CSG2F3 – Sistem dan Logika Digital (SLD)

REPRESENTASI DATA

Tim Dosen SLD

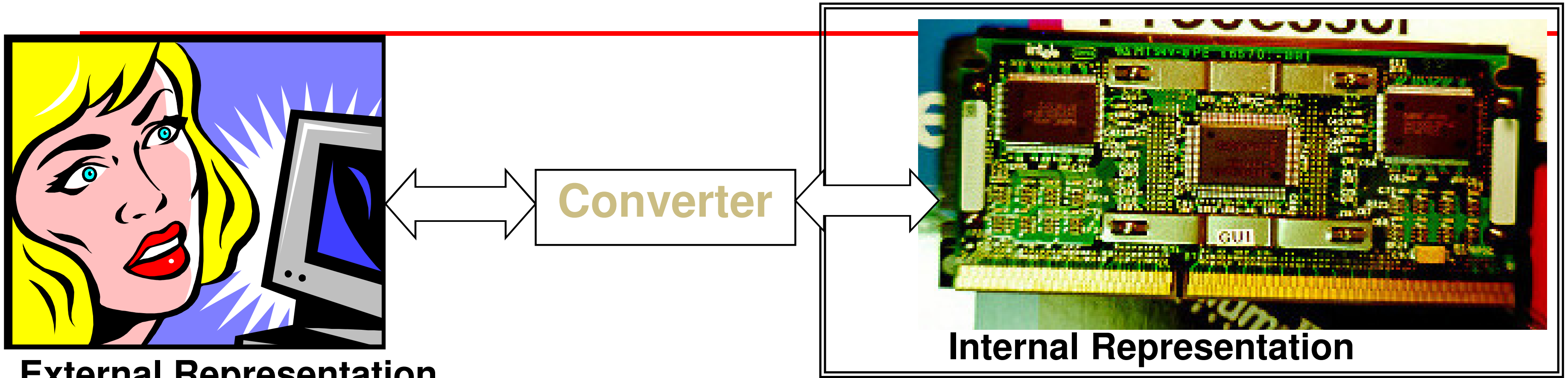
KK Telematika – FIF

Telkom University

Pokok Bahasan

- Representasi data
- Bit, byte, dan word
- Representasi data numerik dan basis bilangan
- Representasi komplemen dua dan bertanda
- Sistem *fixed point* dan *floating point*
- Representasi data bukan numerik (kode karakter)

Representasi data (1)



External Representation

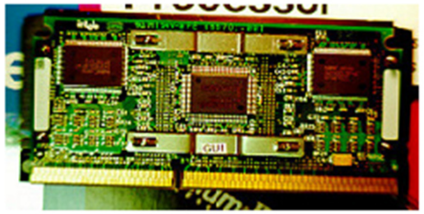
Representasi Eksternal adalah suatu cara untuk merepresentasikan dan memanipulasi informasi oleh *programmer* dengan suatu bahasa pemrograman atau notasi bahasa perintah lainnya –> Agar nyaman bagi *programmer* (user).

Representasi Internal adalah suatu cara untuk menyimpan dan memanipulasi informasi secara aktual di dalam sistem komputer –> Agar mudah dalam membangun perangkat keras.

Informasi ≈ program & data ≈ deretan bit

→ akses/manipulasi terhadap informasi ≈ akses/operasi *(arithmetic/logic)* terhadap deretan bit

Representasi data (2)



$-\infty \ldots \infty$	*Finite precision number*
Decimal : $X_{(10)}$	Binary : $X_{(2)}$

- Bilangan berpresisi terbatas berpeluang memunculkan ‘kesalahan’ (dari segi matematika klasik), tetapi bisa menjadi ‘kebenaran’ sebagai konsekuensi logis dari keterbatasan mesin tersebut
- Kesalahan yang dapat terjadi:
  - *overflow error*
  - *underflow error*
  - *unrepresentable*

ARIANE 5

- Rocket seharga $7 billion
- Diluncurkan pada 4 Juni 1996
- Menyimpang dari jalur 40 detik setelah peluncuran, putus dan meledak.
- Kegagalan tersebut disebabkan ketika komputer yang mengendalikan roket terjadi overflow (membutuhkan komputasi lebih dari 16-bit) dan akhirnya jatuh.
- Ariane 5 memiliki mesin lebih cepat dibandingkan Ariane 4, dan menghasilkan nilai komputasi yang lebih besar untuk kontrol komputer, sehingga menyebabkan overflow dan akhirnya meledak.

Bit dan Byte

- Apa bedanya antara bit dan byte ?
- 1 byte = 8 bit (binary digit)
  - Range Binary: 00000000_2 - 11111111
  - Range Decimal: 0_{10} - 255_{10}
  - Range Hexadecimal: 00_{16} - FF_{16}
    - representasi bilangan basis 16
    - Menggunakan karakter '0' - '9' dan 'A' - '
  - Range Octal: ... - ...
    - 000_8 - 377_8
- 1 nibble = ... bit = ... byte

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Word Size (1)

- Word merupakan sejumlah bit berukuran tetap yang ditangani secara bersama-sama oleh komputer
- Sebuah word dapat merupakan:
  - ukuran register
  - ukuran suatu tipe data
  - jumlah data dalam sekali transfer
  - lebar alamat suatu memori
- Kebanyakan mesin menggunakan 32 bit (4 byte)
- Sistem *high-end* menggunakan 64 bit (8 byte)
- Satuan word adalah byte
- Contoh:
  - Intel: 1 word = 16 bit (8086)
  - Tetap kompatibel dengan, x86, IA-32, IA-64

Word Size (2)

Year	Computer Architecture	Word Size *w*
(1837)	Analytical engine	50 d
1941	Zuse Z3	22 b
1942	ABC	50 b
1944	Harvard Mark I	23 d
1946 (1948) {1953}	ENIAC (w/Panel #16) {w/Panel #26}	10 d
1951	UNIVAC I	12 d
1952	IAS machine	40 b
1952	IBM 701	36 b
1952	UNIVAC 60	*n* d
1953	IBM 702	*n* d
1953	UNIVAC 120	*n* d
1954 (1955)	IBM 650 (w/IBM 653)	10 d
1954	IBM 704	36 b
1954	IBM 705	*n* d
1954	IBM NORC	16 d
1956	IBM 305	*n* d
1957	Autonetics Recomp I	40 b
1958	UNIVAC II	12 d
1958	SAGE	32 b
1958	Autonetics Recomp II	40 b
1959	IBM 1401	*n* d
1959 (TBD)	IBM 1620	*n* d
1960	LARC	12 d
1960	CDC 1604	48 b
1960	IBM 1410	*n* d
1960	IBM 7070	10 d
1960	PDP-1	18 b
1961	IBM 7030 (Stretch)	64 b
1961	IBM 7080	*n* d
1962	UNIVAC III	25 b, 6 d
1962	Autonetics D-17B Minuteman I Guidance Computer	27 b
1962	UNIVAC 1107	36 b
1962	IBM 7010	*n* d
1962	IBM 7094	36 b
1963	Gemini Guidance Computer	39 b
1963 (1966)	Apollo Guidance Computer	15 b
1963	Saturn Launch Vehicle Digital Computer	26 b
1964	CDC 6600	60 b
1964	Autonetics D-37C Minuteman II Guidance Computer	27 b
1965	IBM 360	32 b
1965	UNIVAC 1108	36 b
1965	PDP-8	12 b
1970	PDP-11	16 b
1971	Intel 4004	4 b
1972	Intel 8008	8 b
1972	Calcomp 900	9 b
1974	Intel 8080	8 b
1975	ILLIAC IV	64 b
1975	Motorola 6800	8 b
1975	MOS Tech. 6501 MOS Tech. 6502	8 b
1976	Cray-1	64 b
1976	Zilog Z80	8 b
1978 (1980)	Intel 8086 (w/Intel 8087)	16 b
1978	VAX-11/780	32 b
1979	Motorola 68000	32 b
1982 (1983)	Motorola 68020 (w/Motorola 68881)	32 b
1985	ARM1	32 b
1985	MIPS	32 b
1989	Intel 80486	16 b
1989	Motorola 68040	32 b
1991	Alpha	64 b
1991	Cray C90	64 b
1991	PowerPC	32 b
2000	IA-64	64 b
2002	XScale	32 b

d = desimal; b = bit

Representasi Data

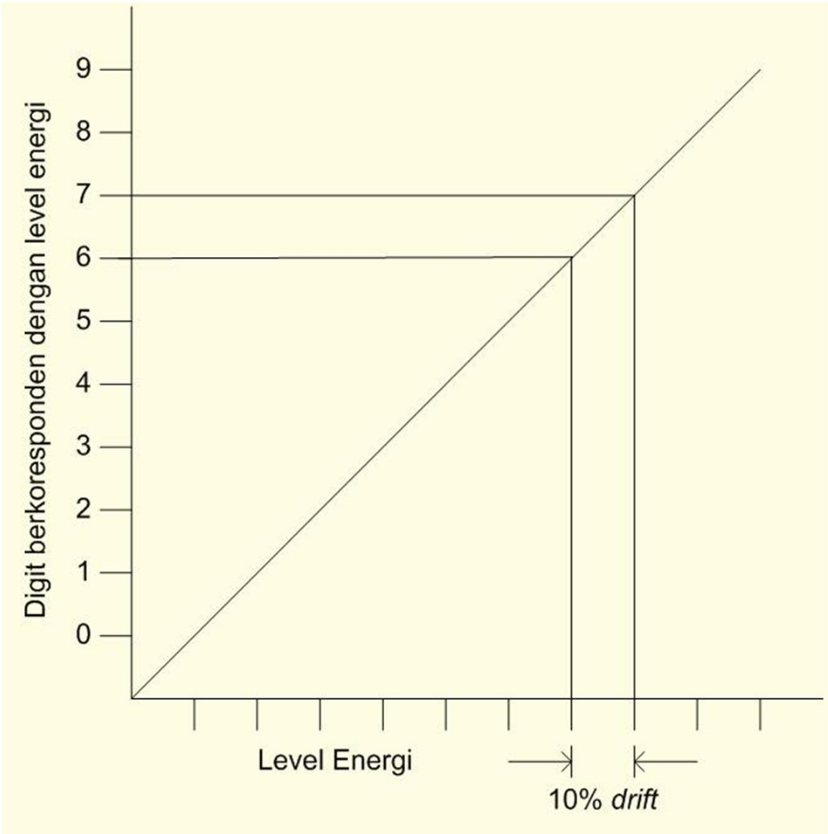
- Contoh ukuran Objek pada C (dalam byte)

Tipe Data C	Compaq Alpha	Typical 32-bit	Intel IA32
int	4	4	4
long int	8	4	4
char	1	1	1
short	2	2	2
float	4	4	4
double	8	8	8
long double	8	8	10/12
char *	8	4	4

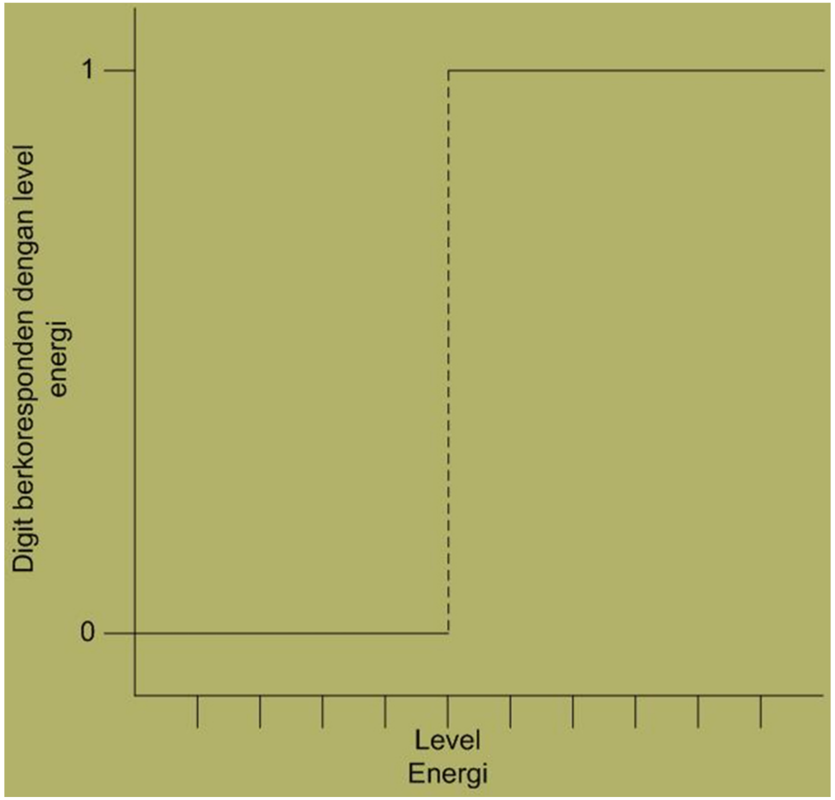
Bilangan Desimal (1)

- Representasi bilangan basis 10
  - Itu kenapa jari tangan dikenal sebagai "digits"
  - Representasi bilangan natural untuk transaksi finansial
- Kenapa komputer sekarang menggunakan sistem biner dan bukan desimal ?
- Implementasi secara elektronik
  - Sukar disimpan
    - ❖ ENIAC (komp. pertama kali) menggunakan 10 *vacuum tubes* per digitnya
  - Sukar dikirimkan
    - ❖ Memerlukan presisi yg tinggi untuk meng-encode sinyal dengan 10 level pada *single wire*
  - Kehandalan komponen elektronika turun sejalan dengan waktu penggunaannya (*drift*)
    - ❖ Perubahan sebesar 10 % saja sudah mengubah nilai
  - Sulit untuk diimplementasikan pada fungsi logika digital
    - ❖ *Addition, multiplication, etc.*

Bilangan Desimal (2)



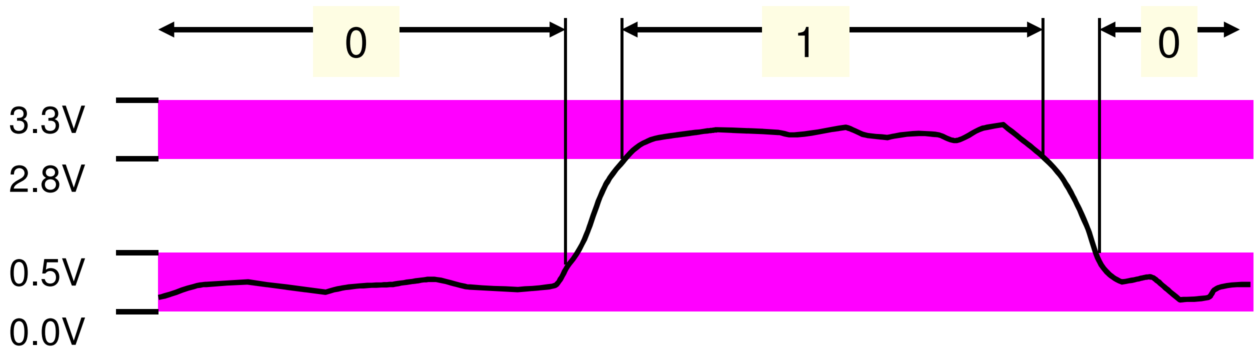
Desimal



Biner

Bilangan Biner

- Representasi bilangan basis 2
  - Representasi 15213_{10} as 11101101101101_2
  - Representasi $1{,}20_{10}$ as $1{,}0011001100110011 \ldots 0011_2$
  - Representasi $1{,}5213 \times 10^4$ as $1{,}1101101101101_2 \times 2^{13}$
- Implementasi Elektronik
  - Mudah untuk disimpan sebagai elemen yang *bistable* (hanya ada 2 nilai yang berbeda jauh)
  - Lebih handal pada *wire* yang noise dan *inaccurate*
  - Mudah diimplementasikan pada fungsi logika digital



Jenis-Jenis Bilangan Biner

- Bilangan bulat biner tak bertanda (*unsigned integer*)
- Bilangan bulat biner bertanda (*signed integer*)
  - *Sign/magnitude*
  - Komplemen 2 (*radix complement*)
  - Komplemen 1 (*diminished radix complement*)
  - *Binary Coded Decimal (BCD)*
- Bilangan pecahan biner (*floating point*)
- Excess 2^{m-1}

Bilangan bulat Biner tak bertanda
(*Unsigned Integer*)

$$d_n d_{n-1} d_{n-2} \ldots d_3 d_2 d_1 d_0 = d_n r^n + d_{n-1} r^{n-1} + d_{n-2} r^{n-2} \,..\, d_3 r^3 + d_2 r^2 + d_1 r^1 + d_0 r^0$$

d = nilai bilangan;
r = radix (basis bilangan) = jumlah simbol maksimum
n = posisi bilangan, LSB = posisi ke-0

Cakupan bilangan yang bisa disajikan: $\mathbf{0 \leq I \leq 2^{m-1}}$

Misal bilangan 16 bit: $0 \leq I \leq 2^{16-1} = 0 \leq I \leq 32768$

Konversi dari N_R ke N_r : R = basis desimal dan r = basis bilangan lainnya

$$N_R = d_n r^n + d_{n-1} r^{n-1} + d_{n-2} r^{n-2} \,..\, d_3 r^3 + d_2 r^2 + d_1 r^1 + d_0 r^0$$

Biner ke desimal: $101011_2 = 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$
$= 32 + 0 + 8 + 0 + 2 + 1 = 43_{10}$

Konversi Bilangan (1)

Desimal ke biner $43_{10} = \ldots_2$

$43 : 2 = 21$; sisa 1 → d_0 *(LSB)*

$21 : 2 = 10$; sisa 1 → d_1

$10 : 2 = 5$; sisa 0 → d_2

$5 : 2 = 2$; sisa 1 → d_3

$2 : 2 = 1$; sisa 0 → d_4

$1 : 2 = 0$; sisa 1 → d_5 Jadi $43_{10} = 101011_2$

Latihan:

a. $10101010_2 = \ldots_{10}$

b. $5000_{10} = \ldots_2$

c. $5000_{10} = \ldots_8$

d. $5000_{10} = \ldots_{16}$

e. $ABCD_{16} = \ldots_{10}$

f. $ABCD_{16} = \ldots_8$

g. $1001011010100101_2 = \ldots_{10}$

h. $1001011010100101_2 = \ldots_8$

i. $1001011010100101_2 = \ldots_{16}$

(solusi)

Konversi Bilangan (2)

Apa kesimpulan yang dapat diperoleh ?

- Konversi bilangan biner ke bilangan oktal atau sebaliknya dapat dilakukan dengan lebih mudah dan lebih cepat dibanding konversi bilangan tersebut ke bilangan desimal
- Konversi bilangan biner ke bilangan heksadesimal atau sebaliknya dapat dilakukan dengan lebih mudah dan lebih cepat dibanding konversi bilangan tersebut ke bilangan desimal
- Konversi representasi data eksternal ke data internal atau sebaliknya memerlukan proses lebih panjang dan lebih rumit

Signed Integer: Sign/magnitude (1)

- Dapat merepresentasikan bilangan negatif
- Simple: Bit terkiri (*Most Significant Bit - MSB*) dianggap sebagai bit tanda (*sign bit*)
  - Bit 0 → bilangan positif
  - bit 1 → bilangan negatif
- Bit selain MSB sebagai nilai *magnitude* absolut bilangan
- Cakupan nilai (I) yang dapat direpresentasikan:

$$-(2^{m-1}-1) \leq I \leq +(2^{m-1}-1)$$

m = banyaknya bit

- Misal:

Untuk bilangan 16 bit: $-(2^{16-1}-1) \leq I \leq +(2^{16-1}-1)$

$= -32767 \leq I \leq +32767$

Signed Integer: Sign/magnitude (2)

- Contoh m = 3:

Biner	Nilai	Biner	Nilai
000	0		
001	+1	101	-1
010	+2	110	-2
011	+3	111	-3

- Masalah: Apakah 000 = +0 sama dengan 100 = -0 ???
  - Bagi manusia: +0 dan –0 adalah sama
  - Bagi komputer: +0 dan –0 adalah beda, karena komputer membandingkan 2 buah bilangan secara bit per bit !!

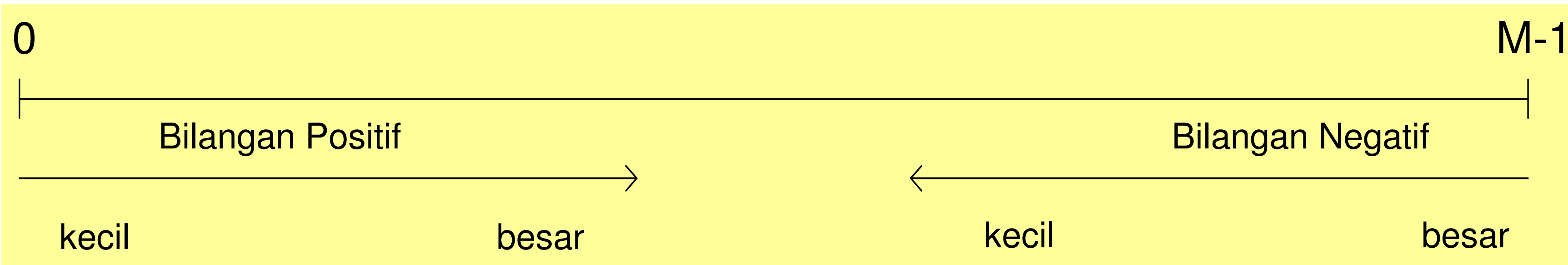
Signed Integer: Komplemen 2 *(Radix Complement)* (1)

- Biner dalam bentuk 2's *complement*
- Disebut sebagai aritmatika modular (modulo)

  $$A = B \pmod{M}$$

  Bilangan berapapun ditambah dengan M hasilnya tetap !
- Bilangan biner dengan jumlah bit = m, mempunyai modulo $M = 2^m$
- $M = 1\underbrace{000...0}_{m\ \text{nol}}$, bilangan terbesar: $\underbrace{111...1}_{m\ \text{satu}} = 2^m - 1 = M - 1$

  = bilangan terkecil



- Bilangan positif, hitung ke atas mulai dari nol: $(+X = X)$
- Bilangan negatif, hitung ke bawah dari modulus M: $(-X = M - X)$

Signed Integer: Komplemen 2 *(Radix Complement)* (2)

- Contoh 1:

  Misal m = 4, maka $M = 2^m = 2^4 = 16$

  $+6_{10} = 0110_2$

  $-7_{10} = \ldots_2$

  $-X = M - X \Rightarrow -7_{10} = 16 - 7 = 9 = 1001_2$ (cara I)

- Contoh 2:

  $-10_{10} = \ldots_2$

  $-10_{10} = 16 - 10 = +6 = 0110_2$

  Jadi $+6_{10} = -10_{10}$??? (ambigu !)

- Solusinya dibuat aturan sbb:

  IF MSB = 0 THEN bilangan adalah POSITIF

  (magnitude = *unsigned integer*)

  ELSE bilangan adalah NEGATIF

  (magnitude = M – X)

  Jadi 0110_2 hanya untuk bilangan $+6_{10}$ saja, -10_{10} = ??

Signed Integer: Komplemen 2 *(Radix Complement)* (3)

m = 4

Desimal	Komplemen 2	Sign/Magnitude
+0	0000	0000
+1	0001	0001
+2	0010	0010
+3	0011	0011
+4	0100	0100
+5	0101	0101
+6	0110	0110
+7	0111	0111
−0	Tidak digunakan	1000
−1	1111	1001
−2	1110	1010
−3	1101	1011
−4	1100	1100
−5	1011	1101
−6	1010	1110
−7	1001	1111
−8	1000	Tidak dapat dg 4 bit

Signed Integer: Komplemen 2 (*Radix Complement*) (4)

- M-1 = 2^{m-1} = 111...1 (satu semua)
- Bilangan biner yang digunakan untuk mengurangi 1 akan menghasilkan biner kebalikannya (1–0 = 1; 1–1 = 0)
  - Pengurangan dengan M-1 = inversi (komplemen) !
- Modifikasi rumus:

  –X = M–X menjadi:

$$-X = \overbrace{(M-1)-X}^{\text{komplemen}} + 1$$

- Contoh: Untuk m = 5, maka $-5_{10} = \ldots_2$

  Cara II: *(lebih sederhana)*

  $+5_{10}$ = 00101 ← nilai X dalam biner

  11010 ← dikomplemenkan: bit 1 → 0, bit 0 → 1

  1 +

  11011 ← setelah ditambah 1

  11011_2 → -5_{10} dalam komplemen 2

Signed Integer: Komplemen 2 (*Radix Complement*) (5)

Latihan: (untuk m = 5)

(a) $-6_{10} = \ldots_2$
(b) $-9_{10} = \ldots_2$
(c) $-13_{10} = \ldots_2$
(d) $-15_{10} = \ldots_2$
(e) $-18_{10} = \ldots_2$
(f) $10101_2 = \ldots_{10}$
(g) $11001_2 = \ldots_{10}$
(h) $10000_2 = \ldots_{10}$
(i) $11111_2 = \ldots_{10}$
(j) $+0_{10} = \ldots_2$
(k) $01010_2 = \ldots_{10}$

(solusi)

Signed Integer: Komplemen 2
(*Radix Complement*) (6)

- Cakupan nilai: $-\left(2^{m-1}\right) \leq I \leq +\left(2^{m-1}-1\right)$
- Contoh untuk bilangan 16 bit: $-(2^{16-1}) \leq I \leq +(2^{16-1}-1)$
- $= -32768 \leq I \leq +32767$ (tipe *sign int*)
- Aritmatika Penjumlah

+	0	1
0	0	1
1	1	0*

* : melibatkan *carry* untuk kolom berikutnya

n= 5 bit

(a)

```
    00101   (+5)
+   00110   (+6)
   ------
    01011   (+11)
```

(b)

```
     00111   (+7)
+    11110   (-2)
   -------
   1 00101   (+5)
```

carry register

Signed Integer: Komplemen 2
(*Radix Complement*) (7)

Berapa komplemen 2 dari 00000 ?

(c)		11011	(-5)
+		11100	(-4)
	1	10111	(-9)

Ke *carry register*

(d) 00000 →

$$\begin{array}{r} 11111 \\ +\quad 1 \\ \hline 1\,|\,00000 \end{array}$$

→ tidak ada ambiguitas +0 dan –0

Komplemen 2 banyak diterapkan di komputer !!

(e)	00101	(+5)
+	01110	(+14)
	10011	(-13) → ??? → ***Overflow***

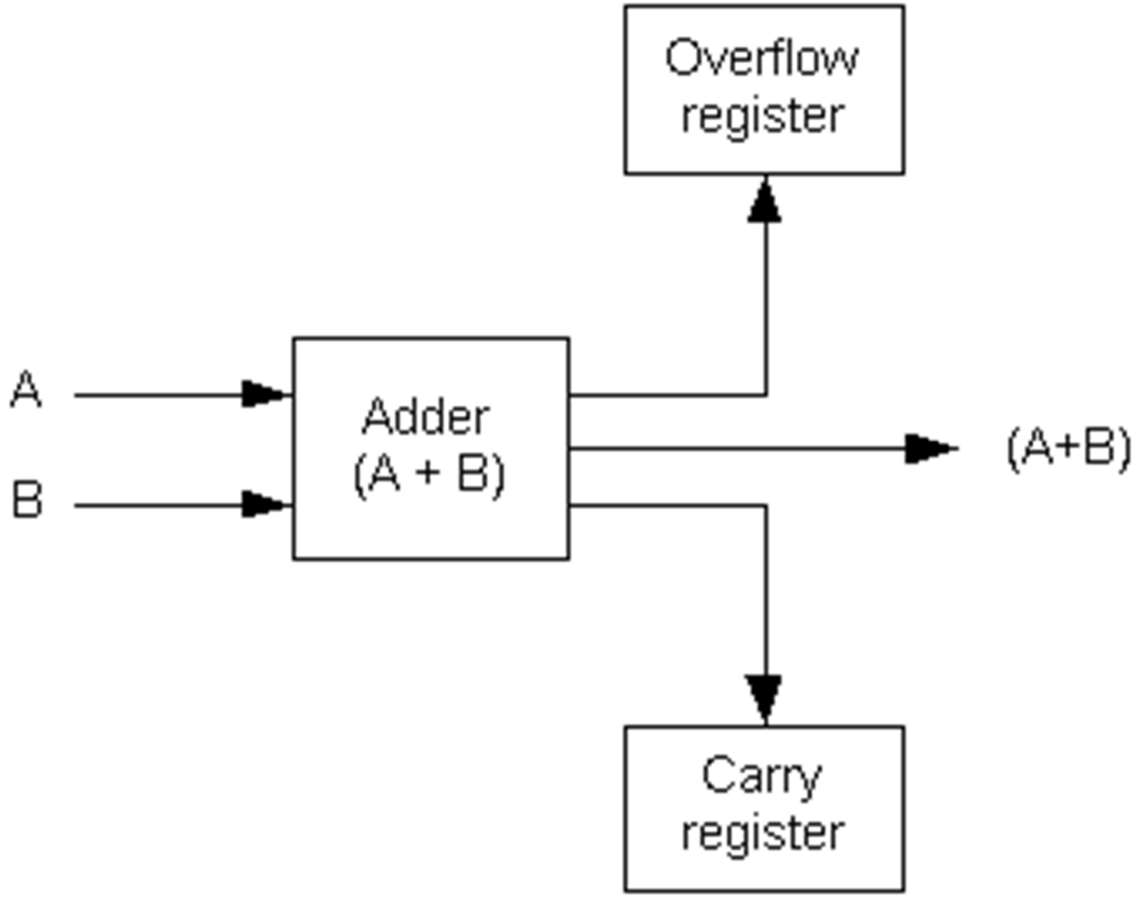
Kapan Overflow terjadi ???:

bilangan positif + bilangan positif = bilangan negatif

bilangan negatif + bilangan negatif = bilangan positif

Signed Integer: Komplemen 2 (*Radix Complement*) (8)

Organisasi fungsional untuk Penjumlahan:



Signed Integer: Komplemen 2
(*Radix Complement*) (9)

Latihan:

Dengan m = 6:

(a) (+6) + (-7)

(b) (+7) + (-6)

(c) (-15) + (-16)

(d) (-20) + (-20)

(e) (+31) + (-31)

(f) (-32) + (+12)

(solusi)

Signed Integer: Komplemen 2 (*Radix Complement*) (10)

- Bagaimana dengan Pengurangan ?
- Dapat dilakukan dengan unit pengurangan + register borrow + register overflow
- Perancang komputer:
  - Lebih suka manfaatkan unit penjumlahan yang sudah ada + unit komplementor
    - Biaya lebih murah
    - Perawatan lebih mudah
  - Modifikasi:
    - $D = Y - X$ diubah menjadi $D = -X + Y$

Signed Integer: Komplemen 2 (*Radix Complement*) (11)

Contoh: m = 4:

(a) (+3) – (+2)

```
   0011   (+3)          ⇉          0011   (+3)
 – 0010   (+2)                   + 1110   (−2)
                                 1|0001   (+1)
```

ke carry register

(b) (+3) – (+5)

```
   0011   (+3)          ⇉          0011   (+3)
 – 0101   (+5)                   + 1011   (−5)
                                   1110   (−2)
```

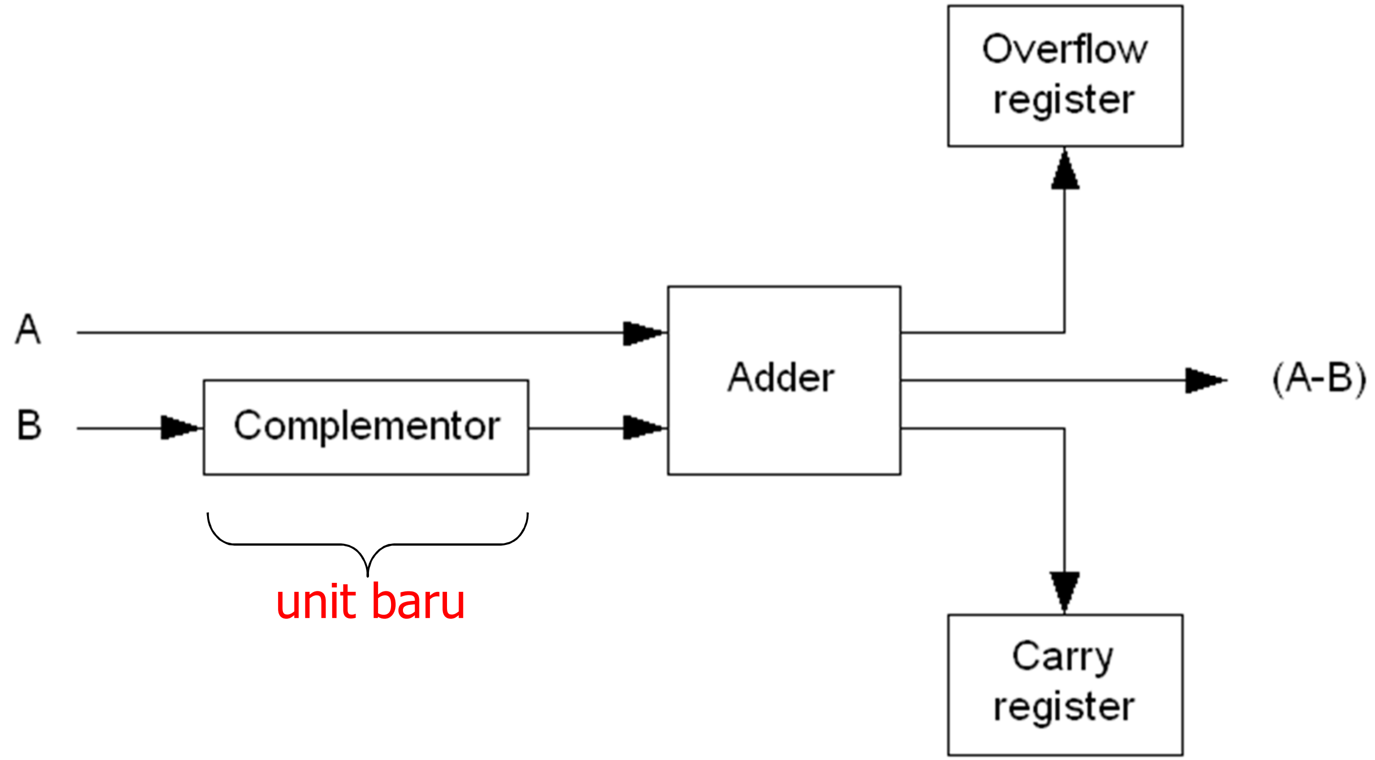
(c) (-2) – (-5)

```
   1110   (−2)          ⇉          1110   (−2)
 – 1011   (−5)                   + 0101   (+5)
                                 1|0011   (+3)
```

ke carry register

Signed Integer: Komplemen 2 (*Radix Complement*) (12)

Organisasi fungsional untuk Pengurangan:



Signed Integer: Komplemen 1
(*Diminished Radix Complement*) (1)

- *Diminished* = mengurangi
- Merupakan varian dari komplemen 2
- Komplemen dilakukan dengan cara:
  - Ganti semua bit 1 dengan 0 dan semua bit 0 dengan 1
  - Tanpa penambahan dengan +1
  - Carry **tidak** dibuang tetapi ditambahkan

$$+X = X$$

$$-X = \overbrace{(M-1)-X}^{\text{komplemen}} + \cancel{1} \Rightarrow -X = (M-1)-X$$

- **Cakupan** nilai:

$$-\left(2^{m-1}-1\right) \le I \le +\left(2^{m-1}-1\right)$$

- Misal:

Untuk bilangan 16 bit: $-(2^{16-1}-1) \le I \le +(2^{16-1}-1)$

$= -32767 \le I \le +32767$ (sama dengan *sign/magnitude*)

Signed Integer: Komplemen 1
(*Diminished Radix Complement*) (2)

Contoh (m=5) :

(a)

$$
\begin{array}{rrl}
 & 00111 & (+7) \\
- & \underline{00011} & (+3)
\end{array}
\qquad \text{komplemen} \rightarrow \qquad
\begin{array}{rrl}
 & 00111 & (+7) \\
+ & 11100 & (-3) \\
\hline
 & 1\;00011 & \\
+ & 1 & \\
\hline
 & 00100 & (+4)
\end{array}
$$

(b)

$$
\begin{array}{rrl}
 & 10101 & (-10) \\
+ & \underline{11100} & (-3)
\end{array}
\qquad
\begin{array}{l}
\text{tetap} \rightarrow \\
\text{tetap} \rightarrow
\end{array}
\qquad
\begin{array}{rrl}
 & 10101 & (-10) \\
+ & 11100 & (-3) \\
\hline
 & 1\;10001 & \\
+ & 1 & \\
\hline
 & 10010 & (-13)
\end{array}
$$

Signed Integer: Komplemen 1
(Diminished Radix Complement) (3)

(c)	00111	(+7)			00111	(+7)
-	01011	(+11)	komplemen →	+	10100	(-11)
					11011	

(d)	10000	(-15)	tetap →		10000	(-15)
+	10000	(-15)	tetap →	+	10000	(-15)
					1 00000	
				+	1	
					00001	(+1) → ???

Muncul kembali ambiguitas +0 dan –0
→ Komplemen 1 jarang digunakan

Binary Coded Decimal (BCD) (1)

- Mengapa BCD digunakan ?
  - Karena konversi bilangan desimal ke komplemen 2 dapat mendominasi waktu eksekusi
- Konversi: tiap digit desimal → empat bit biner
  - Contoh: $0_{10} = 0000_2$; $1_{10} = 0001_2$; ...; $9_{10} = 1001_2$
  - Tanda '+' dan '–' → dengan kombinasi yang belum dipakai, contoh : 1010 = '+' dan 1011 = '–'
- Aplikasi apa yang menggunakan BCD ?
  - Aplikasi yang banyak melibatkan data input maupun output namun sangat sedikit pemrosesan numerik (contoh : *payroll* dan *inventory*)

Binary Coded Decimal (BCD) (2)

desimal	BCD	desimal	BCD
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Sign Digit	BCD 8 4 2 1	Sign	Notes
A	1 0 1 0	+	
B	1 0 1 1	–	
C	1 1 0 0	+	Preferred
D	1 1 0 1	–	Preferred
E	1 1 1 0	+	
F	1 1 1 1	+	Unsigned

- Contoh

1.234_{10} = 0001 0010 0011 0100 1010

1 2 3 4 '+'

$–567_{10}$ = 0101 0110 0111 1011

5 6 7 '–'

Binary Coded Decimal (BCD) (3)

- Apa kekurangan BCD ?
  - Operasi aritmatika lebih lama *(lookup table)* dibanding *sign/magnitude* maupun 2's dan 1's *complement*
  - Penjumlahan bilangan dalam BCD dilakukan per digit desimal (4-bit) dan menghasilkan *carry* desimal (bukan penjumlahan bit per bit)
- Contoh Aritmatika: *(lookup table)*

```
 (0001)(0001)          ← carry
      0110 0011 (+63)
     +0100 1001 (+49)
 ----------------------
 0001 0001 0010 (+112)
```

Binary Coded Decimal (BCD) (4)

- Contoh 2: Jika hasil penjumlahan > 9, tambahkan dengan 6 (0110)

```
                 1
  0000 0011 0101 1001  (+ 359)
 +1001 0101 0110 1001  (+9569)
 -----------------------------
  1001 1000 1100 0010  (+9928)
            1
 +               0110 0110
 --------------------------
  1001 1001 0010 1000
     9    9    2    8
```

Floating Point *(1)*

Bilangan *floating point* (pecahan):

$$d_n d_{n-1} d_{n-2} \ldots d_2 d_1 d_0 \,.\, d_{-1} d_{-2} d_{-3} \ldots =$$

$$d_n r^n + d_{n-1} r^{n-1} + d_{n-2} r^{n-2} \ldots + d_2 r^2 + d_1 r^1 + d_0 r^0 + d_{-1} r^{-1} + d_{-2} r^{-2} + d_{-3} r^{-3} \ldots$$

Contoh :

(a) $110{,}01101_2 = 1x2^2 + 1x2^1 + 0x2^0 + 0x2^{-1} + 1x2^{-2} + 1x2^{-3} + 0x2^{-4} + 1x2^{-5}$

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

$= 6 + 0{,}25 + 0{,}125 + 0{,}03125 = 6{,}40625_{10}$

(b) $101{,}101_2 = 1x2^2 + 0x2^1 + 1x2^0 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3}$

$= 4 + 0 + 1 + ½ + 0 + 1/8$

$= 5 + 0{,}5 + 0{,}125 = 5{,}625_{10}$

Floating Point (2)

(c) $0{,}375_{10} = \ldots_2$

Representasi biner

$$
\begin{array}{r}
0{,}375 \\
\times\ 2 \\
\hline
0{,}750 \\
\times\ 2 \\
\hline
1{,}500 \\
\times\ 2 \\
\hline
1{,}000
\end{array}
$$

$$0{,}375_{10} = 0{,}011_2$$

(d) $0{,}3_{10} =$

Representasi biner

$$
\begin{array}{r}
0{,}3 \\
\times\ 2 \\
\hline
0{,}6 \\
\times\ 2 \\
\hline
1{,}2 \\
\times\ 2 \\
\hline
0{,}4 \\
\times\ 2 \\
\hline
0{,}8 \\
\times\ 2 \\
\hline
1{,}6 \\
\times\ 2 \\
\hline
1{,}2 \\
\times\ 2 \\
\hline
0{,}4 \\
\times\ 2 \\
\hline
0{,}8 \\
\cdot \\
\cdot \\
\cdot
\end{array}
$$

Jika akurasinya 5 digit, maka:

$0{,}3_{10} = 0{,}01001_2$

Floating Point (3)

(e) $0{,}375_{10} = \ldots_4$

```
  0,375
    X 4
  -----
  1,500
    X 4
  -----
  2,000
```

$0{,}375_{10} = 0{,}12_4$

(f) $1{,}234_5 = \ldots_{10}$

$= 1x5^0 + 2x5^{-1} + 3x5^{-2} + 4x5^{-3}$

$= 1 + 2x0{,}2 + 3x0{,}04 + 4x0{,}008$

$= 1 + 0{,}4 + 0{,}12 + 0{,}032$

$= 1{,}552_{10}$

(g) $0{,}984_{10} = \ldots_{16}$

(4 digit akurasi)

```
  0,984
   X 16
   ----
   5904
   984   +
   ------
 15,744
   X 16
   ----
   4464
   744   +
   ------
 11,904
   X 16
   ----
   5424
   904   +
   ------
 14,464
   X 16
   ----
   2784
   464   +
   ------
  7,424
```

$0{,}984_{10} = 0{,}FBE7_{16}$

Floating Point R (Notasi Ilmiah) (1)

- Digunakan untuk merepresentasikan bilangan pecahan di dalam semua komputer
- Notasi *floating point R:*

$$R = \pm M \times B^{\pm E}$$

M = Mantissa

E = Eksponen

B = Basis dari eksponen

- B ≠ basis bilangan
- Basis bilangan selalu bernilai 2 (biner), sementara basis eksponen B tidak harus 2

Floating Point R (Notasi Ilmiah) (2)

- Contoh :

  (a) $1.228{,}8 = 1{,}2288 \times 10^3$

  M = 1,2288; E = +3; B = 10

  (b) Tuliskan 1.228,8 dengan notasi ilmiah dan basis eksponen B = 8 !

  $1.228{,}8 = 2{,}40 \times 8^3$ (caranya ?)

  M = 2,4; E = +3; B = 8
- Basis eksponen ditentukan dan di-set secara *hardware,* tidak bisa diubah oleh user
- Contoh :

  PDP-11 dan VAX-11 → B = 2

  IBM-360 dan 370 → B = 16

  Burroughs B5500 → B = 8
- Bilangan *real* dalam komputer direpresentasikan hanya dengan nilai **M** dan **E,** sehingga R = (±M, ±E)

Floating Point: Normalisasi (1)

- Bagaimana cara merepresentasikan *floating point* secara normal ?

  1.228,8 = 1,2288 * 10^3 ???
  = 0,12288 * 10^4 ???
  = 0,012288 * 10^5 ???
  = ... ???

- *Normalized form:*
  - Agar tidak membingungkan
  - Untuk memaksimalkan akurasi dari representasi bilangan
  - Menempatkan koma tepat di sebelah kiri angka penting pertama
  - Sebelum koma = 0, sesudah koma = bukan nol (jika bisa)
  - Nilai mantissa M: (dalam desimal)

$$\frac{1}{B} \le |M| < 1$$

Floating Point: Normalisasi (2)

- Akurasi: jika mantissa = 5 digit, manakah yang paling akurat ?

  $1.228,8 = 0,12288 * 10^4$???

  $= 0,01228 * 10^5$???

  $= 0,00122 * 10^6$???

- Contoh: Normalisasikan !

  (a) $103,5_{10}$ dengan B = 10 ?

  $103,5_{10} = 0,1035 \times 10^3$

  (b) $0,000011101_2$ dengan B = 2 ?

  $0,000011101_2 = 0,11101_2 \times 2^{-4}$

  (c) $10011,110_2 \times 2^{10}$ dengan B = 2 ?

  $10011,110_2 \times 2^{10} = 0,10011110_2 \times 2^{15}$

Floating Point: Normalisasi (3)

(d) $0,000011_2 \times 8^2$ dengan B = 8 ?

$8 = 2^3 \Rightarrow$ pergeseran per 3 bit !

$0,000011_2 \times 8^2 = 0000,011_2 \times 8^1 = 0,011_2 \times 8^1$

(e) $0,0001_2 \times 16^5$ dengan B = 16 ?

$16 = 2^4 \Rightarrow$ pergeseran per 4 bit !

$0,0001_2 \times 16^5 = \Rightarrow$ sudah normal !

(f) $0,000011_2 \times 16^5$ dengan B = 16 ?

$0,000011_2 \times 16^5 = 00000,11_2 \times 16^4 = 0,11_2 \times 16^4$

Representasi Mantissa dan Eksponen

- Model representasi yang mana yang digunakan untuk merepresentasikan mantissa ?
  - *Sign/magnitude*, komplemen 1, komplemen 2 , atau BCD dapat digunakan, tergantung perancang komputer
  - Contoh:
    - PDP-11 dan VAX-11 menggunakan *sign/magnitude*
- Model representasi yang mana yang digunakan untuk merepresentasikan eksponen ?
  - Ke-4 model representasi dapat digunakan
  - Yang banyak digunakan: notasi bias atau notasi *excess-n*, n = 2^{m-1}

Excess-n (1)

- Cara konversi: $e' = e + 2^{m-1}$
  - e' = eksponen bias
  - e = eksponen sebenarnya
  - m = jumlah bit
- Contoh:

  bilangan 8-bit ($m = 8$) → $n = 2^{m-1} = 128$ → excess 128

  Excess-128 dari -3_{10} = ...

  $-3_{10} + 128_{10} = 125_{10} = 01111101_2$

  $0_{10} = \ldots_2$; $-100_{10} = \ldots_2$; $128_{10} = \ldots_2$
- Range nilai eksponen E:

  $$-2^{m-1} \leq E \leq +(2^{m-1}-1)$$

  Untuk m = 8 bit: −128 ... +127 → e' = 0 ... 255
- Sama dengan komplemen 2 dengan bit tanda yang dinegasikan

Excess-n (2)

- Apa manfaat konversi dari eksponen ke eksponen bias ?
  - Mempermudah atau mempercepat perbandingan 2 buah bilangan

Contoh eksponen bias untuk m = 5 bit

e	**e'**	**Biner**
+15	31	11111
+14	30	11110
⋮	⋮	⋮
+1	17	10001
0	16	10000
−1	15	01111
−2	14	01110
⋮	⋮	⋮
−15	1	00001
−16	0	00000

Berapa jumlah bit untuk Mantissa dan Eksponen ? (1)

- Makin banyak jumlah digit mantisa ⇒ semakin presisi
- Makin banyak digit eksponen ⇒ cakupan (*range*) bilangan yang dapat direpresentasikan makin lebar
- Makin besar basis bilangan B ⇒ cakupan bilangan makin lebar
  - Contoh:
    - ❖ Dengan 6 digit eksponen ⇒ cakupan *signed* eksponen $= -32$ hingga $+31$
    - ❖ Jika $B = 2$ ⇒ cakupan bilangannya 2^{-32} hingga 2^{+31}
    - ❖ Jika $B = 16$ ⇒ cakupan bilangannya 16^{-32} hingga 16^{+31} atau 2^{-128} hingga 2^{+124} !!!

Berapa jumlah bit untuk Mantissa dan Eksponen ? (2)

- Makin besar basis bilangan B $\Rightarrow$ akurasi berkurang
  - Contoh bentuk normal dari $7{,}5_{10}$ dengan 4 bit mantissa:
    - ❖ $7{,}5_{10} = 111{,}1_2$
    - ❖ Dengan basis eksponen 2:
      - ➢ $111{,}1_2 = 0{,}1111 \times 2^3 = 7{,}5_{10}$
    - ❖ Dengan basis eksponen 16:
      - ➢ $111{,}1_2 = 0{,}0111 \times 16^1 = 7{,}0_{10}$ (jauh berbeda !!)
- Pilihan penggunaan basis eksponen ditentukan oleh aspek mana yang lebih dipentingkan (lebar cakupan atau akurasi)

Berapa jumlah bit untuk Mantissa dan Eksponen ? (3)

- Biasanya setiap sistem komputer mempunyai lebih dari satu format *floating point*
- Contoh jumlah bit untuk PDP-11 dan VAX-11:

Format	Mantissa	Eksponen (bit)	Panjang Total (bit)
F	24	8	32
D	56	8	64
G*	53	11	64
H*	113	15	128

* VAX only

- Format F:
  - 24 bit mantissa ≈ akurasi 7 angka desimal (2^{23} = 8388608)
  - 8 bit eksponen ≈ eksponen desimal $10^{\pm 38}$

Berapa jumlah bit untuk Mantissa dan Eksponen ? (4)

- Format D: (*double precision*)
  - 56 bit mantissa ≈ akurasi 16 angka desimal
- Format H:
  - 113 bit mantissa ≈ akurasi 34 angka desimal
  - Cakupan bilangan: $-10^{480} \leq R \leq +10^{480}$
- *Floating point* **overflow**: ⇒ fatal error !!
  - Terjadi jika bilangan yang akan disimpan lebih besar dari eksponen positif
  - Misal bilangan ½ x 2^{200} akan disimpan pada tipe bilangan F
- *Floating point* **underflow**: di-reset ke nol !
  - Terjadi jika bilangan yang akan disimpan lebih kecil dari eksponen negatif
  - Misal bilangan ½ x 2^{-200} akan disimpan pada tipe bilangan F

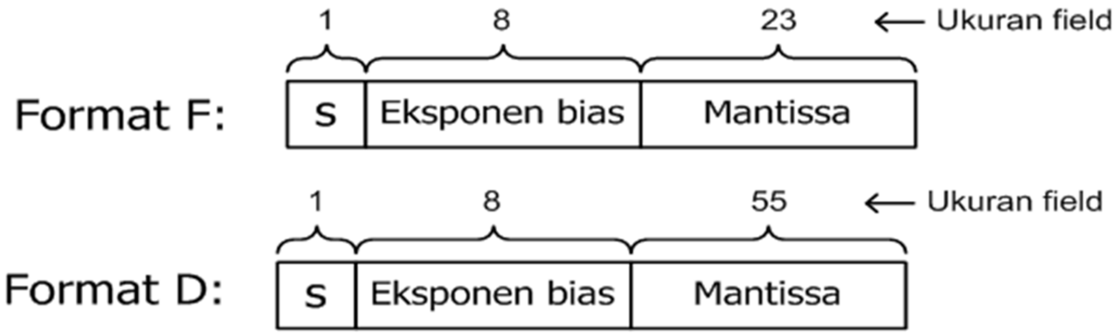
Berapa jumlah bit untuk Mantissa dan Eksponen ? (5)

- Karakteristik *floating point* beberapa komputer

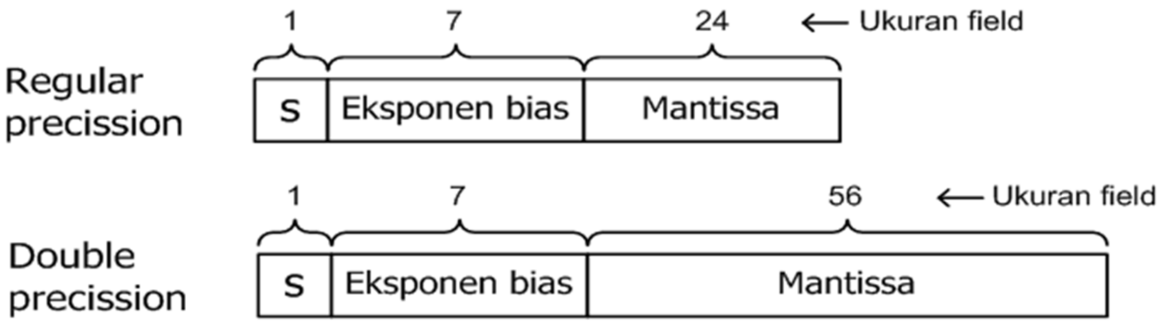
Sistem	Total Bit	Bit Mantissa	Representasi Mantissa	Bit Eksponen	Representasi Eksponen	Basis Eksponen	Norma-lisasi
Burroughs B5500	48	41	Sign / magnitude	7	Sign / magnitude	B = 8	Ya
CDC Cyber-70	60	49	Komplemen 1	11	Excess-1024	B = 2	Ya
Sigma 7	32	25	Komplemen 2	7	Excess-64	B = 16	Ya
Univac 1100/80	36	28	Komplemen 1	8	Excess-128	B = 2	Ya
Cray 1	64	49	Sign / magnitude	15	Excess-2^{14}	B = 2	Ya
Hewlett-Packard 3000	32	24	Komplemen 2	8	Komplemen 2	B = 2	Ya
DEC System 10	36	28	Sign / magnitude	8	Excess-128	B = 2	Ya

Bagaimana Bit-bit *Floating Point* disusun ?

- Komponen bilangan *floating point* :
  - *sign* (sebuah bit, terletak paling kiri / MSB)
  - mantissa (terletak sesudah eksponen)
  - eksponen (terletak antara bit sign dan bit-bit mantissa)
- Distribusi bit *floating point* pada PDP-11 dan VAX-11:



- Distribusi bit *floating point* pada IBM 370:



Optimasi Bit-bit *Floating Point* (1)

- Optimasi: lebih efisien, cepat, akurat
- Fakta untuk B = 2:
  - Setelah normalisasi nilai mantissa selalu $\geq \frac{1}{2}$
  - Bit paling kiri (MSB) mantissa selalu 1
- Maka:
  - Bit MSB mantissa tidak perlu disimpan
  - Lebih akurat 1 digit
  - Mantissa 100...0 ditulis menjadi ~~1~~00...0

Optimasi Bit-bit *Floating Point* (2)

Contoh:

(a) Bagaimana nilai –3/16 desimal disimpan ke dalam format F PDP-11 ?

(1) Normalisasi: $-3/16 = -3/4 \times 2^{-2}$

(2) Nilai eksponen bias: $e' = e + 2^{m-1}$

$e' = -2 + 2^{8-1} = -2 + 128 = 126_{10}$

$e' = 01111110_2$

(3) Nilai mantissa dalam notasi *sign/magnitude*:

$-3/4 = 1 \mid 11000...0$

(4) Buang bit MSB mantissa: $-3/4 = 1 \mid 1000...0$

(5) Hasil lengkap:

$= 1\,0\underline{111}111\underline{0}\,\underline{1}00\underline{000}000\underline{000}000\underline{000}000$

$= 27720000000_8$

Optimasi Bit-bit *Floating Point* (3)

(b) Bagaimana nilai +200,0 desimal disimpan ke dalam format F PDP-11 ?

(1) Normalisasi: $+200 = +200/256 \times 2^8$

$= 25/32 \times 2^8$

(2) Nilai eksponen bias: $e' = e + 2^{m-1}$

$e' = 8 + 2^{8-1} = 8 + 128 = 136_{10}$

$e' = 10001000_2$

(3) Nilai mantissa dalam notasi *sign/magnitude*:

$+25/32 = 0 \mid 1100100...0$

(4) Buang bit MSB mantissa: $+25/32 = 0 \mid 100100...0$

(5) Hasil lengkap:

$= 0\underline{1000}1000\underline{100}1001\underline{000}000\underline{000}000\underline{000}000$

$= 10422000000_8$

Optimasi Bit-bit *Floating Point* (4)

(c) Jika representasi internal pada PDP-11 dengan format F adalah 27734000000_8, angka desimal berapakah yang sedang disimpan ?

(1) Ubah ke biner sebanyak 32 bit:

27734000000_8 =

$10111111011100000000000000000000_2$

(2) Tentukan bilangan positif atau negatif:

MSB = 1 ⇒ bilangan negatif

(3) Cari nilai eksponen:

Eksponen bias (e') = $01111110_2 = 126_{10}$

Eksponen sebenarnya = $126 - 2^{8-1} = 126–128 = –2$

(4) Tambahkan bit MSB mantissa:

$101111110\mathbf{1}11100000000000000000000_2$

(5) Cari nilai mantissa:

Mantissa = 11110000...0

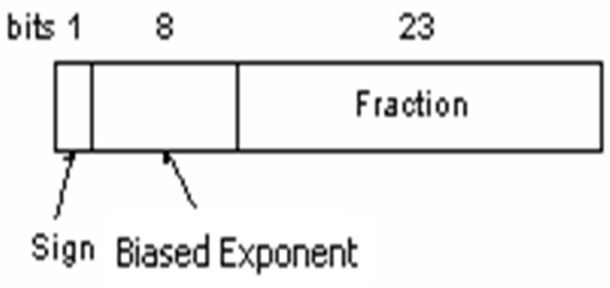
= ½ + ¼ + 1/8 + 1/16 = 15/16

(6) Nilai desimalnya = $–15/16 \times 2^{–2}$

= –15/64 = –0,234375

Floating Point Standard IEEE 754

- Terdapat tiga jenis format:
  - *single precision* (32 bit) } fraksi (mantissa): radix 2;
  - *double precision* (64 bit) } eksponen: *excess-n*
  - *extended precision* (80 bit) → mereduksi *error* akibat pembulatan
  - Contoh: Intel, Motorola, SPARC, MIPS



Single precision format

Double precision format

Normalisasi Standard IEEE 754 (1)

- Bagaimana cara merepresentasikan *floating point* secara normal untuk standar IEEE 754 ?
  - Menempatkan koma tepat di sebelah kanan angka penting pertama
  - Sebelum koma = 1, sesudah koma = boleh nol boleh 1
  - Nilai mantissa M: (dalam desimal)

$$1 \leq |M| < 2$$

  - Bilangan 1 sebelum koma tidak disimpan ke dalam mantissa agar akurasinya lebih baik
  - B = basis mantisa (2 atau 10)

Normalisasi Standard IEEE 754 (2)

- Akurasi: jika mantissa = 5 digit, manakah yang paling akurat ?

  $1.228{,}8 = 0{,}12288 * 10^4$???

  $= 0{,}01228 * 10^5$???

  $= 0{,}00122 * 10^6$???

- Contoh: Normalisasikan !

  (a) $103{,}5_{10}$ dengan B = 10 ?

  $103{,}5_{10} = 1{,}035 \times 10^2$

  (b) $0{,}000011101_2$ dengan B = 2 ?

  $0{,}000011101_2 = 1{,}1101_2 \times 2^{-5}$

  (c) $10011{,}110_2 \times 2^{10}$ dengan B = 2 ?

  $10011{,}110_2 \times 2^{10} = 1{,}0011110_2 \times 2^{14}$

Normalisasi Standard IEEE 754 (3)

(d) $0{,}000011_{10} \times 10^2$?

$0{,}000011_{10} \times 10^2 = 1{,}1_{10} \times 10^{-3}$

(e) $0{,}0001_2 \times 8^5$?

(f) $0{,}000011_2 \times 16^5$ dengan B = 16 ?

Catatan:

Basis bilangan mantissa (B) yang biasa digunakan adalah 2 dan 10

IEEE Standard 754 *Single Precision* (1)

Format:

```
S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF
0 1      8 9                     31
```

S = Sign, E = Eksponen bias, F = Fraction

Jika 0<E<255, maka $V = (-1)^S \times 2^{E-127} \times (1.F)$

dimana:

V = nilai bilangan (dalam desimal)

1.F = nilai Fraction (pecahan) yang sebenarnya (hasil normalisasi)

Contoh:

```
0 10000000 00000000000000000000000 = +1 * 2**(128-127) * 1.0 = 2
0 10000001 10100000000000000000000 = +1 * 2**(129-127) * 1.101 = 6.5
1 10000001 10100000000000000000000 = -1 * 2**(129-127) * 1.101 = -6.5
0 00000001 00000000000000000000000 = +1 * 2**(1-127) * 1.0 = 2**(-126)
```

IEEE Standard 754 *Single Precision* (2)

- Jika E = 0, F = 0, dan S = 1, maka V = –0
- Jika E = 0, F = 0, dan S = 0, maka V = +0

  Contoh:

```
0 00000000 00000000000000000000000 = 0
1 00000000 00000000000000000000000 = -0
```

- Jika E = 0 dan F $\neq$ 0 (nonzero), maka

$$V = (-1)^{S} \times 2^{-126} \times (0.F)$$

dimana "0.F" merupakan nilai Fraction (pecahan) yang tidak dinormalisasikan

Contoh:

```
0 00000000 10000000000000000000000 = +1 * 2**(-126) * 0.1 = 2**(-127)
0 00000000 00000000000000000000001 = +1 * 2**(-126) *
                                     0.00000000000000000000001 =
                                     2**(-149)  (Smallest positive value)
```

IEEE Standard 754 *Single Precision* (3)

- Jika E = 255, F = 0, dan S = 1, maka V = –∞
- Jika E = 255, F = 0, dan S = 0, maka V = +∞

  Contoh:

```
0 11111111 00000000000000000000000 = Infinity
1 11111111 00000000000000000000000 = -Infinity
```

- Jika E = 255, F ≠ 0 (nonzero) maka V = NaN ("Not a number")
  - ❖ NaN digunakan untuk keperluan diagnostik, misal untuk mengetahui darimana NaN dikirimkan
  - ❖ Tanda (*sign*) NaN tidak ada artinya

Contoh:

```
0 11111111 00000100000000000000000 = NaN
1 11111111 00100010001001010101010 = NaN
```

IEEE Standard 754 *Single Precision* (4)

- Contoh 1:
  - Representasikan biner dari –0.75 ke dalam format *IEEE single precision* !

    Jawab:

    ❖Representasi biner: - 0.75 = (- 1/2) + (- 1/4) = - 0.11 x 2^0

    ❖Normalisasi: -1.1 x 2^{-1}

$$V = (-1)^S \times 2^{E-127} \times (1.F)$$

❖Karena bilangannya negatif, maka $(-1)^S = -1$, sehingga sign bit S = 1

❖Bit eksponen E–127 = –1, maka eksponen bias E = -1+127 = 126 = 01111110

❖Bit mantissa 1.F = 1.1, maka mantissa F = 100...0 (23 bit)

❖Jadi –0,75 = 1 01111110 10000000000000000000000

—8 bit— ———— 23 bit ————--

IEEE Standard 754 *Single Precision* (5)

- Contoh 2:
  - Representasikan biner dari -118,625 dalam format *IEEE single precision* !
  - Jawab :
    - ❖Representasi biner: –118.625 = –1110110.101 x 2^0
    - ❖Normalisasi: –1.110110101 x 2^6
    - ❖Karena bilangannya negatif, maka $(–1)^S$ = –1, sehingga sign bit S = 1
    - ❖Bit eksponen E–127 = 6, maka E = 6 + 127 = 133 = 10000101
    - ❖Bit mantissa 1.F = 1.110110101, maka mantissa F = 1101101010...0 (23 bit)
    - ❖Jadi –118.625 = 1 10000101 11011010100000000000000

      —8 bit— ———— 23 bit ————--

IEEE Standard 754 *Double Precision* (1)

- Standar IEEE untuk representasi *double precision floating point* membutuhkan 64 bit word, 0 to 63, dari kiri ke kanan
- Bit pertama *sign bit* S, 11 bit selanjutnya bit-bit eksponen 'E', dan sisanya 52 bit merupakan fraction 'F':

```
S EEEEEEEEEEE FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
0 1        11 12                                                63
```

IEEE Standard 754 *Double Precision* (2)

- Jika 0<E<2047, maka $V = (-1)^S \times 2^{E-1023} \times (1.F)$

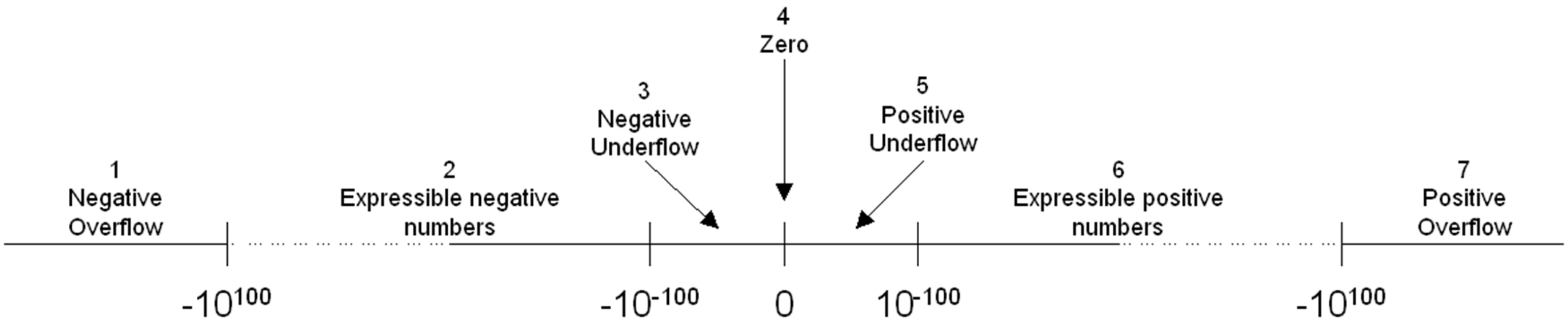
  dimana "1.F" merupakan nilai Fraction (pecahan) yang sebenarnya (normalisasi)
- Jika E = 0, F = 0, dan S = 1, maka V = –0
- Jika E = 0, F = 0, dan S = 0, maka V = +0
- Jika E = 0 dan F ≠ 0 (nonzero), maka

  $$V = (-1)^S \times 2^{-1022} \times (0.F)$$

  dimana "0.F" merupakan nilai Fraction (pecahan) yang tidak dinormalisasikan
- Jika E = 2047, F = 0, dan S = 1, maka V = –∞
- Jika E = 2047, F = 0, dan S = 0, maka V = +∞
- Jika E = 2047, F ≠ 0 (nonzero) maka V = NaN ("Not a number")

Kesalahan-Kesalahan *Floating Point* (1)

(1) *Overflow* dan *underflow*



Jika fraksi = 3 digit desimal bertanda dan eksponen = 2 digit desimal bertanda, maka garis bilangan real terbagi dalam **tujuh** bagian, yakni:

✗ 1. Bilangan negatif yang lebih kecil dari -0.999×10^{99}

✓ 2. Bilangan negatif antara -0.999×10^{99} dan -0.100×10^{-99}

✗ 3. Bilangan negatif dengan nilai <u>magnitude</u> sangat kecil kurang dari 0.100×10^{-99}

✓ 4. Nol

✗ 5. Bilangan positif dengan nilai magnitude sangat kecil kurang dari 0.100×10^{-99}

✓ 6. Bilangan positif antara 0.100×10^{-99} dan 0.999×10^{99}

✗ 7. Bilangan positif yang lebih besar dari 0.999×10^{99}

Kesalahan-Kesalahan *Floating Point* (2)

(2) Kerapatan *(density)*

- Bilangan bilangan real tidak pasti (berubah-ubah)
- Antara dua bilangan real, x dan y, selalu ada bilangan real lain, $z = (x + y)/2$ → *continuum* (rangkaian kesatuan)

(3) Kesalahan pembulatan *(round-off error)*

- Tidak dapat merepresentasikan setiap bilangan *floating point* secara tepat
- Contoh:
  - ❖ $1/5 = 0{,}2_{10} = 0{,}0011001100110011\ldots_2$ (tak terbatas)
  - ❖ Dibatasi dengan jumlah bit mantissa tertentu
  - ❖ Misal 12 bit *sign/magnitude* dan 6 bit eksponen:

$$0{,}2_{10} = 0{,}11001100110_2 \times 2^{-2} = 0{,}798828125 \times ¼$$
$$= 0{,}19971 \text{ (kurang dari 0,2)}$$

"Jangan pernah menguji balik nilai suatu bilangan real !"

Kesalahan-Kesalahan *Floating Point* (3)

(4) Kesalahan propagasi (*propagation error*)

- Terjadi pada operasi aritmatika jika dua bilangan yang dioperasikan memiliki eksponen yang jauh berbeda
- Langkah-langkah untuk menjumlahkan dua bilangan real:
  1. Samakan eksponen kedua bilangan (penskalaan)
  2. Jumlahkan mantissa kedua bilangan
  3. Normalisasikan

$$
\begin{array}{lcrl}
0{,}123 * 10^5 & \textit{Scaling} \rightarrow & 0{,}01230 * 10^6 & \\
0{,}00456 * 10^6 & + & 0{,}00456 * 10^6 & \\
\hline
 & & 0{,}01686 * 10^6 & \leftarrow \text{tdk normal} \\
 & & 0{,}1686 * 10^5 & \leftarrow \text{normal}
\end{array}
$$

$\rightarrow$ masalah belum terlihat

Kesalahan-Kesalahan *Floating Point* (4)

Contoh 2: Penjumlahan +11 dengan +1/4

Jika digunakan 6 bit *sign/magnitude* untuk mantissa (lima bit ditambah satu bit tanda), dan 4 bit *biased exponent*, maka:

	sign	exp	mantissa
$+11 = +11/16 \times 2^4 =$	0	1100	10110
$+1/4 = +1/2 \times 2^{-1} =$	0	0111	10000

Samakan eksponennya:

$+11/16 \times 2^4 = +11/16 \times 2^4 =$ 0 1100 10110

$+1/2 \times 2^{-1} = +1/64 \times 2^4 = +$ <u>0 1100 00000</u> (0,00000←,10000)

($\times 2^5$)↑ 0 1100 10110 → $11/16 \times 2^4 =$ +11 !!

(Propagation error)

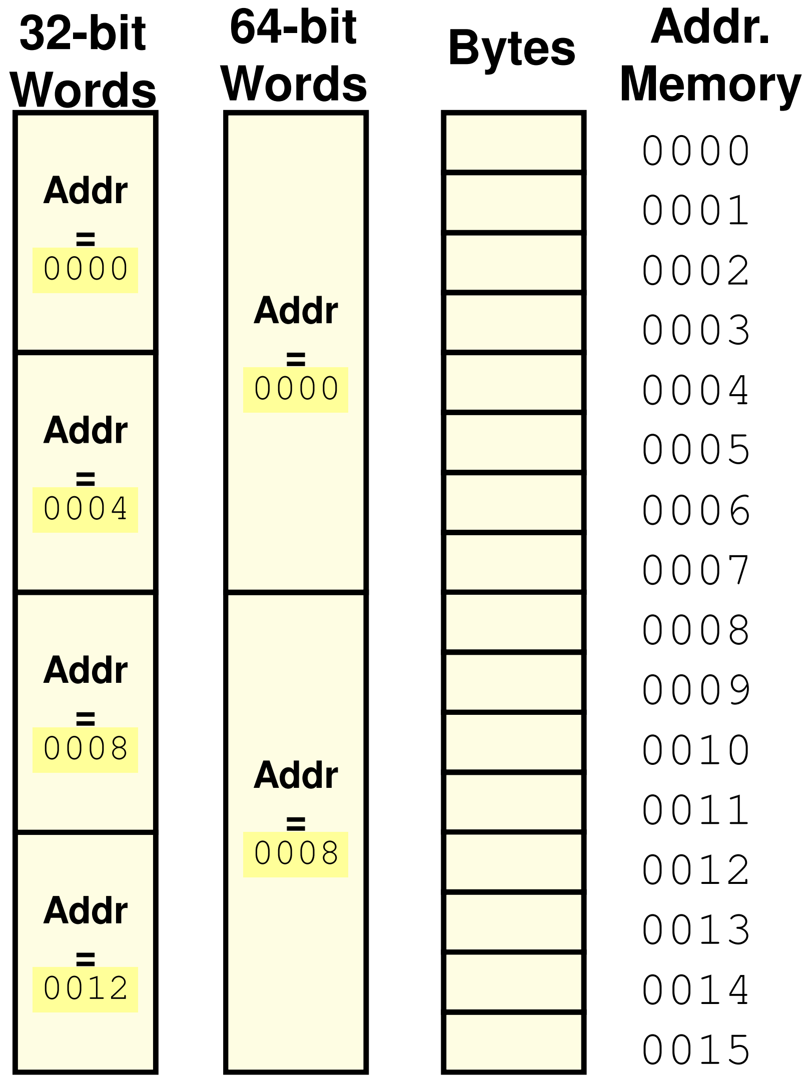
<u>Solusi</u>:

Memperbanyak bit pada mantissa dan eksponen

→ mengurangi dampak masalah

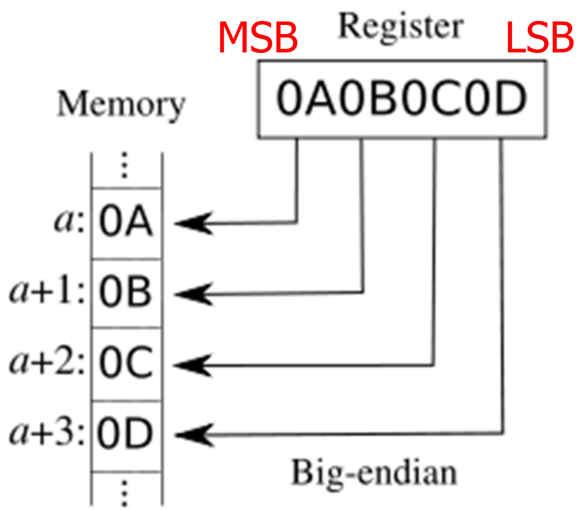
Organisasi Memori Berorientasi Word

- Alamat spesifik dari lokasi Byte
  - Alamat word pertama = alamat awal memori (RAM)
  - Alamat word selanjutnya melompat 4 alamat (32-bit) atau 8 alamat (64-bit)

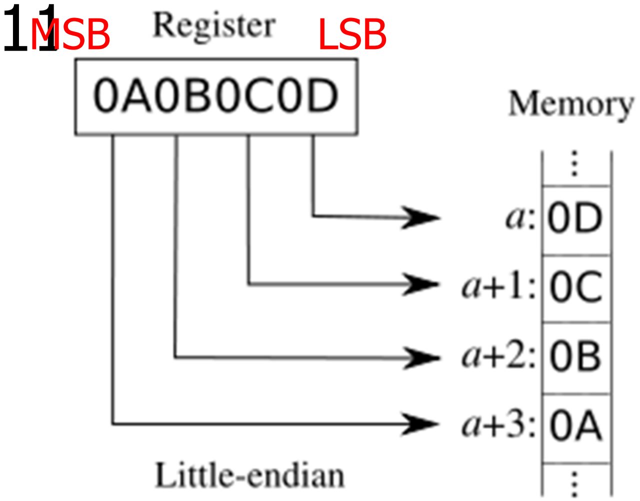


Big Endian dan Little Endian

- Big Endian
  - *Least significant byte* memilik
  - Contoh: Sun, Macintosh, Mot IBM system/360

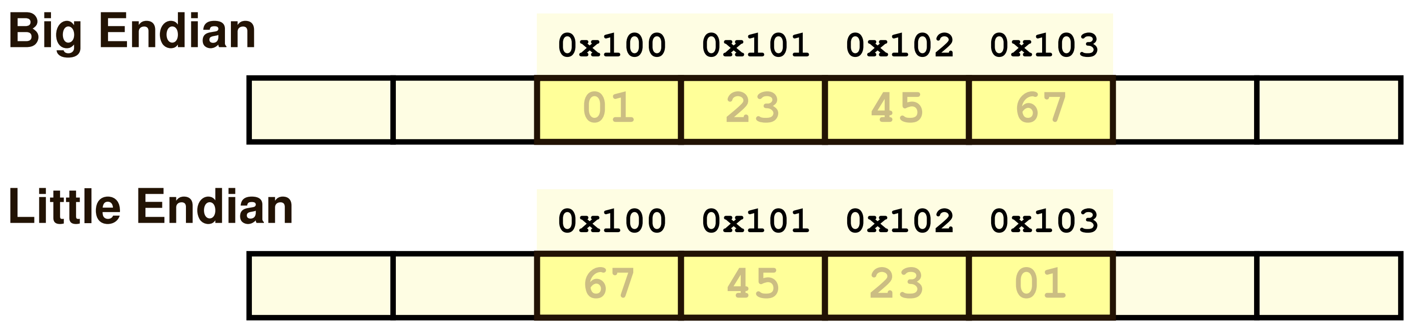


- Little Endian
  - *Least significant byte* memilik
  - Alphas, 6502, Z80, x86, PDP-11



Representasi Variabel

- Contoh
  - Sebuah variable `x` memiliki data sebanyak 4-byte: `0x01234567`
  - Alamat yang diberikan `&x` adalah `0x100`



Representasi Integer

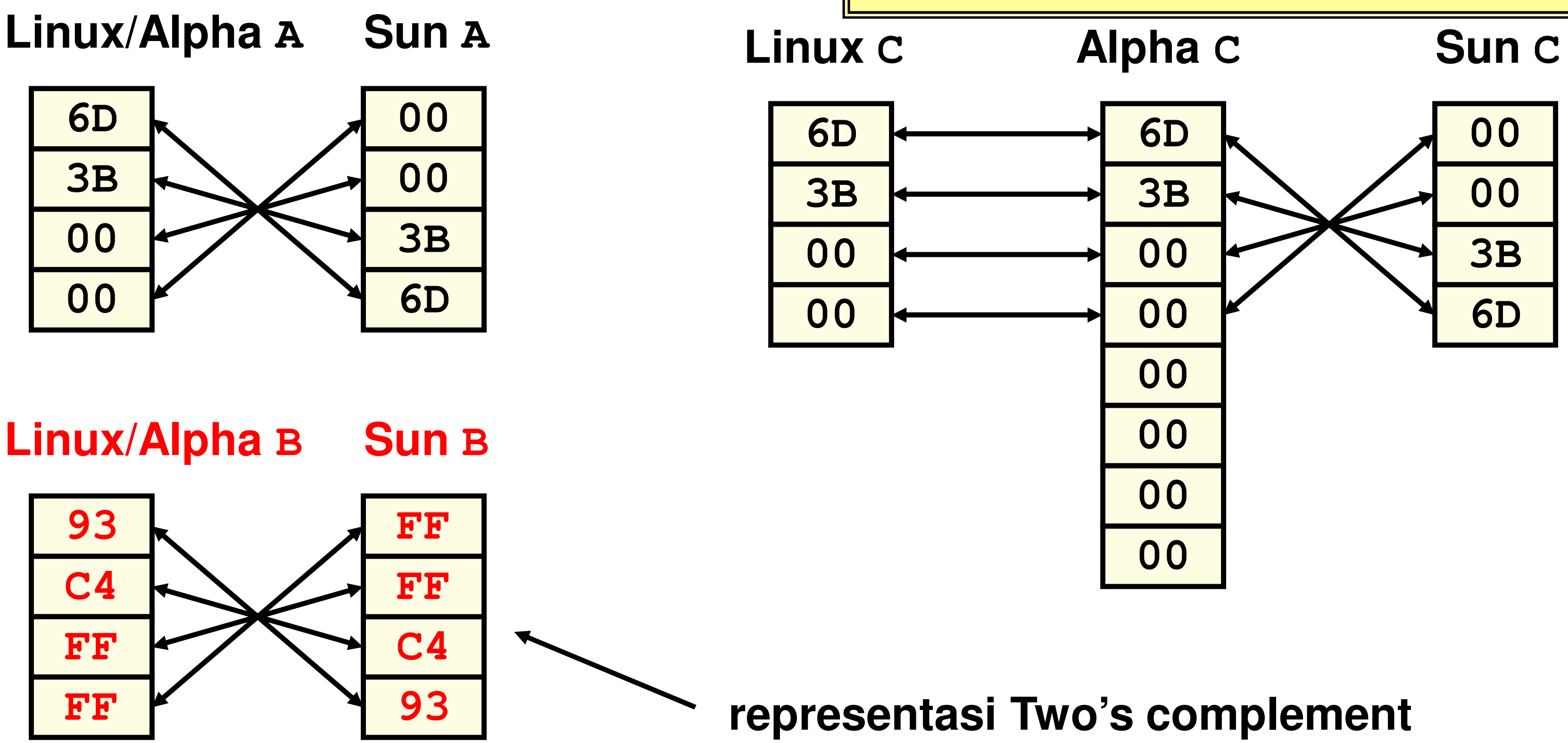
- int A = 15213;
- int B = -15213;
- long int C = 15213;

3 komputer: Linux, Alpha, dan Sun

Linux + Alpha = Little endian; Sun = Big en

Decimal: `15213`
Binary: `0011 1011 0110 1101`
Hex: `3 B 6 D`

Decimal: -15213
Binary: 1100 0100 1001 0011
Hex: C 4 9 3



Representasi Pointer

Contoh: int B = -15213; int *P = &B;

Alpha Address

Hex:	1	F	F	F	F	F	C	A	0
Binary:	0001	1111	1111	1111	1111	1111	1100	1010	0000

Alpha P

A0
FC
FF
FF
01
00
00
00

Sun P

EF
FF
FB
2C

Sun Address

Hex:	E	F	F	F	F	B	2	C
Binary:	1110	1111	1111	1111	1111	1011	0010	1100

Linux P

D4
F8
FF
BF

Linux Address

Hex:	B	F	F	F	F	8	D	4
Binary:	1011	1111	1111	1111	1111	1000	1101	0100

Compilers & mesin yg beda akan merepresentasikan pada lokasi yg beda

Tipe Data Karakter

Tipe yang dominan :

- **ASCII** (*American Standard Code for Information Interchange*)
  standar → proses transfer informasi antar komputer
- **EBCDIC** (*Extended Binary-Coded Decimal Interchange Code*)
  EBCDIC secara internal, ASCII secara eksternal

- ASCII (7 bit/code) vs standar format numerik (kelipatan 8 bit)
  - ASCII diimplementasikan dalam 8 bit
  - Bit ke-8 dapat berupa:
    - selalu bernilai 0, atau
    - flag untuk mendefinisikan *character set expansion, atau*
    - *error detection* (parity genap/ganjil)

Kode ASCII

```
Oct  Dec  Hex   Name
000    0  0x00  NUL
001    1  0x01  SOH, Control-A
002    2  0x02  STX, Control-B
003    3  0x03  ETX, Control-C
004    4  0x04  EOT, Control-D
005    5  0x05  ENQ, Control-E
006    6  0x06  ACK, Control-F
007    7  0x07  BEL, Control-G
010    8  0x08  BS,  Control-H, backspace
. . . . . . . . . . . . . . . . . .
```

Table Karakter EBCDIC

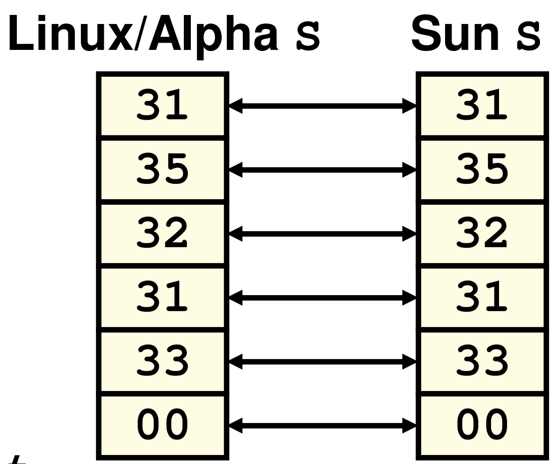
Least significant nibble ->

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	... controls ...															
1																
2																
3									... controls ...							
4			â	ä	à	á	ã	å	ç	ñ	¢	.	<	(	+	|
5	&	é	ê	ë	è	í	î	ï	ì	ß	!	$	*	)	;	^
6	-	/	Â	Ä	À	Á	Ã	Å	Ç	Ñ	¦	,	%	_	>	?
7	ø	É	Ê	Ë	È	Í	Î	Ï	Ì	`	:	#	@	'	=	"
8	Ø	a	b	c	d	e	f	g	h	i	«	»	ð	ý	þ	±
9	°	j	k	l	m	n	o	p	q	r	ª	º	æ	¸	Æ	¤
A	µ	~	s	t	u	v	w	x	y	z	¡	¿	Ð	[	Þ	®
B	¬	£	¥	·	©	§	¶	¼	½	¾	Ý	¨	¯	]	´	×
C	{	A	B	C	D	E	F	G	H	I	-	ô	ö	ò	ó	õ
D	}	J	K	L	M	N	O	P	Q	R	¹	û	ü	ù	ú	ÿ
E	\	÷	S	T	U	V	W	X	Y	Z	²	Ô	Ö	Ò	Ó	Õ
F	0	1	2	3	4	5	6	7	8	9	³	Û	Ü	Ù	Ú	

Representasi String

- String di C:
  - Direpresentasikan dengan *array of characters*
  - Setiap karakter di-encoded ke dalam format ASCII
    - ❖ Standard Encoding: 7-bit
    - ❖ Encoding lain ada, tapi tidak biasa
  - String harus diakhiri dengan null
    - ❖ Karakter akhir = 0
    - ❖ Karakter "0" memiliki kode `0x30`
      - ➢ Digit *i* memiliki kode `0x30`+*i*
- Kompatibiliti
  - Urutan Byte bukan issue
    - ❖ Data merupakan *single byte quantities*
  - Text files secara umum *platform independent*
    - ❖ Kecuali untuk konvensi yang beda

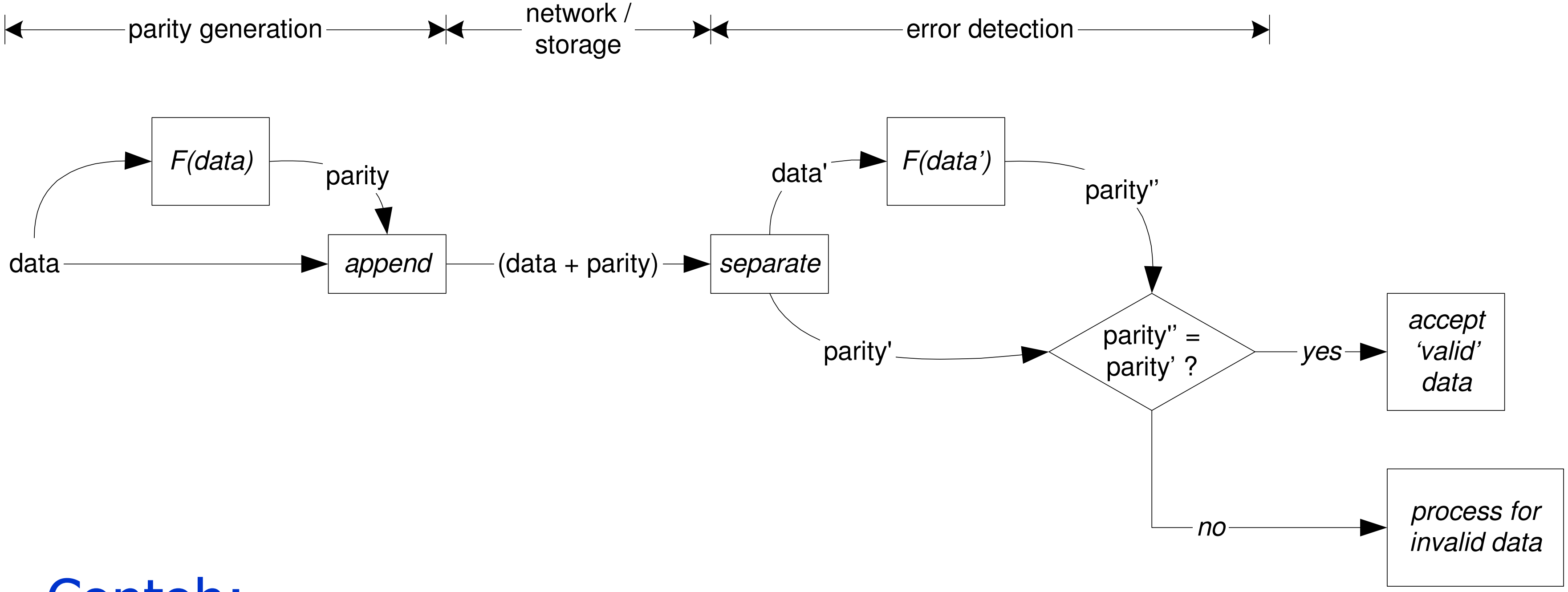
- `char S[6] = "15213";`



Penanganan Kesalahan Dengan Paritas (1)

- Mendeteksi kesalahan data pada level bit
- Bit paritas:
  - bit ekstra yang ditambahkan pada suatu unit data terkecil
  - digunakan dalam proses pengecekan kebenaran data ketika data akan disimpan atau dikirim
  - dihasilkan oleh fungsi generator paritas
- Jenis paritas:
  - Paritas genap *(even)*:
    - ❖ Menambahkan sebuah bit sehingga total bit '1' suatu word berjumlah genap
  - Paritas ganjil *(odd)*:
    - ❖ Menambahkan sebuah bit sehingga total bit '1' suatu word berjumlah ganjil
- Dapat mendeteksi kesalahan bit berjumlah ganjil, lokasi bit tidak bisa diketahui

Penanganan Kesalahan Dengan Paritas (2)



Contoh:

Data = **1001001,** jenis paritas = paritas genap

Jika bit yang dibaca **10010011** → data dianggap valid
Jika bit yang dibaca **10110011** → data dianggap tidak valid
Jika bit yang dibaca **10110010** → data dianggap valid !!

Penanganan Kesalahan Dengan
Hamming Code Single Bit

- Biasa digunakan dalam konteks *Error Control* (deteksi maupun koreksi)
- *Codeword* = bit-bit data + bit paritas/kontrol
- *Hamming distance*: jumlah perbedaan bit dari dua buah *codeword*
  - *Hamming distance* n + 1 → Dapat mendeteksi n *error*
  - *Hamming distance* 2n + 1 → Dapat me-*recover* n *error*

- Contoh *codeword* dengan 7 bit informasi dan 1 bit paritas genap:

0000000	0
0000001	1
0000010	1
0000011	0

Antar *codeword* terdapat 2 bit berbeda

Hamming distance-nya = 2 → hanya dapat mendeteksi 1 bit *error*

Penanganan Kesalahan Dengan *Hamming Code Multi Bit* (1)

- Blok data sebanyak k digit dikodekan menjadi n digit (n>k)
  - Ditulis dengan notasi (n,k)
    - Misal *codeword* terdiri dari 7 bit data + 4 bit *check* → (11,7)
  - Jumlah bit *codeword* harus memenuhi persamaan:

    $$2^K - 1 \geq M+K$$

    K = jumlah bit kontrol; M = jumlah bit data
  - Posisi bit-bit K ditentukan dengan rumus 2^x; x = 0, 1, 2, 3, ...
  - Posisi bit dimulai dari posisi ke-1, bukan ke-0
  - k/n = *code rate* atau *code efficiency*
  - 1 - k/n = *redundancy*
- Dapat mendeteksi kesalahan sebanyak 2 bit
- Dapat menentukan posisi bit yang *error* jika terjadi kesalahan sebanyak 1 bit
- *Recovery* dilakukan dengan meng-*inverse* bit pada posisi yang salah

Penanganan Kesalahan Dengan *Hamming Code Multi Bit* (2)

- Contoh bit-bit data: 1001101 (7 bit)
  Jumlah bit *codeword*: $2^K - 1 \geq M+K$
  Jika K = 3 → $2^3 - 1 \geq 7 + 3$ (salah)
  Jika K = 4 → $2^4 - 1 \geq 7 + 4$ (ok)

Posisi bit:	11	10	9	8	7	6	5	4	3	2	1
Bit-bit data:	1	0	0	K	1	1	0	K	1	K	K

- Cara menentukan bit-bit K:
  - Lakukan penjumlahan modulo 2 (biner) **semua posisi bit yang bernilai 1**

```
11 = 1011
 7 = 0111
 6 = 0110
 3 = 0011
----------- +
     1001
posisi ke: 8421
```

- Hasil:

Posisi bit:	11	10	9	8	7	6	5	4	3	2	1
Bit-bit data:	1	0	0	1	1	1	0	0	1	0	1

Penanganan Kesalahan Dengan *Hamming Code Multi Bit* (3)

- Pendeteksian kesalahan *(decoder):*
  - Jumlahkan (modulo 2) semua posisi bit yang bernilai 1 (termasuk bit *check*)
    - ❖ Jika hasilnya 0 → tidak terjadi kesalahan
    - ❖ Jika hasilnya ≠ 0 → hasil penjumlahan merupakan posisi bit yang salah

```
11 = 1011
 8 = 1000
 7 = 0111
 6 = 0110
 3 = 0011
 1 = 0001
---------- +
      0000
```

tidak terjadi *error* → 0000

- Contoh jika terjadi *error* pada bit ke-11:

```
8 = 1000
7 = 0111
6 = 0110
3 = 0011
1 = 0001
--------- +
    1011
```

1011 ← terjadi kesalahan pada bit ke-1011 (ke-11)

- *Recovery*: *invert* bit ke-11

Penanganan Kesalahan Dengan *Hamming Code Multi Bit* (4)

- Contoh jika terjadi *error* pada bit ke-11 dan bit ke-1:

```
 8 = 1000
 7 = 0111
 6 = 0110
 3 = 0011
----------  +
     1010
```

← kesalahan terdeteksi, posisi bit yang salah tidak diketahui

- Contoh jika terjadi *error* pada bit ke-11, bit ke-1, dan bit ke-10:

```
10 = 1010
 8 = 1000
 7 = 0111
 6 = 0110
 3 = 0011
----------  +
     0000
```

← kesalahan tidak terdeteksi !

Penerapan Pendeteksi Kesalahan Bit

- Digunakan pada aplikasi yang punya batasan *single-bit error*, misalnya: *error correcting semiconductor memory system*
- Jarang digunakan pada komunikasi data atau jaringan komputer, dimana probabilitas terbesar error yang terjadi adalah *burst error*

Pustaka

[HTT02] http://en.wikipedia.org/wiki/

[SCH85] Schneider, Michael G. 1985. "*The Principle of Computer Organization*". 1st edition. John Wiley & Sons. Canada.

[TAN99] Tanenbaum, Andrew S. 1999. "*Structured Computer Organization*". 4th edition. Prentice Hall.