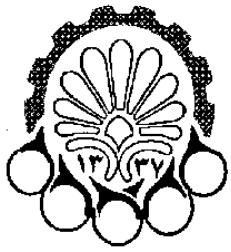


دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر و فناوری اطلاعات

جزوه درس

معماری کامپیوتر

Computer Organization & Design

نسخه ۵.۱

دکتر زرندی

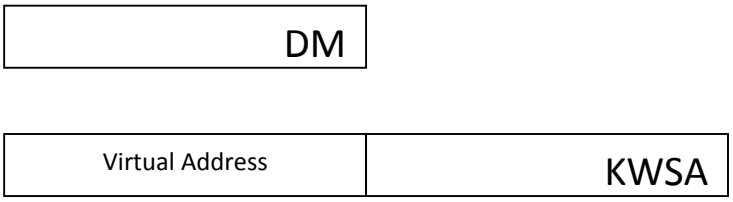
ترم اول سال ۱۳۸۹

فهرست

ضرب کننده ترتیبی:

لذا ایده ضرب کننده ترتیبی را مطرح می‌کنیم:

$$A \times B \rightarrow Q{:}A$$



۱) در این روش هدف، کم کردن پهنای جمع کننده هاست تا 2n بیتی نباشند.

۲) مقدار اولیه دو ثبات Q:Aدر الگوریتمی که در ادامه خواهد آمد، صفر است. (علامت: بین A,Q به این معناست که این دو ثبات به هم وصل یا به عبارتی concat شده‌اند و ۲n بیت جواب را تشکیل داده اند). و مقدار آن را هر بار با حاصلضرب‌های میانی جمع می‌کنیم. حاصلضرب نهایی در Q:A موجود خواهد بود.

۳) هر حا صل ضرب میانی صفر ا ست یا B. (در اینجا داریم B*A را ح ساب میکنیم) بنابراین در هر مرحله Q:A را یا با صفر جمع می‌زنیم یا با عدد B. خود عدد B هم n بیتی است؛ بنابراین یک جمع کننده n بیتی کافی است. چون هر کدام از این حاصلضرب‌های میانی بخواهد جمع شود در مرحله iام iتا شیفت به سمت چپ خورده است و اصلا با تعدادی از بیت‌ها در Q:A برای جمع در آن مرحله کاری نداریم. بنابراین لزومی ندارد که یک جمع کننده ۲n بیتی به کار ببریم.

به عبارت دیگر در هر مرحله باید اندیس i را نگاه کنیم و حاصلضرب میانی را که 0 یاB است به اندازه i تا به سمت چپ شیفت بدهیم و به ازای i+1 امین بیت A به بعد، عمل جمع را انجام دهیم. که در اینصورت به جمع کننده با پهنای بیتی بیش از n نیاز خواهیم داشت. بنابراین به جای چنین کاری، حاصلضرب میانی را ثابت نگاه می‌داریم و به جای شیفت دادن آنها به سمت چپ، Q:A را به سمت راست شیفت می‌دهیم. چون در مرحله i ام به i بیت سمت راست A که قبلا ازآنها برای تعیین حاصل ضربهای میانی استفاده کرده بودیم، نیازی نداریم. (به i بیت سمت را ست A و n-i بیت سمت چپ Q در مرحله iام نیازی نداریم. بنابراین پهنای بیتی جمع کننده به nکاهش می‌یابد.)

با توجه به توضیحات فوق الگوریتم ضرب کننده ترتیبی به صورت زیر خواهد بود:

n→ sc

به A_0 نگاه کن:

اگر $A_0 = 0$؛ هیچ کاری انجام نده!

اگر $A_0 = 1: B + Q \rightarrow EQ$

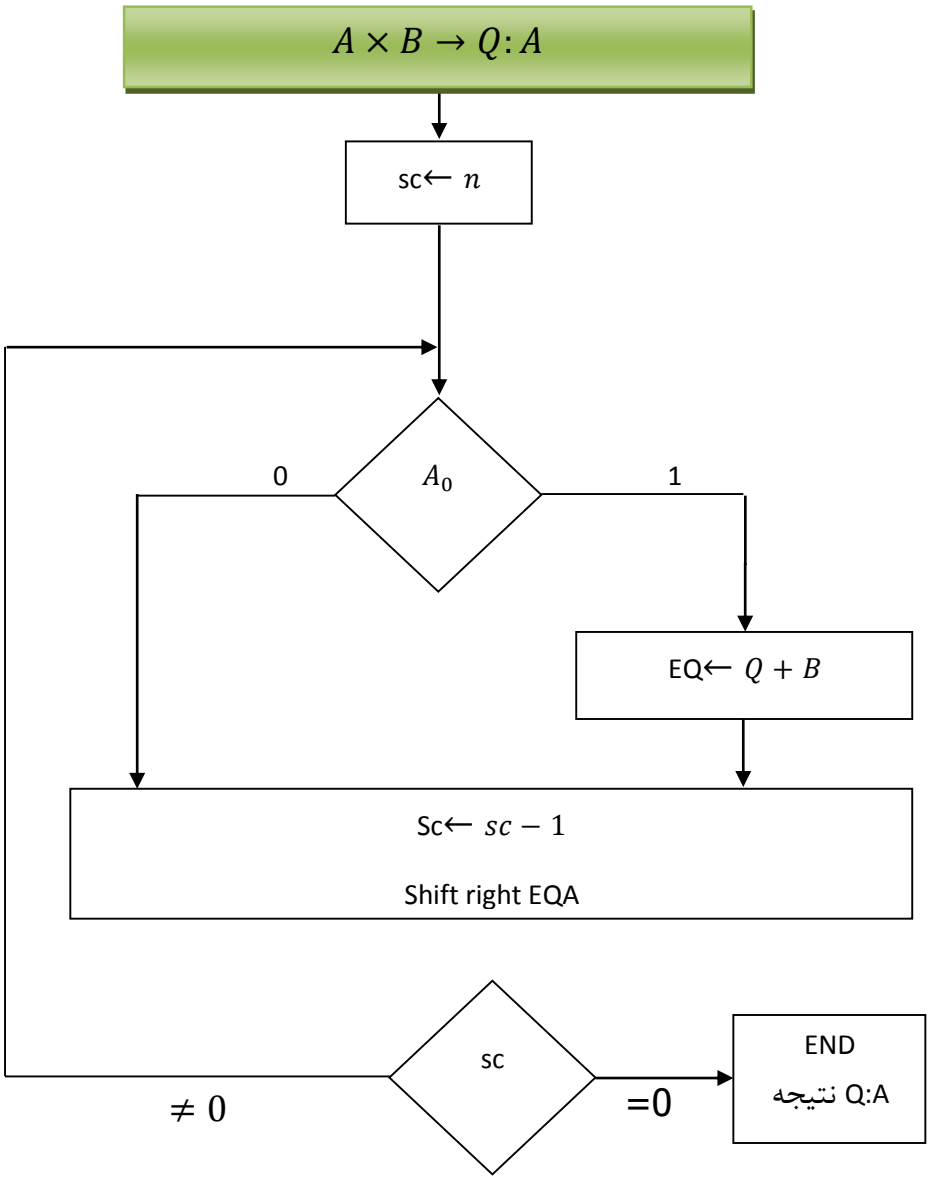
۱) E:Q:A را یک واحد به سمت راست شیفت بده!

۲) $\mathrm{sc} - 1 \rightarrow sc$

۳) اگر sc=0؛ پایان؛ در غیر این صورت برو به مرحله ۲.

در این روش بدبینانه ترین حالت این است که هر n بیت A، یک باشد که در این صورت مدام باید عمل جمع را انجام بدهیم و تاخیر زیاد می‌شود.

خوش بینانه ترین حالت وقتی است که A تماما صفر باشد؛ در این صورت فقط n تا کلاک لازم است.

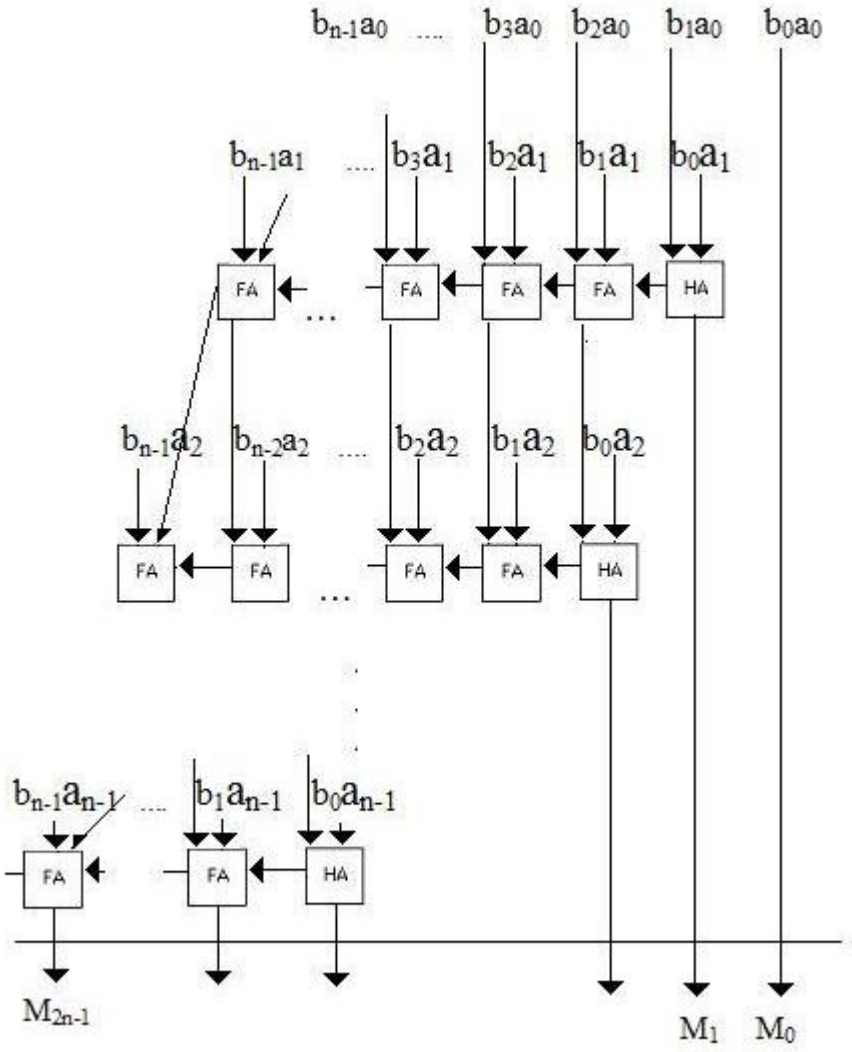


روش فوق یک روش خیلی ساده ا ست که حداقل سخت افزار را م صرف می‌کند؛ چون فقط یک جمع کننده n بیتی و یک فلیپ فلاپ(E) و سه تا ثبات(Q,A,B) می‌خواهد.

اما از نظر زمانی چون این روش یه شـمارنده میخواهد به n تا کلاک نیاز داریم و طول کلاک را هم سـخت افزار جمع کننده می‌سازد

ضرب کننده آرایه ای

واحد ALU ذاتا ترکیبی ا ست، پس ضرب را می‌توان ترکیبی هم پیاده سازی کرد که برای این مقصود از ضرب کننده آرایه‌ای استفاده می‌شود. برای ضرب دو عدد n بیتی a و b مانند شکل زیر عمل می‌کنیم:

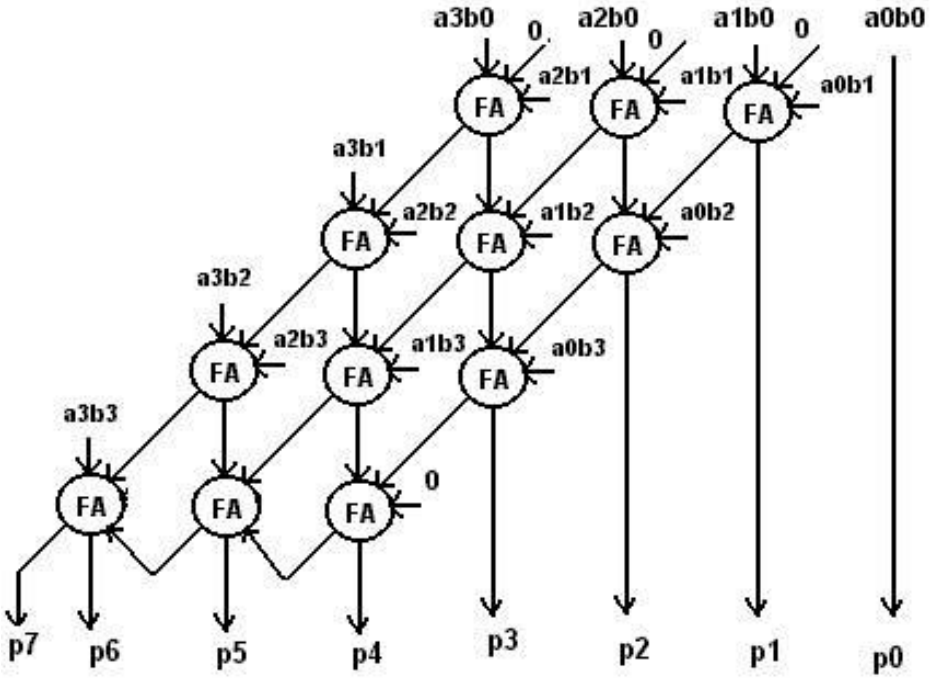


توضیح شکل:

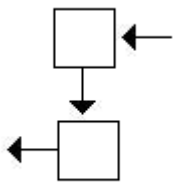
ردیف x ام شـامل a_xb_j $(n \le j \le 0)$ها اسـت. عدد هر ردیف را با عدد هم سـتون ردیف پایینش جمع می‌کنیم و مجموع (sum) را به ردیف بعدی می‌دهیم تا چنانچه ردیف دیگری در پایین موجود بود با این حاصـل جمع، جمع گردد و الا خود sum، M_j یعنی رقم j ام حاصل ضرب خواهد بود. همچنین در این جمع رقم نقلی (Carry out/Cout) ایجاد شـده را به سـمت چپ یعنی عدد هم ردیف در سـتون بعد می‌دهیم تا به عنوان عدد نقلی

ورودی(Carry in/Cin) ا ستفاده گردد. اگرتنها یک ردیف دیگر در ستونی که می‌خواهیم عدد جاری موجود در آن را جمع کنیم وجود داشـــت به جای تـمام جمع کـنـنده(Full Adder/FA) از نیم جمع کـنـنده Half) (Adder/HA استفاده می‌کنیم.

در اینجا ذکر این نکته ضـروری اسـت که در ضـرب کننده آرایه‌ای لزوما عدد نقلی به عدد هم ردیف در سـتون بعدی داده نمی شود و گاهی به ردیف پایین تر در ستون بعد و. .. داده می شود. در هر حال نکته مهم این ا ست که عدد نقلی ایجاد شده در هر ستون به نحوی باید در یکی از جمع‌های ستون بعد شرکت کند. نمونه دیگری از ضرب کننده آرایه‌ای مربوط به ضرب دو عدد ۴ بیتی را می‌توانید در شکل زیر مشاهده کنید:

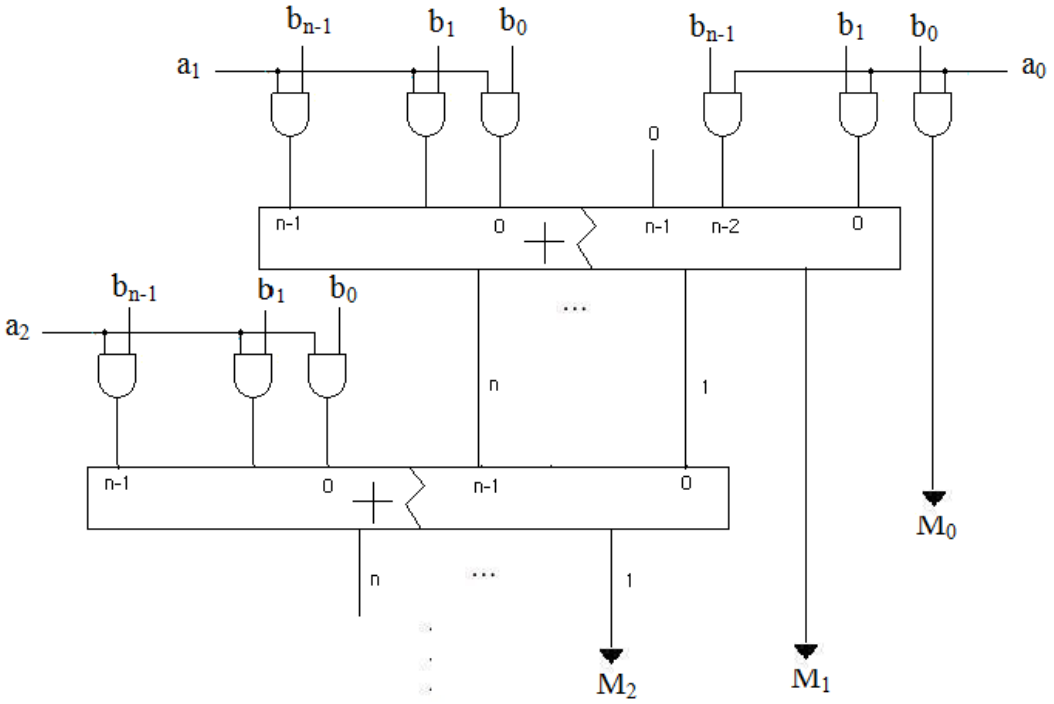


واضح است که در مدار ضرب کننده آرایه‌ای، سخت افزار زیادی مصرف می‌شود یعنی به تعداد n-1، Cascade Adder. برای محا سبه تاخیر باید توجه دا شت که برای تولید M_{2n-1} باید Carryها هم به صورت افقی و هم به صورت عمودی (منظور sumهای تولید شده در هر FA است) حرکت کنند، یعنی:



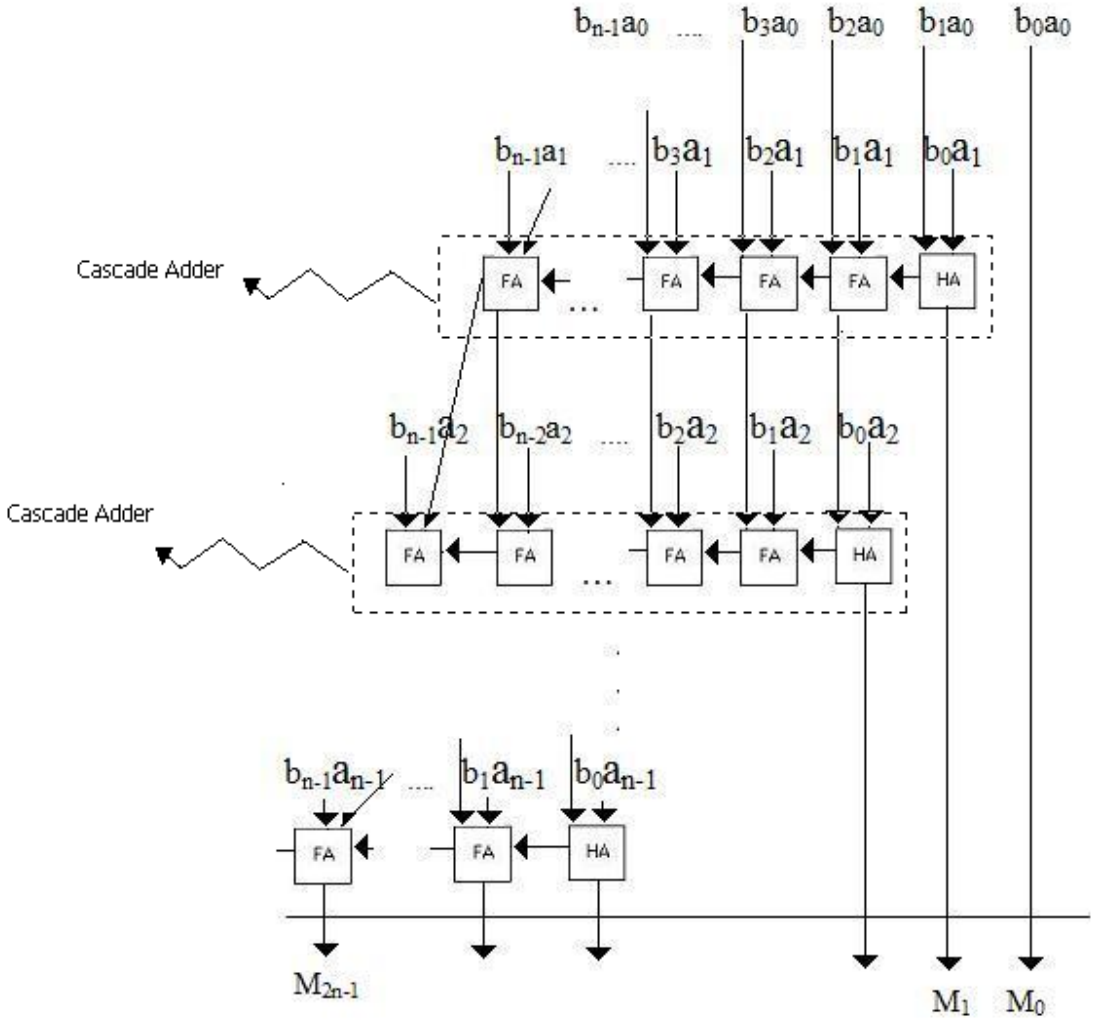
که تاخیر از مرتبه nd است در حالی که ضرب کننده ترتیبی(sequential) تاخیری از مرتبه n^2d داشت. (دقت کنیم که از آنجا که a_ib_jها موازی ایجاد می‌شوند پس تاخیر در تولید M_{2n-1} دقیقا برابر با 3nd+d است.)

نوع دیگری از ضرب کننده‌های آرایه ای:



در این نوع ضرب کننده آرایه‌ای هر بار M_k که راست ترین رقم حاصل ضرب است که تا کنون محاسبه نشده را برمی گردانیم. در این جا نیاز به n تا جمع کننده n بیتی داریم.

در واقع اگر جمع کننده‌ها را Cascade Adder انتخاب کنیم همان ضــرب کننده آرایه‌ای قبلی را به ما می‌دهد، اما استفاده از ضرب کننده‌های دیگر نظیر CLA نوع متفاوتی از ضرب کننده آرایه‌ای را ایجاد می‌کند. برای روشن شدن موضوع به شکل زیر توجه کنید:



باید دقت داشت که ضرب کننده آرایه‌ای که در بالا مورد بحث قرار گرفت تنها برای اعداد مثبت استفاده می‌شود.

ضرب کننده بوث(Booth Algorithm/Multiplier)

در ضـــرب کننده ترتیبی، به تعداد ۱های موجود در مضـــروب، باید عملیات جمع روی مضـــروب فیه را انجام می‌دادیم. برای بهبود این الگوریتم و مستقل کردن ضرب از تعداد این ۱ها شخصی به نام بوث الگوریتم بوث را ارائه کرد که در این الگوریتم تعداد عملیات جمع و یا تفریق روی مضروب فیه تنها به تعداد ۰۱ و ۱۰هایی که در مضروب ظاهر می‌شوند بستگی دارد.

قبل از ذکر این الگوریتم ابتدا باید نمایش دیگری از اعداد را فرا بگیریم.

هر عدد در مبنای ۲ را می‌توان به صورت زیر در نظر گرفت:

...000011...111100.... . 000111.... 111...000

یعنی دنباله از ۱های متوالی و ۰های متوالی.

عدد x را در نظر بگیرید که :

عدد فوق در مبنای ۱۰ برابر است با $2^m - 2^l$

چرا که آن را می‌توان به صورت زیر نوشت:

$$\begin{array}{rl} & \dots 00100 \dots 0000 \dots \\ - & \dots 00000 \dots 0010 \dots \\ \hline x = & \cdots 00011 \dots 1110 \dots \end{array}$$

حال از آنجاییکه می‌توان هر عدد صحیح A را(به طور منحصر به فرد) به صورت مجموعه‌ای از ۱ها و ۰های متوالی یعنی به شکل حاصل جمع x_iها در نظر گرفت (x_iها ساختاری همانند x در بالا دارند یعنی تمام ۱ هایشان پشت سر هم آمده است) پس هر عدد صحیح را می‌توان به فرم $\sum 2^m - \sum 2^l$ نوشت.

با مثال‌هایی موضوع را روشن می‌کنیم:

مثال۱: $+5 = (00000101)_2 = 2^1\text{-}2^0+2^3\text{-}2^2$

مثال۲: $\text{-}10 = (11110110)_2 = 2^3 \text{-}2^1\text{-}2^4$

دقت شود که در مثال بالا رقم آخر یعنی سمت چپ ترین رقم ۱ است و تبدیل ۱ به ۰ رخ نمی‌دهد بنابرین نوشتن $+2^9$ درست نیست همان طور که در محاسبه نیز در نظر گرفته نشده است.

مثال۳: $(11000111)_2 = 2^3\text{-}2^0\text{-}2^6$

همان طور که پیشتر نیز گفته شد این روش تنها به مجموعه ۱های پشت سر هم وابسته است. در این روش به جای اینکه a_0 را نگاه کند دو بیت به دو بیت نگاه می‌کند و لبه‌های بالا رونده و پایین رونده مهمند. این دو رقم ۴ حالت دارد:

هیچ کاری نکن $\Rightarrow$ ۰۰

عمل جمع را انجام بده ⇒ ۰۱

عمل جمع را انجام بده ⇒ ۱۰

هیچ کاری نکن ⇒ ۱۱

در نوشتن الگوریتم بوث از متغیرهای ۱ بیتی E و G و متغیر n بیتی Q استفاده می‌کنیم، که E، carry را شامل می‌شود و G نیز حاوی رقم اول از بین جفت ارقامی است که می‌خواهند بررسی شوند(A_0G بیان کننده یکی از ۴ حالت بالا است.). هدف ما ضرب دو عدد n رقمی A و B است ودر پایان عملیات ضرب Q خارج قسمت و A نیز باقی مانده خواهند بود.

الگوریتم بوث:

۱) $SC \leftarrow n$

۲) A_0G را نگاه کن اگر مساوی بود با:

۰۰: هیچ کاری نکن

۰۱: $EQA \leftarrow Q:A + B$

۱۰: $EQA \leftarrow Q:A - B$

۱۱: هیچ کاری نکن

۳) Shift Right (EQAG)

۴) $SC \leftarrow SC - 1$

۵) اگر $SC = 0$ آنگاه پایان؛ وگرنه برو به مرحله ۲

در این الگوریتم بهترین حالت این است که همه ارقام ۱۱ یا ۰۰ باشند. بدترین حالات نیز زمانی رخ می‌دهند که ارقام به طور متناوب از ۰ به ۱ و از ۱ به ۰ تغییر کنند:

۱) 1010.... 1010

در این حالت به تعداد $\frac{n}{2} - 1$ عمل جمع و $\frac{n}{2}$ عمل تفریق در الگوریتم بوث نیاز است.

۲) 0101.... 0101

در این حالت $\frac{n}{2}$ عمل جمع و $\frac{n}{2}$ عمل تفریق در الگوریتم بوث نیاز است.

تقسیم کننده:

گفتیم که اگر دو عدد n بیتی A,Bرا در هم ضـرب کنیم، حاصـل در حالت ماکزیمم در 2n بیت جا می‌گیرد. در این قسمت می‌خواهیم تقسیم دو عدد را بررسی کنیم. چنانچه عدد 2nبیتی F را بر عدد nبیتی Bتقسیم نماییم حا صل باید درn بیت جا شود؛ البته لزوما تقسیم هر عدد 2n بیتی بر هر عدد nبیتی در n بیت جا نمی شود. به عنوان مثال عدد 2n بیتی 1...11 را در نظر بگیرید. اگر این عدد را بر عدد n بیتی 1...00 تق سیم نماییم وا ضح است که حاصل در n بیت نمی‌گنجد.

بنابراین ممکن اسـت در برخی مواقع با سـرریز مواجه شـویم و سـخت افزار نمی‌تواند محاسـبات را انجام دهد و عددی را که از طول استاندارد تجاوز می‌کند را نگه دارد. این حالات عبارتند از:

۱- اگر مقسوم علیه صفر باشد، حاصل بی نهایت می‌شود که نمی‌توان آن را در n بیت نمایش داد و سرریز رخ می‌دهد.

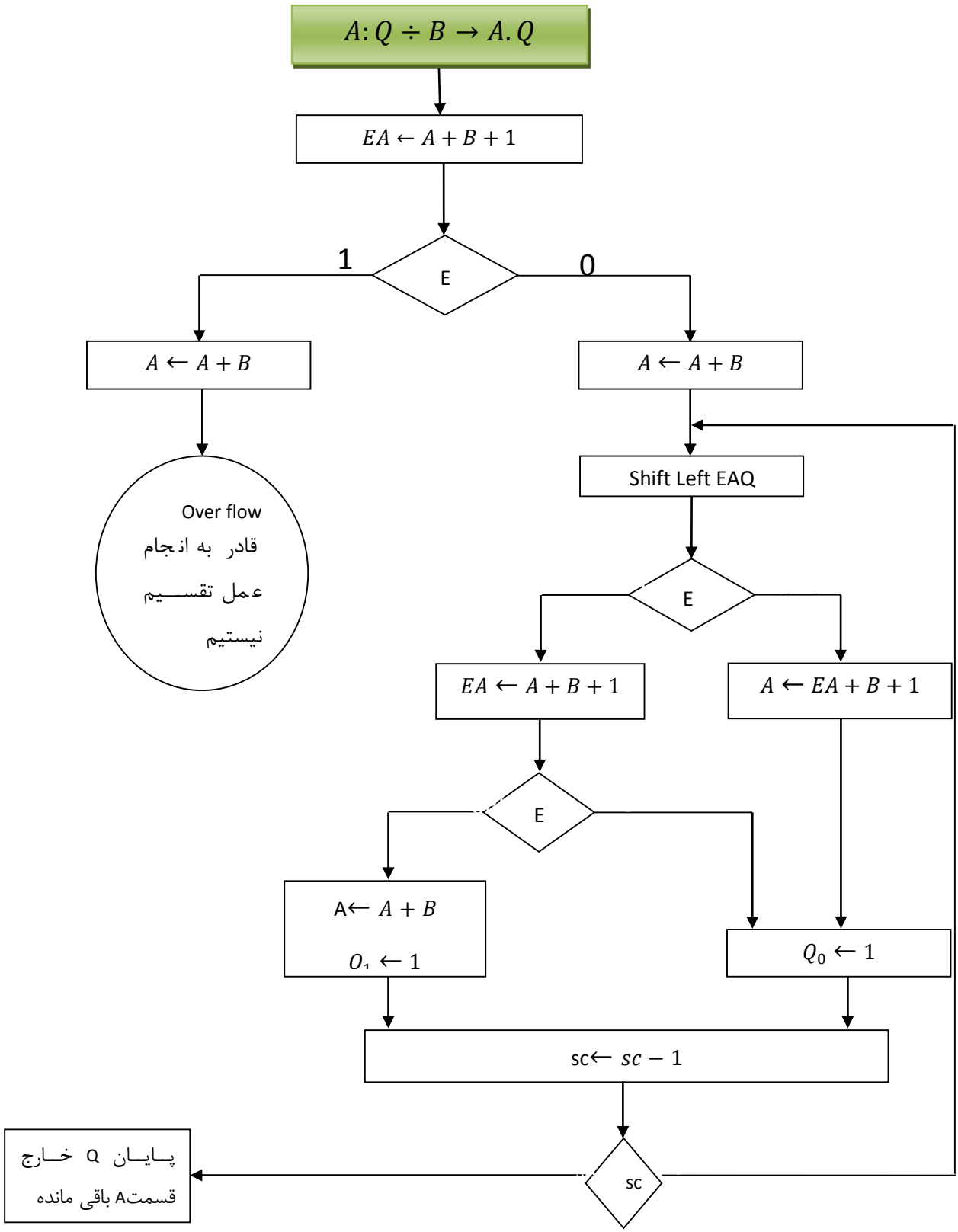
۲- اگر خارج قسمت در n بیت جا نشود، نیز سرریز رخ می‌دهد. در واقع چنانچه عدد ۲n بیتی $F = F_H : F_1$ بر عدد n بیتی Bتقسیم شود، اگر $F_H \geq B$، آنگاه خارج قسمت در nبیت جا نمی‌شود.

اگر کمی دقت کنید متوجه می‌شـوید که حالت یک، در واقع زیر مجموعه‌ای از حالت ۲ اسـت. بنابراین سـاختن سخت افزار حالت ۲ کافی است.

حالت سرریز معمولا با یک شدن فلیپ فلاپ خاصی که به آن فلیپ فلاپ سرریز گوییم مشخص می‌شود.

معمولا فرضمان این است که اعداد مثبت هستند.

فلوچارت الگوریتم تقسیم به صورت زیر است:


$A: Q \div B \rightarrow A.Q$
$EA \leftarrow A + B + 1$
E
1
0
$A \leftarrow A + B$
$A \leftarrow A + B$
Over flow
قادر به انجام عمل تقسیم نیستیم
Shift Left EAQ
E
$EA \leftarrow A + B + 1$
$A \leftarrow EA + B + 1$
E
A$\leftarrow A + B$
$Q_1 \leftarrow 1$
$Q_0 \leftarrow 1$
sc$\leftarrow sc - 1$
sc
پایان Q خارج
قسمتA باقی مانده

در این روش از ۴ ثبات، یک فلیپ فلاپ و یک جمع کننده استفاده کردیم. یک ثبات برای نگهداری مقسوم علیه و دو ثبات برای نگهداری مقسوم.

در این روش ابتدا باید چک کنیم که در تقسیم با سرریز مواجه خواهیم شدیا نه؟ به این منظور مقسوم علیه(B) را از بیت‌های نیمه باارزش تر مقسوم (A)کم میکنیم و در EA قرار می‌دهیم. چنانچه در این عمل carry مساوی یک باشد، همانگونه که در جلسات قبلی ثابت شد A ازB بزرگتر بوده است؛ لذا با حالتی مواجه هستیم که سرریز رخ میدهد و باید فلیپ فلاپ ســرریز را یک کنیم. همچنین بیتهای نیمه پرارزش مقســوم را که در هنگام تفریق خراب نموده ایم باید به حالت قبل برگردانیم. لذا Bرا با A جمع می‌کنیم که A قبلی حا صل شود و آن را در A میریزیم.

اما چنانچه carry حاصل صفر باشد، یعنی تقسیم با سرریز مواجه نمی‌شود. باز هم A را به حالت قبلی خود برمی گردانیم و از آنجایی که مشــخص شــد که در بار اول n بیت ســمت چپ مقســوم از عددn بیتی مقســوم علیه کوچکتر بوده، پس رقم اول خارج ق سمت صفر ا ست. لذا EAQ را یک بیت به سمت چپ شیفت میدهیم. حالا یک بیت از Q آزاد می‌شود که می‌توانیم در مرحله بعد برای نگهداری خارج قسمت از آن استفاده نماییم.

ما پیش‌فرض، بیت خارج قسمت در هر مرحله را برابر با یک می‌گیریم. چنانچه نادرست بود، آن را به صفر تغییر می‌دهیم. برای تشخیص درستی یا نادرستی این فرض مقسوم علیه را از A کم می‌کنیم. طبق همان قضیه‌ای که قبلا گفتیم جنانچه carry حا صل یک با شد، یعنی Aاز مق سوم علیه بزرگتر بوده، لذا خارج ق سمت یک در ست بوده ا ست و بیت صفرام Q را یک میکنیم. همچنین باقیمانده جزئی در این مرحله هم همان حا صل تفریق B از A ا ست که در Aگذا شته شده ا ست. اما چنانچه carry حا صل، صفر با شد، به این معنی ا ست که A از مق سوم علیه کوچکتر بوده ا ست. وا ضح ا ست که در این حالت، خارج ق سمت در این مرحله صفر ا ست. پس علاوه براین که باید بیت صفرام Q را صفر کنیم، باید Aرا هم به همان حالت قبل از عمل تفریق برگردانیم. چون وقتی خارج قسمت صفر باشد، باقیمانده جزیی برابر با با همان A خواهد بود.

حال از شمارنده یکی کم می‌کنیم. چنانچه شمارنده به صفر ر سید عملیات تقسیم به پایان ر سیده ا ست؛ در غیر این صورت برای ادامه عمل تقسیم به مرحله شیفت دادن به چپ بازمی گردیم و عملیات را ادامه می‌دهیم.
در پایان Qخارج قسمت و A باقی مانده نهایی است.

گفتیم در این الگوریتم فرض می‌شــود که یک عدد مثبت 2n بیتی به یک عدد مثبت n بیتی تقســیم می‌شــود. حالا اگر شیوه نمایش عدد اندازه- علامت یا مکمل دو با شد ابتدا باید تکلیف آن را م شخص نمود. به این صورت

که اگر اعداد مثبت باشـــند که همین روند را انجام می‌دهیم. اما اگر یک یا هردو اعداد منفی باشـــند ابتدا معادل مثبت آن را به دســـت می‌آوریم؛ ســـپس عمل تقســـیم را برای آن دو انجام می‌دهیم و در نهایت تعیین علامت می‌کنیم.

شیوه نمایش	جمع و تفریق	ضرب و تقسیم
مکمل دو	ساده	پیچیده
اندازه-علامت	پیچیده	ساده