PERTEMUAN 10

Aritmatika Komputer

1. ARITHMETIC AND LOGIC UNIT (ALU)

Pengertian

Arithmatic and Logic Unit (ALU) adalah salah satu bagian/komponen dalam sistem didalam sistem komputer yang berfungsi melakukan operasi/perhitungan aritmatika dan logika (seperti penjumlahan, pengurangan dan beberapa logika lain). ALU bekerja sama dengan memori, dimana hasil dari perhitungan di dalam ALU di simpan ke dalam memori. Perhitungan dalam ALU menggunakan kode biner, yang merepresentasikan instruksi yang akan dieksekusi (opcode) dan data yang diolah (operand). ALU biasanya menggunakan sistem bilangan biner (two's complement). ALU mendapat data dari register. Kemudian data tersebut diproses dan hasilnya akan disimpan dalam register tersendiri yaitu ALU.

Sejarah ALU

Aritmetika yang terbatas pada jumlah yang sangat kecil artifak kecil yang menunjukkan konsep yang jelas penambahan (+) dan pengurangan (-), yang paling terkenal menjadi tulang Ishango dari Afrika tengah, datang dari suatu tempat antara 20.000 dan 18.000 SM.

Jelas bahwa Babel memiliki pengetahuan yang kokoh dari hampir semua aspek aritmetika dasar oleh 1800 SM, sejarawan meskipun hanya bisa menebak metode yang digunakan untuk menghasilkan hasil aritmetika, seperti yang ditunjukkan. Misalnya, dalam tablet tanah liat Plimpton 322, yang muncul menjadi daftar Pythagoras tiga kali lipat, tetapi tanpa kerja untuk menunjukan bagaimana daftar ini awalnya diproduksi. Demikian pula, Mesir Rhin Mathematical Papyrus (berasal dari sekitar 1650 SM, meskipun jelas salinan teks yang lebih tua dari sekitar 1850 SM) menunjukan bukti penambahan (+), pengurangan (-), perkalian (x), dan pembagian (/) yang digunakan dalam sebagian unit sistem.

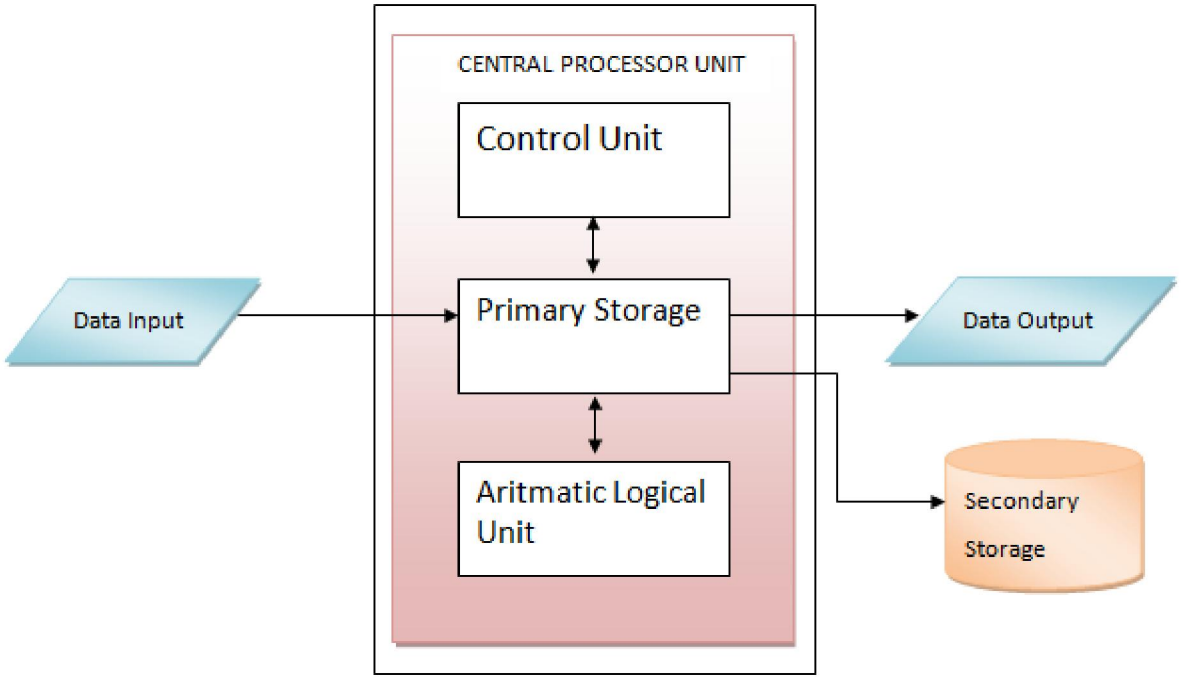
Nicomachus merangkum filsafat Pythagoras pendekatan angka, dan hubungan mereka satu sama lain, dalam Pengenalan aritmatika. Pada saat ini, operasi aritmatika dasar adalah

urusan yang sangat rumit, itu adalah metode yang dikenal sebagai “Metode orang-orang Indian” (Latin Modus Indorum) yang menjadi aritmatika yang kita kenal sekarang. Aritmatika India jauh lebih sederhana daripada aritmatika Yunani karena kesederhanaan system angka India, yang memiliki nol dan notasi nilai tempat. Abad ke - 7 Syria Severus Sebokht uskup disebutkan metode ini dengan kekaguman, namun menyatakan bahwa Metode dari India ini tak tertuliskan. Orang-orang Arab belajar metode baru ini dan menyebutkan Fibonacci (juga dikenal dengan Leonardo dari Paris) memperkenalkan “Metode dari Indian” ke Eropa pada 1202. Dalam bukunya Liber Abaci, Fibonacci mengatakan bahwa dibandingkan dengan metode baru ini, semua metode lain telah kesalahan. Dalam Abad Pertengahan. Aritmatika adalah satu dari tujuh seni liberal diajarkan di universitas.

Operasi pada ALU

Operasi aritmatika adalah operasi penjumlahan dan pengurangan, sedangkan contoh operasi logika adalah logika AND dan OR. ALU melakukan operasi aritmatika yang lainnya seperti pengurangan, dan pembagian dilakukan dengan dasar penjumlahan. Sehingga sirkuit elektronik di ALU yang digunakan untuk melaksanakan operasi aritmatika ini disebut adder. ALU melakukan operasi aritmatika dengan dasar pertambahan, sedang operasi aritmatika yang lainnya, seperti pengurangan, perkalian, dan pembagian dilakukan dengan dasar penjumlahan. sehingga sirkuit elektronik di ALU yang digunakan untuk melaksanakan operasi arithmatika.

Gambar input dan Output pada ALU



Tugas dan Fungsi ALU

Tugas dari ALU adalah melakukan keputusan dari operasi logika sesuai dengan instruksi program. Operasi logika (logical operation) meliputi perbandingan dua buah elemen logika dengan menggunakan operator logika, yaitu :

- sama dengan (=)
- tidak sama dengan (<>)
- kurang dari (<)
- kurang atau sama dengan dari (<=)
- lebih besar dari (>)
- lebih besar atau sama dengan dari (>=)

Arithmatic Logical Unit (ALU) Juga Bertugas membentuk fungsi – fungsi pengolahan data komputer. ALU sering disebut mesin bahasa (machine language) karena bagian ini mengerjakan instruksi – instruksi bahasa mesin yang diberikan padanya. ALU terdiri dari dua bagian, yaitu unit arithmetika dan unit logika boolean, yang masing – masing memiliki spesifikasi dan tugas tersendiri. Fungsi-fungsi yang didefinisikan pada ALU adalah Add (penjumlahan), Addu (penjumlahan tidak bertanda), Sub (pengurangan), Subu (pengurangan tidak bertanda), and, or, xor, sll (shift left logical), srl (shift right logical), sra (shift right arithmetic), dan lain-lain.

Arithmetic Logical Unit (ALU) merupakan unit penalaran secara logic. ALU ini merupakan Sirkuit CPU berkecepatan tinggi yang bertugas menghitung dan membandingkan. Angka-angka dikirim dari memori ke ALU untuk dikalkulasi dan kemudian dikirim kembali ke memori. Jika CPU diasumsikan sebagai otaknya komputer, maka ada suatu alat lain di dalam CPU tersebut yang kenal dengan nama Arithmetic Logical Unit (ALU), ALU inilah yang berfikir untuk menjalankan perintah yang diberikan kepada CPU tersebut.

ALU sendiri merupakan suatu kesatuan alat yang terdiri dari berbagai komponen perangkat elektronika termasuk di dalamnya sekelompok transistor, yang dikenal dengan nama logic gate, dimana logic gate ini berfungsi untuk melaksanakan perintah dasar matematika dan operasi logika. Kumpulan susunan dari logic gate inilah yang dapat melakukan perintah perhitungan matematika yang lebih komplit seperti perintah "add" untuk menambahkan bilangan, atau "devide" atau pembagian dari suatu bilangan. Selain perintah matematika yang lebih komplit, kumpulan dari logic gate ini juga mampu untuk melaksanakan perintah yang berhubungan dengan logika, seperti hasil perbandingan dua buah bilangan.

Instruksi yang dapat dilaksanakan oleh ALU disebut dengan *instruction set*. Perintah yang ada pada masing-masing CPU belum tentu sama, terutama CPU yang dibuat oleh pembuat yang berbeda, katakanlah misalnya perintah yang dilaksanakan oleh CPU buatan Intel belum tentu sama dengan CPU yang dibuat oleh Sun atau perusahaan pembuat mikroprosesor lainnya. Jika perintah yang dijalankan oleh suatu CPU dengan CPU lainnya adalah sama, maka pada level inilah suatu sistem dikatakan compatible. Sehingga sebuah program atau perangkat lunak atau software yang dibuat berdasarkan perintah yang ada pada Intel tidak akan bisa dijalankan untuk semua jenis prosesor,kecuali untuk prosesor yang compatible dengannya.

Seperti halnya dalam bahasa yang digunakan oleh manusia, instruction set ini juga memiliki aturan bahasa yang bisa saja berbeda satu dengan lainnya. Bandingkanlah beda struktur bahasa Inggris dengan Indonesia, atau dengan bahasa lainnya, begitu juga dengan instruction set yang ada pada mesin, tergantung dimana lingkungan instruction set itu digunakan.

Struktur dan Cara Kerja Pada ALU

ALU akan bekerja setelah mendapat perintah dari Control Unit yang terletak pada processor. Contorl Unit akan memberi perintah sesuai dengan komando yang tertulis(terdapat) pada register. Jika isi register memberi perintah untuk melakukan proses penjumlahan, maka PC akan menyuruh ALU untuk melakukan proses penjumlahan. Selain perintah, register pun berisikan operand-operand. Setelah proses ALU selesai, hasil yang terbentuk adalah sebuah register yang berisi hasil atau suatuperintah lainnya. Selain register, ALU pun mengeluarkan suatu flag yang berfungsi untuk memberi tahu kepada kita tentang kondisi suatu processor seperti apakah processor mengalami overflow atau tidak.

ALU (Arithmethic and Control Unit) adalah bagian dari CPU yang bertanggung jawab dalam proses komputasi dan proses logika. Semua komponen pada CPU bekerja untuk memberikan asupan kepada ALU sehingga bisa dikatakan bahwa ALU adalah inti dari sebuah CPU. Perhitungan pada ALU adalah bentuk bilangan integer yang direpresentasikan dengan bilangan biner. Namun, untuk saat ini, ALU dapat mengerjakan bilangan floating point atau bilangan berkoma, tentu saja dipresentasikan dengan bentuk bilangan biner. ALU mendapatkan data (operand, operator, dan instruksi) yang akan disimpan dalam register. Kemudian data tersebut diolah dengan aturan dan sistem tertentu berdasarkan perintah control unit. Setelah proses ALU dikerjakan, output akan disimpan dalam register yang dapat berupa sebuah data atau sebuah instruksi. Selain itu, bentuk output yang dihasilkan oleh ALU berupa flag signal. Flag signal ini adalah penanda status dari sebuah CPU. Bilangan Ineger Bilangan integer (bulat) tidak dikena oleh komputer dengan basis 10. Agar komputer mengenal bilangan integer, maka para ahli komputer mengkonversi basis 10 menjadi basis 2. Seperti kita ketahui, bahwa bilangan berbasis 2 hanya terdiri atas 1 dan 0. Angka 1 dan 0 melambangkan bahwa 1 menyatakan adanya arus listrik dan 0 tidak ada arus listrik. Namun, untuk bilangan negatif, computer tidak mengenal simbol (-). Komputer hanya mengenal simbol 1 dan 0. Untuk mengenali bilangan negatif, maka digunakan suatu metode yang disebut dengan Sign Magnitude Representation. Metode ini menggunakan simbol 1 pada bagian paling kiri (most significant) bit. Jika terdapat angka 18 = (00010010)b, maka -18 adalah (10010010)b. Akan tetapi, penggunaan sign-magnitude memiliki 2 kelemahan. Yang pertama adalah terdaptnya -0 pada sign magnitude[0=(00000000)b; -0=(10000000)b].

Seperti kita ketahui, angka 0 tidak memiliki nilai negatif sehingga secara logika, sign-magnitude tidak dapat melakukan perhitungan aritmatika secara matematis. Yang kedua adalah, tidak adanya alat atau software satupun yang dapat mendeteksi suatu bit bernilai satu atau nol karena sangat sulit untuk membuat alat seperti itu. Oleh karena itu, penggunaan sign magnitude pada bilangan negatif tidak digunakan, akan tetapi diganti dengan metode 2′s complement. Metode 2′s complement adalah metode yang digunakan untuk merepresentasikan bilangan negatif pada komputer. Cara yang digunakan adalah dengan nilai terbesar dari biner dikurangin dengan nilai yang ingin dicari negatifnya. Contohnya ketika ingin mencari nilai -18, maka lakukan cara berikut:

a. ubah angka 18 menjadi biner (00010010)b
b. karena biner tersebut terdiri dari 8 bit, maka nilai maksimumnya adalah 11111111
c. kurangkan nilai maksimum dengan biner 18 -> 11111111 – 00010010 = 11101101
d. kemudian, dengna sentuhan terakhir, kita tambahkan satu -> 11101101 + 00000001 = 11101110

Dengan metode 2′s complement, kedua masalah pada sign magnitude dapat diselesaikan dan komputer dapat menjalankan. Namun, pada 2′s complement, nilai -128 pada biner 8 bit tidak ditemukan karena akan terjadi irelevansi.

ADDER

Adder merupakan rangkain ALU (Arithmetic and Logic Unit) yang digunakan untuk menjumlahkan bilangan. Karena adder digunakan untuk memproses operasi aritmatika, maka adder juga sering disebut rangkaian kombinasional aritmatika. Ada 3 jenis Adder, yaitu:

1. Rangkaian adder yang hanya menjumlahkan dua bit disebut Half Adder.
2. Rangkaian adder yang hanya menjumlahkan tiga bit disebut Full Adder.
3. Rangkaian adder yang menjumlahkan banyak bit disebut Paralel Adder.

· Half Adder.

Rangkain half adder merupakan dasar bilangan biner yang masing-masing hanya terdiri dari satu bit, oleh karena itu dinamakan penjumlah tak lengkap.

1. Jika A=0 dan B=0 dijumlahkan, hasilnya S (Sum) = 0.
2. Jika A=0 dan B=0 dijumlahkan, hasilnya S (Sum) = 1.

3. Jika A=1 dan B=1 dijumlahkan, hasilnya S (Sum) = 0. Dengan nilai pindahan Cy (Carry Out) = 1.

Dengan demikian, half adder memiliki dua masukan (A dan B), dan dua keluaran (S dan Cy).

A	B	S	Cy
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Dari tabel diatas, terlihat bahwa nilai logika dari Sum sama dengan nilai logika dari gerbang XOR, sedangkan nilai logika Cy sama dengan gerbang logika AND. Dari tabel diatas, dapat dibuat rangkaian half adder.

· Full Adder

Full adder adalag mengolah data penjumlahan 3 bit bilangan atau lebih (bit tidak terbatas), oleh karena itu dinamakan rangkaian penjumlah lengkap. Perhatikan tabel dibawah ini.

A	B	C	S	Cy
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

· Paralel Adder

Paralel Adder adalah rangkaian Full Adder yang disusun secara paralel dan berfungsi untuk menjumlahkan bilangan biner berapa pun bitnya, tergantung jumlah Full Adder yang diparalelkan. Gambar dibawah ini menunjukan Paralel Adder yang terdiri dari 4 buah Full Adder yang disusun paralel sehingga membentuk sebuah penjumlahan 4 bit.

2. INTEGER REPRESENTATION

Dalam sistem bilangan biner, [1] semua bilangan dapat direpresentasikan dengan hanya menggunakan digit-digit nol dan satu, tanda minus, dan tanda titik.

$-1101.0101_2 = -13{,}3125_{10}$

Namun, untuk keperluan pengolahan dan penyimpanan komputer, kita tidak perlu merepresentasikan bilangan. Jika kita hanya terbatas pada integer nonnegatif, maka representasinya akan lebih mudah.

Sebuah word 8-bit dapat digunakan untuk merepresentasikan bilangan-bilangan dari 0 hingga

00000000 = 0

00000001 = 1

00101001 = 41

10000000 = 128

11111111 = 255

Secara umum, jika sebuah rangkaian n-bit bilangan biner $a_{n-1}a_{n-2}....a_1a_0$ direpresentasikan sebagai suatu integer tanpa A, nilainya adalah

$A = \sum_i a_i$

Representasi Magnituda Tanda (Sign-magnitude)

Terdapat beberapa konvensi alternatif yang digunakan untuk merepresentasikan bilangan integer negatif seperti halnya bilangan integer positif, semua konpensi tersebut meliputi perlakuan bityang paling signifikan (paling kiri) di dalam word sebagai bit tanda. Jika bit tanda adalah 0, maka bilangan tersebut positif; jika bit tanda adalah 1, maka bilangan tersebut adalah negatif.

Bentuk reprentasi yang paling sederhana yang memakai bit tanda adalah adalah reprentasi magnituda tanda. pada suatu word n-bit, bit *n*-1 paling kanan menampung nilai integer.

+18= 00010010

-18= 10010010 (magnituda tanda)

Secara umum kasus tersebut adalahdapatdiekspresikan sebagai berikut:

Terdapat beberapa kelemahan pada representasi magntuda tanda. salah satunya adalah bahwa penambahan dan pengurangan memerlukan pertimbangan baik tanda bilangan maupun nilai relatifnya untuk menyelesaikan operasi yang diperlukan. Masalah ini adalah menjadi jelas dalam pembahasan pada sub bab 9.3. kelemahan yang lain adalah bahwa terdapatdua reprentasi bilangan 0.

+ 0_{10} = 00000000

- 0_{10} = 10000000 (magnituda tanda)

Hal ini merepotkan, karena akan menyulitkan pemeriksaan bilangan 0 (suatu operasi yang sering dilakukan pada komputer) dibandingkan jika menggunakan representasi tunggal.

Karea kelemahan ini, representasi magnituda tanda jarang digunakan di dalam implementasi bagian bilangan integer ALU. Disamping itu, teknik yang paling umum adalah representation komplemen dua.

Representasi Komplemen Dua

Seperti halnya magnituda tanda, representasi komplemen dua menggunakan bit yang paling signifikan sebagai bit tanda, yang memudahkannya untuk mengetahui apakah suatu integer bernilai positif atau negatif. Representasi ini berbeda dengan penggunaan representasi magnitude tanda dalam cara dengan bit-bit lainnya diinterpretasikan. Tabel 9.1 menekankan karakteristik penting representasi komplemen dua dan aritmetika, yang dibahas dalam bagian ini dan yang berikutnya.

Sebagian besar perlakuan representasi komplemen dua menfokuskan pada aturan untuk menghasilkan bilangan-bilangan negatif, dengan tidak ada bukti formal bahwa teknik “bekerja“. Di samping itu, pembahasan kita tentang integer-integer komplemen dua dalam

bagian ini dan dalam Subbab 9.3 berdasarkan pada (DATT93), yang menyatakan bahwa representasi

Range	-2^{n-1} sampai $2^{n-1} - 1$
Jumlah representasi nol	Satu
Negasi	Ambil komplemen Boolean dari setiap bit yang bersesuaian dengan bilangan positif, kemudian tambahkan 1 untuk menghasilkan pola bit yang dilihat sebagai integer tanpa tanda.
Perluasan panjang bit	Tambahkan posisi bit tambahan ke kiri da nisi dengan bit magnituda tanda asli.
Aturan overflow	Jika dua bilangan-bilangan dengan tanda yang sama (keduanya bilangan positif atau keduanya bilangan negatif) ditambahkan, maka terjadi overflow jika dan hanya jika hasilnya mempunyai tanda berkebalikan.
Aturan pengurangan	Untuk mengurangi B dari A, ambil komplemen kedua dari B dan menambahkannya ke A.

Komplemen dua adalah terbaik untuk dipahami oleh pendefenisiannya dalam kaitannya dengan penjumlahan bit-bit berbobot, seperti ketika kita melakukan sebelumnya pada representasi tanpa tanda dan magnitude tanda. Keuntungan perlakuan yang demikian adalah bahwa representasi itu tidak meninggalkan keraguan apapun dimana aturan operasi-operasi aritmatika di dalam dua notasi komplemen tidak dapat bekerja untuk beberapa kasus khusus.

Perhatikan suatu integer n-bit, A, dalam representasi komplemen dua. Jika A bilangan positif, maka bit tanda, a_{n-1}, adalah nol. Bit-bit lainnya merepresentasikan nilai bilangan dalam mode yang sama dengan magnitude tanda:

$$A = \sum_{i=0}^{n-1} 2^i a_i$$

untuk $A \geq 0$

Bilangan nol akan didefenisikan sebagai bilangan positif dan sehingga mempunyai bit tanda 0 dan nilai keseluruhan 0. Kita dapat melihat bahwa cakupan integer positif yang dapat direpresentasikan mulai dari 0 (semua bit nilainya adalah 0) hingga $2^{n-1} – 1$ (semua bit nilainya adalah 1). Bilangan yang lebih besar maupun akan memerlukan bit yang lebih banyak.

Sekarang, untuk bilangan negatif A (Ad “0), bit tanda, a_{n-1}, adalah satu. n – 1 bit sisanya dapat mengambil salah satu dari nilai-nilai 2^{n-1}. Oleh karena itu, cakupan integer negative yang dapat direpresentasikan adalah mulai dari 1 hingga- $2^{n-1.}$ Kita bermaksud menugaskan nilai-nilai bit ke integer negative sehingga aritmatika dapat ditangani di dalam mode secara langsung, mirip dengan aritmatika integer tanpa tanda. Pada representasi integer tanpa tanda, untuk menghitung nilai integer dari representasi bit, bobot bit-bit yang paling signifikan adalah $+2^{n-}1$. Untuk representasi dengan bit tanda, representasi menghasilkan tercapainya aritmatika yang diinginkan, seperti yang akan kita lihat pada sub bab 9.3, jika bobot bit yang paling signifikan adalah $-2^{n-}1$. Hal ini merupakan kontrovensi yang digunakan dalam representasi komplemen dua, menghasilkan ekspresi berikut untuk bilangan-bilangan negatif:

$$A = \sum_{i=0}^{n-2} 2iai \quad \text{untuk } A \geq 0 \qquad (9.2)$$

Pada kasus bilangan integer positif, $a_{n-1}=0$, maka suku $-2^{n-1}a_{n-1}= 0$. Dengan demikian, Persamaan (9.2) mendefenisikan representasi komplemen dua untuk bilangan positif dan negative.

Tabel 9.2 membandingkan magnitude tanda dan representasi komplemen dua integer 4-bit. Walaupun dari sudut pandang manusia komplemenn dua merupakan representasi yang canggung, kita akan lihat bahwa itu memudahkan operasi-operasi aritmatika yang paling utama, penambahan dan pengurangan. Dengan alas an ini, representasi ini sangat umum, representasi ini sangat umum digunakan sebagai representasi prosessor untuk integer.

Suatu ilustrasi yang bbermanfaat tentang sifat representasi komplemen dua adalah kotak nilai, dimana nilainya terdapat jauh pada sisi kanan kotak adalah 1 (2^0) dan setiap posisi berikutnya ke sebelah kiri merupakan nilai gandanya, sampai posisi yang paling kiri, yang ditiadakan. Seperti yang anda lihat pada gambar 9.2a, bilangan komplemen dua yang paling negtif yang dapat direpresentasikan adalah $2^{n-1,}$ jika terdapat sembarang bit selain bit tanda satu, maka bit iitu akan menambahkan jumlah positive terhadap bilangan. Selain itu, jelas bahwa bilangan negative harus mempunyai suatu bilangan 1 pada posisi yang paling kirinya dan bilangan positive harus mempunyai suatu bilangan 0 pada posisi tersebut. Jadi bilangan positif yang paling besar adalah suatu 0 diikuti oleh semua 1, yang sama dengan $2^{n-1} – 1$.

Bagian lainnya pada gambar 9.2 mengilustrasikan penggunaan kotak nilai untuk mengkonversikan komplemen dua ke bilangan decimal dan dari bilangan decimal menjadi komplemen dua.

Tabel 9.2

Representasi Alternatif untuk Integer 4-bit

Representasi Bilangan Desimal	Representasi Magnituda Tanda	Representasi Komplemen Dua	Representasi Terbias
+8	-	-	1111
+7	0111	0111	1110
+6	0110	0110	1101
+5	0101	0101	1100
+4	0100	0100	1011
+3	0011	0011	1010
+2	0010	0010	1001
+1	0001	0001	1000
+0	0000	0000	0111
-0	1000	-	
-1	1001	1111	0110
-2	1010	1110	0101
-3	1011	1101	0100

-4	1100	1100	0011
-5	1101	1011	0010
-6	1110	1010	0001
-7	1111	1001	0000
-8	-	1000	-

Gambar 9.2

Penggunaan Kotak Nilai untuk Konversi Antara Dua Bilangan Komplemen Biner dan Bilangan Desimal

-128	64	32	16	8	4	2	1

(a) Suatu delapan posisi kotak nilai komplemen dua

-128	64	32	16	8	4	2	1
1	0	0	0	0	0	1	1

-128 + 2 + 1 = -125

(b) konversi biner 10000011 ke decimal

Konversi Antara Panjang Bit yang Berlainan

Kadang-kadang kita perlu mengambil integer n-bit dan menyimpannya di dalam bit m, di mana m > n. Pada notasi magnitude tanda, hal ini mudah terpenuhi: Sederhananya cukup memindahkan bit tanda ke posisi paling kiri yang baru dan mengisi sisanya dengan nol.

+18 = 00010010 (magnituda tanda, 8 bit)

+18 = 0000000000010010 (magnituda tanda, 16 bit)

-18 = 10010010 (magnituda tanda, 8 bit)

-18 = 1000000000010010 (magnituda tanda, 16 bit)

Prosedur ini tidak berlaku bagi integer negative komplemen dua. Dengan menggunakan contoh yang sama,

+18 = 00010010 (komplemen dua, 8 bit)

+18 = 0000000000010010 (komplemen dua, 16 bit)

-18 = 11101110 (komplemen dua, 8 bit)

-32,658 = 10000000001101110 (komplemen dua, 16 bit)

Selanjutnya sampai baris terakhir mudah dilihat dengan menggunakan kotak nilai pada Gambar 9.2. Garis terakhir dapat diverifikasi dengan menggunakan Persamaan 9.2 atau kotak nilai 16-bit.

Di samping itu, aturan integer komplemen dua akan memindahkan bit tanda ke posisi paling kiri yang baru dan mengisinya dengan salinan bit tanda. Untuk bilangan-bilangan positif, isi dengan nol, dan untuk bilangan-bilangan negative, isi dengan satu. Hal ini disebut sebagai perluasan tanda.

-18 = 11101110 (komplemen dua, 8 bit)

-18 = 1111111111101110 (komplemen dua,16 bit)

Untuk mengetahui mengapa aturan ini bisa berlaku, mari kita memperhatikan sekumpulan bilangan integer biner n-bit $a_{n-1}a_{n-2}...a_1a_0$ yang diinterpretasikan sebagai integer komplemen dua A, sehingga nilainya adalah :

$$A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Jika A bilangan positif, maka aturan tersebut jelas akan berlaku. Sekarang, jika A negative dan kita akan membuat representasi m-bit, dengan m > n. maka:

$$A = -2^{m-1} a_{m-1} + \sum_{i=0}^{m-2} 2^i a_i$$

Kedua nilai harus sama

$$-2^{m-1} a_{m-1} + \sum_{i=0}^{m-2} 2^i a_i = -2^{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

$$-2^{m-1} a_{m-1} + \sum_{i=n-1}^{m-2} 2^i a_i = -2^{n-1}$$

$$2^{m-1} a_{m-1} + \sum_{i=n-1}^{m-2} 2^i a_i = -2^{m-1}$$

$$1 + \sum_{i=0}^{n-2} 2^i + \sum_{i=n-1}^{m-2} 2^i a_i = 1 + \sum_{i=0}^{m-2} 2^i$$

$$\sum_{i=n-1}^{m-2} 2^i a_i = \sum_{i=n-1}^{m-2} 2^i$$

$$\Rightarrow a_{m-2} = ... = a_{n-2} = a_{n-1} = 1$$

Dalam mengerjakan persamaan menjadi persamaan kedua, kita memerlukan n-1 bit paling sedikit signifikan tidak boleh berubah pada kedua representasinya. Kemudian kita akan sampai pada persamaan terakhir, yang hanya akan benar jika semua bit yang berada pada posisi n-1 sampai m-2 adalah 1. Maka dengan demikian aturan ekstensi tanda akan berfungsi.

Representasi Titik Tetap

Terakhir, kita menyebutkan bahwa representasi yang telah dibahas dalam bagian ini kadang-kadang dikenal sebagi titik tetap. Hal ini karena titik radiksnya (titik biner) tetap dan diasumsikan akan berada di sebelah kanan dari digit yang paling kanan. Pemprogram dapat menggunakan representasi yang sama untuk bilangan pecahan biner dengan melakukan peskalaan bilangan-bilangan yang bersangkutan sehingga titik biner secara implisit berada pada lokasi lain.

3. INTEGER ARITHMETIC

Bagian ini akan membahas fungsi-fungsi aritmatik bilangan dalam representasi komplemen dua

Negasi

Pada notasi komplemen dua, pengurangan sebuah bilangan integer dapat dibentuk dengan menggunakan aturan berikut :

Anggaplah komplemen Boolean seluruh bit bilangan integer (termasuk bit tanda)

Perlakukan hasilnya sebagai sebuah unsigned binary integer, tambahkan 1.

Misal : 18 = 00010010 (komplemen dua)

Representasi Integer Positif, Negatif Dan Bilangan 0

- Bila sebuah bilangan integer positif dan negatif yang sama direpresentasikan (sign-magnitude), maka harus ada representasi bilangan positif dan negatif yang tidak sama.
- Bila hanya terdapat sebuah representasi bilangan 0 (komplemen dua), maka harus ada representasi bilangan positifdan negatif yang tidak sama.
- Pada kasus komplemen dua, terdapat representasi bilangan n-bit untuk -2n, tapi tidak terdapat untuk 2n.

Aturan Untuk Mendeteksi Overflow

1. Aturan Overflow :

Bila dua buah bilangan ditambahkan, dan keduanya positif atau keduanya negatif, maka akan terjadi overflow bila dan hanya bila hasilnya memiliki tanda yang berlawanan, seperti pada contoh halaman 18 ((e),(f))

2. Aturan Pengurangan :

   Untuk mengurangkan sebuah bilangan (subtrahend) dari bilangan lainnya (minuend), anggaplah komplemen dua subtrahend dan tambahkan hasilnya ke minuend.

Pembulatan

Teknik pembulatan yang sesuai dengan standard IEEE adalah sebagai berikut :

1. Pembulatan ke Bilangan Terdekat : Hasil dibulatkan ke bilangan terdekat yang dapat direpresentasi.
2. Pembulatan Ke Arah : Hasil dibulatkan ke atas ke arah tak terhingga positif.
3. Pembulatan Ke Arah : Hasil dibulatkan ke atas ke arah tak terhingga negatif.
4. Pembulatan Ke Arah 0 : Hasil dibulatkan ke arah 0

4. FLOATING POINT REPRESENTATION

Representasi Integer oleh Biner

Dalam sistem bilangan biner ada 4 macam sistem untuk merepresentasikan integer

- representasi unsigned integer
- representasi nilai tanda (sign magnitude)
- representasi bias
- representasi komplemen dua (2's complement)

Unsigned Integer

Untuk keperluan penyimpanan dan pengolahan komputer diperlukan bilangan biner yang terdiri atas 0 dan 1

1 byte (8 bit binary digit) dapat digunakan untuk menyatakan bilangan desimal dari 0 – 255

Kelemahan Unsigned Integer

- Hanya dapat menyatakan bilangan positif
- Sistem ini tidak bisa digunakan untuk menyatakan bilangan integer negatif

Representasi Nilai Tanda (sign magnitude)

- Karena kelemahan unsigned integer
- Dikembangkan beberapa konvensi untuk menyatakan bilangan integer negatif

Konvensi

- Perlakuan bit yang paling kiri (MSB) di dalam byte sebagai tanda
- Bila MSB = 0 maka bilangan tersebut positif
- Jika MSB = 1 maka bilangan tersebut negatif

Kelemahan sign magnitude

- Adanya representasi ganda pada bilangan 0, yaitu

Representasi BIAS

- Digunakan untuk menyatakan exponen (bilangan pemangkat) pada representasi floating point
- Dapat menyatakan bilangan bertanda, yaitu dengan mengurutkan dari bilangan negatif terkecil dapat dijangkau sampai bilangan positif paling besar yang bisa dijangkau
- Mengatasi permasalahan pada sign magnitude yaitu +0 dan -0

Representasi Komplemen 2

Merupakan perbaikan dari representasi nilai bertanda (sign magnitude) yang mempunyai kekurangan pada operasi penjumlahan dan pengurangan serta representasi nilai 0.

Bilangan Negatif Pada 2's Complement

1. Sistem bilangan dalam 2's complement menggunakan bit paling kiri (MSB) sebagai bit tanda dan sisanya sebagai bit nilai seperti pada sign magnitude
2. Bilangan negatif dalam 2's complement dibentuk dari:

- komplemen satu dari bilangan biner semula (nilai positif)
- menambahkan 1 pada LSB

5. FLOATING POINT ARITHMETIC

Bentuk Bilangan Floating Point

Bilangan Floating Point memiliki bentuk umum : + m * b e , dimana m (disebut juga denganmantissa), mewakili bilangan pecahan dan umumnya dikonversi ke bilangan binernya, e mewakili bilangan exponentnya, sedangkan b mewakili radix (basis) dari exponent.

Contoh :

Pada gambar diatas, menunjukkan tentang panjang bit pada bilangan floating point m = 23 bit, e = 8 bit, dan S (bit sign) = 1. Jika nilai yang tersimpan di S adalah 0, maka bilangan tersebut adalah positif dan jika nilai yang tersimpan pada S adalah 1, maka bilangan tersebut adalah negatif. Bilangan *exponent* pada contoh diatas, hanya dapat digunakan pada bilangan positif 0 hingga 255. Untuk dapat menggunakan bilangan exponent negatif dan positif, nilai bulat yang disebut dengan*bias*, dikurangkan dengan bilangan pada kolom exponent dan menghasilkan bilangan exponent akhir. Misalkan pada contoh diatas menggunakan *bias* = 128, maka bilangan exponent akhirnya memiliki range antara 128 (disimpan sebagai 0 pada kolom exponent) hingga +127 (disimpan sebagai 255 pada kolom exponent). Berdasarkan bentuk seperti ini, bilangan exponent +4 dapat digunakan dengan menyimpan 132 pada kolom exponent, sedangkan bilangan exponent 12 dapat digunakan dengan menyimpan 116 pada kolom exponent. Anggap b = 2, maka bilangan floating point seperti 1,75 dapat menggunakan salah satu dari bentuk umum seperti pada gambar berikut:

Macam-macam bentuk bilangan *floating point*

Untuk mempermudah operasi bilangan floating point dan menambah tingkat presisinya, maka bilangan tersebut dibuat dalam bentuk ternormalisasi (*normalized forms*). Suatu bilangan floating point telah ternormalisasi jika *most significant bit (MSB)* dari mantissanya adalah 1. Karena itu, diantara ketiga bentuk diatas dari bilangan 1,75, maka bentuk yang telah ternormalisasi adalah bentuk yang paling atas, dan disarankan untuk digunakan.

Karena nilai MSB dari bilangan Floating Point yang telah ternormalisasi selalu 1, maka bit ini tidak disimpan, sehingga nilai mantissa yang tersimpan adalah 1.*m*. Sehingga untuk bilangan *floating point* bukan nol yang ternormalisasi memiliki bentuk (-1)S * (1.*m*) * 2*e-128*

Aritmetika Floating Point Penjumlahan / Pengurangan

Hal yang sulit dari penjumlahan dua bilangan exponent adalah jika bilanganbilangan tersebut memiliki bentuk exponensial yang berbeda. Unutk memecahkannya, maka sebelum ditambahkan bilangan exponensialnya harus disetarakan terlebih dahulu, atau bilangan dengan nilai *exponent* lebih kecil disamakan dulu ke bilangan*exponent* yang sama dengan bilangan lain.

Langkah-langkah yang dilakukan untuk menambah/mengurangkan dua bilangan *floating point* :

1. Bandingkan kedua bilangan, dan ubah ke bentuk yang sesuai pada bilangan dengan nilai exponensial lebih kecil
2. Lakukan operasi penjumlahan / pengurangan
3. Lakukan normalisasi dengan 'menggeser' nilai *mantissa* dan mengatur nilai Exponensialnya

Contoh : Jumlahkan dua bilangan *floating point* 1,1100 * 2 4 dan 1,1000 * 2 2

1. Sesuaikan : 1,1000 * 2 2 diubah menjadi 0,0110 * 2 4
2. Jumlahkan : hasil penjumlahan 10,0010 * 2 4
3. Normalisasi : hasil setelah dinormalisasi adalah 0,1000 * 2 6 (dianggap bit yang diijinkan setelah koma adalah 4).

Operasi penjumlahan/pengurangan dua bilangan *floating point*diilustrasikan dengan skema seperti pada gambar berikut :

Perkalian

Perkalian dari dua bilangan *floating point* dengan bentuk X = *mx * 2 a dan* Y = *mx * 2 b* setara dengan X * Y = (*mx * my*) * 2 a+b

Algoritma umum untuk perkalian dari bilangan *floating point* terdiri dari tiga langkah:

1. Hitung hasil exponensial dengan menjumlahkan nilai exponent dari kedua Bilangan
2. Kalikan kedua bilangan *mantissa*
3. Normalisasi hasil akhir

Contoh : Perkalian antara dua bilangan *floating point X* = 1,000 * 2 2 dan *Y* = 1,010 * 2 1

1. Tambahkan bilangan exponennya : 2 + (1) = 3

2. Kalikan *mantissa:* 1,0000 * 1,010 = 1,010000

Hasil perkaliannya adalah 1,0100 * 2 3

Perkalian dari dua bilangan *floating point* diilustrasikan menggunakan skema seperti tampak pada gambar berikut :

Pembagian dari dua bilangan *floating point* dengan bentuk X = *mx * 2 a dan* Y = *mx * 2 b* setara dengan X / Y = (*mx / my*) * 2 ab

Algoritma umum untuk pembagian dari bilangan *floating point* terdiri dari tiga langkah :

1. Hitung hasil exponensial dengan mengurangkan nilai exponent dari kedua bilangan
2. Bagi kedua bilangan *mantissa*
3. Normalisasi hasil akhir

Contoh : Pembagian antara dua bilangan *floating point X* = 1,0000 * 2 2

dan *Y* = 1,0100 * 2 1

1. Kurangkan bilangan exponennya : 2 – (1)= 1
2. Bagi *mantissa:* 1,0000 / 1,0100 = 0,1101

Hasil pembagiannya adalah 0,1101 * 2 1

Pembagian dari dua bilangan *floating point* diilustrasikan menggunakan skema seperti tampak pada gambar berikut :

Floating Point standard IEEE

IEEE membuat dua bentuk bilangan *floating point* standard. Bentuk*basic* dan bentuk *extended*. Pada tiap bentuk tersebut, IEEE menentukan dua format, yaitu *singleprecision* dan *double precision format*. *Single precision format* adalah model 32bit sedangkan *double precision format* adalah 64bit. Pada *single extended format*setidaknya menggunakan 44 bit, sedangkan pada *double extended format* setidaknya menggunakan 80 bit.

Pada *single precision format*, mengijinkan penggunaan bit tersembunyi, kolom exponentnya adalah 8bit. Bentuk *single precision*ditunjukkan pada gambar berikut.

IEEE single precision Format

Jika jumlah bit bilangan exponent adalah 8, maka nilainya memiliki 256 kombinasi, diantara angka-angka tersebut, dua kombinasi digunakan sebagai nilai

khusus:

1. e = 0 bernilai nol (jika m = 0) dan nilai terdenormalisasi (jika m≠ 0)
2. *e* = 255 bernilai + ∞ (jika *m = 0*) dan nilai tak terdefinisi (jika *m*≠ 0)

IEEE Double Precision Format

Bentuk ini memiliki kolom exponent 11 bit dan kolom nilai mantissasebesar 52 bit. Bentuknya seperti tampak pada gambar.

KESIMPULAN

Karena computer semula digunakan untuk menggambarkan orang yang perkerjaannya melakukan perhitungan aritmatika, dengan atau tanpa alat bantu, tetapi arti kata ini kemudian dipindahkan kepada mesin itu sendiri. Asal mulanya, pengolahan informasi hampir eksklusif berhubungan dengan masalah aritmatika, tetapi komputer modern dipakai untuk banyak tugas yang tidak berhubungan denganmatematika. Dalam arsitektur von Neumann yang asli, ia menjelaskan sebuah Unit Aritmatika dan Logika, dan sebuah Unit Kontrol. Dalam komputer-komputer modern, kedua unit ini terletak dalam satu sirkuit terpadu (IC - Integrated Circuit), yang biasanya disebut CPU (Central Processing Unit).

Unit Aritmatika dan Logika, atau Arithmetic Logic Unit (ALU), adalah alat yang melakukan pelaksanaan dasar seperti pelaksanaan aritmatika (tambahan, pengurangan, dan semacamnya), pelaksanaan logis (AND, OR, NOT), dan pelaksanaan perbandingan (misalnya, membandingkan isi sebanyak dua slot untuk kesetaraan). Pada unit inilah dilakukan "kerja" yang sebenarnya.ALU ini adalah merupakan Sirkuit CPU berkecepatan tinggi yang bertugas menghitung dan membandingkan. ALU sendiri merupakan suatu kesatuan alat yang terdiri dari berbagai komponen perangkat elektronika termasuk di dalamnya sekelompok transistor, yang dikenal dengan nama logic gate, dimana logic gate ini berfungsi untuk melaksanakan perintah dasar matematika dan operasi logika. Instruksi yang dapat dilaksanakan oleh ALU disebut dengan instruction set. Tugas lain dari ALU adalah melakukan keputusan dari suatu operasi logika sesuai dengan instruksi program. Operasi logika meliputi perbandingan dua operand dengan menggunakan operator logika tertentu, yaitu sama dengan (=), tidak sama dengan (<>), kurang dari (<), kurang atau sama dengan (<=), lebih besar dari (>), dan lebih besar atau sama dengan (>=).