Ir. Heru Nurwarsito, M.Kom
Barlian Henryranu P, ST, MT
Eko Saksi Pramukantoro, S.Kom, M.Kom

1. PENDAHULUAN
- Pengantar
- Tujuan
- Latar belakang

2. bilangan
2.1 bilangan positif
2.2 bilangan negatif
2.3 **Konversi antara panjang Bit yang berbeda**

3. operasi aritmatik
3.1 penjumlahan
3.2 perkalian
3.3 pengurangan
3.4 pembagian

1. PENDAHULUAN

1.1 Pengantar

ALU, singkatan dari *Arithmetic And Logic Unit* (bahasa Indonesia: unit aritmatika dan logika), adalah salah satu bagian dalam dari sebuah mikroprosesoryang berfungsi untuk melakukan operasi hitungan aritmatika dan logika. Contoh operasi aritmatika adalah operasi penjumlahan dan pengurangan, sedangkan contoh operasi logika adalah logika AND dan OR. tugas utama dari ALU (Arithmetic And Logic Unit)adalah melakukan semua perhitungan aritmatika atau matematika yang terjadi sesuai dengan instruksi program. ALU melakukan operasi aritmatika yang lainnya. Seperti pengurangan, pengurangan, dan pembagian dilakukan dengan dasar penjumlahan. Sehingga sirkuit elektronik di ALU yang digunakan untuk melaksanakan operasi aritmatika ini disebut *adder.* ALU melakukan operasi arithmatika dengan dasar pertambahan, sedang operasi arithmatika yang lainnya, seperti pengurangan, perkalian, dan pembagian dilakukan dengan dasar penjumlahan. sehingga sirkuit elektronik di ALU yang digunakan untuk melaksanakan operasi arithmatika ini disebut *adder*. Tugas lalin 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

1.2 Tujuan

Modul ini membahas fungsi dari unit aritmatika dan logika (ALU) dan berfokus pada representasi angka dan teknik untuk melaksanakan operasi aritmatika. Prosesor biasanya mendukung dua jenis aritmatika: integer, atau titik tetap, dan floating point. untuk kedua kasus, bab yang pertama mengkaji representasi angka dan kemudian membahas operasi aritmatika. IEEE penting 754 floating-point standar diperiksa secara detail.

1.3 Latar Belakang

Karena transaksi elektronik logika dengan arus yang sedang aktif atau tidak aktif telah ditemukan, maka untuk mewakili kuantitas dapat dengan mudah dimengerti oleh aritmatika komputer ketika nilainya diubah dalam bentuk biner. Jadi, daripada harus berbeda sepuluh angka, 0, 1, 2, 3, 4, 5, 6, 7, 8, dan 9, di aritmetika biner, hanya ada dua digit berbeda, 0 dan 1. Ketika pindah ke kolom berikutnya, bukan mewakili angka kuantitas yang sepuluh kali lebih besar, hanya merupakan sebuah kuantitas yang dua kali lebih besar. Dengan demikian, angka pertama ditulis dalam biner sebagai berikut:

Desimal		Biner
Nol	0	0
Satu	1	1
Dua	2	10
Tiga	3	11
Empat	4	100
Lima	5	101
Enam	6	110
Tujuh	7	111
Delapan	8	1000
Sembilan	9	1001
Sepuluh	10	1010
Sebelas	11	1011
Dua belas	12	1100

Dengan demikian, hanya ada bilangan 0 & 1 untuk merepresentasikan semua angka. Lalu bagaimana dengan bilangan yang bernilai minus (negative, kurang dari 0) ? Itu akan dijawab dalam sub-bab yang membahas tentang bilangan biner negatif.

2. BILANGAN

2.1 Bilangan Positif

Seandainya semua integer positif, konversi ke biner biasa tinggal disesuaikan dengan panjang bit register yang tersedia. Misal data akan disimpan ke dalam register 8-bit :

```
00000000 : 0
00000001 : 1
00101001 : 41
10000000 : 128
11111111 : 255
```

Tidak ada masalah dalam konversi bilangan biner, baik dari bilangan desimal ke bilangan biner dan juga sebaliknya.

2.2 Bilangan Negatif

Membahas bilangan negatif, ternyata ada masalah dalam konversi bilangan biner negatif. Lihat saja contoh di bawah ini :

```
+18 : 00010010
-18 : 10010010
+10 : 00001010
-10 : 10001010
+0  : 00000000
-0  : 10000000
```

Dari contoh +18 dan -18 masih tidak ada masalah, begitu juga dengan contoh pada +10 dan -10. Lalu bagaimana dengan +0 dan -0? Padahal hanya ada 1 nilai 0, tidak ada nol negatif dan nol positif.Maka dari itu ada solusi dalam mengatasi kekurangan pada konversi bilangan negatif ini. Solusinya adalah dengan merepresentasikan bilangan biner dengan menggunakan komplemen-2.

Pada contoh ini akan dijelaskan bagaimana mencari bilangan biner negatif dari -18 menggunakan metode representasi dengan komplemen 2. Berikut adalah caranya :

1. Tentukan biner positif dari 18.
2. Negasikan biner dari 18.
3. Jumlahkan hasil negasi dari biner 18 dengan 1

```
1. Biner dari         18 : 00010010
2. Negasi biner dari 18 : 11101101
3. Jumlahkan hasil negasi :       1+

Hasil penjumlahan negasi :11101110

Jadi, biner dari -18 adalah 11101110
```

Sebagai informasi, metode ini digunakan di komputer sekarang. Metode konversi bilangan biner menggunakan komplemen 2 hanya memiliki 1 biner yang bernilai desimal 0 sehingga lebih unggul dibanding metode sebelumnya.

Tetapi, ada kekurangan yang dihasilkan dari metode ini. Terjadi ketimpangan representasi nilai negatif dan positif untuk jumlah bit tertentu. Misal untuk 8-bit, range bilangan bulat yang terwakili adalah -128 hingga 127 (bukan 128, tapi 128-1)

```
Range bilangan : (-n) sampai (n-1)
```

Inilah yang terjadi di komputer kita. Silahkan cek range bilangan untuk setiap tipe data yang dimengerti oleh komputer.

2.3 Konversi antara panjang Bit yang berbeda

Mungkin saja terdapat perbedaan jumlah bit antar register. Ada yang berjumlah 8 bit hingga 16 bit. Ini adalah peraturan konversi antara panjang bit yang berbeda :

Pada bilangan positif harus ditambahkan dengan nilai 0 (digaris bawahi) di bagian depan. Contoh :

```
+18 :          00010010

+18 : 00000000 00010010
```

Sedangkan pada bilangan negatif harus ditambahkan dengan nilai 1 (digaris bawahi) di bagian depan. Contoh :

```
-18 :          10010010

-18 : 11111111 10010010
```

3. OPERASI ARITMATIKA

Pada pembahasan sebelumnya telah dibahas tentang konversi bilangan biner negatif menggunakan komplemen-2. Ternyata metode ini memudahkan komputer dalam operasi matematika dasar yang terdiri dari penjumlahan, pengurangan, perkalian, dan pembagian. Dalam operasi penjumlahan dan pengurangan, komputer hanya menggunakan rangkaian penjumlahan karena untuk operasi pengurangan bisa diselesaikan dengan solusi :

```
8- 2 = 8 + (-2)
```

Pada contoh di atas, komputer akan mencari nilai minus 2 kemudian ditambahkan dengan 8. Sehingga nilai yang dihasilkan akan sama dengan 8 dikurangi 2.

3.1 Penjumlahan

Seperti operasi penjumlahan pada bilangan desimal, kita bisa menjumlahkan beberapa bilangan biner dengan mudah. Simak contoh berikut :

```
 8 : 1000
 2 : 0010 +
-----------
10: 1010
```

Pada contoh di atas tidak ada aturan khusus karena hanya ada penjumlahan antara 0 dengan 0 dan antara 0 dengan 1. Lalu bagaimana untuk penjumlahan bit 1 dengan 1? Simak contoh berikut :

```
10: 1010
 2 : 0010 +
-----------
12: 1100
```

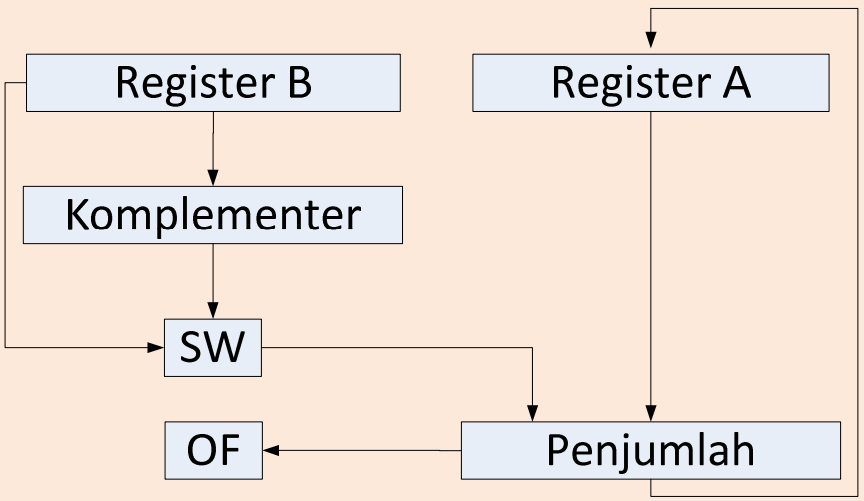
Terdapat kekhususan dalam penjumlahan bit 1 dengan 1. Hasil dari penjumlahan bit 1 dengan 1 adalah nol. Kemudian biner di sebelah kirinya akan ditambah dengan bit bernilai 1 yang merupakan simpanan dari bit yang dijumlahkan tersebut.

3.2 Pengurangan

Seperti yang telah dibahas sebelumnya, di aritmatika komputer tidak dikenal

pengurangan. Yang dikenal oleh aritmatika komputer adalah penjumlahan dari bilangan negatif. Langkah untuk pengurangan di aritmatika komputer adalah :

1. Mencari nilai biner negatif dari bilangan pengurang.
2. Menjumlahkan bilangan yang dikurangi dengan bilangan negatif dari pengurang.



Gambar 1. Algoritma Penjumlahan Komputer

3.3 Perkalian

Operasi perkalian lebih rumit dibanding operasi penjumlahan atau pengurangan, baik dalam hardware maupun software. Ada beberapa jenis algoritma yang digunakan untuk operasi perkalian dalam komputer :

a. Perkalian Unsigned Integer
b. Perkalian Komplemen-2
c. Perkalian menggunakan Algoritma Booth

Apapun algoritma yang dipakai, ada istilah universal yang digunakan untuk operasi perkalian dalam komputer. Contohnya untuk perkalian antara 2 dan 3 yang menghasilkan 6. Angka 2 merupakan *multiplicand*, angka 3 merupakan *multiplier*, dan angka 6 merupakan *product* dari perkalian tersebut.

3.3.1 Perkalian Unsigned Integer

Perkalian menggunakan algoritma unsigned integer hampir sama dengan operasi perkalian untuk bilangan desimal. Bedanya, dalam operasi ini digunakan bilangan biner dengan aturan AND. Contoh, perkalian antara 11 dan 13 akan diproses seperti berikut :

```
        1011 (11)
        1101 (13) x
        ----------
        1011
       0000
      1011
     1011       +
    ----------
    10001111 (143)
```

Pengalian meliputi pembentukan beberapa perkalian parsial untuk setiap digit dalam multiplier. Perkalian parsial ini kemudian dijumlahkan untuk mendapatkan hasil pengalian akhir. Pengalian dua buah integer biner n-bit menghasilkan product sampai 2n-bit.

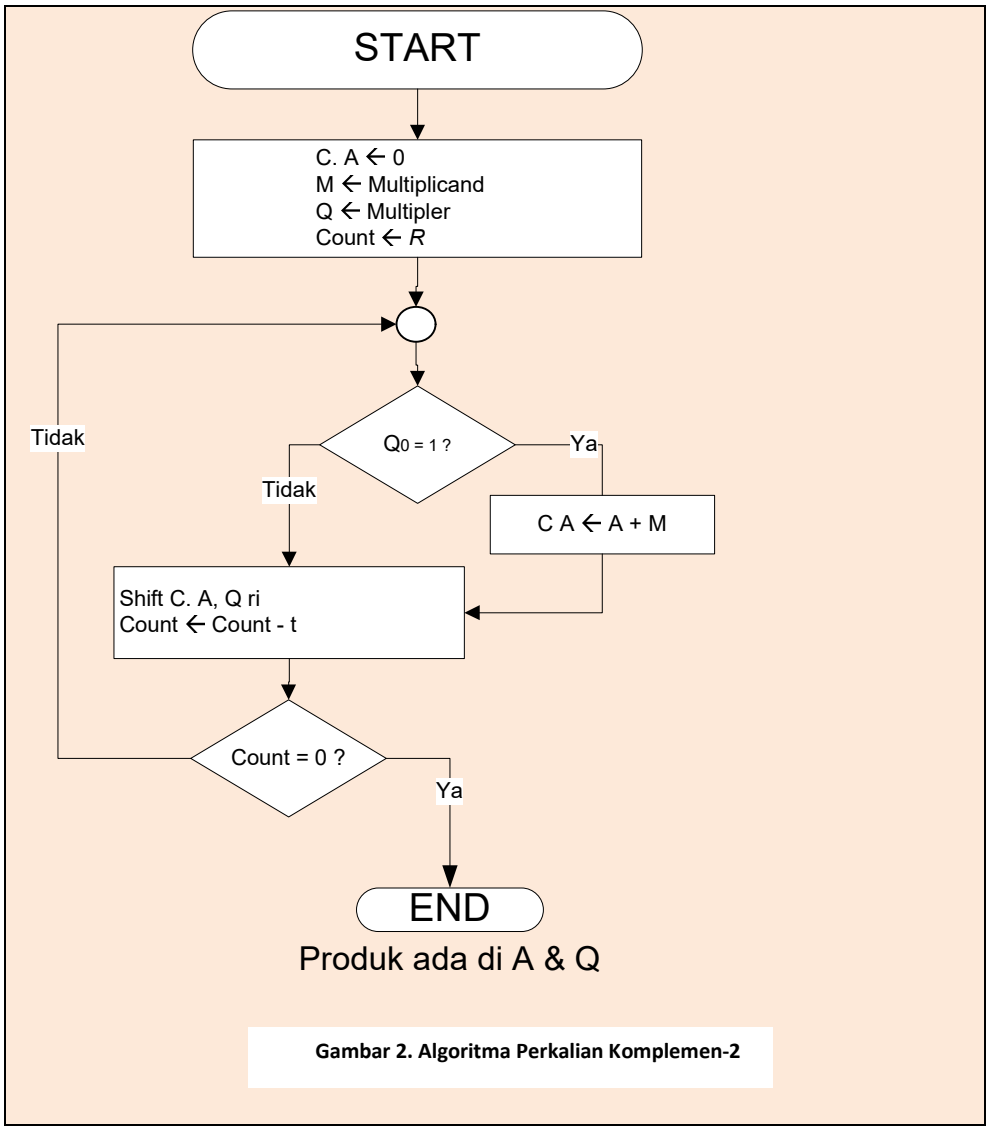
3.3.2 Perkalian Komplemen-2

Perkalian komplemen-2 sedikit lebih rumit dibandingkan dengan algoritma sebelumnya. Berikut adalah algoritma untuk perkalian yang menggunakan komplemen-2 :

1. Control Logic membaca bit-bit multiplier satu persatu.
2. Bila Q_0 = 1, multiplicand ditambahkan ke register A; hasilnya disimpan ke register A; setelah itu seluruh bit di register C, A, dan Q digeser ke kanan 1 bit.
3. Bila Q_0 = 0, tidak terjadi penambahan; seluruh bit di register C, A, dan Q digeser ke kanan 1 bit.
4. Proses tersebut dilakukan secara berulang untuk setiap bit multiplier.

5. Hasil perkalian akhir tersimpan di register A dan Q.

Secara singkat, algoritma di atas bisa digambarkan sebagai berikut



Gambar 2. Algoritma Perkalian Komplemen-2

Masuk ke implementasi, berikut adalah contoh dari operasi perkalian antara 11 dan 13 menggunakan algoritma komplemen-2 :

11 : 1011 → Multiplicand (Adder)

13 : 1101 → Multiplier (Q)

C	A	Q		
0	0000	1101	*Initial Value*	
0	1011	1101	*Hasil penjumlahan*	Siklus I
0	0101	1110	*Hasil geser kanan*	
0	0010	1111	*Hasil geser kanan*	Siklus II
0	1101	1111	*Hasil penjumlahan*	Siklus III
0	0110	1111	*Hasil geser kanan*	
1	0001	1111	*Hasil penjumlahan*	Siklus IV
0	1000	1111	*Hasil geser kanan*	

Product perkalian antara *1011* dan *1101* adalah *1000 1111*

3.3.3 Algoritma Booth

Ada algoritma lain untuk operasi perkalian, yakni algoritma Booth. Di algoritma Booth terdapat register Q (multiplier) , M (multiplicand), A (accumulator), dan register 1-bit di kanan yang ditandai dengan Q_1. Hasil perkalian disimpan di register A dan Q.

Berikut adalah algoritma Booth yang digunakan dalam menyelesaikan operasi perkalian :

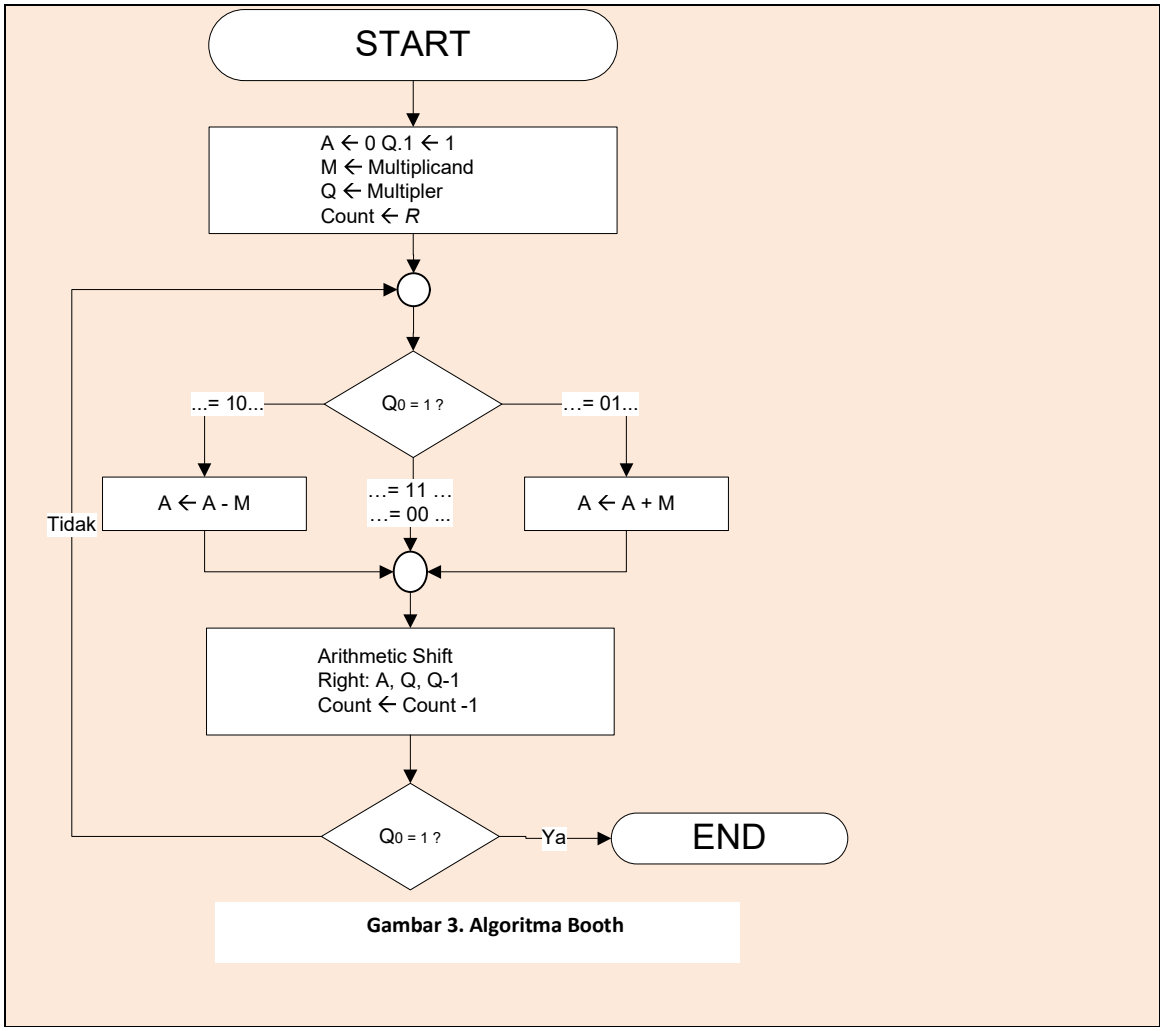
1. A dan Q_1 diinisialisasi 0.
2. Control Logic memeriksa bit-bit multiplier satu-persatu beserta bit di kanannya.
3. Jika kedua bit sama (1-1 atau 0-0), maka seluruh bit di A, Q, dan Q_1 digeser 1-bit ke kanan.

4. Jika kedua bit bernilai berbeda, ada 2 kondisi; multiplicand ditambahkan ke register A jika kedua bit bernilai (0-1); multiplicand dikurangkan ke register A jika kedua bit bernilai (1-0).

5. Proses tersebut dilakukan secara berulang untuk setiap bit multiplier.

6. Hasil perkalian akhir tersimpan di register A dan Q.

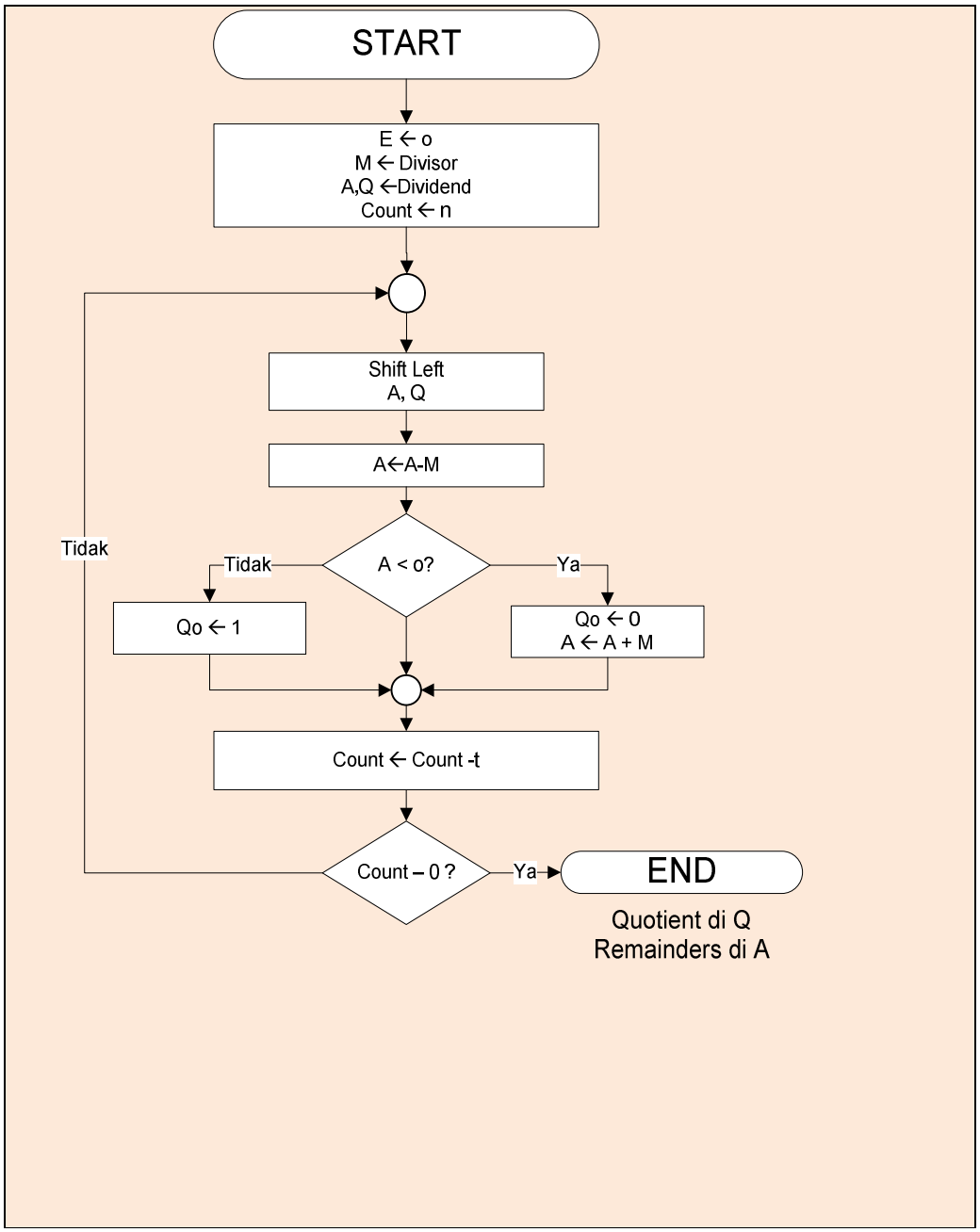
Secara singkat, algoritma di atas bisa digambarkan sebagai berikut :



Gambar 3. Algoritma Booth

3.4 Pembagian

Operasi pembagian memiliki dua operand yang diantaranya *dividend* dan *divisor*. *Divident* adalah bilangan yang dibagi, sedangkan *divisor* adalah bilangan pembagi. Flowchart berikut merupakan gambaran algoritma untuk melakukan operasi pembagian.


START
E ← o
M ← Divisor
A,Q ←Dividend
Count ← n
Shift Left
A, Q
A←A-M
A < o?
Tidak
Ya
Tidak
Qo ← 1
Qo ← 0
A ← A + M
Count ← Count -t
Count – 0 ?
Tidak
Ya
END
Quotient di Q
Remainders di A

Kalkulasi biner dalam pembagian antara 147 dengan 11 dicontohkan seperti berikut :

```
147 = 10010011 → Dividend (A,Q)
11  = 1011     → Divisor (M)
-11 = 0101 (-M)

  E    A      Q
  0   1001   0011   Start
  1   0010   0110   Geser kiri
      0101          A-M
  1   0111          Hasil A-M
  1   0111   0111   Set Q0
  0   1110   1110   Geser kiri
      0101          A-M
  1   0011          Hasil A-M
  1   0011   1111   Set Q0
  0   0111   1110   Geser kiri
      0101          A-M
  0   1100          Hasil A-M
      1011          A+M
  1   0111   1110   Hasil A+M
  0   1111   1100   Geser kiri
      0101          A-M
  1   0100          Hasil A-M
  1   0100   1101   Set Q0
```

Masuk ke implementasi, berikut adalah contoh dari operasi perkalian antara 8 dan 4 menggunakan algoritma Booth :

```
8 : 1000 (multiplicand) → -8 : 1000

4 : 0100 (multiplier) → Q

   A       Q     Q1

 0000    0100   0   Initial Value

 0000    0010   0   Hasil geser kanan    Siklus I

 0000    0001   0   Hasil geser kanan    Siklus II

 1000    0001   0   Hasil pengurangan
                                         Siklus III
 1100    0000   1   Hasil geser kanan

 0100    0000   1   Hasil penjumlahan
                                         Siklus IV
 0010    0000   0   Hasil geser kanan

Product perkalian antara 1000 dan 0100 adalah 0010 0000
```

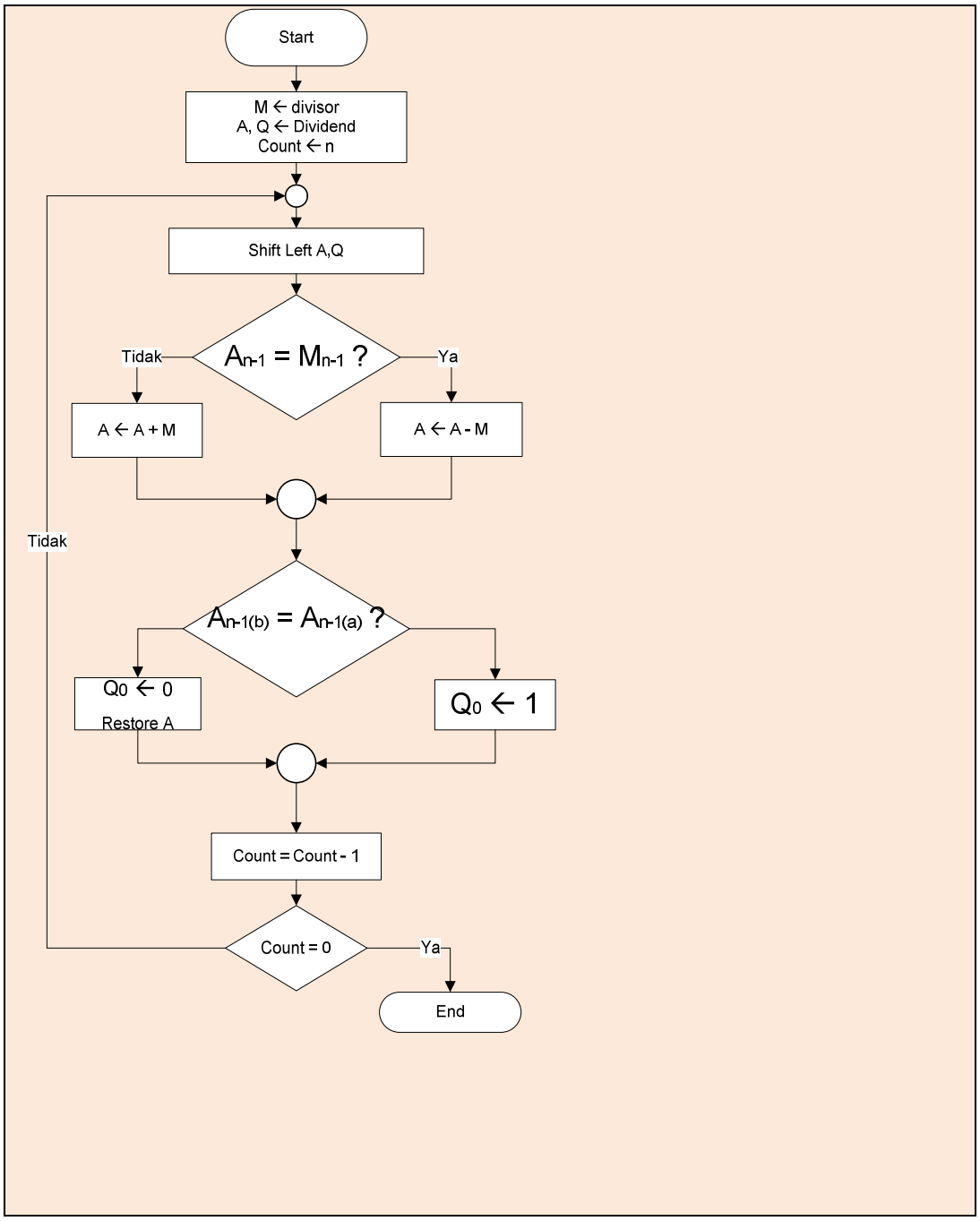
Coba perhatikan kembali contoh di atas, jika terjadi pengurangan oleh multiplicand terhadap register A, bit paling kiri hasil geser kanan 1-bit akan diisi dengan 1. Sebaliknya, jika tidak terjadi pengurangan maka bit paling kiri hasil geser kanan 1-bit akan diisi dengan biner 0.

3.4.1 Pembagian Komplemen-2

Langkah untuk melakukan pembagian dengan menggunakan metode komplemen-2 adalah sebagai berikut :

1. Muatkan divisor ke M, dividend ke A dan Q. Divident diekspresikan sebagai komplemen-2 2-nbit.
2. Geser A dan Q 1-bit ke kiri.
3. Bila M dan A memiliki tanda yang sama, lakukan A← A-M; bila tandanya berbeda, A← A+M
4. Operasi tersebut akan berhasil bila tanda A sesudah dan sebelum operasi sama
   - bila berhasil (A dan Q = 0), set $Q_0 \leftarrow 1$
   - bila gagal (A dan Q <> 0), reset Q_0 ←0 dan simpan A sebelumnya
5. Ulangi langkah 2 sampai 4 untuk setiap posisi bit di Q
6. Bila tanda divisor dan dividend sama maka quotient ada di Q, jika tidak quotient adalah komplemen-2 dari Q.
7. Remainder ada di A.

Sedangkan flowchart dari algoritma di atas adalah :


Start
M ← divisor
A, Q ← Dividend
Count ← n
Shift Left A,Q
An-1 = Mn-1 ?
Tidak
Ya
A ← A + M
A ← A - M
Tidak
An-1(b) = An-1(a) ?
Q0 ← 0
Restore A
Q0 ← 1
Count = Count - 1
Count = 0
Ya
End

Contoh :

Pembagian antara (-7) dan 3 :

A	Q	M = 0011
0000	0111	Initial Value
0000	1110	Shift
1101		Substract
0000	1110	Restore
0001	1100	Shift
1110		Substract
0001	1100	Restore
0011	1000	Shift
0000		Substract
0000	1001	Set $Q_0 = 1$
0001	0010	Shift
1110		Substract
0001	0010	Restore

Pembagian antara 7 dan (-3) :

A	Q	M = 1101
0000	0111	Initial Value
0000	1110	Shift
1101		Add
0000	1110	Restore
0001	1100	Shift
1110		Add
0001	1100	Restore
0011	1000	Shift
0000		Add
0000	1001	Set $Q_0 = 1$
0001	0010	Shift
1110		Add
0001	0010	Restore

Pembagian antara (-7)/3 dan 7/(-3) akan menghasilkan remainder yang berbeda. Hal ini disebabkan operasi pembagian didefinisikan sebagai berikut :

D = Q * V + R

Dengan

D = dividend

Q = quotient

V = divisor

R = remainder

REFERENSI

Stalling, W. COMPUTER ORGANIZATION AND ARCHITECTURE *DESIGNING FOR PERFORMANCE* EIGHTH EDITION, prentice hall 2010

PROPAGASI

A. Pertanyaan (Evaluasi mandiri)

1. Briefly explain the following representations: sign magnitude, twos complement, biased.
2. Explain how to determine if a number is negative in the following representations: sign magnitude, twos complement, biased.
3. What is the sign-extension rule for twos complement numbers?

B. QUIZ -mutiple choice (Evaluasi)