tanda data.
- Instruksi semu ARM adalah ADR (memuat alamat lokasi memori) dan LDR (memuat nilai 32 bit ke dalam register).
- Instruksi penyimpanan prosesor ARM adalah STR (menyimpan satu kata), STRB (menyimpan satu byte), dan STRH (menyimpan setengah kata).
- ARM menawarkan beberapa mode pengalamatan dan mereka adalah pra-indeks, pra-indeks dengan offset langsung, pra-indeks dengan register offset, pra-indeks dengan register berskala, pra-indeks dengan register offset dan write back, pasca-indeks dengan segera. makan offset, pasca-indeks dengan register offset, dan pasca-indeks dengan offset register berskala.
- Data dapat direpresentasikan dalam memori dalam bentuk Big Endian dan Little Endian.
- Di Big Endian, byte paling signifikan dari sebuah kata disimpan di alamat terendah.
- Dalam Little Endian, byte paling tidak signifikan dari sebuah kata disimpan di alamat terendah.
- Bab 10 mencakup cara menggunakan alat pengembangan Keil dan menjalankan dan men-debug program, aturan pemrograman, arahan, dan program sampel.

Masalah

1. Lacak instruksi berikut; asumsikan daftar dimulai di lokasi memori 0x00000018 dan menggunakan ARM Big Endian:
   ADR R0, DAFTAR ; Muat R0 dengan alamat daftar lokasi memori

MOV R10, #0x2
(a) LDR R1, [R0]
(b) LDR R2, [R0, #4]!
(c) LDRB R3, [R0], #1
(d) LDRSB R5, [R0], #1
(e) LDRSH R6, [R0]
DAFTAR DCB 0x34, 0xF5, 0x32, 0xE5, 0x01, 0x02, 0x8, 0xFE

2. Kerjakan soal #1 bagian A dan B menggunakan Little Endian.
   (a) R1 0xE532F534
   (b) R2 0xFE080201

3. Apa isi register R7 setelah eksekusi program berikut?

```
ADR R0, LIST
LDRSB R7, [R0]
LIST DC 0xF5
```

4. Apa isi register Ri untuk instruksi beban berikut? Asumsikan R0 memegang alamat daftar menggunakan Little Endian.
   (a) LDR R1, [R0]
   (b) LDRH R2, [R0]
   (c) LDRB R3, [R0], #1
   (d) LDRB R4, [R0]
   (e) LDRSB R5, [R0], #1
   (f) LDRSH R6, [R0]
   Daftar DCB 0x34, 0xF5, 0x32, 0xE5, 0x01, 0x02

5. Memori berikut diberikan, tunjukkan isi setiap register, dan asumsikan R1 = 0x0001000 dan R2 = 0x00000004 (gunakan Little Endian).
   (a) LDR R0, [R1]
   (b) LDR R0, [R1, #4]
   (c) LDR R0, [R1, R2]
   (d) (d) LDR R0, [R1, #4]!

1000	23
	13
	56
	00
1004	45
	11
	21
	88
1008	03
	08
	35
	89
100C	44
	93

6. Apa alamat dan isi efektif R5 setelah menjalankan instruksi berikut? Asumsikan R5 berisi 0x 18 dan r6 berisi 0X00000020.
   (a) STR R4, [R5]
   (b) STR R4, [R5, #4]
   (c) STR R4, [R5, #8]
   (d) STR R4, [R5, R6]
   (e) STR R4, [R5], #4

BAB 10
PERAKITAN ARM DENGAN ALAT PENGEMBANG KEIL

Tujuan: Setelah menyelesaikan bab ini, Anda diharapkan dapat:

- Menjelaskan alat pengembangan fungsi.
- Menjelaskan fungsi dari cross-assembler.
- Daftar beberapa alat pengembangan untuk menjalankan program bahasa assembly.
- Instal alat pengembangan Keil.
- Jalankan dan debug program.
- Gunakan template program untuk menulis program Anda sendiri.
- Pelajari aturan pemrograman.
- Mewakili data dalam memori untuk program bahasa rakitan.
- Pelajari penerapan arahan.
- Membedakan berbagai jenis direktif data.
- Jalankan program langkah demi langkah dan amati isi setiap register.

10.1 PENDAHULUAN

Produsen prosesor mempublikasikan dokumentasi yang berisi informasi tentang prosesor mereka, seperti daftar register, fungsi setiap register, ukuran bus data, ukuran bus alamat, dan daftar instruksi yang dapat dieksekusi. Setiap CPU memiliki set instruksi yang diketahui yang dapat digunakan programmer untuk menulis program bahasa assembly. Set instruksi khusus untuk setiap jenis prosesor. Misalnya, prosesor Pentium mengimplementasikan set instruksi yang berbeda dari prosesor ARM. Program yang ditulis menggunakan set instruksi prosesor dikatakan ditulis dalam bahasa rakitan. Fungsi assembler adalah mengubah bahasa assembly menjadi kode mesin (biner) yang dapat dieksekusi oleh CPU.

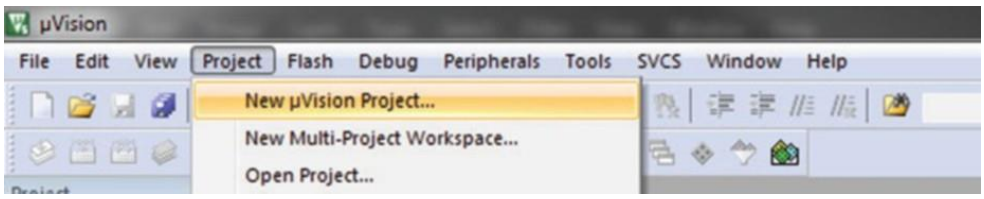
Ketika assembler berjalan pada satu prosesor tetapi dapat merakit instruksi untuk prosesor yang berbeda dengan set instruksi yang berbeda itu disebut cross-assembler. Simulator prosesor adalah alat pengembangan utama, karena memungkinkan lingkungan pengujian yang dapat dikontrol dalam pengaturan seperti workstation Windows atau Linux. Mereka juga dapat memfasilitasi transfer atau download program ke prosesor target. Alat pengembangan berikut adalah beberapa yang mendukung prosesor ARM:

1. Alat Mikrokontroler ARM Keil (www.keil.com)
2. Meja Kerja Tertanam IAR (www.iar.com)
3. Perakit ARM GNU (www.gnu.org)

10.2 ALAT PENGEMBANGAN KEIL UNTUK PERAKITAN ARM

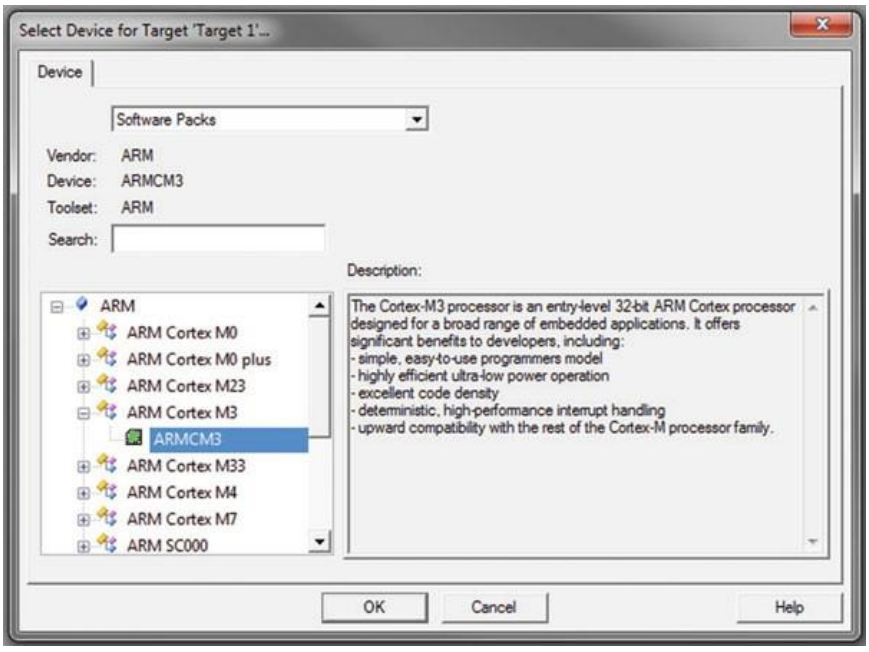
Untuk contoh dalam buku ini, Keil Vision® IDE (lingkungan pengembangan terintegrasi) dari Keil's Microcontroller Development Kit (MDK) versi 5 digunakan. Versi gratis dari perangkat lunak ini dapat merakit dan mensimulasikan eksekusi instruksi ARMv7 asalkan ukurannya di bawah 32 K. Unduhan tersedia dari situs web Keil: http://www.keil.com.

Saat pertama kali diinstal, dialog berjudul Pemasang Paket mungkin terbuka setelah penginstal selesai. Utilitas ini membantu pengguna dalam mengunduh dan menginstal lingkungan. Vision untuk mengaktifkan simulasi papan dan perangkat yang berbeda. Namun, secara default, beberapa template perangkat telah diinstal sebelumnya dengan Vision (Gambar. 10.1).



Gambar 10.1 Membuat proyek baru di Keil μVision® IDE v5.22

- Untuk memulai, buka Vision dan pilih Project → Proyek μVisi Baru.. ..
- Beri nama proyek Anda dan pilih lokasi untuk menyimpannya.
  Setelah menyimpan, sebuah dialog akan terbuka dan meminta Anda untuk Memilih Perangkat untuk Target "Target 1" Tergantung pada apakah Anda telah menginstal paket tambahan apa pun dari Penginstal Paket, layar ini mungkin terlihat berbeda. Beberapa prosesor ARM disertakan dengan instalasi default. Untuk contoh dalam buku ini, ARM Cortex M3 (ARMCM3) dipilih.
- Pilih ARM Cortex M3 → ARMCM3 dan tekan OK. (Gambar. 10.2)

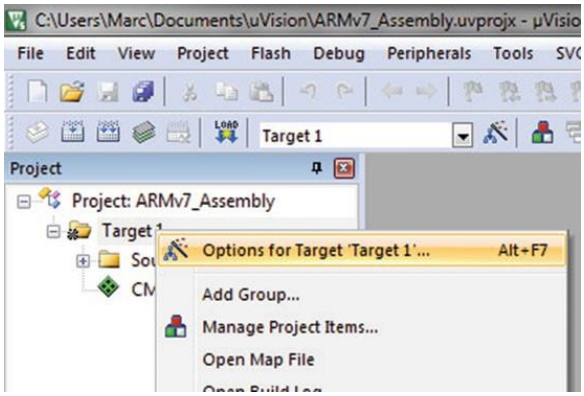


Gambar 10.2 Memilih prosesor ARM Cortex M3

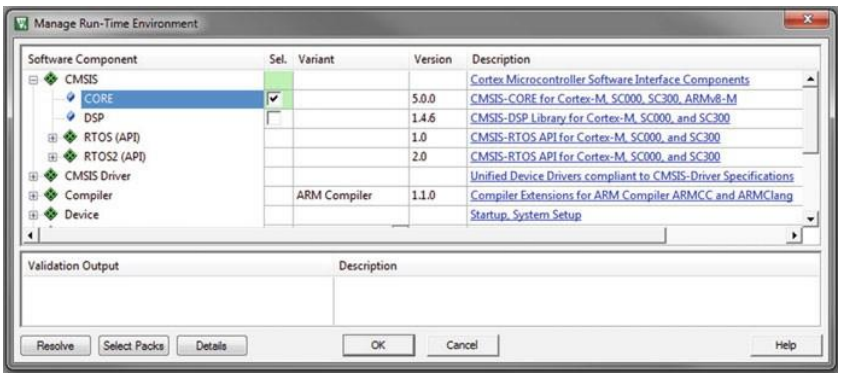
Untuk setiap prosesor, Vision memiliki beberapa perpustakaan yang tersedia. Beberapa sangat penting, seperti konfigurasi start-up, sementara yang lain adalah ekstensi opsional untuk mengaktifkan fungsionalitas yang lebih luas, seperti driver Ethernet dan antarmuka Grafik. Untuk menjalankan program perakitan ARMv7 sederhana, hanya komponen awal yang diperlukan.

- Pilih CMSIS → INTI dan tekan OK. (Gambar. 10.3)

Sekarang, Anda harus memiliki proyek kosong yang terbuka di panel Proyek, yang secara default ada di sisi kiri jendela, di bawah bilah Menu. Selanjutnya, proyek perlu dikonfigurasi untuk menggunakan simulator untuk menjalankan program.

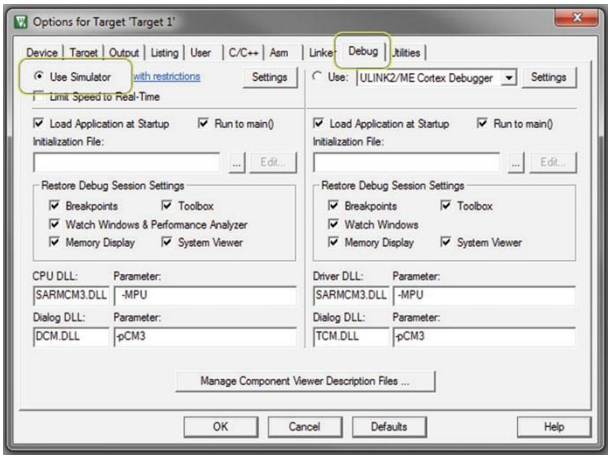


Gambar 10.3 Pustaka CORE untuk menjalankan prosesor ARMCM3



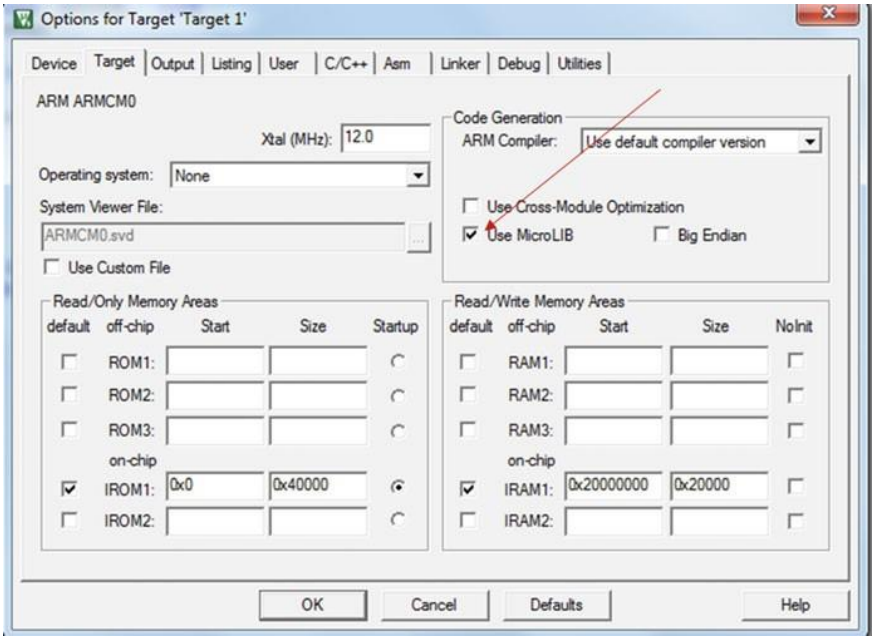
Gambar 10.4 Mengubah konfigurasi proyek

- Klik kanan pada folder Target 1 dan pilih Opsi untuk Target "Target 1"... (Gambar. 10.4)
- Klik pada tab Debug dan pilih Gunakan Simulator. (Gambar. 10.5)
-



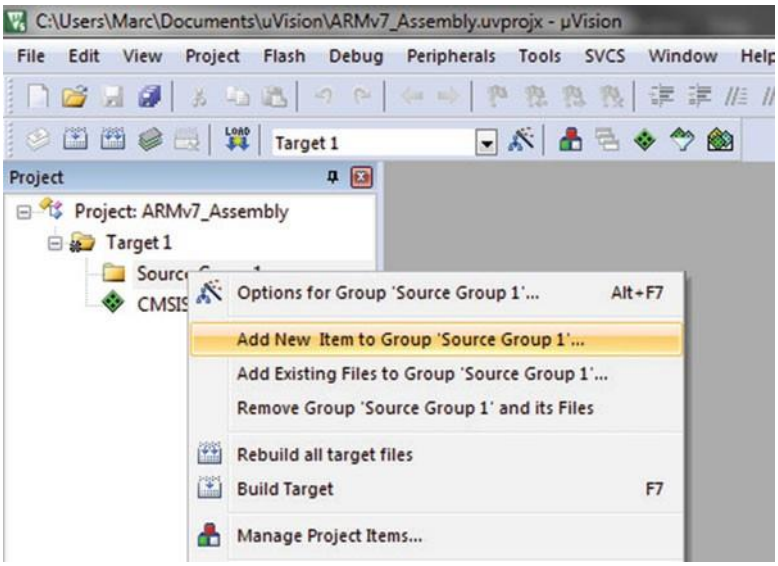
Gambar 10.5 Mengatur proyek untuk menggunakan simulator

- Klik Target dan pilih Use MicroLIB seperti yang ditunjukkan pada Gambar 10.6



Gambar 10.6 Opsi untuk Target1 (Gunakan LIB Mikro)

- Tambahkan item baru ke proyek dengan klik kanan folder Grup Sumber 1 di panel proyek seperti Gambar 10.7

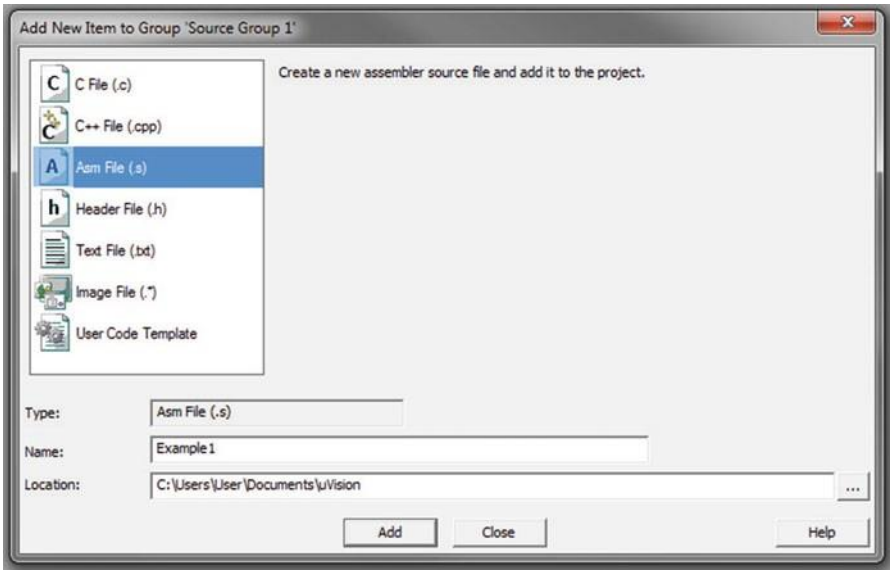


Gambar. 10.7 Menambahkan item ke proyek

- Buat file sumber assembler baru (file Asm atau .s), beri nama, lalu simpan seperti yang ditunjukkan pada Gambar 10.8

  Program perakitan mengikuti struktur tertentu dan memiliki apa yang disebut arahan. Ini akan dijelaskan nanti dalam bab ini, tetapi mereka menginstruksikan assembler tentang bagaimana menyusun sebagian besar program akhir. Sama seperti banyak program bahasa tingkat tinggi yang berjalan di dalam "fungsi" atau "metode" utama, contoh berikut menggunakan rutinitas Vectors dan Reset_Handler (yang dipanggil secara default oleh prosesor yang kami pilih).

Contoh 10.1 Program ini membaca dua bilangan dan menyimpan jumlah tersebut dalam R3 seperti yang ditunjukkan pada Gambar 10.9.



Gambar. 10.8 Buat file sumber perakitan

Example1.s

```
1     AREA RESET,DATA,READONLY
2         EXPORT __Vectors
3
4  __Vectors
5     DCD 0x20001000 ;stack pointer value when stack is empty
6     DCD Reset_Handler ; reset vector
7     ALIGN
8
9     AREA MYCODE, CODE, READONLY
10
11
12    ENTRY
13    EXPORT Reset_Handler
14 Reset_Handler
15
16    MOV R1, #Q
17    MOV R2, #P
18    ADD R3, R1, R2
19 P  EQU 5
20 Q  EQU 6
21 STOP B STOP
22
23    END; end of the program
```

Two underscores

Gambar 10.9 Kode sumber untuk Contoh 10.1

Merakit Program

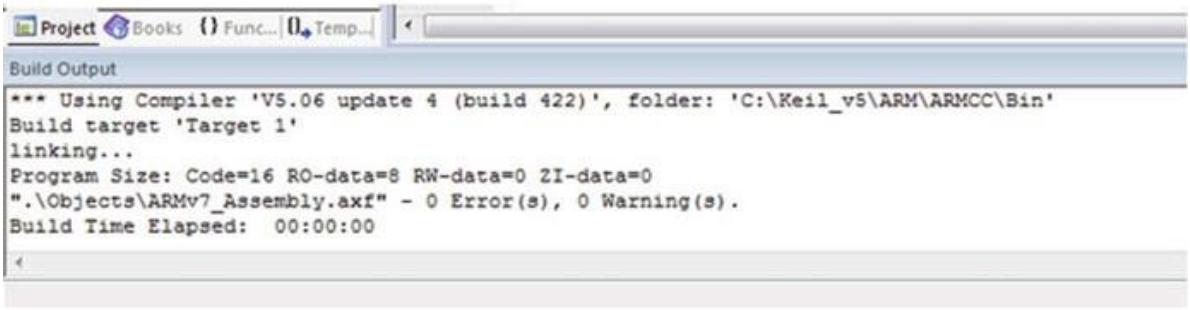
Setelah kode sumber siap untuk dirakit, Anda mungkin meminta assembler untuk menerjemahkan dan menjalankan program dalam simulator. Langkah pertama adalah membuat program perakitan dengan meminta assembler menerjemahkan kode sumber Anda. Ada pintasan keyboard dan tombol menu untuk melakukannya.

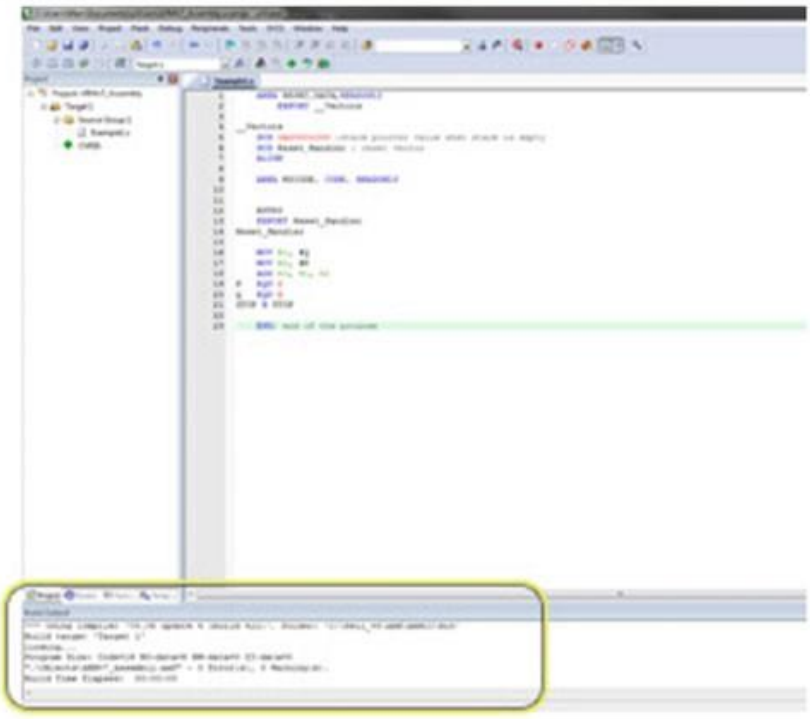
Kedua opsi yang tercantum di bawah ini tersedia pada bilah menu seperti yang ditunjukkan pada Gambar 10.10.

- F7 akan membangun semua file target (mis., Semua file sumber di Target 1).

- Ctrl + F7 akan membangun file yang sedang aktif (mis., File sedang dimodifikasi).

Gambar 10.10 Tombol yang digunakan untuk merakit program





Gambar 10.11 Membangun Output untuk perakitan yang sukses

Panel Build Output di bagian bawah jendela akan menampilkan kesalahan, peringatan, atau jika proyek berhasil dibangun. Pembangunan yang berhasil akan terlihat seperti Gambar 10.11, sedangkan kegagalan akan memberikan deskripsi kesalahan untuk membantu pemrogram menemukan di mana kode tersebut salah.

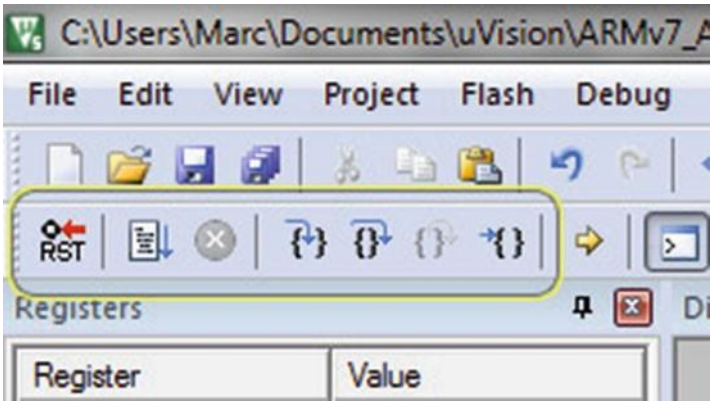
Menjalankan Debugger/Simulator

Sekarang setelah Anda mengkompilasi sepotong kode, Anda ingin men-debug kode untuk pengujian. Untuk memulai debugger, klik **Debug** → **Start/Stop Debug Session** dari menu bar seperti yang ditunjukkan pada Gambar 10.12. (µVision mungkin meminta peringatan bahwa itu sedang berjalan dalam mode evaluasi – jika demikian, cukup tekan OK.)

Gambar 10.12 Memulai/menghentikan debugger

Secara default, register akan ditampilkan di panel Registers di sisi kiri. Jika Anda mengklik "+" di sebelah baris xPSR, Anda dapat melihat bit bendera saat ini – ini adalah **Program Status Register (PSR).**

Panel Disassembly akan menampilkan beberapa data saat ini, seperti instruksi kode mesin dan catatan baris apa dalam kode sumber yang sedang dieksekusi, dan di alamat memori apa.



Gambar 10.13 Lokasi tombol Menu Debug

Untuk menjalankan program, Anda dapat menggunakan tombol/penekanan tombol berikut (Gambar. 10.13):

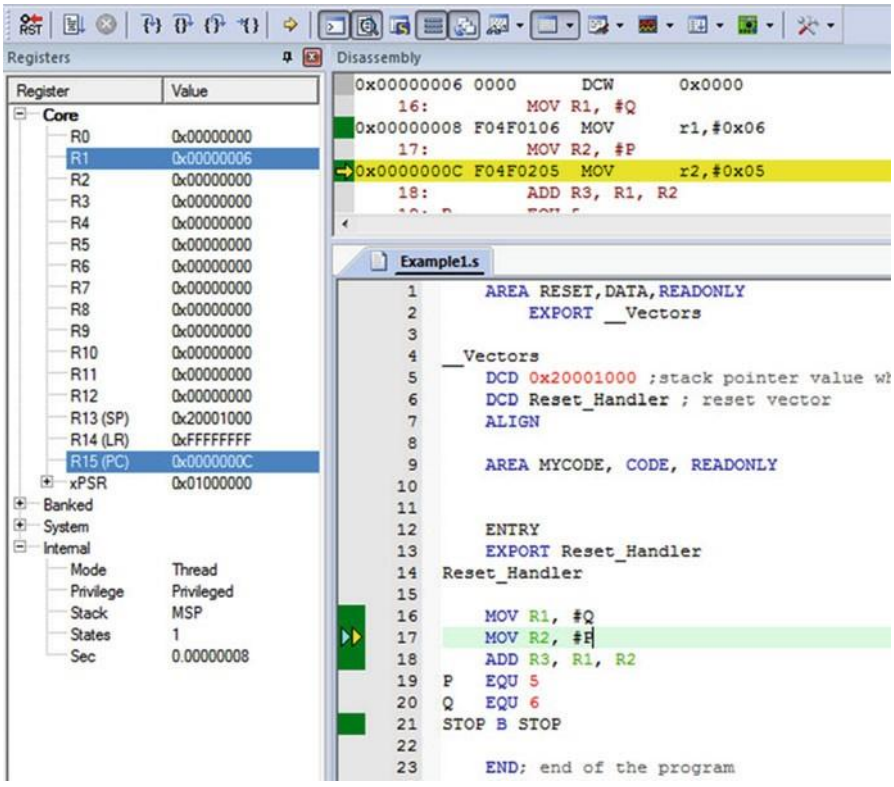
- **Reset** – Reset CPU dan restart program.
- **Jalankan (F5)** – Jalankan seluruh program hingga breakpoint berikutnya (jika disetel).
- **Berhenti** – Menghentikan kode yang sedang dijalankan.
- **Langkah (F11)** – Jalankan baris instruksi saat ini.
- **Step Over (F10)** – Menjalankan satu langkah di atas suatu fungsi.
- **Step Out (Ctrl + F11)** – Selesaikan fungsi saat ini dan berhenti.
- **Jalankan ke Garis Kursor (Ctrl + F10)** – Jalankan program hingga mencapai garis kursor saat ini.

Contoh 10.2 Menjalankan kode dari Gambar 10.8.

- Pertama, ketik ulang kode persis seperti yang ditunjukkan pada Gambar 10.8.

- Rakit kode dengan menyimpan file (Ctrl + S) lalu tekan F7 untuk menerjemahkan file bahasa assembly ke kode mesin. Panel Build Output akan terlihat seperti Gambar 10.11.
- Jalankan simulator/debugger dengan menekan Ctrl + F5 atau di bawah menu Debug. Tekan OK jika prompt EVALUATION MODE muncul.
- Tekan F11 atau untuk "melangkah" satu baris dan jalankan instruksi, dan perhatikan perubahan di setiap panel. Saat debugging, simulator akan menyorot data yang sedang berjalan atau diubah. Lihat Gambar 10.14 untuk contoh yang akan menyerupai layar Anda setelah "langkah" pertama. Ini akan mengeksekusi bit kode pertama kita, MOV R1, #Q, dan menyimpan nilai Q (6) ke dalam register R1.
  - Debugger μVision akan menyorot instruksi berikutnya dalam kode sumber (MOV R2, #P), register yang dimodifikasi (R1, R15), dan menunjukkan di panel Disassembly kode mesin dan alamat memori instruksi dengan label aslinya kode perakitan.
- Lanjutkan ke “langkah” melalui kode, tekan F11 atau tombol langkah setiap kali sampai program telah dijalankan. Perhatikan perubahan di setiap register di setiap langkah.

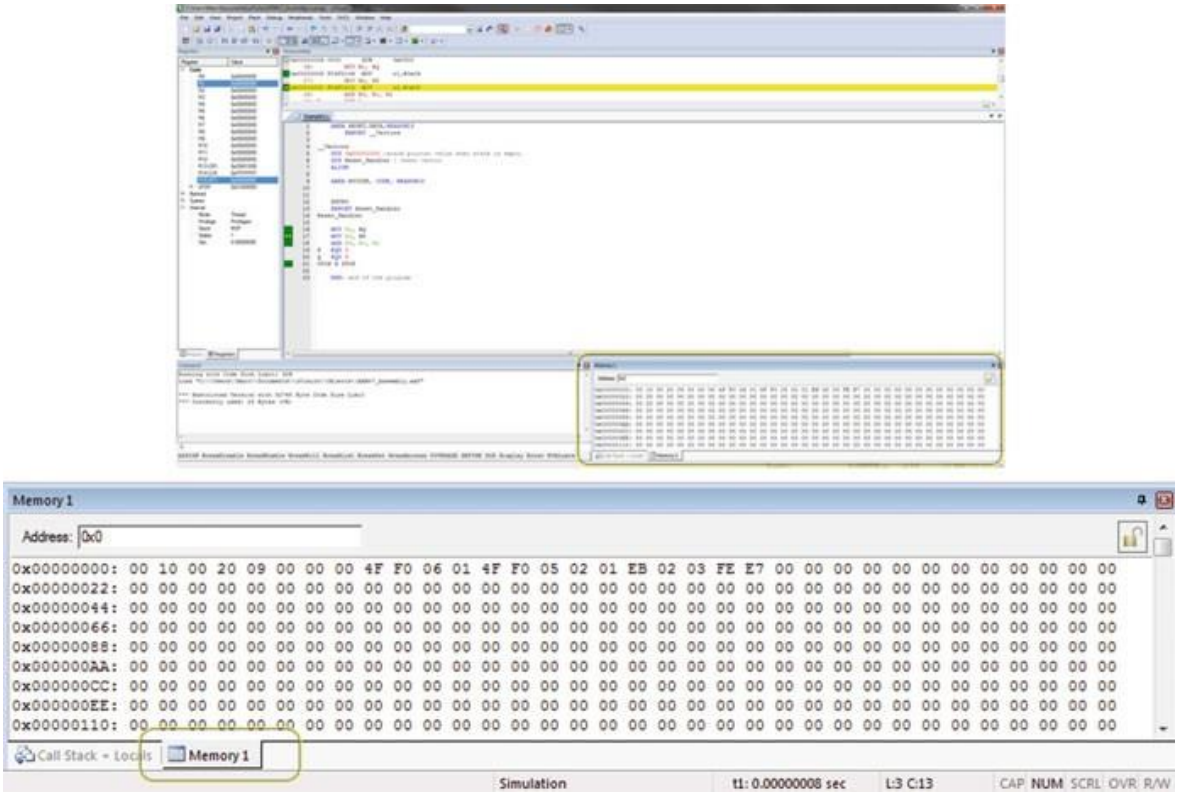
Cara lain untuk mengamati fungsionalitas program Anda adalah dengan melihat memori saat ini. Semua instruksi yang diterjemahkan dari kode sumber dan data yang telah ditentukan disimpan dalam memori. Panel Disassembly akan menampilkan instruksi alamat heksadesimal 32-bit di memori di sebelah setiap instruksi, misalnya.



Gambar 10.14 Debugger secara otomatis menyoroti informasi yang paling relevan

Untuk melihat nilai saat ini dalam memori saat debugging, buka panel Memori 1 di sudut kanan bawah (Gambar. 10.15). Hingga empat jendela tersedia, dan secara default Memori 1 harus diaktifkan. Jika tidak, mungkin diaktifkan di bawah bilah Menu dengan mengklik **View** → **Memory Windows** → **Memory 1**. (Jendela yang diaktifkan akan memiliki ikon ini di sebelahnya:).

Untuk mencari memori, masukkan alamat heksadesimal dari bagian memori yang ingin Anda lihat. Karena merupakan prosesor byte-addressable, setiap alamat menyimpan satu byte informasi, dan ditampilkan sebagai dua digit heksadesimal.



Gambar. 10.15 Lokasi jendela Memori dan tab panel

```
    AREA RESET,DATA,READONLY
        EXPORT __Vectors

__Vectors
    DCD 0x20001000 ;stack pointer value when stack is empty
    DCD Reset_Handler ; reset vector
    ALIGN

    AREA MYCODE, CODE, READONLY


    ENTRY
    EXPORT Reset_Handler
Reset_Handler

   YOUR PROGRAM CODE HERE

   ; comments like this are ignored by the assembler
   ; mark them with a semicolon and use them to make
   ; notes throughout your programs


STOP B STOP

    END; end of the program
```

Gambar. 10.16 Template program untuk prosesor ARMCM3 di Keil Vision v5

Template Program

Gambar 10.16 berisi contoh template kode untuk menulis program untuk ARM Cortex M3. Kode sumber yang terletak di antara label Reset_Handler dan STOP dijalankan saat program dijalankan. (Semua label tidak boleh memiliki spasi putih seperti spasi atau tab di sebelah kiri – label harus berupa teks pertama di kolom, dan karakter pertama di baris.)

10.3 ATURAN PEMROGRAMAN

Aturan CASE Instruksi, simbol, dan label dapat ditulis dalam huruf besar atau kecil tetapi tidak dapat digabungkan (misalnya, MOV atau mov benar, tetapi MoV atau moV tidak).

Komentar Programmer dapat menulis komentar setelah titik koma (;)

```
MOV R1, R2; Moving contents of R2 to R1
```

10.4 REPRESENTASI DATA DAN MEMORI

Prosesor ARM mendefinisikan kata sebagai 4 byte dan setengah kata sebagai 2 byte. Data dapat direpresentasikan dalam bentuk bilangan heksadesimal, desimal, dan biner (Gambar. 10.17). Data dan kode disimpan dalam memori. Gambar 10.18 menunjukkan diagram blok memori. Alamat mendefinisikan lokasi data dalam memori. Setiap lokasi memori prosesor ARM menampung satu byte. Dalam bahasa rakitan, label – seperti yang ditunjukkan pada Gambar 10.14 – mewakili alamat dalam memori untuk data tersebut. Ketika kode diterjemahkan, assembler secara otomatis memutuskan alamat mana yang akan digunakan sebagai pengganti kode memberi label dan menggantinya jika sesuai dalam program.

Basis	Awalan	Contoh
Desimal (10)	-	17
Heksadesimal (16)	0x	0x11
Biner (2)	2	2_00010001

Gambar 10.17 Format representasi numeric

Label	Data
List	0x23
List+1	0x4A
List+2	0x56
List+3	0xF5

Memory Address	Data
0x400	0x23
0x401	0x4A
0x402	0x56
0x403	0xF5

Gambar 10.18 Blok diagram memori jika List disimpan pada alamat 0x400

	Big Endian	Little Endian
0x400	0x20	0x0F
0x401	0x00	0x00
0x402	0x00	0x00
0x403	0x0F	0x20

Gambar 10.19 Big Endian vs Little Endian

Karena setiap lokasi memori menampung satu byte, pemrogram harus berhati-hati saat bekerja dengan kata dan setengah kata. Prosesor dapat berfungsi dengan pemesanan Big Endian atau Little Endian. Gambar 10.19 menunjukkan angka 0x2000000F yang disimpan pada 0x400 di kedua sistem.

- ARM Cortex M3 yang digunakan untuk contoh ini adalah prosesor Little Endian.

10.5 ARAHAN

Direktif adalah perintah assembler yang dijalankan oleh assembler. Arahan tidak pernah menghasilkan kode mesin apa pun. Direktif digunakan untuk menunjukkan awal kode atau data dan akhir program. Direktif sederhana adalah END, yang menandai akhir dari sebuah program. Beberapa direktif yang paling berguna yang digunakan oleh ARM Assembler adalah sebagai berikut:

- **AREA** – mendefinisikan segmen memori
- **ENTRY** – menentukan awal program
- **EQU** – digunakan untuk menetapkan konstanta ke label
  - `X EQU 6`
  - `MOV R1, #X ; R1 = #6`

Arahan Data

Arahan data menentukan jenis dan ukuran data.

- **DCB** (Tentukan Byte Konstan)
- **DCW** (Tentukan Setengah Kata Konstan)
- **DCD** (Tentukan Kata Konstan)
- **SPACE** (Cadangan blok memori yang di-nolkan)

DCB (Tentukan Byte Konstan). Direktif ini digunakan untuk mengalokasikan satu atau lebih byte dalam memori.

```
list1 DCB 0xF,10,2_00010001
```

- `list1 : 0x0F`
- `list1 + 1: 0x0A`
- `list1 + 2: 0x11`

DCW (Tentukan Setengah Kata Konstan). Arahan ini mendefinisikan setengah kata konstan (16 bit, 2 byte) dan membutuhkan dua lokasi memori per setengah kata.

```
list1 DCW 0xFF00, 0x13
```

- `list1 : 0x00`
- `list1 + 1: 0xFF`
- `list1 + 2: 0x13`
- `list1 + 3: 0x00`

DCD (Define Constant Word) DCD digunakan untuk mendefinisikan sebuah kata (32 bit, 4 byte) dan membutuhkan empat lokasi memori per kata.

```
list1 DCD 0x12345678, 0xFF
```

- list1 : 0x78
- list1 + 1: 0x56
- list1 + 2: 0x34
- list1 + 3: 0x12
- list1 + 4: 0xFF
- list1 + 5: 0x00
- list1 + 6: 0x00
- list1 + 7: 0x00

String Karakter. Urutan karakter disebut string karakter. Dalam perakitan ARM, string harus diakhiri dengan null karena harus diakhiri dengan nilai 0 saat ditentukan.

- `List1 DCB "Assembly",0`
- `List2 DCB "I have $250",0`

Assembler memecah string menjadi byte dan menyimpannya masing-masing dalam urutan Little Endian.

Karakter Tunggal. Saat menyimpan satu karakter dalam register atau lokasi memori, karakter tersebut harus berada di dalam tanda kutip tunggal. Assembler mengubah ASCII menjadi heksadesimal.

- `List DCB 'A'`

SPACE. Cadangan lokasi memori untuk digunakan nanti.

- `List SPACE 20`
  – Cadangan 20 byte memori mulai dari *List*.

10.6 MEMORI DI VISION V5

Memori dalam μVision disimulasikan, dan program mencoba mereplikasi kondisi dunia nyata. Banyak mikroprosesor menggunakan kombinasi ROM (memori hanya-baca) dan RAM dan akan memiliki program yang disimpan dalam ROM untuk merespons penyalaan, reboot, atau situasi lain di mana RAM tidak akan disebarkan atau dapat diandalkan.

Untuk melakukannya, memori ditandai sebagai tidak ada, beberapa, atau semua tanda izin berikut:

- *Exec* – Memori menyimpan instruksi dan dapat dieksekusi.
- *Baca* – Memori dapat dibaca.
- *Tulis* – Memori dapat ditulisi.

Secara default, area di memori tempat sebagian besar directive akan menyimpan datanya ditandai sebagai exec dan read, tetapi tidak write. Ini meniru ROM karena instruksi yang disimpan di sana dapat dibaca dan dijalankan, tetapi upaya untuk menulis ke lokasi tersebut (dengan STR atau yang serupa) tidak akan berhasil. Assembler juga tidak mungkin memberi tahu Anda tentang hasil ini.

Contoh 10.3 Apa isi list1 setelah dieksekusi, jika disimpan dalam ROM?

```
ADR R0, list1 ;store the address of list1 in R0
MOV R1, #2 ;store 2 in R1
LDRB R2, [R0] ;load the byte at list1 into R2
```

```
ADD R3, R1, R2 ; R3←R1+R2 (0+2 = 2)
STRB R3, [R0] ;store R3 (#2) at list1
List1 = 0x00
```

Sementara Vision dan perpustakaan sumber ARM memungkinkan program untuk lebih mudah menggunakan data yang disimpan dalam RAM daripada ROM, itu melampaui cakupan bab ini. Namun, mengetahui alamat mana yang ditandai sebagai *baca* dan *tulis*, Anda masih dapat menggunakan RAM untuk menyimpan data ke dalam memori. Secara default, untuk prosesor **ARMCM3**, lokasi memori 0x20000000 – 0x20020000 ditandai sebagai RAM baca dan tulis di μVision.

Contoh 10.4 Apa isi memori pada 0x200000000 setelah eksekusi instruksi ini?

```
list1 DCB 0x0

ADR R0, list1 ; store the address of list1 in R0
MOV R1, #2 ;store 2 in R1

LDRB R2, [R0] ;load the byte at list1 into R2

ADD R3, R1, R2 ; R3←R1+R2 (0+2 = 2)

MOV R4, #0x20000000 ; store our initial RAM address in R4
STRB R3, [R4] ;store R3 (#2) at 0x20000000
List1 = 0x00
0x20000000 0x02
```

10.7 RINGKASAN

- Simulasi prosesor yang berjalan pada prosesor yang berbeda disebut cross-assembler.
- Ada beberapa alat pengembangan untuk prosesor ARM seperti Keil, IAR, dan GNU.
- Instruksi dan label ARM dapat ditulis dalam huruf besar atau kecil tetapi tidak dapat digabungkan.
- Prosesor ARM mendefinisikan kata sebagai 4 byte, setengah kata sebagai dua byte, dan mereka dapat direpresentasikan dalam biner (2_1000011), hex (0x24), atau desimal 45.
- Direktif data adalah DCB (Define Constant Byte), DCW (Define Constant Half Word), DCD (Define Constant Word), dan SPACE (menyimpan blok memori yang di-nolkan).
- DCB digunakan untuk mewakili satu byte di lokasi memori seperti LIST DCB 0x32
- DCW digunakan untuk mewakili setengah kata di lokasi memori seperti LIST DCW 0x3245
- DCD digunakan untuk merepresentasikan sebuah kata di lokasi memori seperti LIST DCD 0x87673245
- Arahan SPACE digunakan untuk mencadangkan lokasi memori dengan nilai nol di semua lokasi seperti: LIST SPACE 20
- String karakter harus diakhiri dengan null (0) seperti Daftar DCB “WELCOME”, 0
- Kateter tunggal di lokasi memori harus berada di dalam tanda kutip tunggal seperti:
- Daftar DB 'A'

Masalah

1. Tulis program untuk menambahkan elemen List1 dan simpan di List2.

```
List1    DCB 0x23, 0x45, 0X23, 0x11
List2    DCB 0x0
```

2. Tulis program untuk mencari bilangan terbesar dan simpan di lokasi memori List3.

```
List1     DCD     0x23456754
List2     DCD     0X34555555
List3     DCD       0x0
```

3. Tulis program, cari jumlah data di LIST lokasi memori, dan simpan SUM di jumlah lokasi memori menggunakan loop.

```
List    DCB   0x23, 0x24, 0x67, 0x22, 0x99
SUM     DCD   0x0
```

4. Tampilkan isi register R1 sampai R5 setelah eksekusi program berikut:

```
AREA  NAME, CODE, READONLY
       EXPORT SystemInit
       EXPORT __main
        ENTRY
 SystemInit

__main

     AREA Directives, CODE, READONLY
          ENTRY

          ADR R0, LIST1
          LDRB R1, [R0]
          LDRB R2, [R0, #1]!
          LDRB R3, [R0,#1]!
          LDRB R4, [R0,#1]!
          LDRB R5, [R0,#1]
List      DCB   0x23, 0x24, 0x67, 0x22, 0x99
          align
          END
```

5. Tulis bahasa assembly untuk menghapus posisi bit 0, 3, 5, dan 6 dari R12; bit lainnya harus tidak berubah (menggunakan instruksi ARM).
   Solution:

```
LDR  R1, #0x00000069
BIC  R12, R12, R1
```

6. Tulis program bahasa assembly untuk HLL berikut:

```
IF R1 = R0
 Then
ADD R3, R0, #5
 Else
SUB R3, R0, #5
```

7. Tulis sebuah program untuk membaca lokasi memori LIST1 dan LIST2 dan menyimpan jumlahnya dalam LIST3.

```
LIST1   DCD 0x00002345
LIST2   DCD 0X00011111
LIST3   DCD 0x0
```

8. Buatlah program perkalian dua bilangan dengan subrutin.

9. Buatlah program untuk menjumlahkan delapan bilangan menggunakan pengalamatan tidak langsung.

```
LIST DCB 0x5, 0x2,0x6,0x7 ,0x9,0x1,0x2,0x08
```

10. Buatlah program untuk menjumlahkan delapan bilangan menggunakan pengalamatan Post-index.

```
LIST DCB 0x5, 0x2,0x6,0x7 ,0x9,0x1,0x2,0x08
```

11. Tulis program untuk mengonversi bahasa HLL berikut ke instruksi ARM.

```
IF R1=R2 AND R3>R4 then

R1= R1 +1
Else
R3=R3 +R3*8
Endif
```

12. Apa isi R4 setelah eksekusi program berikut.

```
__main

   LDR R1, =0xFF00FF
ADR R0, LIST1
LDR R2, [R0]
AND R4, R2, R1

LIST1    DCD 0X45073487
```

13. Buatlah program untuk mengonversi HLL berikut ke bahasa assembly.

```
If R1=R2 then
R3= R3+1
IF R1<R2 Then
R3=R3-1
If R1>R2 Then
R3=R3-5
```

14. Tulislah subrutin untuk menghitung nilai Y dimana Y X*2 + x + 5, asumsikan x diwakili oleh

```
LIST DCB 0x5

LIST1 DCB  0x5
```

15. Tulis program untuk memutar R1 16 kali; asumsikan R1 berisi 0x12345678.

16. Tulis sebuah program untuk membandingkan dua angka dan menyimpan angka terbesar dalam LIST lokasi memori.

```
M1     EQU   5
N1     EQU   6
LIST2  DCB   0x0
```

17. Tulis program untuk membaca DAFTAR lokasi memori kata dan menghapus posisi bit B4 hingga B7 dari register R5; asumsikan R5 berisi 0XFFFFFFF.

```
LDR R0, =0x000000F0
LDR R5, =0xFFFFFFFF
```

18. Tulis program untuk memuat Register R1, R2, R3, dan R4 dari lokasi memori LIST

```
LIST DCD  0x12345AAA, 0x0000BBBB, 0x0000CCCC, 0X00000DDD
```

DAFTAR PUSTAKA

A. H. Sutopo, 2012. “Teknologi informasi dan komunikasi dalam Pendidikan”. Yogyakarta: Graha Ilmu.

E. Prastyo dan N. 2015, “Pengembangan Media Pembelajaran Interaktif Menggunakan Adobe Flash CS3 Pada Mata Diklat PLC Di Jurusan Teknik elektro Industri,” Pengembanan Media Pembelajaran Interaktif.

G. Depari. 2013. Teknik Digital. Bandung: Nuansa Aulia.

H. A. Fatta. 2007. Analisis dan Perancangan Sistem Informasi Untuk Keunggulan Bersaing Perusahaan dan Organisasi Modern, Yogyakarta: Andi.

Hall, D. V. 1993. Microprocessors and Interfacing: Programming and Hardware, 2/E. Lake Forest: Glencoe Division of Macmillan/McGraw-Hill School Publishing Company.

Hariyadi, S., and M. Kom. 2017. "Digital Trainer Laboratorium Teknik Elektro Ft-Umsb." Menara Ilmu.

Hill, F. J. and Peterson, G. R. 1981. Switching Theory and Logical Design. New York: John Wiley & Sons, Inc.

M. Azhar Arsyad. 2008. Media Pembelajaran, Jakarta: Rajafrafindo Persada.

Maini, Anl K. 2007. “Digital Electronics Principles, Devices and Applications”, Weley, England.

Mano, M. M. 1992. Computer System Architecture (3rd Edition). Englewood Cliff: Prentice Hall, Inc.

Mismail, B. 1998. Dasar-Dasar Rangkaian Logika Digital. Bandung: Penerbit ITB.

Murdocca, M. and Heuring, V. P. 1999. Principles of Computer Architecture. Englewood Cliff: Prentice Hall, Inc.

P. Moh. Nazir, Metode Penelitian, Bogor: Ghalia Indonesia, 2014.

R. A. M.Shalahudin, Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Objek, Bandung: Informatika, 2016.

R. Rickman. 2016. Unity Tutorial Game Engine, Bandung: Informatika,

Smith, R. J. and Dorf, R. C. 1992. Circuits, Devices and Systems. New York: John wiley & Sons, Inc.

Sugiyono. 2017. Metode Penelitian dan Pengembangan Research and Development, Bandung: Alfabeta.

Tarigan, Pernantin, “Dasar Teknik Digital”, 2012, Graha Ilmu, Yokyakarja.

Tocci, R. J. & Widmer, R. S. 2001. Digital Systems: Principles and Applications, 8th Edition. Englewood Cliff: Prentice Hall, Inc.

Tokheim, Roger. 2008. “Digital Electronics Principles and Applications”, Seventh Edition, Mc Graw-Hil,US.

W. Sanjaya. 2012.Media Komunikasi Pembelajaran, Jakarta: Kencana Prenada Media Group.

Wasito S., 1994. " Data Sheet Book 1 Data IC Linier, TTL dan CMOS", Elex Media Komputindo, Jakarta.

Winder, Steve. 2008. "Power Supplies for LED Drivers", Elsevier & Newnes, UK.

Y. Tadjiri. 2015. "Pembangunan Media Pembelajaran Interaktif Elekronika Dasar Untuk SMK Jurusan Tekni Elektronika Industri Study Kasus SMK Prakarya Internasional Bandung," Jurnal Ilmiah Komputer dan Informatika.

DASAR KOMPUTER DIGITAL

Dr. Ir. Agus Wibowo, M.Kom, M.Si, MM

BIO DATA PENULIS

Penulis memiliki berbagai disiplin ilmu yang diperoleh dari Universitas Diponegoro (UNDIP) Semarang. dan dari Universitas Kristen Satya Wacana (UKSW) Salatiga. Disiplin ilmu itu antara lain teknik elektro, komputer, manajemen dan ilmu sosiologi. Penulis memiliki pengalaman kerja pada industri elektronik dan sertifikasi keahlian dalam bidang Jaringan Internet, Telekomunikasi, Artificial Intelligence, Internet Of Things (IoT), Augmented Reality (AR), Technopreneurship, Internet Marketing dan bidang pengolahan dan analisa data (komputer statistik).

Penulis adalah pendiri dari Universitas Sains dan Teknologi Komputer (Universitas STEKOM) dan juga seorang dosen yang memiliki Jabatan Fungsional Akademik Lektor Kepala (Associate Professor) yang telah menghasilkan puluhan Buku Ajar ber ISBN, HAKI dari beberapa karya cipta dan Hak Paten pada produk IPTEK. Penulis juga terlibat dalam berbagai organisasi profesi dan industri yang terkait dengan dunia usaha dan industri, khususnya dalam pengembangan sumber daya manusia yang unggul untuk memenuhi kebutuhan dunia kerja secara nyata.

YAYASAN PRIMA AGUS TEKNIK

PENERBIT :

YAYASAN PRIMA AGUS TEKNIK

JL. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-623-5734-63-7 (PDF)

9 786235 734637

Dr. Ir. Agus Wibowo, M.Kom, M.Si, MM

DASAR KOMPUTER DIGITAL

YAYASAN PRIMA AGUS TEKNIK

PENERBIT :
YAYASAN PRIMA AGUS TEKNIK
JL. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id