بنام خداوند جان و خرد

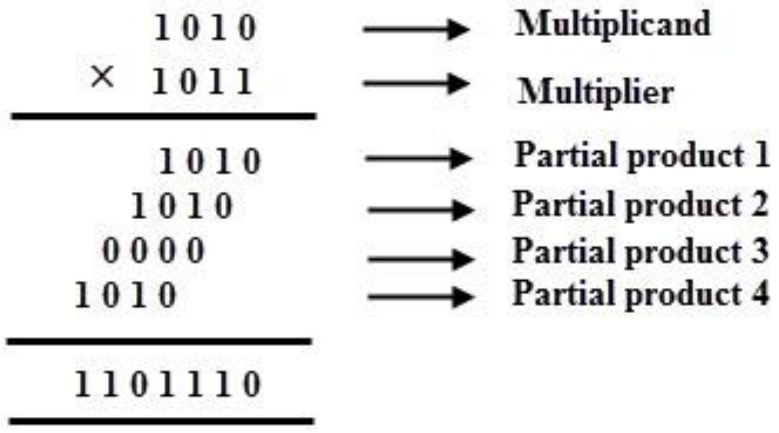
معماری کامپیوتر

جلسه پانزدهم: ضرب‌کننده-تقسیم‌کننده

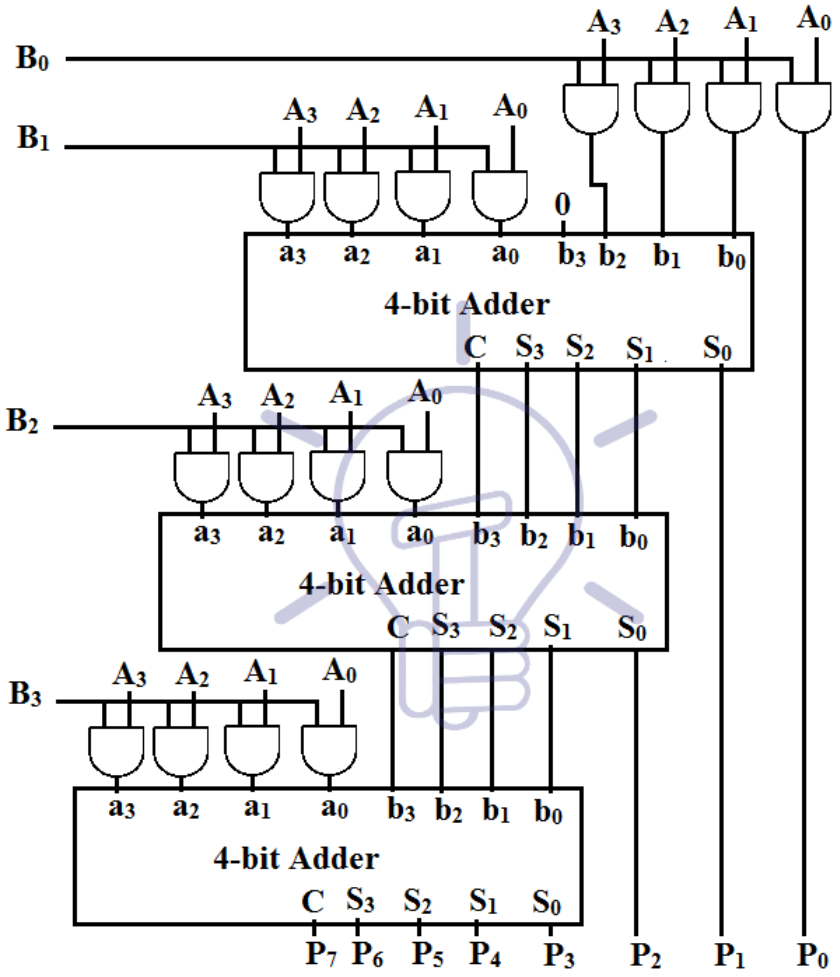
ضرب‌کننده (Multiplier)

- الگوریتم

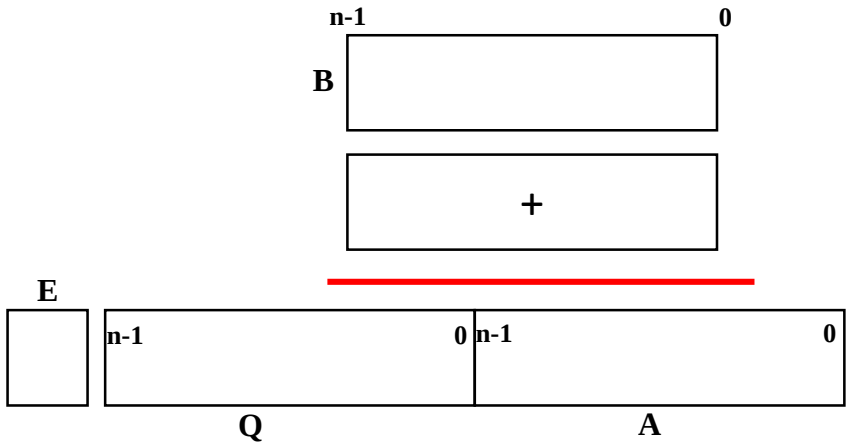
- استخراج حاصل‌ضرب‌های میانی (ضرب بیت i ام B در A) و شیفت آن‌ها به به اندازه i بیت به چپ
- ذخیره تمامی حاصل‌ضرب‌های میانی
- جمع حاصل‌ضرب‌های میانی و ذخیره حاصل در 2n بیت

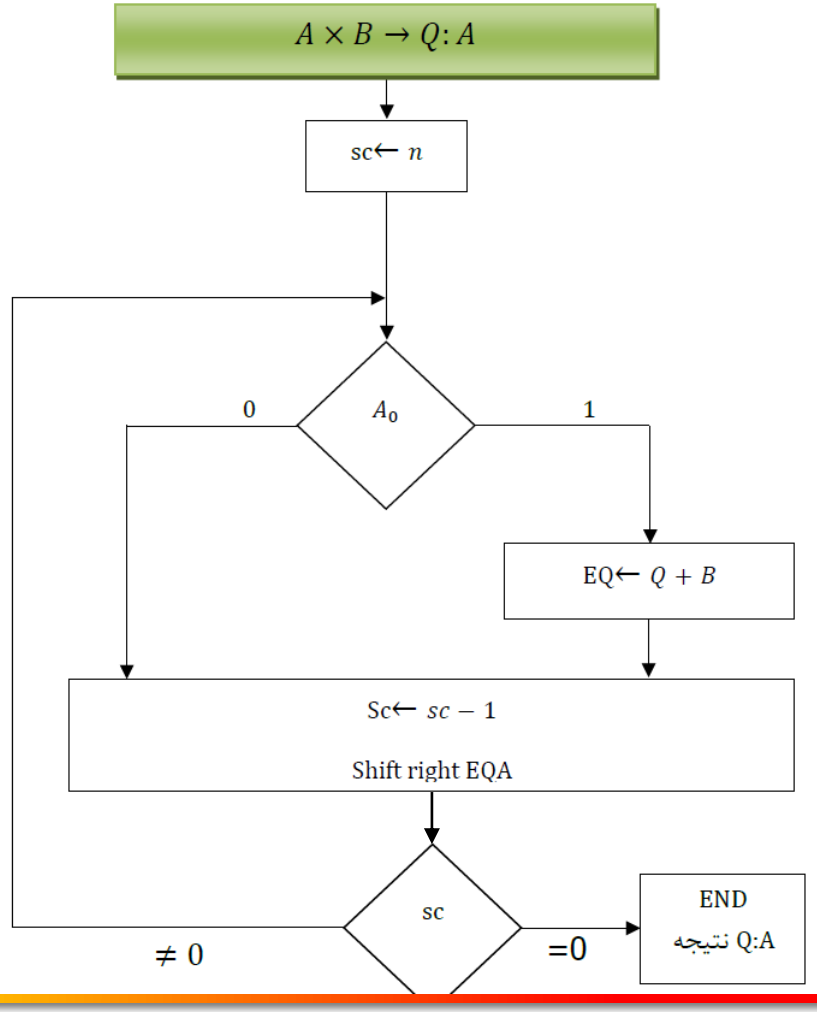


ضرب‌کننده (Multiplier)

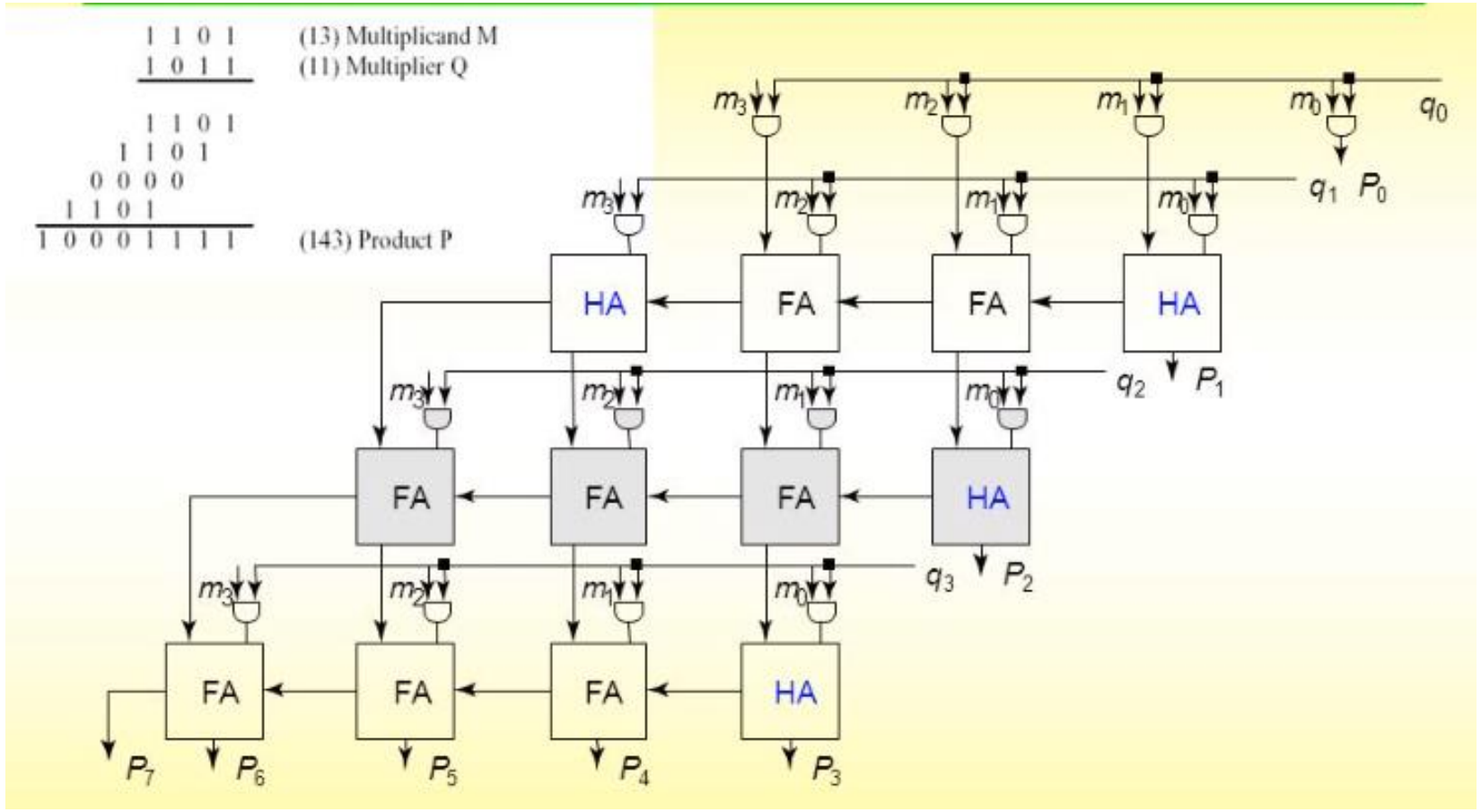


ضرب‌کننده ترتیبی





ضرب‌کننده آرایه‌ای



ضرب‌کننده بوث (Booth)

- در ضرب‌کننده آرایه‌ای
  - بسیاری از عبارات برابر صفر است و عملیات جمع اضافه باعث افزایش تاخیر می‌شود
- در ضرب‌کننده ترتیبی
  - به تعداد بیت‌های یک عدد دوم عملیات جمع انجام می‌دادیم
- مستقل کردن ضرب از تعداد یک‌ها و کاهش دادن تعداد عملیات جمع
  - معرفی ضرب‌کننده booth

ضرب‌کننده بوث (Booth)

- روش Booth
  - باهدف پوشش‌دهی عملیات ضرب برای اعداد علامت‌دار ارائه شد
    - عدد علامت‌دار در فرمت مکمل ۲
  - در این روش هدف، یافتن نقاط قطع توالی بین رشته‌های یک و صفر است (دنباله‌های ۱۰ و ۰۱)
    - در عملیات ضرب عادی، اگر در مضروب‌فیه چندین یک متوالی داشته باشیم، جمع را ادامه می‌دهیم
    - اگر چندین صفر متوالی داشته باشیم، شیفت می‌دهیم.
    - اگر از یک به صفر یا از صفر به یک برویم، تغییر روش می‌دهیم

ضرب‌کننده بوث (Booth)

- هدف روش Booth
  - عملیات ضرب برای اعداد علامت‌دار
  - کاهش دادن عملیات جمع در فرایند ضرب
  - تعریف یک سیستم کدگذاری و نمایش جهت
    - محدودسازی عملیات جمع به موارد تغییر مقدار از صفر به یک یا برعکس
    - تبدیل رشته یک‌های متوالی به صفرهای متوالی
    - در مواجهه با یک‌های متوالی هم مانند صفرهای متوالی فقط شیفت می‌دهیم و نیازی به جمع نیست

ضرب‌کننده بوث (Booth)

- کدگذاری بوث:
  - به زبان ساده‌تر هدف این کدگذاری پیاده‌سازی حالت زیر است:

$$01111 = 10000 - 1$$

  - در این حالت اگر دنباله یک‌ها از بیت k ام تا بیت mام گسترده شده‌اند
    - لازم است 2^{k+1} را از 2^m کم کنیم
    - بدین ترتیب دنباله یک‌ها به دنباله‌ای از صفرها تبدیل می‌شود
    - درنظر گرفتن رفتار یکسان و شیفت بدون جمع درحالتی که دنباله‌ای از یک‌ها یا صفرها داریم
    - مشابه آن است که برای محاسبه n*15 بگوییم (n*1)-(n*16) که با یک تفریق و شیفت بدست می‌آید

ضرب‌کننده بوث (Booth)

- در این الگوریتم سعی بر آن است که عملیات ضرب عمدتا با شیفت انجام گیرد
  - کدگذاری بوث: نوشتن اعداد به‌صورت توانی از دو و درجه‌دهی با یک یا منفی یک
    - درجه اولین یک از سمت راست عدد را ۱- قرار می‌دهیم
    - تا انتهای توالی یک ادامه داده و درجه اولین صفر را ۱ قرار می‌دهیم
    - تا انتهای توالی صفر پیش رفته و درجه اولین یک بعدی را ۱- قرار می‌دهیم
    - و ...
  - نکته: اگر پرارزش‌ترین بیت عدد، یک بود، آن را درجه‌گذاری نمی‌کنیم (علامت عدد)

ضرب‌کننده بوث (Booth)

- نمایش اعداد در کدگذاری بوث

+15: 00001111 = $2^4 - 2^0$

1 -1

-10: 11110110 = $-2^4 + 2^3 - 2^1$

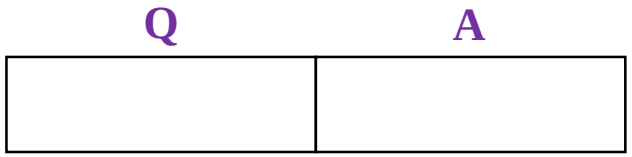
-1 1 -1

101110101 = $(2^7 - 2^4) + (2^3 - 2^2) + (2^1 - 2^0)$

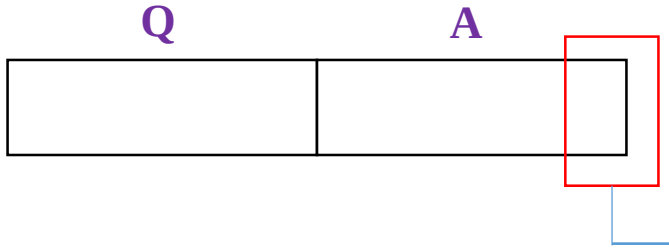
1 -1 1-11-1

پیاده‌سازی ضرب‌کننده بوث (Booth)

- در این الگوریتم تغییر مقدار مهم است
  - الگوهای 10 یا 01
- مشابه ضرب‌کننده ترتیبی با متصل کردن A و Q یک آرایه 2n بیتی می‌سازیم
  - خروجی نهایی در این آرایه ذخیره می‌شود (مشابه ضرب‌کننده ترتیبی)
  - مقداردهی اولیه Q به صفر



پیاده‌سازی ضرب‌کننده بوث (Booth)

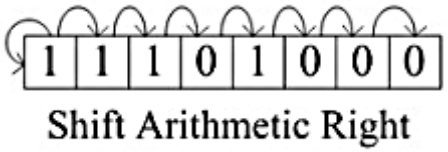


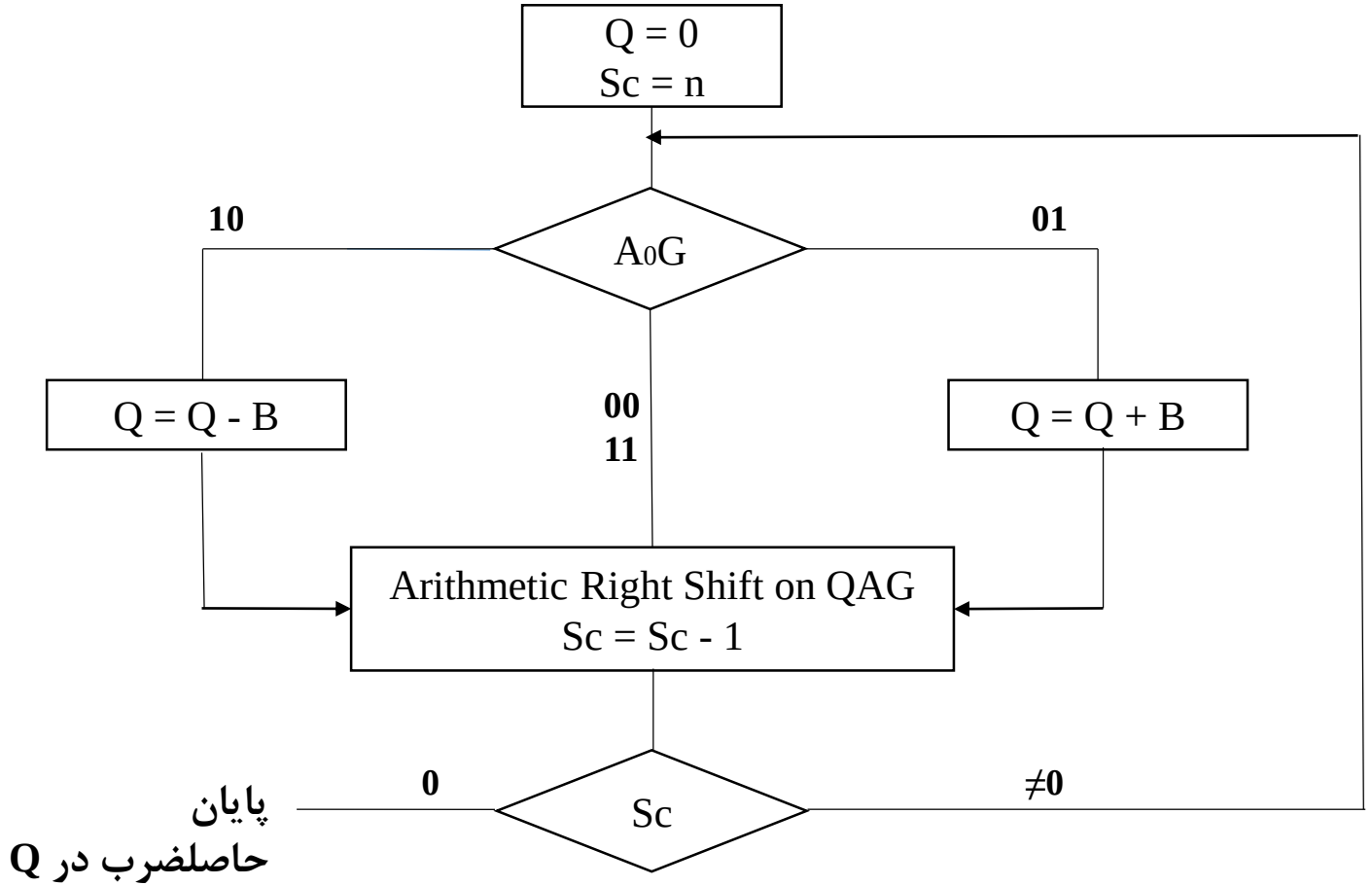
- در هر مرحله از ضرب
  - به‌دنبال تغییر مقدار از صفر به یک یا برعکس هستیم
  - دو بیت سمت راست A را چک می‌کنیم
    - **00 یا 11:** عدم تغییر مقدار پس حاصل را به سمت راست شیفت می‌دهیم
    - **10:** تغییر مقدار داریم پس عملیات تفریق چون درجه بیت پرارزش در اینجا ۱- است (کدگذاری بوث)
      - Q - B → Q.A
    - **01:** تغییر مقدار داریم پس عملیات جمع چون درجه بیت پرارزش در اینجا ۱+ است (کدگذاری بوث)
      - Q + B → Q.A

پیاده‌سازی ضرب‌کننده بوث (Booth)

- بدترین حالت در ضرب بوث:
  - رشته عدد بصورت یک در میان صفر و یک باشد (...1010 یا ...0101)
  - مجبوریم همواره عملیات جمع انجام دهیم (از درجه n)
- برای پیاده‌سازی ضرب بوث
  - یک بیت در سمت راست آرایه Q.A قرار می‌دهیم (G)
    - در هر مرحله A_0G را با چک کرده و تصمیم می‌گیریم

الگوریتم ضرب‌کننده بوث (Booth)





الگوریتم ضرب‌کننده بوث (Booth)

مثال: دو عدد روبرو را توسط الگوریتم booth در یکدیگر ضرب کنید.

A = 010101
B = 001110

حل:

QBG = 0000000011100 ,

Sc = 6 → B_0G = 00 →ASR→ QBG = 0000000001110

Sc = 5 → B_0G = 10 → Q = Q - A = 000000 + 101011 = 101011

QBG = 1010110001110 → ASR→ 1101011000111

Sc = 4 → B_0G = 11 → ASR→ Q = 1110101100011

Sc = 3 → B_0G = 11→ ASR→ Q = 1111010110001

Sc = 2 → B_0G = 01 → Q = Q + A = 111101 + 010101 = 010010

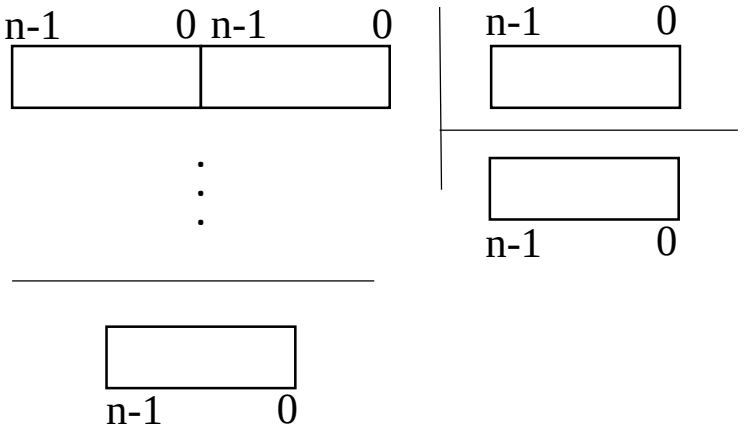
QBG = 0100100110001→ ASR→ 0010010011000

Sc = 1 → B_0G = 00 → ASR→ QBG = 0001001001100

Sc = 0 → Finish , QA = 000100100110 ≈ 294

تقسیم‌کننده (Divider)

- حاصل تقسیم یک عدد 2n بیتی بر عددی n بیتی به n بیت فضا نیاز دارد
- روش تقسیم:
  - از مقسوم n بیت جدا می‌کنیم اگر بزرگتر از مقسوم‌علیه بود
    - یک در خارج قسمت می‌گذاریم
  - درغیراینصورت
    - صفر گذاشته و روال را برای n+1 بیت تکرار می‌کنیم



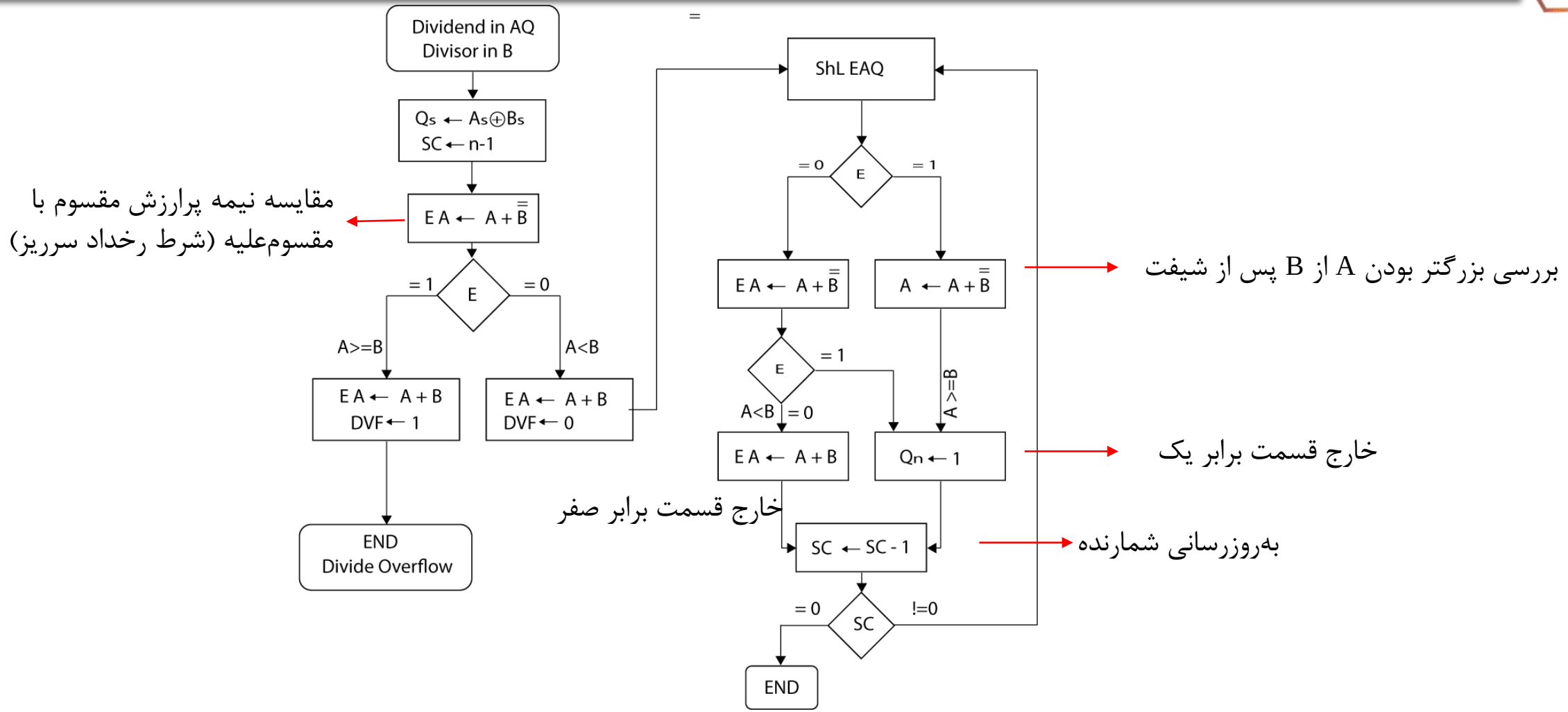
تقسیم‌کننده (Divider)

- گاهی بر حسب شرایط n بیت فضا برای خارج قسمت تقسیم کم است
  - مانند تقسیم 11…11 بر 000…1
  - شرایط سرریز (overflow) در تقسیم
    - مقسوم‌علیه برابر صفر باشد که حاصل برابر بی‌نهایت شده و در n بیت نمی‌گنجد
    - اگر n بیت پرارزش مقسوم از مقسوم‌علیه بزرگتر باشد، خارج قسمت در n بیت جا نشده و سرریز رخ می‌دهد
      - مانند مثال بالا
  - شرایط یک و دو سرریز در تقسیم را می‌توان در حالت کلی مورد دوم خلاصه نمود
  - سرریز باید مدیریت شود و در پیاده‌سازی سخت‌افزاری لحاظ گردد

الگوریتم تقسیم‌کننده (Divider)

- الگوریتم برای دو عدد صحیح بدون علامت
  - مقسوم (A.Q)، مقسوم‌علیه (B)، خارج قسمت (A)، باقی‌مانده (Q)
  - چک کردن رخداد سرریز:
    - اگر A>B: سرریز رخ می‌دهد قادر به انجام تقسیم نیستیم
    - اگر A<B: رقم اول خارج قسمت صفر است و یک بیت از Q را به A اضافه می‌کنیم
      - در خارج قسمت 1 قرار داده و B را از A کم می‌کنیم
      - عملیات را تا انتها براساس شمارنده‌ای که برابر n تنظیم شده ادامه می‌دهیم

الگوریتم تقسیم‌کننده (Divider)



الگوریتم تقسیم‌کننده (Divider)

مثال: دو عدد روبرو را توسط الگوریتم divider بر یکدیگر تقسیم کنید.

X = 10011000
Y = 1100

حل:

Sc = 4 : AQ = 10011000, B = 1100

EA = A+B'+1 = 1001 + 0011+ 1 = 01101

E = 0 → Overflow = 0 → A = A + B = 1101 + 1100 = 1001

EAQ = 010011000 →SL → EAQ = 100110000

E=1 → A = EA+B'+1 = 10011 + 0011 + 1 = 0111 → Q0 = 1

Sc = 3: AQ = 01110001, B = 1100

EA = A+B'+1 = 0111 + 0011 + 1 = 01011

E = 0 → Overflow = 0 → A = A + B = 1011 + 1100 = 0111

EAQ = 001110001 → SL → EAQ = 011100010

E= 0 → EA = A+B'+1 = 1110 + 0011 + 1 = 10010 →E = 1 → Q0 = 1

الگوریتم تقسیم‌کننده (Divider)

Sc = 2: AQ = 00100011, B = 1100

EA = A+B’+1 = 0010 + 0011 + 1 = 00110

E =0 → Overflow =0 → A = A + B = 0110 + 1100 = 0010

EAQ = 000100011→ SL → 001000110

E = 0 → EA = A+B’+1 = 0100 + 0011 + 1 = 01000→E=0 → Q0 = 0

EA = A+ B = 1000+ 1100 = 10100

Sc =1: AQ = 01000110, B = 1100

EA = A+B’+1 = 0100 + 0011 + 1 = 01000

E = 0 → Overflow = 0 → A = A + B = 1000 + 1100 = 0100

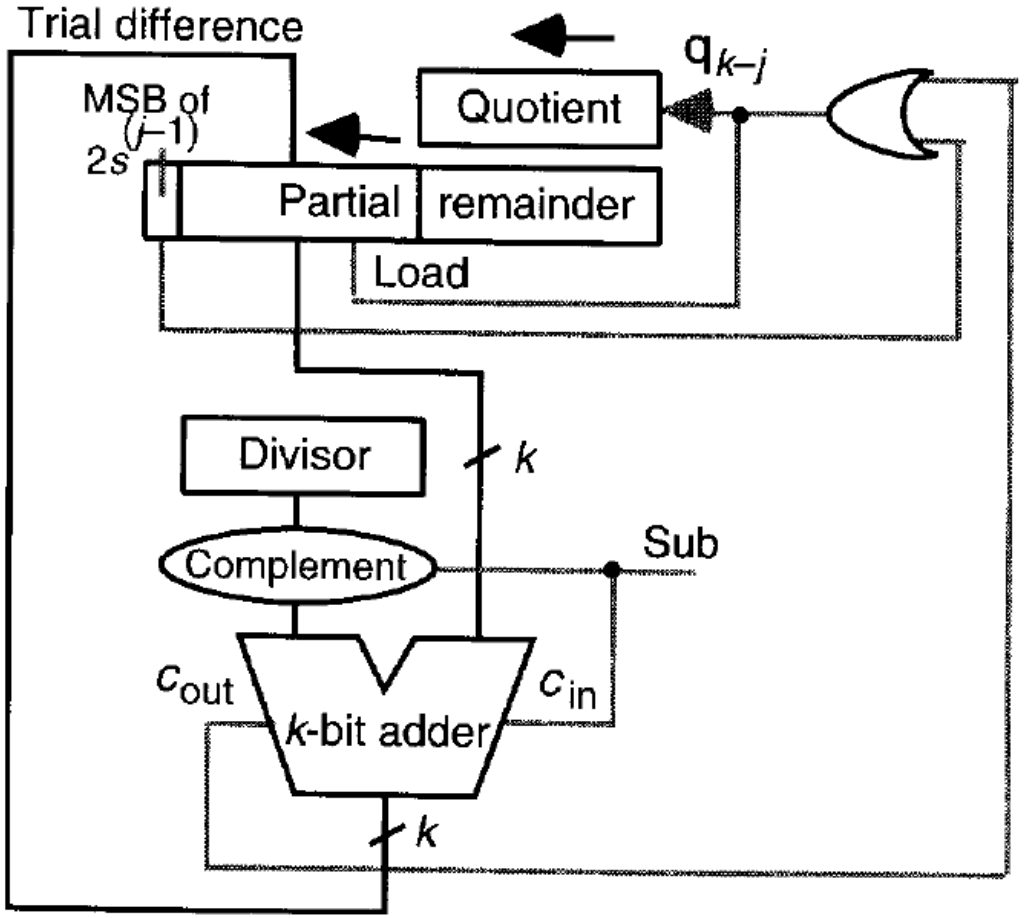
EAQ = 001000110 → SL → 010001100

E = 0 → EA = A+B’+1 = 1000+ 0011 + 1 = 01100 →E = 0 → Q0 = 0

EA = A + B = 1100 + 1100 = 11000

Sc = 0 → Finish , Q = 1100 , A = 1000

سخت‌افزار تقسیم‌کننده (Divider)



تقسیم‌کننده (Divider) علامت‌دار

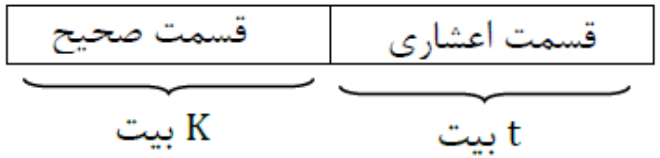
- تا اینجا فرض کردیم دو عدد مثبت را بر یکدیگر تقسیم می‌کنیم
- اگر اعداد از نوع صحیح و علامت دار باشند:
  - ابتدا مشخص می‌کنیم شیوه نمایش چگونه است (مکمل ۲ یا اندازه علامت)
  - اگر اعداد مثبت باشند که همان روال قبلی را عینا تکرار می‌کنیم
  - اگر یکی یا هردو اعداد منفی بود
    - معادل مثبت آن را به‌دست آورده و عملیات تقسیم را مشابه حالت گفته شده انجام می‌دهیم
    - در انتها تعیین علامت کرده و مقادیر را به‌روزرسانی می‌کنیم

محاسبات برای اعداد اعشاری

- برای ذخیره اعداد اعشاری دو دیدگاه وجود دارد
  - اعداد اعشاری ممیز ثابت (Fixed Point Numbers)
    - پیاده‌سازی ساده
    - استفاده غیربهینه از فضای ذخیره‌سازی
  - اعداد اعشاری ممیز شناور (Floating Point Numbers)
    - پیاده‌سازی پیچیده
    - استفاده بهینه و منعطف از فضای ذخیره‌سازی

محاسبات برای اعداد اعشاری

- ممیز ثابت:
  - تعداد بیت‌های تخصیص داده شده به قسمت صحیح و اعشاری ثابت است
    - بخش صحیح به‌صورت مکمل ۲
    - بخش اعشاری به‌صورت بدون علامت
    - منجر به خطا در محاسبات
    - استفاده ناکارامد از فضای ذخیره‌سازی
      - عدم توانایی ذخیره‌سازی برخی اعداد
      - خالی ماندن فضا در بیشتر موارد



01101100

0110.1100

$2^2 + 2^1 + 2^{-1} + 2^{-2} = 6.75$