



دانشکده فناوری اطلاعات و کامپیوتر

معماری کامپیوتر

دکتر حمید حسن پور

## هووالدظیو

### فهرست

## ۲- آشنایی با سطوح طراحی و ادوات مورد استفاده در رجیسترها

۱-۲ : سطوح طراحی در کامپیوتر

۲-۲ : مالتی پلکسر (Multi plexer)

۳-۲ : کدگشا (Decoder)

۴-۲ : کدگذار (encoder)

۵-۲ : آرایه منطقی (Logic Array)

۶-۲ : گذرگاه (BUS)

۱- BUS اختصاصی (dedicated Bus)

۲- BUS مشترک (Common BUS)

## ۳- طراحی پردازنده

۱-۳ : وظیفه اصلی CPU

۲-۳ : وظیفه فرعی Cpu

۳-۳ : ارتباط CPU با I/O و حافظه اصلی

Memory Mapped I/O : ۱-۳-۳

I/O Mapped I/O : ۲-۳-۳

۴-۳ : ماشین وان نیومن (Von Neuman)

۳-۴-۱ : دستورالعملهای ماشین وان نیومن

۳-۴-۲: روشهای گسترش ماشین وان نیومن

۳-۵: پردازش موازی (Parallel Processing)

## فصل دوم- طراحی واحد محاسبه و منطق

### ۱- نحوه نمایش داده در کامپیوتر

۱-۱: اعداد بی سی دی (BCD)

۱-۲: اعداد افزایش سه تایی (EX - 3)

۱-۳: کد گری (Gray)

۱-۴: کد گری - افزایش سه تایی (EX-3 Gray)

۱-۵: اعداد ممیز شناور (Floating Point)

### ۲- مدارات جمع کننده

۲-۱: طراحی جمع کننده کامل (Full Adder)

۲-۲: طراحی جمع کننده سریال (Serial Adder)

۲-۳: طراحی جمع کننده با رقم نقلی پله‌ای (Ripple Carry Adder)

۲-۴: طراحی جمع کننده با پیش بینی رقم نقلی (Carry Lookahead Adder)

### ۳- مدارات ضرب کننده

۳-۱: ضرب کننده برای دو عدد قدر مطلق علامت

۲-۳ : ضرب دو عدد متمم 2 یا 2's

۳-۳ : روشهای سریع ضرب

۱-۳-۳ : روش ضرب Booth

۲-۳-۳ : ضرب به کمک مدارات ترکیبی (ضرب آرایه‌ای)

#### ۴- مدارات تقسیم کننده

۱-۴ : تقسیم به روش مقایسه‌ای

۲-۴ : تقسیم به روش Restoring

۳-۴ : تقسیم به روش Non Restoring

#### ۵- اعمال اصلی در اعداد ممیز شناور

۱-۵ : عمل جمع در اعداد ممیز شناور

۲-۵ : عمل ضرب در عدد ممیز شناور

۳-۵ : مدار جمع و تفریق دو عدد ممیز شناور

### فصل سوم - طراحی واحد کنترل و حافظه

#### ۱- طراحی واحد کنترل

۱-۱ : روش سیم بندی شده

State Table Method : ۱-۱-۱

Delay Element Method : ۲-۱-۱

Sequence Counter Method : ۳-۱-۱

۲-۱ : روش ریز برنامه سازی

۳-۱: طراحی واحد کنترل به روش ریزبرنامه سازی (Micro Programmed)

## ۲- سازمان حافظه (Memory Organization)

۱-۲: جنس حافظه

۱-۱-۲: حافظه نیمه رسانا (Semiconductor)

۲-۱-۲: حافظه حلقه های مغناطیسی (Magnetic Core)

۲-۲: سلسله مراتب ذخیره سازی اطلاعات

۳-۲: انواع Cache

۲-۳-۱ Direct Mapping

۲-۳-۲ Set – Associative

۲-۳-۳ Associative mapping

۴-۲: حافظه انجمنی

## ۳- روشهای متفاوت انتقال اطلاعات بین کامپیوتر و دستگاههای جانبی

۱-۳: برنامه ریزی ورودی / خروجی (programed I/O)

۲-۳: وقفه (Intrupt)

۳-۳: دسترسی مستقیم به حافظه ((Direct Memory Access) DMA

## ۲- آشنایی با سطوح طراحی و ادوات مورد استفاده در رجیسترها

### ۱-۲ : سطوح طراحی در کامپیوتر

همانطور که می دانید طراحی در کامپیوتر در سه سطح انجام می شود.

(۱) سطح گیت (Gate Level)

(۲) سطح ثبات (Register Level)

(۳) سطح پردازنده (Processor Level)

به عنوان مثال طراحی یک ساعت دیجیتال را در نظر می گیریم .

### ۱-۱-۲-۱-۲ در سطح گیت

یک مدار مجتمع ( IC ) را در نظر بگیرید که زمان را می شمارد و به صورت Seven Segment نمایش می دهد .

IC timer

(شکل ۱-۲)

### ۱-۲-۲-۲ در سطح ثبات

در اینجا به کمک گیت‌های موجود طراحی را انجام می‌دهیم و ثبات‌هایی که کارهای بخصوصی را انجام می‌دهند با یکدیگر ارتباط دارند .

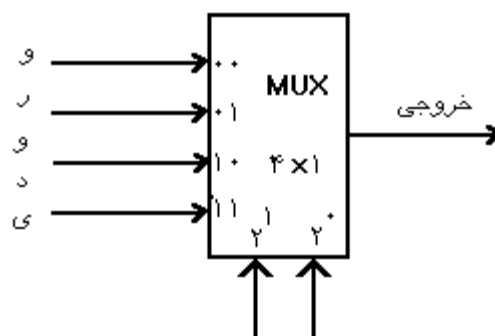
### ۲-۱-۳- در سطح پردازنده

در این سطح واحد پردازنده سیستم طراحی می‌شود، که این بحث مربوط به درس ریزپردازنده‌ها می‌باشد . در درس معماری طراحی رجیسترها مورد بحث است و بیشتر مطالب مربوط به طراحی پیاده‌سازی این قسمت از کامپیوتر است.

### ۲-۲: مالتی پلکسر (Multi plexer)

مالتی پلکسر یک مدار ترکیبی است که به کمک آن می‌توان اطلاعات واقع در چند مسیر را به یک مسیر هدایت نمود . اگر در مالتی پلکسرها  $2^n$  مسیر ورودی وجود داشته باشد ؛  $n$  خط انتخاب برای این مسیرها وجود دارد که به کمک آن می‌توان اطلاعات موجود در یک مسیر ورودی را به خروجی منتقل نمود.

در مالتی پلکسر  $MUX(m \times n)$ ، تعداد مسیرهای ورودی  $n$  و تعداد بیت‌های یک مسیر می‌باشد . در شکل زیر یک مالتی پلکسر  $4 \times 1$  را داریم .



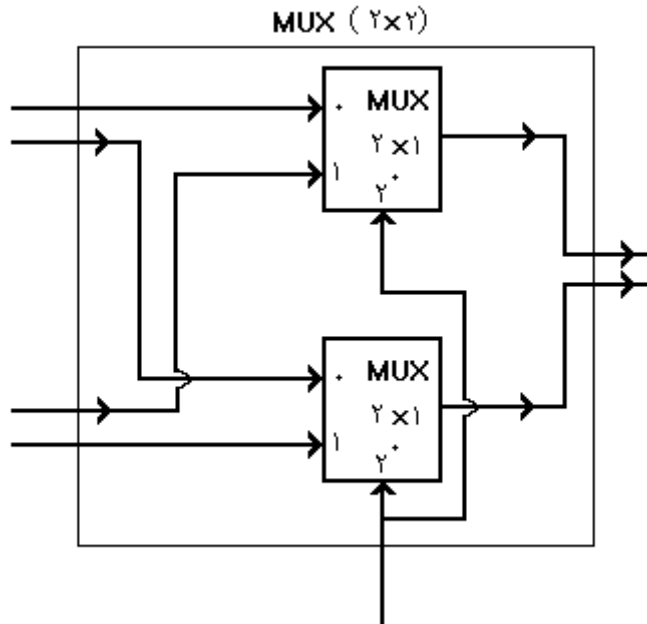
(شکل ۲-۲)

**نکته:** مسیر هر چند بیتی باشد خط انتخاب همیشه یک بیتی است و باید همواره ارزش آن معلوم شود. خروجی نیز یک مسیر می‌باشد که ممکن است هر چند بیتی باشد.

روشهایی وجود دارد که بتوان مالتی پلکسرها را به گونه‌ای به هم مرتبط نمود تا تعداد مسیرهای ورودی و یا تعداد بیتهای خروجی را افزایش داد.

افزایش تعداد ورودی را می توان براساس اتصال درختی مالتی پلکسرها و عمل افزایش تعداد بیتهای خروجی را می توان با اتصال موازی مالتی پلکسرها انجام داد.

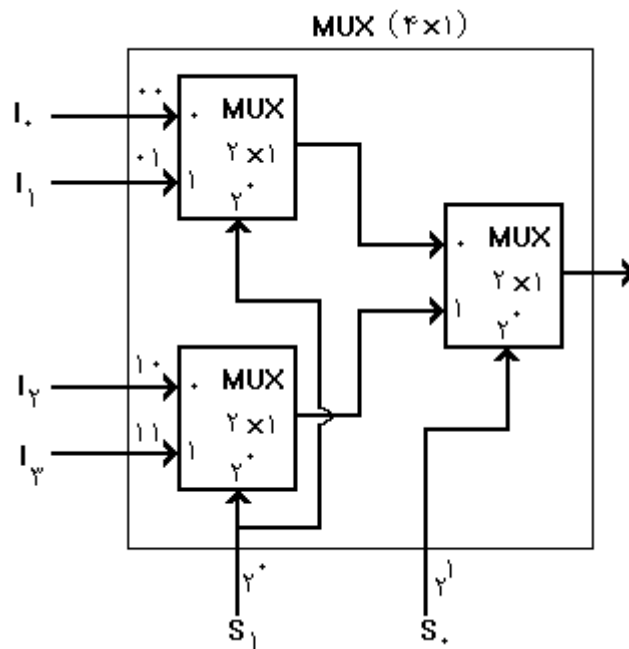
مثال : با استفاده از  $Mux (2 \times 1)$  ،  $Mux (2 \times 2)$  را طراحی کنید .



(شکل ۲-۳)

مثال : می خواهیم با استفاده از  $Mux (2 \times 1)$  ، یک  $Mux (4 \times 1)$  بسازیم .





(شکل ۲-۴)

### ۲-۲-۱- کاربردهای مالتی پلکسر

۱- برای هدایت اطلاعات از بین چندین منبع ورودی، به یک مقصد خروجی مشترک .

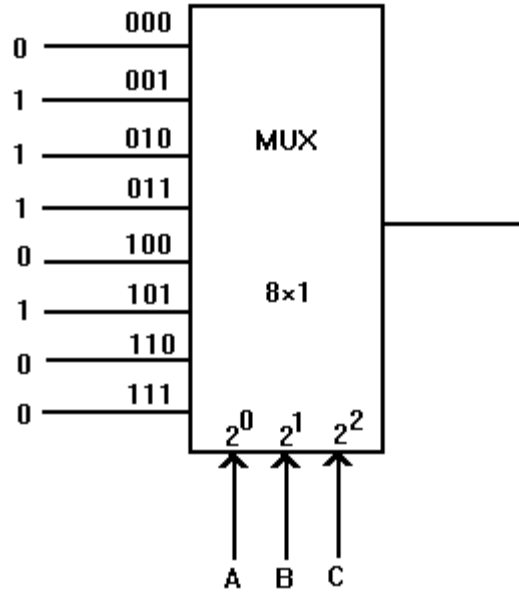
۲- برای پیاده سازی توابع بولی و ترکیبی.

۳- برای طراحی سیستم باس یا گذرگاه (BUS).

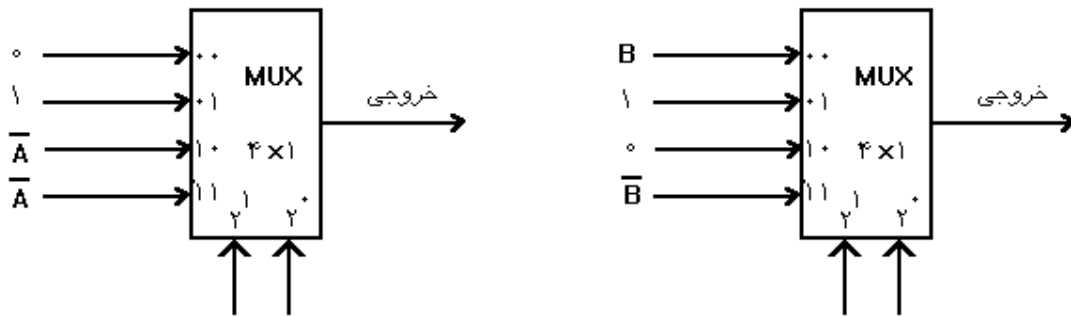
### ۲-۲-۱- کاربرد Mux در طراحی توابع

هر تابع  $n$  متغیره را می توان به کمک  $MUX(2^n \times 1)$  پیاده سازی نمود. در این پیاده سازی متغیرهای تابع به خطوط انتخاب متصل شده و بسته به آنکه این تابع حاوی چه میترمهایی باشد در ورودی برای شماره های مربوطه یک قرار می دهیم.

به عنوان مثال : تابع  $F(A,B,C) = \Sigma (1,2,3,5)$  را به کمک Mux  $(8 \times 1)$  پیاده سازی نمایید.  
 سپس همین تابع را با استفاده از  $(4 \times 1)$  Mux طراحی کنید.



(شکل ۲-۵)



| A \ BC    | $\bar{B}\bar{C}$ | $\bar{B}C$ | $B\bar{C}$ | $BC$ |
|-----------|------------------|------------|------------|------|
| A         | ۴                | ⑤          | ۶          | ۷    |
| $\bar{A}$ | ۰                | ①          | ②          | ③    |

| B \ AC    | $\bar{A}\bar{C}$ | $\bar{A}C$ | $A\bar{C}$ | $AC$ |
|-----------|------------------|------------|------------|------|
| B         | ②                | ③          | ۶          | ۷    |
| $\bar{B}$ | ۰                | ①          | ۴          | ⑤    |

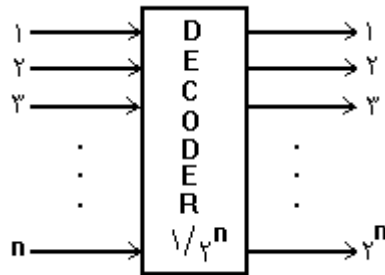
(شکل ۲-۶)

۲-۳: کدگشا (Decoder)

مداری ترکیبی است با  $n$  ورودی و  $2^n$  خروجی که با اعمال یک کد به ورودی آن تنها

یک

خط از خروجی فعال می شود.



(شکل ۲-۷)

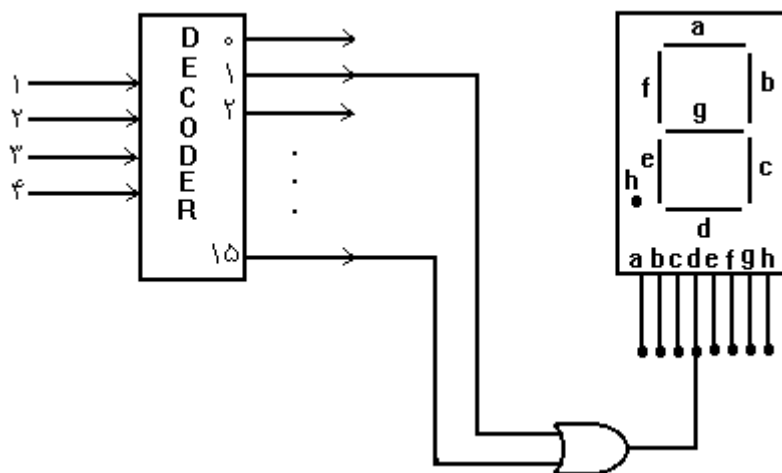
۲-۳-۱- کاربردهای کد گشا

۱- نمایش اعداد بی سی دی بر روی Seven Segment

۲- آدرس دهی خطوط حافظه

۳- پیاده سازی توابع ترکیبی

کاربرد اول - نمایش اعداد بی سی دی بر روی Seven Segment "7seg"

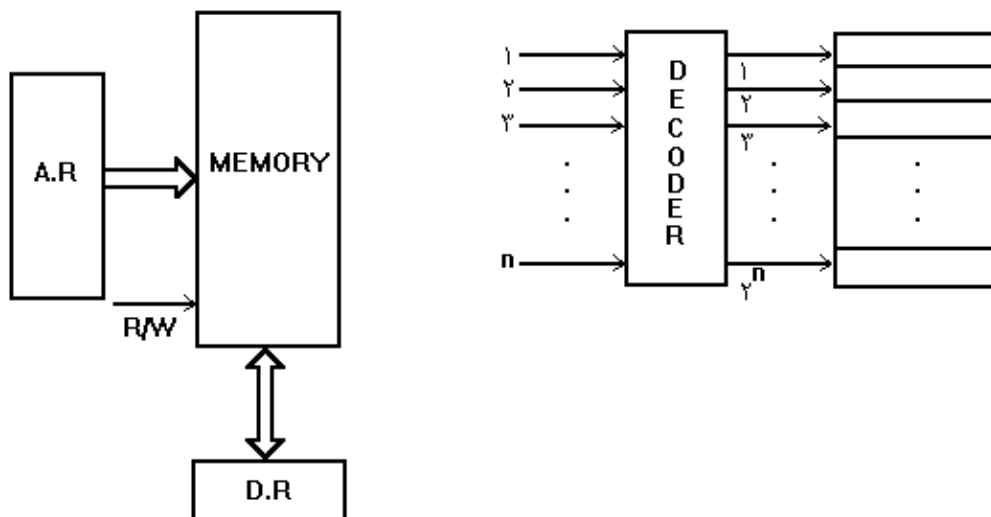


(شکل ۲-۸)

برای مثال Segment d ، هم در 0 هم در 9 وجود دارد. پس این دو سیم را بوسیله گره به هم وصلی نمی کنیم ، چون IC می سوزد. بنابراین باید از یک گیت OR برای این منظور استفاده شود.

### کاربرد دوم - آدرس دهی خطوط حافظه

مثالی از یک آدرس دهی که توسط کدگشا انجام می گیرد، ثبات آدرس می باشد که آدرس محلی از حافظه را به صورت کد نگاه می دارد که این کد بوسیله کد گشا رمز گشایی می شود.



(شکل ۲-۹)

### کاربرد سوم - پیاده سازی توابع ترکیبی

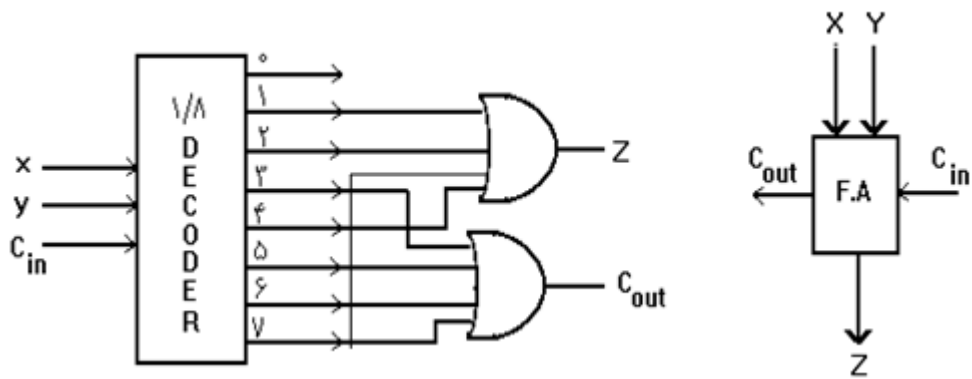
هر تابع با n ورودی و m خروجی را می توان به کمک یک کدگشا  $\frac{1}{2^n}$  و m عدد گیت OR

پیاده

سازی نمود.

مثال : با استفاده از کد گشا و گیت‌های مورد نیاز یک جمع کننده کامل (Full adder) بسازید .

| x | y | $C_{in}$ | Z | $C_{out}$ |
|---|---|----------|---|-----------|
| ۰ | ۰ | ۰        | ۰ | ۰         |
| ۰ | ۰ | ۱        | ۱ | ۰         |
| ۰ | ۱ | ۰        | ۱ | ۰         |
| ۰ | ۱ | ۱        | ۰ | ۱         |
| ۱ | ۰ | ۰        | ۱ | ۰         |
| ۱ | ۰ | ۱        | ۰ | ۱         |
| ۱ | ۱ | ۰        | ۰ | ۱         |
| ۱ | ۱ | ۱        | ۱ | ۱         |

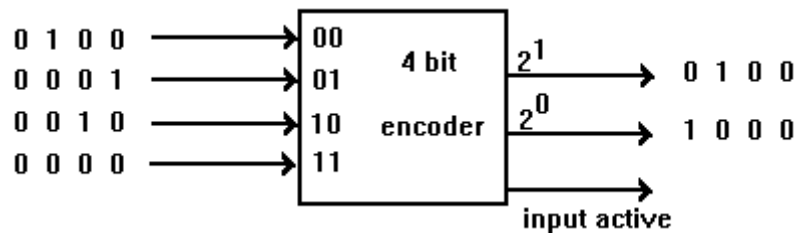


(شکل ۲-۱۰)

۴-۲: کدگذار (encoder)

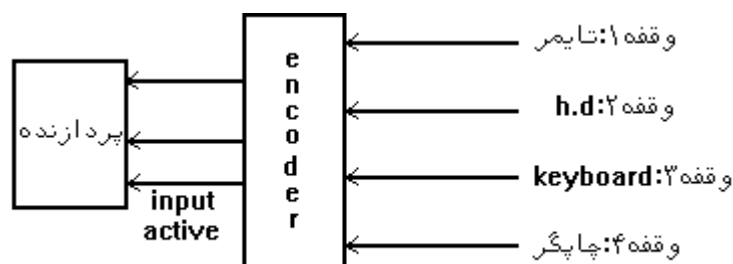
یک مدار ترکیبی با  $2^n$  ورودی و  $n$  خروجی می باشد. در کدگذار در صورتیکه پایه‌ای از ورودی را فعال کنیم در خروجی شماره آن پایه در اختیار قرار می گیرد و در واقع عکس عمل کدگشا را انجام می دهند.

شکلی از یک کدگذار با ۴ ورودی و ۲ خروجی



(شکل ۲-۱۱)

از کاربردهای آن می توان دریافتن آدرس وقفه های مختلفی که به پردازنده ای فرستاده می شود اشاره کرد.



(شکل ۲-۱۲)

دیگر نیاز نیست که برای هر وسیله که وقفه نیاز دارد ، خطوط جداگانه‌ای برای آدرس ایجاد کرد و با یک کدگذار می توان  $2^n$  وسیله را با  $n$  خط ، آدرس دهی نمود.

در اینجا ما به دو مشکل برمی خوریم :

اشکال (۱) : اگر بطور همزمان در ورودی بیش از یک خط فعال باشد ، کدنامشخصی رادر خروجی می دهد که این عیب کدگذار است لذا کدگذار ها را با رعایت حق تقدم می سازند. بدین ترتیب که در صورت فعال شدن بیش از یک ورودی به طور همزمان ، ورودی که حق تقدم بیشتری داشته باشد فعال می شود که به این کدگذار ها Priority encoder یعنی کدگذار با رعایت حق تقدم می گویند.

اشکال (۲) : عیب دیگر کدگذار این است که اگر ورودی ۰۰۰۱ ویا ۰۰۰۰ باشد ، خروجی ۰۰ می شود . بنابر این بایدخطی به کد گذاراضافه شود که آن را input Active می نامند. در صورتیکه هر یک از ورودیها فعال باشد این خط فعال می شود واگر هیئکدام از ورودیها فعال نباشد خط input Active نیز فعال نمی شود. بدینوسیله این مشکل نیز حل می شود.

## ۲-۵ : آرایه منطقی (Logic Array)

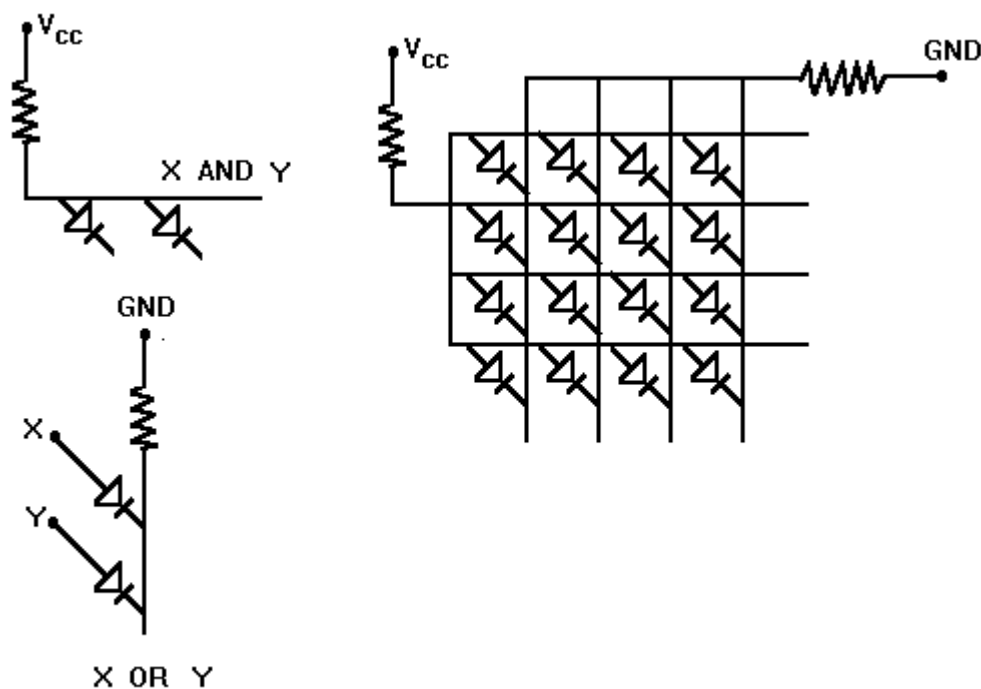
یک مدار ترکیبی است که به صورت آرایه های از دیودها می باشد . که در آن آرایه به کمک دیودها می توان صفرها ویک ها را داشته باشیم . به کمک دیودها ودر نتیجه آرایه منطقی می توان گیتهای AND ویا OR را ساخت . از آنجاییکه هر تابع ترکیبی را می توان به کمک این گیتها ساخت،بنابر این از آرایه های منطقی می توان برای ثیاده سازی هر نوع تابعی استفاده نمود. اغلب در مواقعی که تابع شلوغ باشد استفاده از آرایه منطقی بسیار مناسب خواهد بود. بیشتر در طراحی واحد کنترل کامپیوتر از آرایه های منطقی استفاده می شود . ازانواع آرایه منطقی ، می توان به PLA وROM اشاره نمود.

PLA مخفف Programing Logic array می باشد، که به آن آرایه منطقی قابل برنامه ریزی می گویند.

### ۲-۵-۱- آرایه منطقی برنامه ریزی (PLA)

به کمک PLA می توان مجموعه توابع ترکیبی را به صورت یک مدار دو سطحی متناظر با مجموع حاصلضربها تولید نمود. از کاربردهای اصلی PLA، می توان به طراحی توابع ترکیبی موجود در واحد کنترل اشاره نمود. هر سطر از PLA یک گیت AND و هر ستون یک گیت OR می باشد.

در زیر ساختمان داخلی یک PLA نشان داده شده است:



(شکل ۲-۱۳)

**نکته:** خطهای موجود بین سطرها نشانگر فیوز می باشد که همان دیوداست.

**نکته:** در  $PLA (p \times q)$ ،  $p$  تعداد سطر و  $q$  تعداد ستون می باشد.

**مثال:** به کمک PLA توابع زیر را تهیه سازی کنید.



$$F_1 = \overline{x_1 x_2} + \overline{x_2 x_3} + x_1 x_3$$

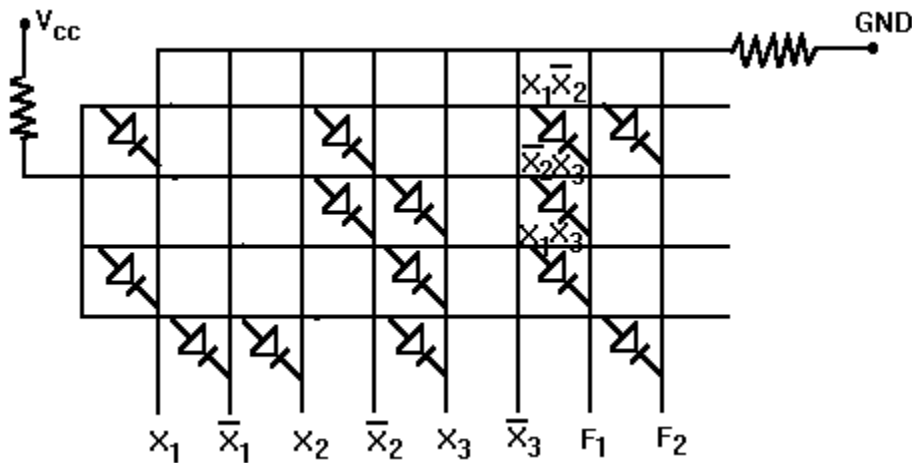
$$F_2 = x_1 \overline{x_2} + \overline{x_1} x_2 x_3$$

حل : ابتدا باید تعداد سطرها و ستونهای مورد نیاز برای پیاده سازی را مشخص کنیم. چون PLA راباسطر و ستونش می شناسند. برای پیاده سازی توابع باید دید که چه مقدار گیت AND مستقل، مورد نیاز است که تعداد این گیتهای AND مستقل برابر با تعداد سطر مورد نیاز در PLA می باشد در این مثال، ۴گیت AND مستقل وجود دارد. برای بدست آوردن ستونها باید مقدار متغیر را در دو ضرب کرده و آنرا با تعداد توابع جمع کنیم.

$$q = \text{not آنها} + \text{متغیرها} + \text{تعداد توابع} = \text{تعداد ستونها}$$

$$\text{یا } q = m + 2 \times n$$

متغیرها                      تعداد توابع



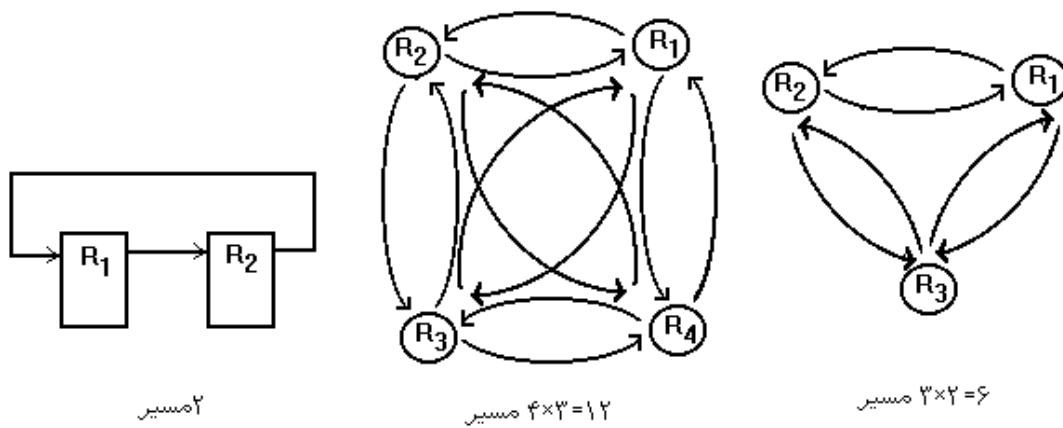
(شکل ۲-۱۴)

۶-۲: گذرگاه (BUS)

مجموعه‌ای از سیمهای رابط برای انتقال یک کلمه N بیتی از یک مبدا به یک مقصد که

معمولاً مبداها و مقصدها ثابت‌های کامپیوتر می‌باشند. در واقع مسیری که برای انتقال

اطلاعات وجود دارد را BUS می‌گوییم.



(شکل ۲-۱۵)

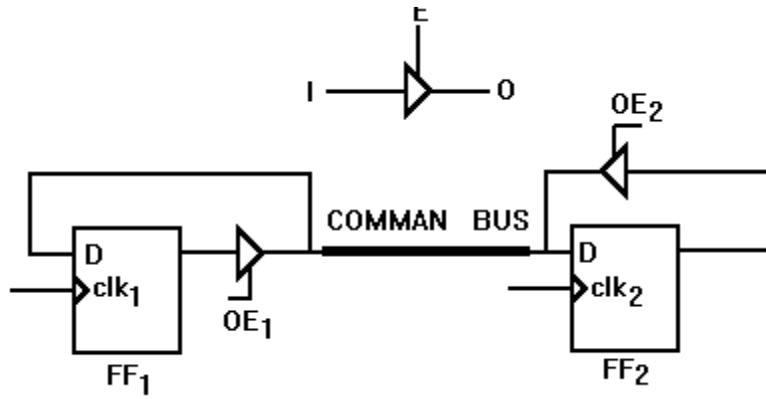
باس (BUS) بردو نوع است:

۱- باس اختصاصی

۲- باس مشترک

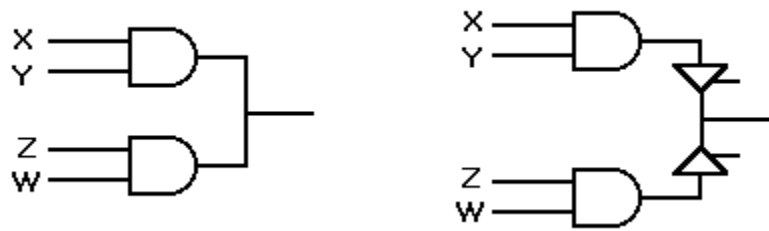
۶-۲-۱- BUS اختصاصی (dedicated Bus)





(شکل ۲-۱۷)

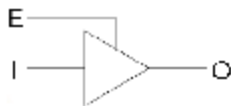
**نکته:** نمی توان خروجی دو گیت یا فیلیپ فلاپ را به هم وصل کرد چون ممکن است یکی High (بالا 5V) و دیگری low (پایین 0V) باشد در نتیجه IC ما می سوزند.



(شکل ۲-۱۸)

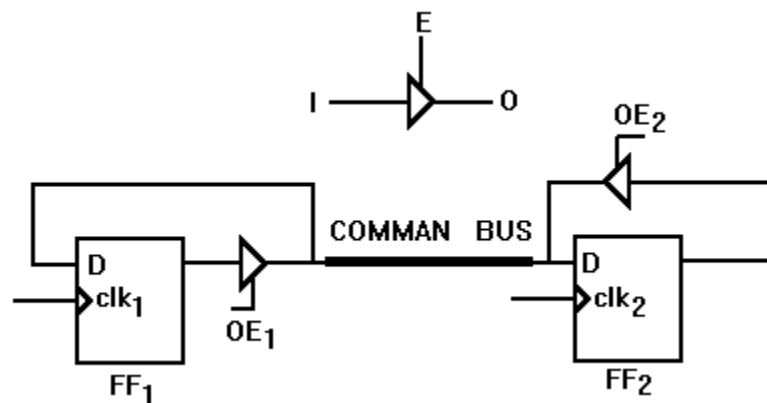
### ۲-۶-۳- بافر یا درایو یا بافر درایو

بافر درایو را به صورت زیر نمایش می دهند:



این درایو دارای یک سویچ (E) است که اگر صفر باشد، مدار قطع و اگر یک باشد مدار متصل است و ولتاژ ورودی و خروجی با هم برابرند و جریان را تقویت می کند که به آن 3-State Buffer یا بافر سه حالت می گویند.

مثال : نحوه انتقال داده بین ثبات A و ثبات B



(شکل ۲-۱۹)

FF1-----> FF2

۱- باید ثبات A فعال باشد تا اطلاعات از آن به ثبات B برود.

1)  $OE1=1$  ,  $OE2=0$

۲- ضمن اینکه ثبات A فعال است به ثبات B دستور ذخیره کردن را می دهیم .

2)  $OE1=1$  ,  $OE2=0$  ,  $Clk2=1$

حال اگر بخواهیم اطلاعات موجود در ثبات B را به A منتقل کنیم بدین صورت عمل می نماییم.

FF2 -----> FF1

۱- باید ثبات B فعال باشد تا اطلاعات از آن به ثبات A برود.

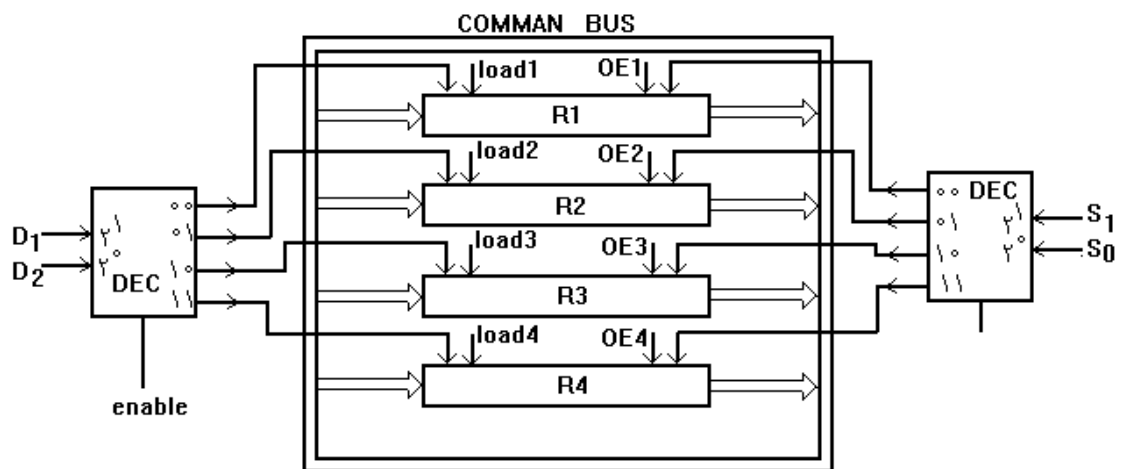
1)  $OE2=1$  ,  $OE1=0$

۲- ضمن اینکه ثابت B فعال است به ثابت A دستور ذخیره کردن را می دهیم.

2)  $OE2=1$  ,  $OE1=0$  ,  $Clk1=1$

۲-۶-۴- طراحی دو نوع BUS مشترک

۲-۶-۴-۱- طراحی یک BUS مشترک بین چهار ثابت که خروجی آنها از نوع 3-State می باشد.



(شکل ۲-۲۰)

مثال: اگر بخواهیم اطلاعات  $R_1$  را به  $R_3$  منتقل کنیم.

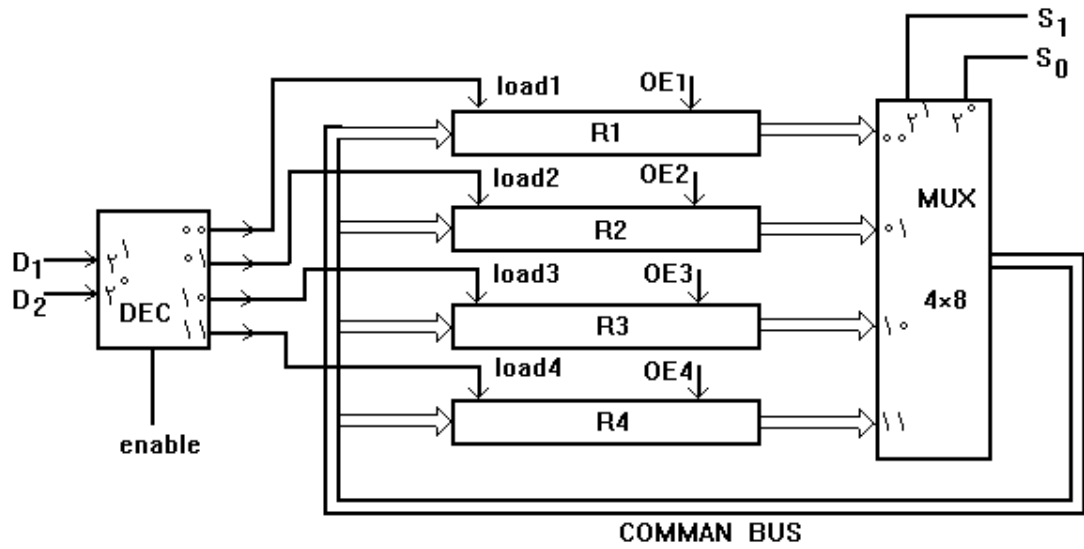
$$R_1 \text{-----} \rightarrow R_3$$

1)  $S_1 S_0 = 00$  ,  $enable = 0$  ,  $D_1 D_0 = 10$

2)  $S_1 S_0 = 00$  ,  $enable = 1$  ,  $D_1 D_0 = 10$

برای اینکه بخواهیم هربار یک ثابت انتخاب شود از کدگشا استفاده کرده ایم که در حقیقت کدگشاها در اینجا انتخابگر مبدأ و مقصد می باشند.

۲-۶-۴-۲- طراحی یک باس مشترک بین چهار ثابت که خروجی آنها از نوع 3-State نیست.



(شکل ۲-۲۱)

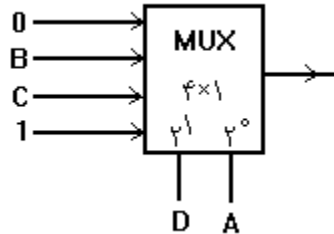
مثال: اگر بخواهیم محتوای  $R_2$  را در  $R_4$  بریزیم داریم .

$$R_2 \text{ -----} \rightarrow R_4$$

- 1)  $S_1 S_0 = 01$  ,  $D_1 D_0 = 11$  ,  $\text{enable} = 0$
- 2)  $S_1 S_0 = 01$  ,  $D_1 D_0 = 11$  ,  $\text{enable} = 1$

تمرینات

۱- عبارت بولی (  $4 \times 1$  ) Mux زیر را بنویسید؟



۲- با استفاده از هر تعداد (  $2 \times 1$  ) Mux یک (  $4 \times 2$  ) Mux طراحی کنید؟

۳- تابع  $f(A, B, C) = \Sigma(0, 2, 6, 7)$  را در سه حالت مختلف با (  $4 \times 1$  ) Mux طراحی کنید؟

۴- با استفاده از دو مالتی پلکسر در توابع موجود یک مدار تفریق کننده کامل (Full-Subtractor) را پیاده سازی نمایید؟

۵- با استفاده از کدگشا (Decoder) و گیت‌های OR لازم یک ضرب کننده دو بیتی طراحی کنید.

۶- به کمک کدگشا (Decoder) و گیت‌های OR لازم یک مدار Full adder- Full Subtractor طراحی نمایید.

۷- مداری طراحی کنید که یک عدد دو بیتی را در یافت و آن را به تابعی که در زیر آمده است اعمال نموده و نتیجه را بر روی 7Seg نمایش دهد. (به کمک کدگشا)



$$x \leq 2$$

$$x > 2$$

$$f(x) = \begin{cases} x^2 + 1 \\ F \end{cases}$$

۸- به کمک کدگشا (Decoder) و گیت‌های مورد نیاز یک مقایسه گر دو بیتی طراحی نمایید.

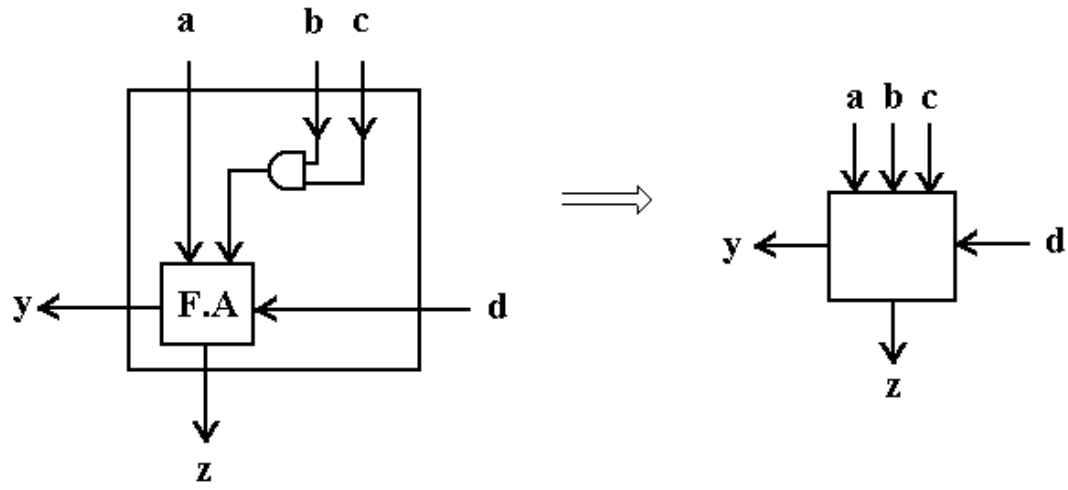
۹- یک کدگذار (Encoder) با رعایت حق تقدم 4 بیتی با استفاده از گیت‌های AND و OR طراحی نمایید.

۱۰- مداری طراحی کنید که بتواند 4 ثبات 8 بیتی را به نحو خواسته شده با هم جمع نماید و حاصل را در ثبات دیگر مورد نظر قرار دهد در این طراحی ثباتها را از نوع 3-State بگیریید و جمع کننده را به صورت یک واحد 4 بیتی.

۱۱- یک مدار ترتیبی طراحی کنید که عدد باینری بدون علامت و با طول n را در 3 ضرب نماید. عدد n بصورت سریال از بیت کم ارزشتر به مدار وارد و حاصل یعنی 3x n به صورت سریال در طرف دیگر خارج می گردد.

۱۲- یکی از بخشهای CPU مدار شیفت دهنده به نام Buffer Shift می باشد که توسط آن می تواند شیفتهای سریع به تعداد بیت‌های خواسته شده به سمت چپ یا راست اعمال نمود. یک Buuffer Shift چهار بیتی طراحی کنید که توسط آن بتوان تا 4 بیت انتقال به چپ یا راست انجام داد. (ثبات را در این مسأله چهاربیتی بگیریید.)

۱۳- چگونه می توان به کمک سلول زیر یک ضرب کننده 3 بیتی طراحی کرد .



### ۳- طراحی پردازنده

#### ۳-۱: وظیفه اصلی CPU

پردازنده قسمتی از یک کامپیوتر می باشد که وظیفه اصلی آن اجرای دنباله‌ای از دستورالعمل‌های موجود در حافظه اصلی می باشد. برای این کار هر دستورالعمل قبل از اجرا شدن بایستی از حافظه اصلی خوانده شود (Fetch) و سپس دستورالعمل رمزگشایی شود (Decoding) آنگاه به داخل پردازنده منتقل شده و سپس دستورالعمل اجرایی شود (Execute).

مجموعه زیر دستوراتی که ، برای اجرای کامل یک دستور لازم است چرخه دستورالعمل یا Inst.Cycle گفته می شود.

چرخه دستورالعمل به دو گروه تقسیم می شود:

۱- Fetchh.Cycle: واکنشی دستور از حافظه .

۲- Execute Cycle: decode کردن یا رمز گشایی دستور و اجرای زیر دستورات لازم .

مراحل اجرای یک دستورالعمل:

واکنشی دستورالعمل: برداشت دستورالعمل بعدی از حافظه

کدبرداری از دستورالعمل: بررسی دستورالعمل برای مشخص شدن اینکه:

چه عملی باید توسط دستورالعمل انجام گیرد (به عنوان مثال جمع)

چه عملوندهایی مورد نیازند، و نتایج باید کجا قرار گیرند.

واکنشی عملوندها: عملوندها برداشت می شوند.

اجرا: اجرای عملیات بر روی عملوندها

بازنویسی نتیجه: نوشتن نتیجه در محل مخصوص

دستورالعمل بعدی: تعیین اینکه دستورالعمل بعدی از کجا گرفته شود.

چه چیزی در یک ISA (معماری مجموعه دستورالعمل) مشخص می شود؟

ISA: Instruction Set Architecture

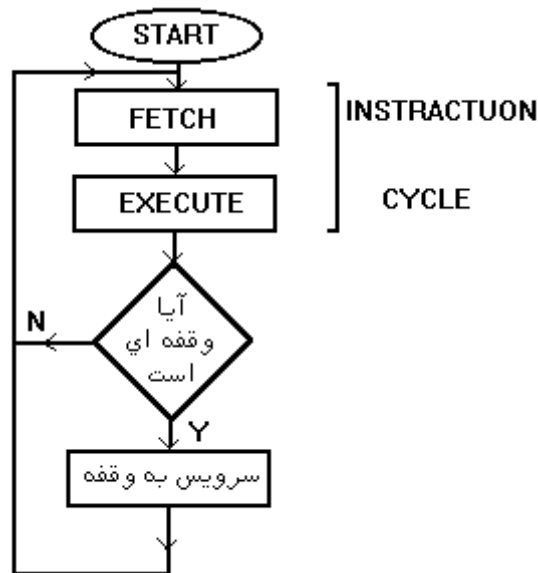
کدبرداری از دستورالعمل: اعمال و عملوندها چگونه تعیین می گردند؟

واکشی عملوندها: عملوندها ممکن است کجا باشند؟ چه تعداد؟

اجرا: چه اعمالی می تواند انجام گیرد؟ چه نوع داده و چه اندازه هایی؟

بازنویسی نتایج: نتایج کجا نوشته می شوند؟ چه تعداد؟

دستورالعمل بعدی: دستورالعمل بعدی را چگونه می توان انتخاب نمود؟



(شکل ۱-۳)

ریزعمل:

انجام کارهای جزئی برای رسیدن به هدف اصلی را ریز عمل گویند.

**: Cpu Clock time**

مدت زمان لازم برای اجرای یک ریزدستورالعمل (Micro Inst) را Cpu Clock time

$(t_{cpu})$  می نامند.

**: Cpu Clock Rate**

تعداد ریزعملهایی که در واحد زمان انجام می شود را به عنوان Cpu Clock Rate گویند. که

معمولاً با واحد هرتز (مگا هرتز) بیان می شود و آن را بصورت  $\frac{1}{t_{cpu}}$  نمایش می دهند ، که ملاک اندازه گیری سرعت پردازنده هاست .

برای مثال اگر سرعت Cpu کامپیوتری 100 MHZ باشد یعنی  $100 \times 2^{20}$  ریز عمل رادر یک ثانیه انجام می دهد.

**۲-۳ : وظیفه فرعی Cpu**

Cpu علاوه بر اجرای دستورالعمل ، بر واحدهای دیگر کامپیوتر نیز نظارت دارد مانند نظارت بر نقل و انتقال اطلاعات از حافظه به دستگاههای ورودی و خروجی (I/o device) و یا نظارت بر کارهای I/o . اما اینگونه کارها به ندرت انجام می گیرد. از این جهت تا لحظه دریافت تقاضا از دستگاههای جانبی CPU توجهی به آنها نمی کند. در صورتی که دستگاه I/o تقاضای سرویس داشته باشد تقاضا را با فرستادن سیگنال Intrupt یا وقفه به Cpu اعلام می کند.

**سؤال :** چرا پردازنده در پایان اجرای دستورالعمل جاری به وقفه توجه می کند؟

برای جواب دادن به این سؤال به مثال زیر توجه کنید.

**مثال :** اگر کامپیوتر در حال اجرای برنامه ای که حاوی دستورالعملهای زیر، به زبان ماشین باشد .

ADD Ax, Num

INC Bx

اگر دستورالعمل اول را دریافت کرده و سپس وقفه فرارسد ابتدا دستورالعمل اول را اجرا کرده و بعد وقفه را اجرا می نماید و بعد از پاسخ به وقفه به سراغ اجرای دستورالعمل دوم می رود. زیرا اگر در حین اجرای دستورالعمل اول به وقفه توجه کند، نتیجه و یا محتوای ثابتی که بر روی آن کارانجام گرفته است دست نخورده باقی می ماند یعنی مانند این است که دستورالعمل اول وجود نداشته است و چون (PC (Program Counter) ما یک واحد اضافه شده است بنابراین امکان دسترسی مجدد به دستورالعمل اول را نخواهیم داشت بنابراین برنامه ما به اشتباه اجرا می شود.

همیشه در طراحی CPU سعی بر آن است که حداقل هزینه و حداکثر سرعت را داشته باشیم. به این خاطر در طراحی مدارات کامپیوتر اگر چه همه کارهای آن بکمک عملیاتی چون جمع، ضرب و تفریق و تقسیم انجام می شود، اما از لحاظ سخت افزاری تنها یک مدار جمع کننده وجود دارد که همه این کارها را انجام می دهد.

اگر قرار باشد یک پردازنده با سرعت بالا عمل کند، بایستی ادواتی که با آن درگیری مستقیم دارند نیز، هم سرعت با آن باشند. (زیرا در غیر اینصورت سرعت نهایی پایین می آید) از جمله ادواتی که با CPU رابطه مستقیم دارد، حافظه اصلی کامپیوتر است.

امروزه می دانیم برای اجرای برنامه‌ها ظرفیت زیادی از حافظه مورد نیاز است. اگر قرار باشد که این حافظه با حجم زیاد خود هم سرعت پردازنده باشد، به عبارتی از تکنولوژی ساخت پردازنده استفاده کند، قیمت حافظه و در نتیجه قیمت کامپیوتر به مراتب بالا می رود. از این جهت برای کاهش قیمت معمولاً حافظه‌ها را با سرعتی حتی تا یک دهم سرعت پردازنده در اختیار آن قرار می دهند. (البته در PC ها اینطور نیست بلکه ضریب سرعت بین حافظه و پردازنده بین ۴ تا ۶ می باشد) بدین ترتیب قیمت حافظه و همچنین قیمت کامپیوتر در کل مطلوب خواهد بود. اما مشکل اختلاف سرعت بین حافظه با ظرفیت بالا و پردازنده را به کمک حافظه دیگری که از نظر سرعت و تکنولوژی همسان با پردازنده است (که این حافظه را اصطلاحاً Cache می نامند) برطرف می سازند.

### سرعت حافظه

مدت زمان لازم برای انجام دو read یا دو Write متوالی را سرعت حافظه گویند. که معمولاً با

سرعت Cpu به صورت  $\frac{t_{mem}}{t_{cpu}}$  مقایسه می شود. (مقدار آن بین 1 تا 10 می باشد) اگر سرعت

حافظه برابر با Cpu باشد ،  $\frac{t_{mem}}{t_{cpu}}$  برابر یک می شود. در این صورت فضای حافظه اصلی به

صورت یک ثبات (Register) تلقی خواهد شد (کارایی یک کامپیوتر نیز با این نسبت سنجیده می شود).

### ۳-۳: ارتباط CPU با I/O و حافظه اصلی

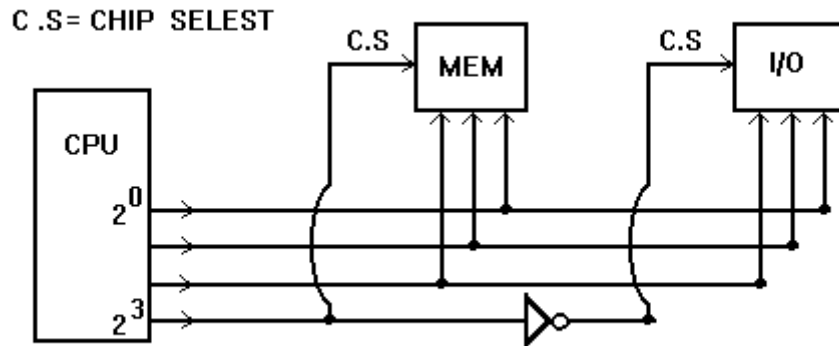
می دانیم که پردازنده علاوه بر حافظه اصلی ، دستگاههای مختلف I/O را دارا می باشد که بایستی بتواند آنها را آدرس دهی نماید . نحوی آدرس دهی حافظه اصلی و دستگاه I/O در سیستم های کامپیوتری به طور عمده به دو دسته تقسیم می شود :

دسته اول : Memory Mapped I/O

دسته دوم : I/O Mapped I/O

### Memory Mapped I/O : ۱-۳-۳

در این روش از کل فضای آدرس دهی حافظه اصلی ، بخشی از آن به I/O های مختلف تعلق دارد و بقیه فضای باقیمانده به حافظه اصلی اختصاص می یابد. بنابراین فضای آدرس دهی I/O باید در فضای آدرس دهی حافظه ، Mapped شود.

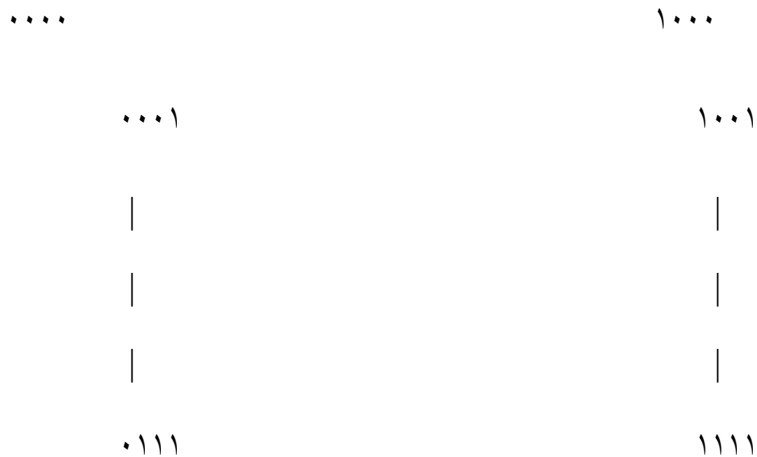


(شکل ۳-۲)

در اینجا فرض شده که CPU هیچ وسیله جانبی ندارد و فقط چهار خط آدرس دارد.

**نکته:** تا زمانی که با ارزش ترین بیت دارای ارزش صفر است، I/O آدرس دهی می شود و وقتی دارای ارزش یک باشد حافظه اصلی آدرس دهی می شود.

با این آدرسها حافظه اصلی فعال می شود. با این آدرسها I/O فعال می شود.



زمانی از این روش استفاده می شود که سرعت I/O و سرعت حافظه با هم برابر باشد. وقتی سرعتها برابر نیست باید وقفه نرم افزاری ایجاد نماییم.

مثال: LDA 2

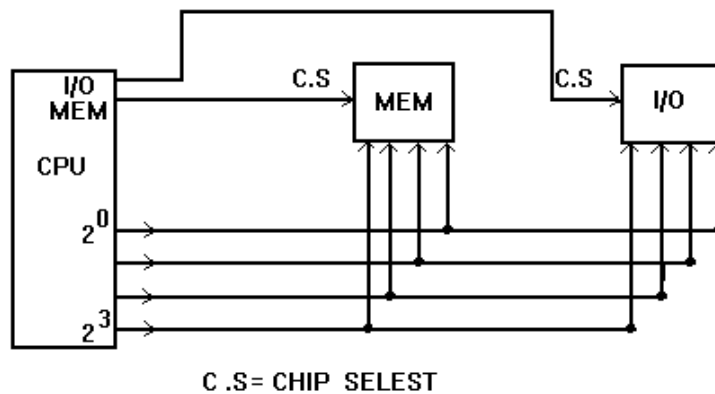
توقف زمان Delay



در مثال فوق در دستور LDA 2 از یکی از وسایل جانبی I/O با آدرس 2 ، مقداری را Load می کند . یعنی بدین صورت از ابزارهای جانبی استفاده می کنیم و چون سرعت I/O با سرعت حافظه اصلی برابر نیست بصورت نرم افزاری یک توقف ایجاد می نمایم . مثلاً بادستور Delay این عمل را انجام می دهیم ، تا کمی توقف زمانی ایجاد شود تا عمل I/O قبلی پایان یابد و سپس عمل بعدی که استفاده از وسایل جانبی می باشد ، صورت گیرد .

### I/O Mapped I/O : ۲-۳-۳

هنگامی که حافظه و I/O هم سرعت نباشد دستورالعملهای مختلفی برای I/O و حافظه وجود دارد و نیز فضای آدرس دهی حافظه ، از فضای آدرس دهی I/O مجزا است و حتی دو دسته دستورالعمل مجزا برای کارکردن با حافظه و I/O وجود دارد.



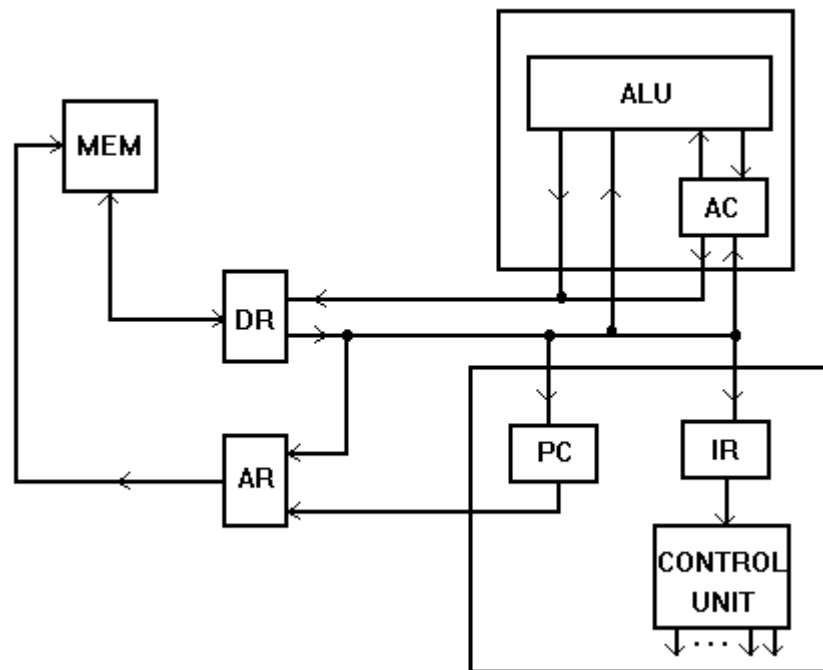
(شکل ۳-۳)

برای مثال دستورات I/O با in و out همراه هستند ولی دستورات حافظه با MOV همراه هستند

نکته : معمولاً بطور واقعی در سیستمهای کامپیوتر مشاهده می شود که خطوط آدرس دهی به حافظه بطور کامل می باشد ولی همی این خطوط به I/O برای آدرس دهی متصل نمی باشد زیرا تعداد I/O به اندازی تعداد آدرسهای حافظه نیست.

۳-۴ : ماشین وان نیومن (Von Neuman)

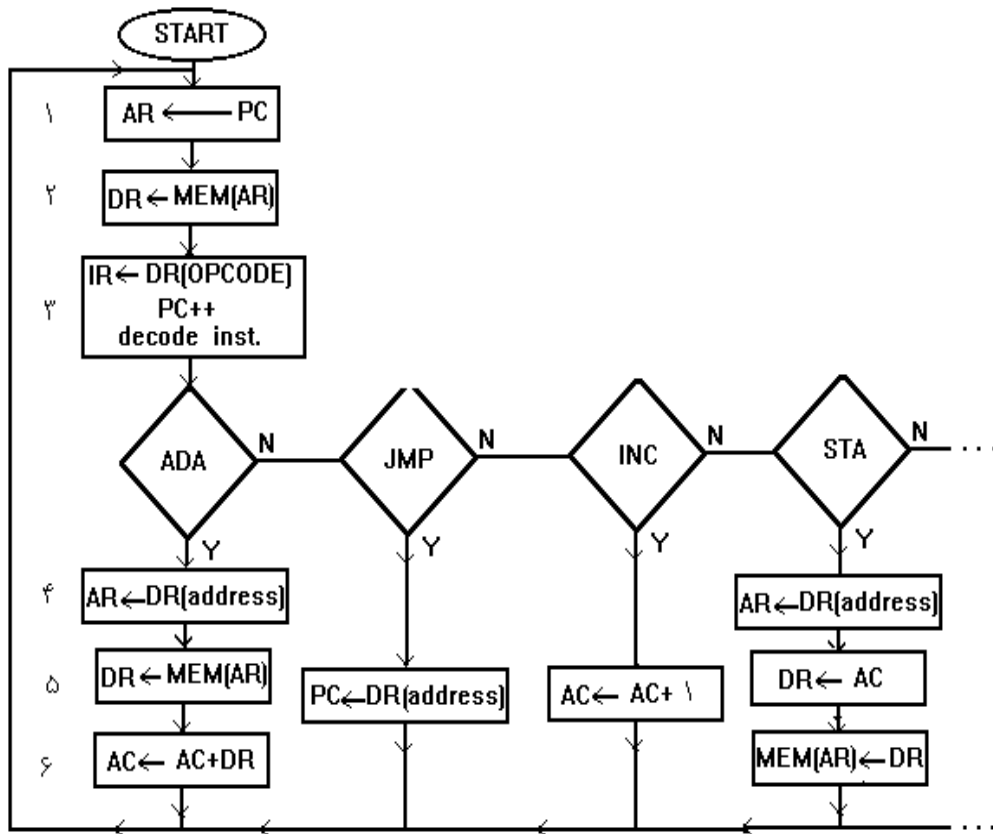
وان نیومن به همراه همکارانش یک کامپیوتر به نام IAS ساخت که پایه و اساس کامپیوترهای امروزی می باشد و کامپیوتر ایشان یک کامپیوتر تک آدرسه بود .



(شکل ۳-۴)

۳-۴-۱ : دستورالعملهای ماشین وان نیومن :

وقتی بخواهیم یک پردازنده طراحی کنیم باید اول بدانیم چه سخت افزارهایی را نیاز داریم و این انتخاب سخت افزارها بسته به دستوراتی است که ریز پردازنده نیاز دارد.



(شکل ۳-۵)

با توجه به دستورات بالا مثلا اگر بخواهیم محتوای AC را با سطر ۱۰۰ حافظه جمع کنیم ( ADA 100 ) ، ابتدا پردازشگر به سطری از حافظه که این دستور درون آن نوشته شده است رفته و این دستور را درون DR قرار می دهد و Decode شده و قسمت های آن مجزا می شوند و بخش آدرس آن که ۱۰۰ است ، درون AR قرار می گیرد . سپس به خانه ۱۰۰ حافظه مراجعه شده و اطلاعات آن درون DR قرار می گیرد . حال به راحتی می توان محتوای ثبات های AC و DR را باهم جمع نمود .

مراحل دستور های بالا باید گفت که این دستورات را باید طوری ادغام کنیم که همزمان اجرا شدن آن تاثیری روی هم نداشته باشد و باعث بالاتر رفتن سرعت برنامه می شود . البته دستور PC++ در مرحله دوم هم می توانست قرار داشته باشد . زیرا خللی در اجرای برنامه نخواهد داشت . انجام

این مراحل به ترتیب است یعنی اگر مرحله اول رخ ندهد ، مرحله دوم هم نمیتواند اتفاق بیفتد که این مسئله مربوط به پردازش موازی است که بعدا بحث خواهد شد.

\* آیا می توانید بوسیله ریز پردازنده‌ای که در اختیار دارید یک دستورالعمل شرطی بکار برید ؟  
خیر ، چون در این ماشین ما دارای فلاگهای خاصی نیستیم ، که بر اساس آن دستورات شرطی را اعمال کنیم .

\* آیا این ماشین می تواند دستور  $LDA\ M\ 100$  را اجرا کند ؟ بله

با قراردادن 100 بطور مستقیم در AC این دستور امکان پذیر است .  $AC \leftarrow AR\ (DR)$

\* آیا این ماشین می تواند دستور  $JnG\ 100$  را انجام دهد ؟

خیر ، زیرا فلاگ  $n$  را برای تشخیص منفی بودن حاصل عملیات دارا نمی باشد.

\* آیا این ماشین می تواند دستور  $LDA\ I\ 100$  را انجام دهد ؟ I یک اشاره گر است بنابراین ۲ بار به حافظه باید مراجعه کنیم ، یک بار به آن مراجعه کنیم که آدرس را بدست آوریم و بار دیگر از آدرسی در داخل حافظه می خوانیم تا Data را بدست آوریم .

$DR \leftarrow MEM\ (AR)$

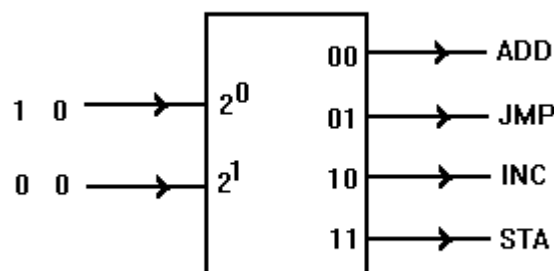
$AR \leftarrow DR\ (Address)$

$DR \leftarrow MEM\ (AR)$

\* آیا این ماشین می تواند دستور  $LDA\ X\ 100$  را انجام دهد؟

خیر ، باید محتوای ثبات Indexing Reg با AC جمع شود که چنین دستوری را در ماشین نداریم. پس باید یک ثبات Index Reg به آن اضافه کنیم تا آن دستور قابل اجرا باشد.

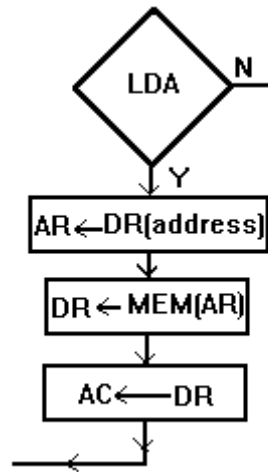
**نکته :** به علت استفاده از کدگشا در قسمت‌های مقایسه‌ای ، این قسمت‌ها شماره گذاری نشده است. یعنی واحد زمانی برای آنها در نظر گرفته نمی شود . همانطور که می دانید در مدار های ترکیبی اتلاف وقت نداریم . بدین صورت که کد گشا یکی از دستورات را تشخیص داده و فوراً بدون اتلاف وقت ریز دستورات مربوط به آن دستورالعمل را اجرامی نماید . این تشخیص دستور، توسط کدگشا از روی Opcode آن دستورالعمل می باشد .



(شکل ۳-۶)

**مثال :** دستورالعمل LDA 5 چگونه اجرا می شود ؟

LDA 5 که محتوای خانه ۵ از حافظه را روی AC بار می کند. در ابتدا PC اشاره می کند به سطری که دستور LDA 5 در آن قرار دارد و آن را در DR قرار می دهد . در این مرحله بایستی آدرس DR را که همان ۵ است را به AR انتقال دهد و سپس محتوای این خانه جدید که آدرسش در همان AR می باشد را به DR منتقل می نماید و سپس DR را به AC می ریزد. چون بین حافظه و ALU هیچ ارتباطی وجود ندارد مرحله آخر الزامی است.



(شکل ۳-۷)

**نکته :** وقتی دستگاه راه اندازی می شود، PC طوری طراحی شده که به جای مشخصی اشاره کند و باید سیستم عامل را در آن آدرس مشخص قرار دهیم تا دستورات از آن آدرس شروع ، و به اجرا درآیند تا سیر منطقی اجرای دستورات صورت گیرد. یعنی به ابتدای برنامه سیستم عامل که روی ROM قرار دارد رفته و آن را روی RAM می آورد و سپس به ابتدای برنامه ای که خودش Load کرده است ، رفته و سیستم عامل را اجرا می کند .

**مثال :** مدت زمانی که طول می کشد تا دستورالعمل جمع اجرا شود چند ثانیه می باشد؟

(فرض کنید سرعت CPU برابر یا 60 MHZ باشد)

مدت زمانی که طول می کشد تا دستورالعمل جمع اجرا شود .

$$6 \times \frac{1}{60 \times 2^{20}} = \frac{1}{2^{21}} \text{ یا } \text{تعداد مراحل}$$

تعداد ریز عملها

فرکانس (قطر پالس ها)

- تمرین : ده دستورالعمل از کامپیوتری که در اختیار دارید را در نظر بگیرید .
- الف ) ابتدا حدس بزنید که این دستورالعمل ها چه تعداد ریز دستور نیاز دارند .
- ب ) با استفاده از کتاب ها و منابعی که در اختیار دارید این تعداد را بدست آورید .
- ج ) با استفاده از برنامه محاسبه کنید . ( استفاده از حلقه )
- د ) این ده دستورالعمل را در یک برنامه به کار ببرید .
- نکته : برنامه ی خود را با استفاده از زبان اسمبلی بنویسید .

### ۳-۴-۲: روشهای گسترش ماشین وان نیومن

می توان به راحتی قابلیت های بیشتری به ماشین پیشنهادی Von - neuman داد.

۱) می دانیم که اگر تعداد ثبات های یک پردازنده بیشتر باشد، می توان برنامه را با سرعت بیشتری اجرا نمود . به راحتی می توان ثبات های بیشتری به این ماشین اضافه نماییم .

۲) این ماشین فقط دستورالعمل های جمع و تفریق را انجام می دهد و می توان به راحتی با افزودن مدارات جزئی به آن قابلیت ضرب و تقسیم را نیز به آن بخشید .

۳) ماشین Von neuman نمی تواند دستورات پرشی از نوع شرطی را اجرا کند . اما به راحتی می توان با اضافه کردن Flag Register به آن ، این قابلیت را به ماشین Von neuman داد.

۴) اگر در پردازنده ای سرعت حافظه اصلی نسبت به CPU کم باشد ، می توان با دراختیار گذاشتن حافظه نوع Cache ، در عمل خواندن و نوشتن از حافظه سرعت را بهبود بخشید .

### ۳-۵ : پردازش موازی (Parallel Processing) :

پردازش موازی اصطلاحی است برای مشخص کردن دسته وسیعی از روشهایی که برای پردازش همزمان اطلاعات با هدف افزایش سرعت محاسباتی یک سیستم کامپیوتری به کار می رود .

یک سیستم پردازش موازی می تواند به جای پردازش ترتیبی دستورالعمل ها به سبک کامپیوترهای نسبی ، اطلاعات را به طور همزمان پردازش کند و به این ترتیب به سرعت بالاتری دست یابد . مثلاً در حالی که یک دستورالعمل در Alu در حال اجرا است دستور بعدی می تواند از حافظه خوانده شود .

سیستم می تواند دو Alu و یا بیشتر داشته باشد و در یک زمان دو دستورالعمل بابتیتر را اجرا کند . علاوه براین می تواند دارای دو یا چند پردازنده باشد که همزمان کار کنند.

هدف از پردازش موازی تسریع قابلیت پردازش کامپیوتر و افزایش توان عملیاتی آن ، یعنی میزان پردازش قابلیت انجام در طول یک فاصلی زمانی معین است . در پردازش موازی حجم سخت افزار افزایش می یابد و به همراه آن هزینه سیستم نیز بالا می رود . اما پیشرفت های تکنولوژی هزینه های سخت افزاری را قدری کاهش داده است که به همین علت تکنیکهای پردازش موازی از نظر اقتصادی به صرفه شده است .

نمونه ای از پردازش موازی در زیر آمده است

مثال ۱ : Parallel adder در مقابل Serial adder



Parallel adder نوعی جمع کننده می باشد که مجموعه‌ای از بیت ها را با تیش بینی رقم نقدی جمع می کند و Serial adder در هر لحظه یک بیت را با بیت متناظر جمع می بندد .

مثال ۲ : مراحل Fetch , Execute یک دستورالعمل

اعمال Fetch , Execute در دو مرحله مستقل صورت می گیرند . بنابراین هنگامی که دستورالعملی در حال اجرا است می توانیم دستور جدید را Fetch کنیم . در آن موقع هردو کار صورت می گیرد، یکی واکنشی دستورالعمل جدید و دیگری اجرای دستورالعمل قبلی .

در مثال اول از پردازش موازی ، عناصر پردازش را افزایش می دهیم . (یعنی تعداد F.A ها را افزایش می دهیم ) در مثال دوم انجام یک عمل به چند قسمت مستقل تقسیم می شود .

بطور کلی پردازش موازی به دو طریق امکان پذیر است

۱- Multiunit Parallel Process

۲- Pipelining Parallel Process

مثال اول از نوع Multiunit می باشد ، چون به سخت افزار وهزینه بیشتری نیازمند است ومثال دوم براساس Pipeline می باشد که یک واحد به چند تکه تقسیم می شود بطورکلی برای پیاده سازی پردازش موازی به سخت افزار بیشتری نیاز است که این نیازمندی در روش Multiunit مشهودتر است.

کامپیوترها را از لحاظ نحوی پردازش به چهار گروه تقسیم می کنند.

۱- SISD (Single Instruction Stream Single Data Stream)

این کامپیوترها دارای یک CPU و یک ALU می باشند و در آن از پردازش موازی استفاده نمی شود. یعنی فقط یک دستورالعمل در هر لحظه انجام می شود. نظیر پردازنده های 8085, 80280

### ۲- SIMD (Single Instruction Stream Multiple Data Stream)

پردازش موازی از نوع Multiunit دارای یک CPU و چند ALU است و نیز دارای دستوراتی برای پردازش بر روی بردارها می باشد.

### ۳- MISD (Multiple Instruction Stream Single Data Stream)

این کامپیوترها بصورت Pipeline عمل می کنند. نمونه ای از کامپیوتر که براین اساس است کامپیوتر ۱-Carry می باشد که یک سوپرکامپیوتر است.

### ۴- MIMD (Multiple Instruction Stream Multiple Data Stream)

این کامپیوترها دارای چند CPU و چند ALU هستند که پردازش موازی را چه از لحاظ Pipeline و چه از لحاظ Processor انجام می دهد. نمونه ای از کامپیوتر که براین اساس عمل می کند IBM 308 می باشد.

کامپیوترهای شخصی (PC) تا حدی براساس شیوه سوم عمل می کنند (یعنی MISD) در کامپیوترهای شخصی 8080 به بالا، این به دو قسمت تقسیم می شود. قسمت اول وظیفه دریافت دستورالعمل (Fetch) را برعهده دارد و قسمت دوم وظیفه اجرای دستورالعمل را برعهده دارد.

قسمت اول دارای یک صف است که دستورالعمل از حافظی Fetch شود وارد پشته (Stack) می شود سپس این دستورالعمل به قسمت دوم می رود و اجرا می گردد. در زمانی که قسمت اول بیکار است، دستورالعمل دوم و سوم را می گیرد تا کار اجرای دستورالعمل اول تمام شود. پس از پایان اجرای دستورالعمل اول، دستورالعملهای دوم و سوم به ترتیب اجرا می گردد.

در کامپیوتر ۸۰۸۸ این صف ظرفیت شش دستورالعمل را دارد و در کامپیوتر ۸۰۸۶ این صف ظرفیت نه دستورالعمل را دارد.

مثالی از Pipeline در ALU

در نظر بگیرید که آرایه  $a, b$  را از هم کم کنیم و حاصل را در  $c$  بریزیم. برای آنکه بشود کاری را بصورت Pipeline انجام داد، باید آنرا به قسمتهای مستقل تقسیم کنیم. اگر یک عمل به  $n$  قسمت مستقل تقسیم شود می توان گفت که سرعت اجرا تقریباً  $n$  برابر می شود. هر چه تعداد تقسیمات زیاد شود، سرعت اجرا هم بیشتر می گردد. برای آنکه  $a, b$  را از هم کم کنیم در ابتدا از  $b$  مکمل یک می گیریم و سپس با یک جمع می کنیم و حاصل را با  $a$  جمع بسته و در  $c$  قرار می دهیم.

$$\bar{b} \quad (1)$$

$$\bar{b} + 1 \quad (2)$$

$$a + \bar{b} + 1 \quad (3)$$

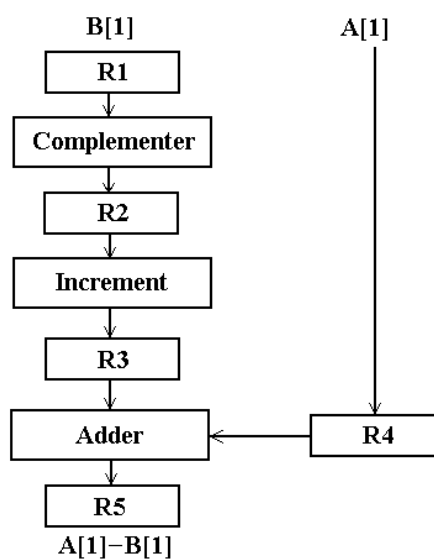
$$a + \bar{b} + 1 \rightarrow c \quad (4)$$

دیاگرام و نتیجه حاصل از دیاگرام برای محاسبه  $c[I]=A[I]-B[I]$

| NO | R1 | R2          | R3              | R4  | R5  |
|----|----|-------------|-----------------|-----|-----|
| 1  | B1 | ---         | ---             | --- | --- |
| 2  | B2 | $\bar{B}_1$ | ---             | --- | --- |
| 3  | B3 | $\bar{B}_2$ | $\bar{B}_{1+1}$ | A1  | --- |
| 4  | B4 | $\bar{B}_3$ | $\bar{B}_{2+1}$ | A2  | C1  |
| 5  | B5 | $\bar{B}_4$ | $\bar{B}_{3+1}$ | A3  | C2  |
| 6  | B6 | $\bar{B}_5$ | $\bar{B}_{4+1}$ | A4  | C3  |

بین هر مرحله از این برنامه باید یک ثابت قرار داشته باشد. زیرا لزومی ندارد که زمان اجرای هر مرحله باهم برابر باشد و بعد بر اساس آنها مراحل بعدی انجام شود. در روش Pipelining هیچگاه خروجی هر مرحله نباید با ورودی اتباط داشته باشد. در غیر این صورت نمی توان آن را به این حالت نوشت.

اولین محصول را در لحظه ۴ داریم و پس از آن در هر لحظه یک محصول داریم و این ۴ لحظه اول همان تاخیری است که باعث می شود بگوئیم تقریباً سرعت  $n$  برابر می شود و باید دقت کنیم که اولین  $A$  در لحظه ۳ وارد می شود.

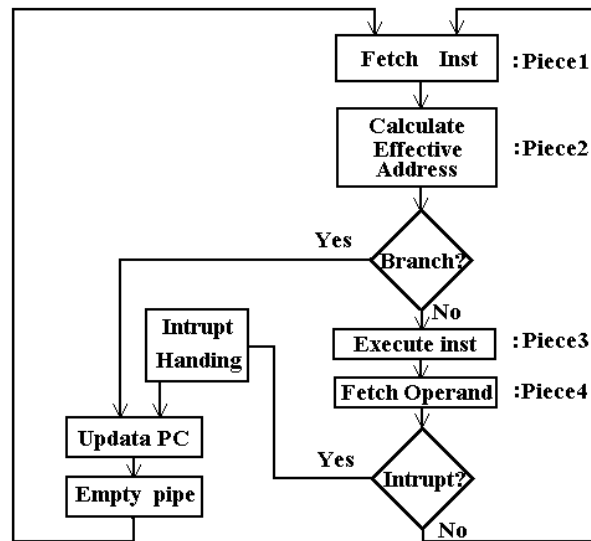


(شکل ۳-۸)

مثالی از پردازش موازی در اجرای دستورالعمل در CPU

در صورتیکه بتوان مراحل اجرای یک دستورالعمل را به  $n$  قسمت مستقل تقسیم کرد، می توان سرعت اجرای دستور را تقریباً  $n$  برابر نمود. در اجرای یک دستورالعمل حداقل می توان چرخه دستور را به دو قسمت مستقل و مساوی Execute, Fetch تقسیم نمود. در دیاگرام زیر چرخه

اجرای دستورالعمل به چهار قسمت مستقل تقسیم شده است .



(شکل ۳-۹)

در اینجا باید آدرس موثر حساب شود پس از آدرس باید آرگومان مربوط را Fetch شود و سپس دستورالعمل اجرا گردد. یعنی آرگومان در AC ذخیره شود. می دانیم که در تقسیم بندی رمزی اجرای دستورالعمل در دو قسمت Fetch, Execute است اما تقسیم بندی جدید همی قسمت ها را شامل نمی شود ، یعنی با دستورالعملهای زیر نیازی به بخش محاسبه آدرس موثر نداریم زیرا Operand ها در داخل ثبات هستند اما بهرحال باید از مراحل بگذرند که هر مرحله برای خود یک تاخیر دارد. سرعت مراحل به اجبار باید با هم برابر باشد زیرا دستورالعملها بصورت موازی اجرا می شود . در اجرای دستورالعمل پرش Update PC می رسیم . برای این دستور باید صفت خاصی در نظر گرفت که برای همین از Empty Pipe استفاده می شود که اگر دستورالعمل از نوع وقفه باشد به برنامه سرویس دهی وقفه می رود و وقفه را سرویس دهی می کند . وقتی که از دستورالعمل پرش و یا وقفه استفاده می شود کارایی روش Pipeline کاسته می شود . مثلاً در نظر بگیرید دارای دو عملیات A,B باشیم .

A : LDA 100

B : JMP 50

A را در ابتدا وارد می کنیم ، قطعاً باید A زودتر اجرا شود و سپس B اجرا گردد . حال اگر A وارد سیر غیر پرشی و B وارد سیر پرشی شود باید سرعت هر دو با هم برابر باشد ، چرا که در غیر

اینصورت باعث ایجاد اختلال در الگوریتم می شود . به همین دلیل در دو دستور ابتدا سطر ۱۰۰ ، Load می شود سپس عمل پرش صورت می گیرد. یعنی دو سطر باید هماهنگ وهم سرعت باشند . همه ی دستورات و مراحل باید خودشان را با کند ترین دستوالعمل تطبیق دهند.

### تمرینات

- ۱- وظایفی که CPU بعهده دارد را بصورت فلوجارتی نمایش دهید؟
- ۲- توضیح دهید که با چندنوع کدگذار (Encoder) می توان چهار وقفه سخت افزاری را به یک پردازنده اعمال نمود؟
- ۳- اگر سرعت CPU کامپیوتری 133 MHZ باشد ، تعداد ریزعملهای انجام شده در مدت دو ثانیه در این کامپیوتر چه مقدار می تواند باشد؟
- ۴- مدت زمانی که طول می کشد تا دستوالعمل ذخیره سازی اجرا شود چند ثانیه می باشد؟ (فرض کنید سرعت CPU برابر با 100 MHZ است .)
- ۵- مدت زمان اجرای قطعه برنامه زیر را در کامپیوترهایی با سرعت 100 MHZ و 133 MHZ بدست آورده وبا یکدیگر مقایسه نمایید؟

- |     |     |
|-----|-----|
| 1   | LDA |
| 2   | ADA |
| DEC |     |
| 3   | SBA |
| INC |     |
| 4   | STA |

۶- یک ماشین دارای دستورات ۱۶ بیتی است وهر آدرس حافظه ۶ بیتی می باشد . بعضی از دستورات تک آدرسی و بعضی دو آدرسی می باشند . اگر n دستوالعمل دو آدرسی باشند ، حداکثر تعداد دستورات ممکن تک آدرسی چندتا است؟

$$(۱) \quad 2^6 \times (16 \times n) \quad (۲) \quad 2^{16} - 2^{12} \times n \quad (۳) \quad 2^{10} - 2^6 \times n \quad (۴) \quad 2^{10} - n$$

۷- به ماشین وان نیومن امکاناتی را اضافه کنید که توسط آن دستورات با آدرس دهی نوع Indexing را بتوان اجرا نمود؟

۸- فلوچارت اجرایی دستورالعملهای زیر را ترسیم نمایید؟

- 1) LDA        m        10
- 2) LDA        I        10
- 3) LDA        x        10

۹- به ماشین وان نیومن امکاناتی را اضافه نمایید که توسط آن بتوان دستورات پرش شرطی را اجرا نمود؟

۱۰- کدام دستورالعمل این عملیات را به ترتیب انجام می دهد؟

- 1) MAR <----- R2
- 2) DR <----- M[MAR]
- 3) MAR <----- DR
- 4) M[MAR] <----- R1 , R2 <----- R2 + 1

(۱) Move مستقیم R1 با اضافه شدن خودکار ثبات R2 (Post - Increment)

(۲) Move غیرمستقیم R1 و Move غیرمستقیم R2 اضافه شدن خودکار R2

(۳) Move مستقیم R1 با اضافه شدن خودکار ثبات R2

(۴) Push مستقیم R1 براساس ثبات R2

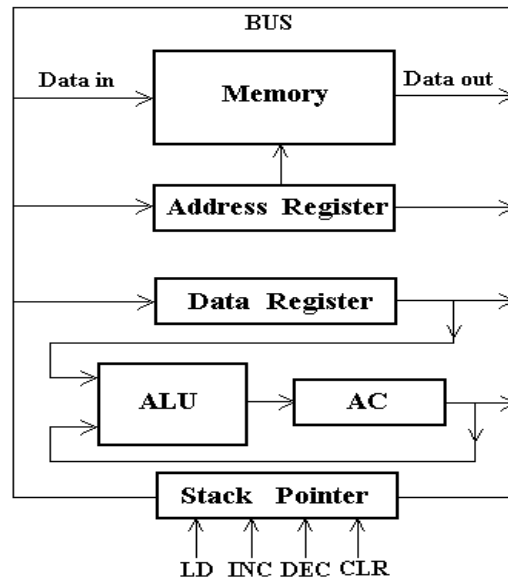
۱۱- از یک کامپیوتر با سازمان اکومولاتور ، باس و تعدادی از ثباتها و حافظه و مسیرهای مبادله

اطلاعات نشان داده است . SP به محل ثربالای نشسته اشاره می کند . اگر دستورات POP

PUSH AC , AC در این کامپیوتر تعبیه شده باشند، تعداد Clock لازم برای فقط سیکل های

اجرایی (Execute cycle) دو دستور POP , PUSH بترتیب چیست ؟

- (۱) ۲ و ۴
- (۲) ۲ و ۴
- (۳) ۲ و ۳
- (۴) ۳ و ۳



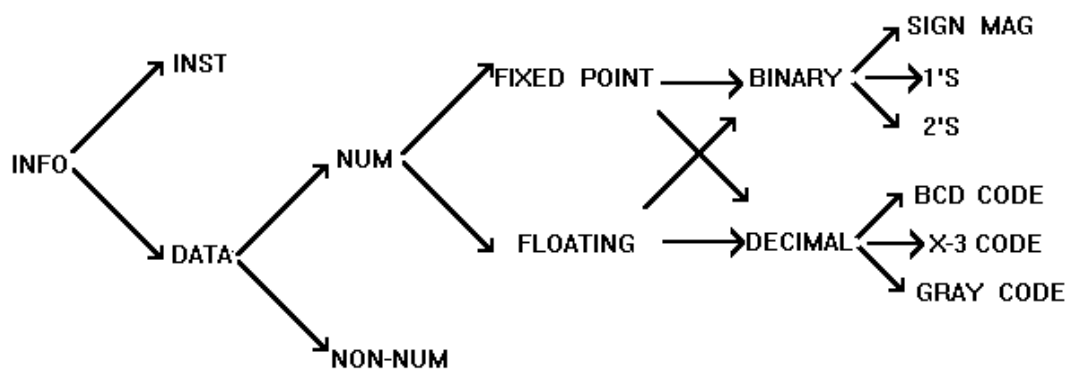
۱۲- در پیاده سازی .....انجام یک عمل را به چند قسمت مجزا تقسیم می کنند . زمان لازم برای اجرای یک دستورالعمل در حالت کلی برابر با مجموع مدت تأخیر در هر قسمت نیست ، بلکه برابر با ..... می باشد . با پیاده سازی این روش سرعت اجرای دستور افزایش می یابد اما اگر .....استفاده شود کارایی آن از بین می رود.



## فصل دوم- طراحی واحد محاسبه و منطق

### ۱- نحوه نمایش داده در کامپیوتر

برای طراحی یک Alu لازم است بدانیم محاسبات مورد نظر می خواهد بر روی داده ای با چه قرار داد نمایش اعمال شود.



( شکل ۱-۱ )

معرفی کدهای دهدهی (Decimal Code) و آشنایی با ضوابط حاکم بر آنها در انجام محاسبات .

| رقم در مبنای 10 | BCD  | EX - 3 | Gray | EX - 3 Gray |
|-----------------|------|--------|------|-------------|
| ۰               | ۰۰۰۰ | ۰۰۱۱   | ۰۰۰۰ | ۰۰۱۰        |
| ۱               | ۰۰۰۱ | ۰۱۰۰   | ۰۰۰۱ | ۰۱۱۰        |
| ۲               | ۰۰۱۰ | ۰۱۰۱   | ۰۰۱۱ | ۰۱۱۱        |

|   |      |      |      |      |
|---|------|------|------|------|
| ۳ | ۰۰۱۱ | ۰۱۱۰ | ۰۰۱۰ | ۰۱۰۱ |
| ۴ | ۰۱۰۰ | ۰۱۱۱ | ۰۱۱۰ | ۰۱۰۰ |
| ۵ | ۰۱۰۱ | ۱۰۰۰ | ۰۱۱۱ | ۱۱۰۰ |
| ۶ | ۰۱۱۰ | ۱۰۰۱ | ۰۱۰۱ | ۱۱۰۱ |
| ۷ | ۰۱۱۱ | ۱۰۱۰ | ۰۱۰۰ | ۱۱۱۱ |
| ۸ | ۱۰۰۰ | ۱۰۱۱ | ۱۱۰۰ | ۱۱۱۰ |
| ۹ | ۱۰۰۱ | ۱۱۰۰ | ۱۱۰۱ | ۱۰۱۰ |

(جدول ۱ - ۱)

### ۱-۱- اعداد بی سی دی (BCD)

چون در تبدیل اعداد به باینری با تقسیمات متوالی وقت زیادی گرفته می شود، لذا در بعضی از کامپیوترها از کد BCD استفاده می کنند.

$$(37)_{10} = (100101)_2$$

$$\text{BCD} = 0011 \ 011$$

3      7

در مبنای ۲                      در مبنای ۱۰

$$A - B = A + \bar{B} + 1, A + 9's(B) + 1$$

### عیب اول BCD

وقتی به مبنای ۱۰ می رویم، بدست آوردن 9's اعداد BCD مشکل می باشد و نیازمند طراحی مدارات جدیدی است. مثلاً در تبدیل عدد ۳ به ۶ که در واقع عدد ۶ متمم ۳ در روش 9's می باشد مشکل وجود دارد. یعنی نمی توان به سادگی با تبدیل صفرها به یک ها و یک ها به

صفرها متمم یک عدد را از روش 9's بدست آورد . ولی در روش EX-3 می توان به راحتی 9's یک عدد را با تبدیل یکها به صفر و صفرها به یک بدست آورد .

مثال :  $(1)_{10} = (0100)_{X-3} \xrightarrow{9's} (1011)_{X-3} = (8)_{10}$

**عیب دوم BCD**

عیب دوم این است که حاصل جمع دو عدد BCD ، الزاماً BCD نیست . در مثال ۱، دو عدد BCD با هم جمع شدند و حاصل نیز BCD است ولی در مثال ۲ دو عدد BCD با هم جمع شده اند و حاصل BCD نمی باشد .

\* برای این که عدد را به BCD تبدیل کنیم باید مقادیر را که در رنج BCD نیستند با ۶ جمع کنیم .

|   |   |
|---|---|
| مثال ۱  | مثال ۲  |
| $\begin{array}{r} 0011 \\ + \quad 7 \\ \hline 0110 \end{array}$ | $\begin{array}{r} 0111 \\ + \quad 3 \\ \hline 0101 \end{array}$ |
| $\begin{array}{r} 0110 \\ + \quad 5 \\ \hline 1001 \end{array}$ | $\begin{array}{r} 0101 \\ + \quad 6 \\ \hline 1100 \end{array}$ |

۱۰۰۱ حاصل BCD نیست + ۱۱۰۰ ۱۲ ۱۱۰۰ + ۹

دو عدد با هم جمع شوند

$$\begin{array}{r} 110 \\ \hline \end{array}$$

و حاصل BCD  $(12)_{10}$  ۱/۰۰۱۰ است.

حاصل BCD است .

۲-۱- اعداد افزایش سه تایی (EX - 3)

با اضافه کردن عدد ۳ به کد BCD اعداد X-3 بوجود می آید . اعداد ۰ به ۹ و ۱ به ۸ و .... با ایجاد یک 9's به هم تبدیل می شوند بنابراین مشکل اول در کد BCD ، در کد EX-3 حل می شود ولی مشکل دوم در کد EX-3 نیز وجود دارد.

مثال ۲

مثال ۱

|        |      |         |  |       |
|--------|------|---------|--|-------|
| ۰۱۱۰   | +    | ۱۰۱۰    |  | ۳     |
| ۱۰۰۱   | ۵    | ۱۰۰۰    |  | ۶     |
| ۱۱۱۱ - | ۱۲   | ۱۰۰۱۰ + |  | ۹     |
| ۰۰۱۱   | ۱۱۰۰ | ۰۰۱۱    |  | ۱۰۱۰۱ |

در مثال ۱ انتظار جواب ۹ برای کد EX-3 یعنی ۱۱۰۰ را داریم که باید ۳ واحد از آن کم کنیم تا جواب درست حاصل شود و در مثال ۲ انتظار جواب ۱۲ را داریم که باید با ۳ جمع شود تا ۱۰۱۰۱ که معادل ۱۲ است بدست آید . پس در بعضی از مواقع ۳ واحد کم می شود و در بعضی از مواقع ۳ واحد اضافه می شود .

نکته : هر جا که رقم نقلی رخ دهد ۳ واحد اضافه می کنیم و اگر رقم نقلی رخ ندهد ۳ واحد کم می کنیم .

نکته : کد EX-3 بهتر از کد BCD می باشد .

نکته : در سرعت ها و فرکانس های بالا ، به دلیل میدان مغناطیسی بالا ، Noise ایجاد می شود .

### ۱-۳-۱- کد گری (Gray)

در این کد هر عدد، با عدد قبلی و بعدی آن در یک بیت اختلاف دارد . در این روش احتمال خطا و Noise بسیار کاهش می یابد چون عملاً از یک شمارنده (Counter) استفاده می شود . البته در کد BCD و EX-3 چون خطوط تغییر زیاد می باشند، ممکن است فلیپ فلاپهای موجود در شمارنده همزمان عمل نکنند و بدین صورت عمل کنند.

|      |      |      |      |
|------|------|------|------|
| ۰۰۱۱ | ۰۰۱۰ | ۰۱۱۰ | ۰۱۰۰ |
|------|------|------|------|

۳ -----> ۴

بنابراین احتمال اینکه هنوز فلیپ فلاپی باشد که اثر نکرده باشد و سیستم، آن عدد را به عنوان عدد بعدی قبول کند وجود دارد، لذا دچار اشتباه می شود . (مانند مراقبین جلسه که با گفتن اجرای دستور شماره N عمل شماری N را انجام می دهند) . ولی در Gray Code چون فقط یک فلیپ فلاپ تغییر می کند این خطا به حداقل می رسد.

۳ -----> ۴

کد گری محاسبه پذیر نیست و برای انتقال داده هاست و با استفاده از آن اطلاعات واقعی از یک محیط نمونه گیری شده و در جای دیگر انتقال داده می شود . کاربرد بعدی آن در طراحی شمارنده واحد کنترل می باشد .

### ۱-۳-۱- کاربردهای کد خاکستری

کاربرد اول : طراحی شمارنده واحد کنترل

در داخل واحد کنترل عملیات براساس شماره ونظم خاصی انجام می گیرند و برای اینکه این مراحل درست و منظم انجام شوند نیاز به روشی است که اعدادی که نشان دهنده ی هر مرحله است ، به درستی بعد از هم شمارش شوند که این کار با استفاده از کد گری امنیت بیشتری خواهد داشت .

### کاربرد دوم : در انتقال اطلاعات

(۱) مثلاً اگر بخواهیم متوسط دمای هوای محیط را در یک روز بدست آوریم ، کامپیوتری داریم که تشخیص می دهد ، درجه هوا طبق یک مسیر پیوسته افزایش یا کاهش می یابد. مثلاً درجه هوا از ۳ به ۴ تغییر می کند ، لذا باید ۳ بیت عوض گردد که زمان زیادی می برد و احتمال خطا وجود دارد.

۳ ----> ۰۰۱۱

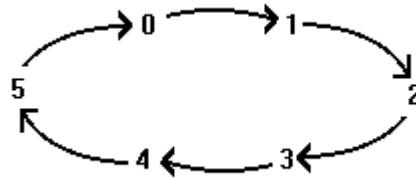
۴ ----> ۰۱۰۰

### (۲) در تبدیل اطلاعات آنالوگ به دیجیتال

انتقال اطلاعات از آنالوگ به دیجیتال بدین صورت انجام می گیرد که اگر برای مثال خطوط تغییرات زیاد باشد، ایجاد Noise می کند. لذا باید با سرعت کمتری انتقال دهیم تا Noise آن کمتر گردد. در مخابرات از سیستم Gray استفاده می شود تا Noise بوجود نیاید .

**نکته :** کد Gray در محاسبات به کار نمی رود ، چون اگر عمل محاسبه از رنج خارج شود امکان هیچگونه اصلاحی وجود ندارد . ولی در انتقال اطلاعات و شمارنده های واحد کنترل کاربرد بسیار وسیعی دارد . در مخابرات می توان از آنها برای کم کردن حجم کابلها استفاده نمود .

نکته : در ماشین Von Neuman حداکثر با ۶ عمل دستورات انجام می شود. پس شمارندی آن بصورت زیر عمل می کند و باید Gray باشد. یعنی هر عدد با عدد قبلی فقط در یک بیت اختلاف داشته باشد .



( شکل ۱ - ۲ )

### ۱-۴ - کد گری - افزایش سه تایی (EX-3 Gray)

در تبدیل عدد ۹ به عدد ۰ که در واقع عدد ۹ عدد قبل از ۰ است ، بیش از یک بیت اختلاف وجود دارد ( ۰۰۰۰ > ۱۱۰۱ ) لذا کد جدیدی بنام کد EX-3 Gray را معرفی می کنیم ، که در آن از EX-3 معادل Gray می گیریم .

### ۱-۵ - اعداد ممیز شناور (Floating Point)

اعداد ممیز شناور که با نمای علمی نمایش داده می شوند بصورت زیر می باشند.

توان

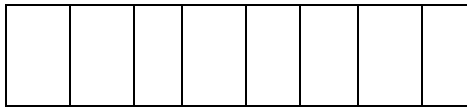
$$N = MB^E$$

پایه      مانتیس

$$125 = 125 \times 10^2$$

مثال :

نقطه اعشار وجود ندارد و علامتی فرضی است . فقط مانتیس و توان درج می شود . اگر پایه B یک عدد ثابت فرض شود . (B=Const) در ارائه مطلب هیچ کمبودی احساس نخواهد شد . اگر B را ثابت د ر نظر بگیریم در آنصورت هر عدد را می توان بصورت  $N = (M, E)$  نشان داد.



E . M

نقطه اعشار

در کامپیوترهای مختلف نیز یک قرار داد ثابت وجود دارد . بخشی از مکان حافظه را به مانتیس و بخشی را به توان اختصاصی می دهند . در کامپیوترها نقطه اعشار در سمت چپ قسمت M بصورت ثابت فرض می شود و مانند مثال زیر نقطه اعشار بصورت شناور نمی باشد . اعداد باید در کامپیوتر بصورت نرمال ذخیره شوند .

مثال :

$$125 = 1.25 \times 10^2 = 12.5 \times 10^1 = 125.0 \times 10^0$$

سؤال : چرا در داخل کامپیوتر از اعداد با ممیز شناور استفاده می شود ؟

چون در این روش دامنه نمایش اعداد وسیعتر می شود .

فرض کنید که ما چهار بیت فضا داشته باشیم که با این چهار بیت محدوده 0 تا 15 را در روش Fixed Point می توانیم نشان دهیم ولی در Floating Point به صورت زیر عمل می شود .

|      | Fixed Point | B = 2 | B = 16 |
|------|-------------|-------|--------|
| 0000 | 0           | 0     | 0      |
| 0001 | 1           | 0.25  | 0.25   |
| 0010 | 2           | 0.5   | 0.5    |
| 0011 | 3           | 0.75  | 0.75   |



|      |    |     |      |
|------|----|-----|------|
| 0100 | 4  | 0   | 0    |
| 0101 | 5  | 0.5 | 4    |
| 0110 | 6  | 1   | 8    |
| 0111 | 7  | 1.5 | 12   |
| 1000 | 8  | 0   | 0    |
| 1001 | 9  | 1   | 64   |
| 1010 | 10 | 2   | 128  |
| 1011 | 11 | 3   | 198  |
| 1100 | 12 | 0   | 0    |
| 1101 | 13 | 2   | 1024 |
| 1110 | 14 | 4   | 2048 |
| 1111 | 15 | 6   | 3072 |

(جدول ۱-۲)

در اعداد ممیز شناور (Floating Point) دامنه نمایش افزایش می یابد. اما دقت نمایش ثابین می آید. (یعنی اعداد مابین آن را نمی توان نمایش داد) و از بالا به ثابین یعنی از سمت اعداد کوچک به سمت اعداد بزرگ که می آیم، دقت اعداد کمتر می شود. ولی اعداد رنج ثابین دقت خوبی دارند.

\* در اعداد ممیز شناور (Floating Point) هر چه پایه (Base) بیشتر باشد، دامنه نمایش بیشتر اما دقت نمایش کمتر خواهد بود.

#### ۱-۵-۱- معایب اعداد ممیز شناور

عیب اول: هر چه مقدار پایه (Base) بیشتر باشد، دامنه وسیعتر و دقت اعداد کمتر است.

عیب دوم : یک عدد چندین فرم نمایش را داراست ، که کامپیوتر دچار مشکل می شود و مقایسه توسط کامپیوتر دشوار می گردد. برای رفع عیب دوم اعداد ممیز شناور عدد را بصورت نرمال در می آوریم و آنها را در داخل کامپیوتر ذخیره می کنیم .

### ۱-۵-۲- نرمال سازی

اولاً) وقتی عددی رامی خواهیم در کامپیوتر ذخیره کنیم باید بگونه‌ای باشد که مانتیس آن عددی بین صفر و یک باشد (یعنی  $0 < M < 1$ )

ثانیاً) اولین رقم بعد از ممیز صفر نباشد .

برای مثال اگر عدد  $2/5$  را بخواهیم در داخل کامپیوتر نشان دهیم می توانیم صورتهای مختلف زیر را برای آن داشته باشیم :

$$2.5 = 25 \times 10^{-1} = 0.25 \times 10^1 = 0.025 \times 10^2$$

طبق قاعده نرمال سازی دو مورد  $25 \times 10^{-1}$  و  $0.025 \times 10^2$  مورد قبول نیستند . زیرا در عدد  $25 \times 10^{-1}$  مانتیس باید عددی بین صفر و یک باشد ، که اینطور نیست و در عدد  $0.025 \times 10^2$  اولین رقم بعد از ممیز باید غیر صفر باشد که در اینجا هم رعایت نشده است .

نحوه نمایش اعداد Floating Point در IMB 360/370 به صورت زیر است .

اعداد ممیز شناور فضای ۴ بایتی یعنی ۳۲ بیتی را اشغال می کنند و مبنا (Base) عدد ۱۶ می باشد . از این ۳۲ بیت سمت چپ ترین بیت ، بیت علامت می باشد و ۷ بیت متعلق به توان و ۲۴ بیت باقی مانده متعلق به مانتیس عدد می باشد.

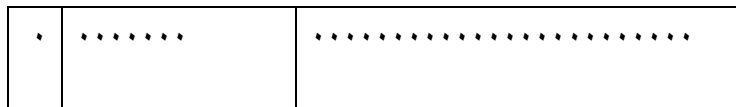
|  |       |        |
|--|-------|--------|
|  | 7 Bit | 24 Bit |
|--|-------|--------|





نکته : اگر دو عدد نرمال را با هم جمع کنیم یا در هم ضرب کنیم حاصل ممکن است نرمال نباشد و باید دوباره به نرمال تبدیل گردد. به همین خاطر در کامپیوتر، اعمال روی اعداد ممیز شناور زمان زیادی می گیرد.

سؤال : اگر مانتیس ما صفر باشد ماشین چگونه با آن برخورد می نماید؟



بیت                      E                      M

علامت

برای نرمال سازی حداکثر تا ۶ بار عمل Shift Left را انجام می دهد و سپس متوقف می شود. چون در عمل نرمال سازی به ازای هر شیفت به سمت چپ ، یک واحد از توان کم می گردد و در اقداماتی که برای نرمال سازی عدد صفر به عمل می آید، توان به سمت منفی تر شدن میل می کند. باید کاری کنیم که منفی ترین عدد شکل صفر باشد . برای رفع این مشکل باید توان بصورت

اریب دار، ذخیره گردد . اگر n بیت برای توان داشته باشیم نیاز است که اریب برابر  $2^{n-1}$  باشد .

$$۱۰۰۰۰۰۰ <-----> ۰۱۱۱۱۱۱$$

$$-(2)^{7-1} = -64 \qquad +(2)^{7-1} - 1 = 63$$

با اضافه کردن ۶۴ واحد به توان بصورت قدرمطلق نه بصورت علامت دار (چون در آن صورت از رنج خارج می گردد) می توان این مشکل را حل نمود . با این روش منفی ترین عدد ، عددی خواهد بود که همه بیتهایش صفر است و از دیدگاه کامپیوتر عدد به فرم  $0 \times 16^{-64}$  ، صفر واقعی می باشد .



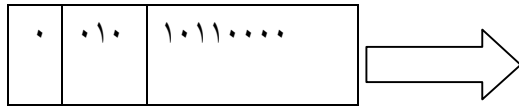


در مورد محاسبات بین اعداد اعشاری به مثال زیر توجه کنید . البته این محاسبات با توجه به کامپیوتر های IBM است :

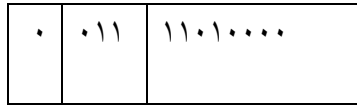
فرض کنید دو عدد  $A$  و  $B$  را می خواهیم باهم جمع کنیم . چون بیت های توان  $3$  تا است پس میزان اریب  $2 \times 2$  یعنی  $4$  خواهد بود . ابتدا باید توان ها را یکی کنیم . مسلما نمی توان عدد  $B$  را به چپ شیفت دهیم زیرا بیت پر ارزش آن از بین خواهد رفت به همین خاطر عدد  $A$  را به راست شیفت می دهیم . سپس حاصل جمع دو مانتیس را در  $C$  قرار می دهیم . در انتهای جمع چون رقم نقلی داریم ، کلیه ی اعداد مانتیس حاصل را به راست شیفت داده و یکی به توان اضافه می کنیم و سپس عدد  $C$  را نرمال سازی می کنیم .



A :



B :

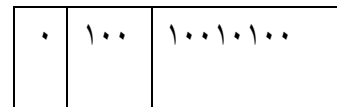
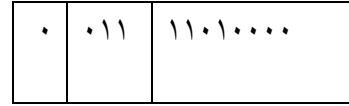


C :

A :



B :



تمرینات

- ۱- مزایا و معایب کد BCD را بیان نمایید؟
- ۲- علت افزایش ۶ واحد به کد BCD در برخی از عملیات ریاضی انجام گرفته بر روی این کد چیست؟ (با مثال توضیح دهید)
- ۳- مزایا و معایب کد افزایش سه تایی (EX-3) را بیان نمایید؟
- ۴- استفاده از ..... برای ..... مفید است. زیرا کدهای مجاور متوالی تنها در یک بیت با هم اختلاف دارند.
- ۵- مزایا و معایب کد Gray و همچنین موارد استفاده این کد را بیان نمایید؟
- ۶- لزوم استفاده از کد Gray EX-3 را بیان نمایید؟
- ۷- در n بیت اگر عددی به فرم علمی ذخیره شود، ..... نسبت به اعداد با ممیز ثابت در اختیار خواهد بود که البته ..... این روش کمتر است.
- ۸- نمایش اعداد به فرم ممیز شناور چه مزایایی دارد و چرا بر روی اعداد با ممیز شناور عمل نرمال سازی انجام می گیرد؟
- ۹- عملیات بر روی اعداد ممیز شناور در مقایسه با ممیز ثابت، توسط کامپیوتر سریعتر انجام می گیرد. چرا؟
- ۱۰- عدد  $(80A00000)_{16}$  موجود در حافظه کامپیوتر IBM 360 / 370 نشان دهنده چه عدد Floating Point در مبنای دسیمال می باشد؟

## ۲- مدارات جمع کننده

در طراحی مدار جمع کننده کامپیوتر معمولاً از روش متمم دو (2's) استفاده می شود و باید مدار طراحی شده دارای سرعت کافی و بالایی باشد. زیرا تمامی عملیات جمع، تفریق و ضرب و تقسیم توسط این مدار انجام خواهد گرفت. جمع کننده یک مدار ترکیبی است و هر مدار ترکیبی را می توان در دو سطح پیاده سازی نمود. هر چه تعداد سطوح کمتر گردد عمل جمع سریعتر صورت می گیرد.

اگر چه می توان از لحاظ تئوری برای طراحی جمع کننده  $n$  بیتی آنرا در دو سطح (سریعترین حالت ممکن) پیاده سازی نمود. اما از لحاظ عملی امکان پذیر نیست. زیرا برای طراحی چنین مداری، قطعاً به گیتی با  $n$  ورودی نیاز خواهد بود که اگر  $n$  عدد بزرگی باشد، عملاً گیتهایی با ورودی بالا کمتر وجود دارند. به این خاطر نمی توان مدار جمع کننده را برای حالتی که  $n$  عدد بزرگی باشد در دو سطح پیاده سازی نمود. بدین جهت مجبوریم که دو عدد  $n$  بیتی را به گروههای  $m$  بیتی تقسیم کنیم و سپس با هم جمع ببندیم و رقم نقلی (Carry) هر گروه را به گروه بعدی منتقل کنیم.

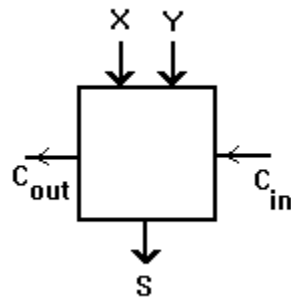
## انواع جمع کننده ها

## ۱-۲: طراحی جمع کننده کامل (Full Adder)

این مدار دو عدد تک بیتی را با هم جمع می کند و اگر رقم نقلی هم از قبل باشد در عملیات اثر می دهد.

$$S = x \oplus y \oplus c_{in}$$

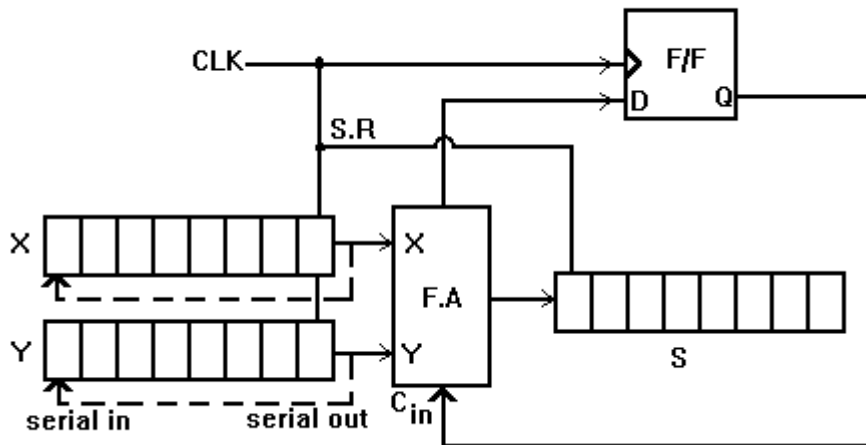
$$c_{out} = xy \oplus c_{in} (x \oplus y)$$



(شکل ۱-۲)

۲-۲: طراحی جمع کننده سریال (Serial Adder)

این مدار برای جمع دو عدد  $n$  بیتی با حداقل قیمت بکار می رود ولی زمان زیادی مصرف می کند.



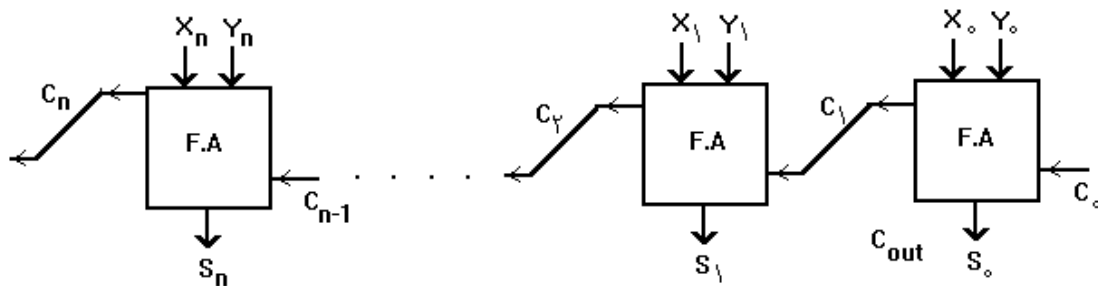
(شکل ۲-۲)

\* هر بار که  $clk$  زده می شود، یک شیفت به راست در ثباتهای  $S, Y, X$  صورت می گیرد.

\* اگر میزان تاخیر F.A برابر  $d$  باشد و میزان تاخیر فلیپ فلاپ (F/F) برابر  $D$  باشد آنگاه برای  $n$  بیت، جمع زمانی معادل  $n(d+D)$  صرف خواهد شد. پس کمترین هزینه را داریم ولی زمان زیادی مصرف می شود.

\* برای اینکه محتوای ثباتهای  $y, x$  از بین نرود، باید وقتی شیفت به راست می دهیم دوباره Serial out - را به Serial in وصل کنیم تا دوباره در انتهای عملیات عدد  $x, y$  را بطور کامل داشته باشیم.

۳-۲: طراحی جمع کننده با رقم نقلی پله‌ای (Ripple Carry Adder)



(شکل ۳-۲)

$$d_A = \text{زمان تأخیر در فلیپ فلاپ اول}$$

هر F.A پس از اعمال ورودیها بعد از یک  $d_A$  خروجی را به ما می دهد. برای  $n$  بیت، زمانی معادل  $nd_A$  نیاز می باشد. زیرا هر F.A باید منتظر رقم نقلی F.A قبلی باشد و سرعت در این روش نسبت به جمع کننده سریال بیشتر است و تعداد F.A ها برای یک عدد  $n$  بیتی برابر  $n$  تا می باشد، پس هزینه آن بیشتر از جمع کننده سریال می شود.

۲-۴ : طراحی جمع کننده با پیش بینی رقم نقلی (Carry Lookahead Adder)

اگر بتوان رقم نقلی را پیش بینی کرد ، دیگر نیازی نیست که منتظر بمانیم تا رقم نقلی F.A قبلی به F.A بعدی برسد. بنابراین ، این روش که همان جمع کننده با پیش بینی رقم نقلی می باشد را پیشنهاد می کنیم تا عملیات جمع سریعتر انجام گیرد .

| x | y | $c_{out}$ |   |
|---|---|-----------|---|
| ۰ | ۰ | ۰         | ۱ |
| ۰ | ۱ | $c_{in}$  | ۲ |
| ۱ | ۰ | $c_{in}$  | ۳ |
| ۱ | ۱ | ۱         | ۴ |

حالت اول

این حالت نیاز به رقم نقلی ندارد و اصطلاحاً در این حالت رقم نقلی دهن می شود.

حالت دوم و سوم

این حالت وابسته به رقم نقلی می باشد و اصطلاحاً رقم نقلی انتشار پیدا می کند .

$$(P_i = \text{Propagated Carry})$$

حالت چهارم

این حالت نیاز به رقم نقلی ندارد چون خودش رقم نقلی را بطور خودکار تولید می

نماید و اصطلاحاً در این حالت رقم نقلی تولید می شود .

$$(G_i = \text{Generated Carry})$$

وضعیت انتشار ۱  $P_i = x_i \oplus y_i$

وضعیت تولید ۲  $G_i = x_i y_i$

۳  $c_{out} = xy + (x \oplus y) c_{in}$  داشتیم

در دو سطح طراحی می گردد.  $C_i = G_i + P_i C_{i-1} \rightarrow$

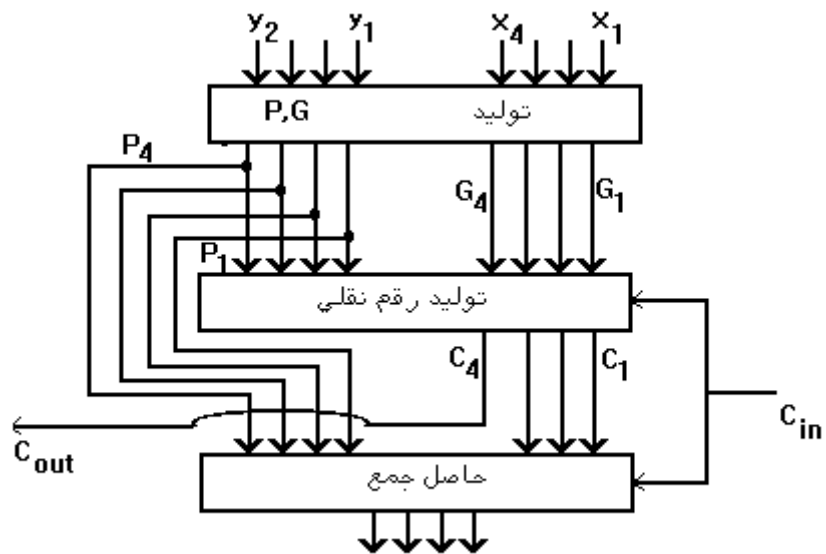
$C_0 = c_{in}$

$C_1 = G_1 + P_1 c_{in}$

$C_2 = G_2 + P_2 C_1 = G_2 + P_2 (G_1 + P_1 c_{in}) = G_2 + P_2 G_1 + P_2 P_1 c_{in}$

$C_3 = G_3 + P_3 C_2 = G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 c_{in}) = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 c_{in}$

$C_4 = G_4 + P_4 C_3 = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 c_{in}$



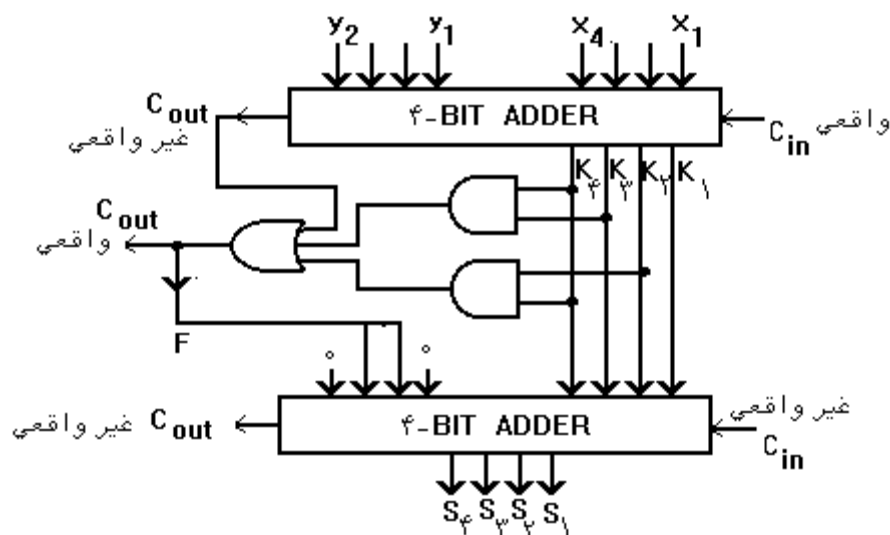
(شکل ۲-۴)

$S_i = x_i \oplus y_i \oplus c_{in} = P_i \oplus c_{in}$

در مجموع مدار ما دارای چهار سطح است. برای یک مدار ۱۰ بیتی هم نیاز به چهار سطح می باشد. می دانیم که هر F.A دو سطحی است، پس کل این مدار به اندازه دو F.A تأخیر دارد

بنابراین از نظر زمانی این روش سریعتر است. اما اگر این مدار را برای ۱۰۰ بیت طراحی کنیم ممکن است در مکان تولید رقم نقلی مشکل پیش آید، زیرا که ما نیاز به گیت‌هایی با ۱۰۰ ورودی داریم که این کارمشکلی است چون گیتها نهایتاً دارای ۸ ورودی می باشند. پس تعداد سطوح در مکان تولید رقم نقلی زیاد می شود که زمان تأخیر یا delay این مدار جمع کننده افزایش می یابد.

مثال: جمع کننده‌ای برای جمع دو عدد بی سی دی (BCD) طراحی کنید. (۴ بیتی)



(شکل ۲-۵)

در صورتی حاصل عملیات جمع، BCD نیست که یکی از اعداد 10, 11, 12, 13, 14, 15 حاصل شود یا در حاصل عملیات  $C_{out}$  رخ دهد که در این دو صورت تابع F فعال می شود.

نکته: هر دو Carry out، غیر واقعی هستند و تابع F رقم نقلی ماست.

$$\frac{K_4 K_3}{K_2 K_1}$$



|    |    |    |    |    |
|----|----|----|----|----|
|    | ۰۰ | ۰۱ | ۱۱ | ۱۰ |
| ۰۰ |    |    | ۱  |    |
| ۰۱ |    |    | ۱  |    |
| ۱۱ |    |    | ۱  | ۱  |
| ۱۰ |    |    | ۱  | ۱  |

$$F = K_4 \bar{K}_3 K_2 \bar{K}_1 + K_4 \bar{K}_3 K_2 K_1 + K_4 K_3 \bar{K}_2 \bar{K}_1 + K_4 K_3 \bar{K}_2 K_1 + K_4 K_3 K_2 \bar{K}_1 + K_4 K_3 K_2 K_1$$

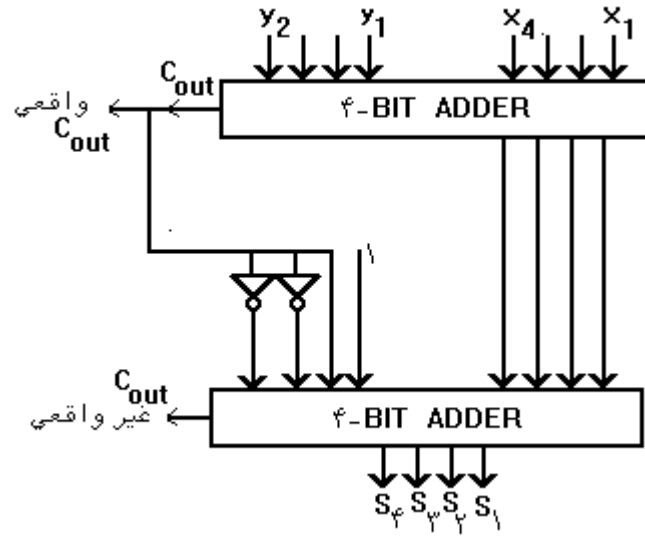
$$F = K_4 K_3 + K_4 K_2 + C_{OUT}$$

مثال : جمع کننده‌ای برای جمع دو عدد افزایش سه تایی (Execcss-3) طراحی کنید ؟

حل : در این حالت اگر پس از عملیات جمع  $C_{out}$  داشته باشیم ۳ واحد به حاصل عملیات

اضافه می‌کنیم و اگر  $C_{out}$  نداشته باشیم ۳ واحد از حاصل عملیات کم می‌کنیم .

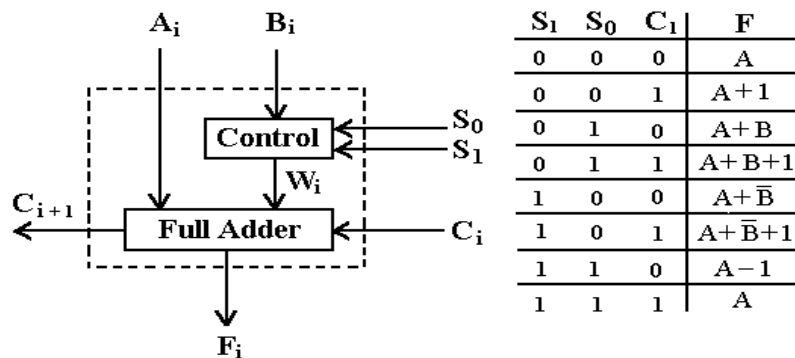
|     |                  |                 |           |   |
|-----|------------------|-----------------|-----------|---|
| عدد | نمایش در مبنی دو |                 |           |   |
| +3  | ۰                | ۰               | ۱         | ۱ |
| -3  | ۱                | ۱               | ۰         | ۱ |
|     | $\bar{C}_{out}$  | $\bar{C}_{out}$ | $C_{out}$ | ۱ |

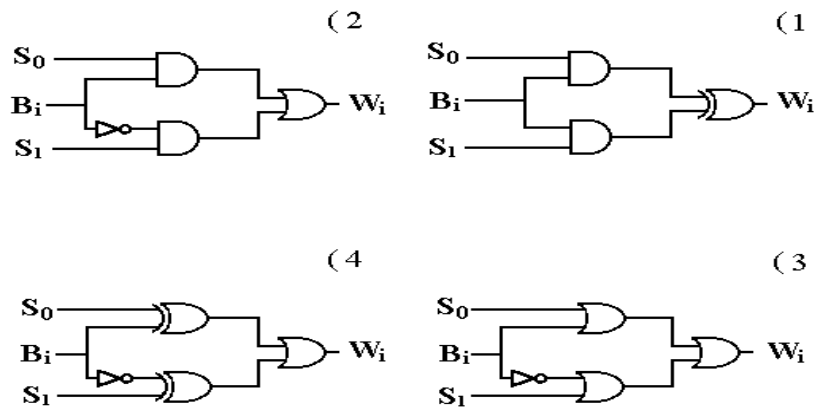


(شکل ۶-۲)

تمرینات

- ۱- مداری برای جمع دو عدد BCD به روش Pipelining طراحی نمایید؟
  - ۲- مداری برای جمع دو عدد EX-3 به روش Pipelining طراحی نمایید؟
  - ۳- چگونه می توان جمع کننده‌ای برای جمع دو عدد Gray طراحی نمایید؟
  - ۴- برای جمع دو عدد n بیتی می توان با ترکیب دو سطحی سریعترین مدار را پیاده سازی نمود. اما این ایده در عمل مورد استفاده قرار نمی گیرد . چرا؟
  - ۵- برای ساخت یک ALU از بلوکهایی مشابه بلوک زیر استفاده می شود .
- c1 بیت Gray متصل به اولین بلوک است و S0 , S1 برای تمام بلوکها یکسان است . مدار کنترل باید چگونه باشد تا توابع زیر را بتوان تولید کرد .





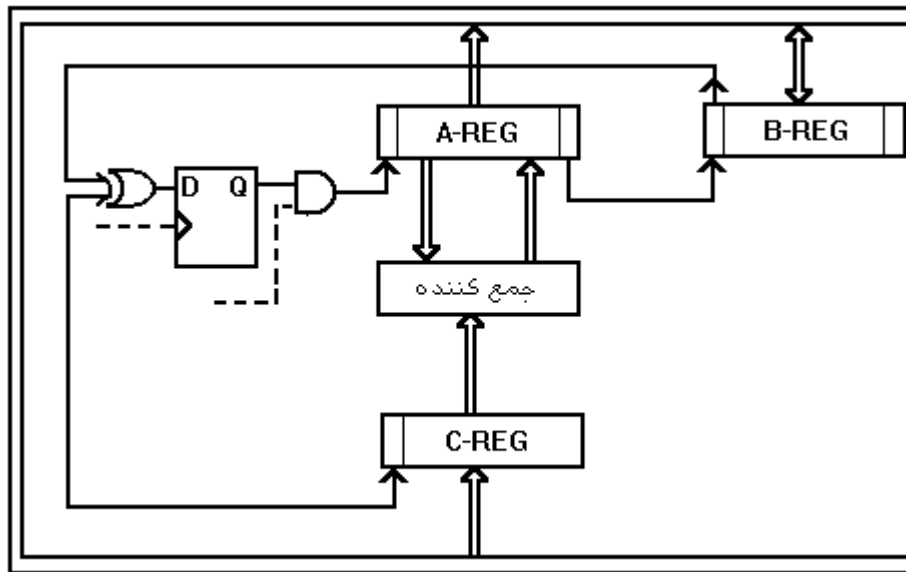
### ۳- مدارات ضرب کننده

در داخل کامپیوتر عمل ضرب توسط جمع های متوالی انجام می شود . برای عمل ضرب از 3 ثبات استفاده می شود . وقتی دو عدد  $n$  بیتی در هم ضرب می شوند به یک فضای  $2n$  بیت نیاز دارند.

#### ۳-۱: ضرب کننده برای دو عدد قدر مطلق علامت

چون ضرب دو عدد قدر مطلق علامت برخلاف جمع آنها دارای مدار و عمل ساده تری است . لذا ابتدا آنرا توضیح می دهیم تا با الگوریتم عمل آشنا تر شوید . در این عمل که با الهام از روش ضرب دستی به وقوع می پیوندد ، باید برای عمل ضرب برای دو عدد  $n$  بیتی یک ثبات  $2n$  بیتی را در نظر بگیریم و برای کل عمل ضرب باید  $n$  بار شیفت به راست صورت گیرد .

#### مدار ضرب کننده برای ضرب دو عدد قدر مطلق علامت

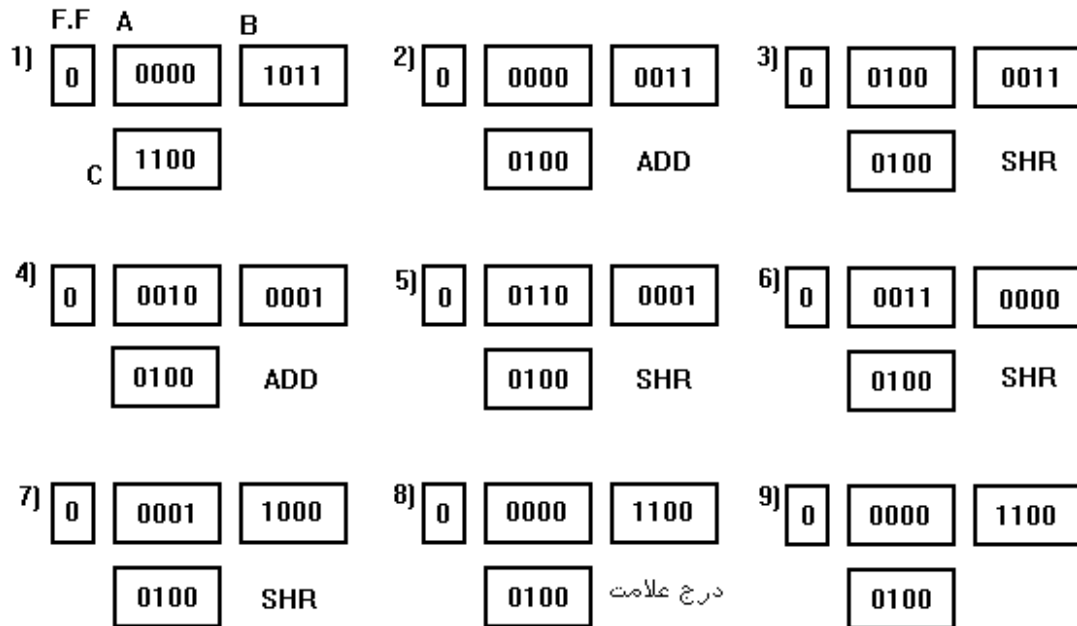


(شکل ۳-۱)

ابتدا به بیت کم ارزش عدد دوم (ثبات B) توجه می شود، اگر یک بود ثبات C و ثبات A را با هم جمع می کند و حاصل (ثبات B, A) به سمت راست شیفت داده می شود. حاصل عملیات که  $2n$  بیتی می باشد، در دو ثبات B, A قرار می گیرد.

در فلیپ فلاپ، بیت علامت حاصل ضرب که از یای انحصاری کردن بیت علامت ثبات B با C بدست می آید را ذخیره می کنیم و سپس بیت علامت ثبات B و ثبات C را Clear می کنیم یعنی قدر مطلق می گیریم. سپس طبق الگوریتم قبل در هم ضرب می نماییم و در پایان هم بیت علامت ضرب را در مکان مربوطه قرار می دهیم.

مثال: دو عدد قدر مطلق علامت 3-، 4- را به کمک مدار ضرب کننده چهار بیتی به روش قدر مطلق علامت در هم ضرب نمایید و کلیه مراحل را نشان دهید؟



به تعداد ShR که در مدار انجام می شود بیت داریم . چون اعداد چهار بیتی هستند باید چهار مرتبه شیفت به راست داشته باشیم ، قدم آخر نیز تأثیر بیت علامت است که در فلیپ فلاپ D نگهداری شده است .

### ۲-۳ : ضرب دو عدد متمم 2 یا 2's

یک روش این است که مدار همان مدار سخت افزاری قبلی باشد و با یک کارنرم افزاری دو عدد را از متمم 2 به قدر مطلق علامت تبدیل کرده و پس از عمل ضرب عدد قدرمطلق را به متمم 2 تبدیل نمود که چون این عملیات وقت گیر می باشد پس به صرفه نیست.

روش دیگر اینکه مداری طراحی کنیم که عیناً دو عدد متمم دو را در هم ضرب کند . دو عدد با توجه به علامتشان دارای چهار حالت نسبت به هم می باشند:

$$x \times y = ?$$

$$X = x_m \dots \dots \dots x_1 x_0$$

$$Y = y_n \dots \dots \dots y_1 y_0$$

- 1)  $x_n = 0, y_n = 0$
- 2)  $x_n = 1, y_n = 0$
- 3)  $x_n = 0, y_n = 1$
- 4)  $x_n = 1, y_n = 1$

حالت اول : هر دو عدد مثبت باشند (  $y_n = 0, x_n = 0$  )

که این روش عیناً براساس ضرب دو عدد قدر مطلق علامت حل می شود .

حالت دوم : اگر  $x_n$  منفی و  $y_n$  مثبت باشد (  $y_n = 0, x_n = 1$  )

این حالت را با یک مثال دنبال می کنیم.

مثال: حاصلضرب دو عدد 3-، 5- را به روش دستی نمایش دهید؟

حل : چون دو عدد چهار بیتی هستند به فضای 8 بیتی برای نمایش حاصلضرب نیاز داریم.

|            |     |
|------------|-----|
| ۱۱۰۱       | -3  |
| ۰۱۰۱ ×     | ۵ × |
| ۰۰۰۰۱۱۰۱   | -15 |
| ۰۰۰۰۰۰۰۰   |     |
| ۰۰۱۱۰۱۰۰   |     |
| ۰۰۰۰۰۰۰۰ + |     |
| ۰۱۰۰۰۰۰۱   |     |

مشاهده می شود که حاصل عملیات نادرست می باشد . علت اینست که چون عدد منفی از یک مکان 4 بیتی به مکان 8 بیتی منتقل می شود، می بایست آن عدد منفی را بسط علامت بدهیم . پس داریم :



$$\begin{array}{r}
 1101 \\
 0101 \times \\
 \hline
 11111101 \\
 \dots\dots\dots \\
 11110100 \\
 \dots\dots\dots + \\
 \hline
 111110001
 \end{array}$$

$$(11110001)_2 = -(00001111)_2 = -15$$

حال می خواهیم این عملیات را در کامپیوتر انجام دهیم .

در واقع ما نیازی به فلیپ فلاپ قبلی نداریم . زیرا این فلیپ فلاپ برای ضبط بیت علامت بود که به آن نیازی نیست ، چون هر بار که حاصل ثبات A را در ثبات C می ریزیم ، اگر نیاز به شیفت به راست باشد ، ثبات C را بسط علامت می دهیم و این عمل توسط سیمی که بیت علامت را در خودش کپی می کند صورت می گیرد .

**نکته:** در عمل ضرب هنگامی که می خواهیم شیفت به راست انجام دهیم این کار حتماً باید با حفظ علامت صورت گیرد .

**حالت سوم:** اگر  $X_n$  مثبت و  $Y_n$  منفی باشد (  $y_n = 0, x_n = 1$  )

این حالت را با یک مثال دنبال می کنیم.

**مثال:** حاصلضرب دو عدد 5 , 3- را به روش دستی نمایش دهید؟

**حل:** چون دو عدد چهار بیتی هستند ، به فضای ۸ بیتی برای نمایش حاصلضرب نیاز داریم .

$$\begin{array}{r} 1101 \times \\ \hline \end{array} \quad \begin{array}{r} -3 \times \\ \hline -15 \end{array}$$

.....101

.....

....10100

..101000 +

..1000001

مشاهده می شود که حاصل عملیات نادرست می باشد . برای رفع این مشکل باید به ازای آخرین بیت از عدد دوم ، در صورت یک بودن به جای عمل جمع ، عمل تفریق را انجام دهیم .

1101

1101 ×

.....101

.....

....10100 +

.....1001

..101000 -

.....11110001

$$(11110001)_2 = -(00001111)_2 = -15$$

حال می خواهیم این عملیات را در کامپیوتر انجام دهیم.

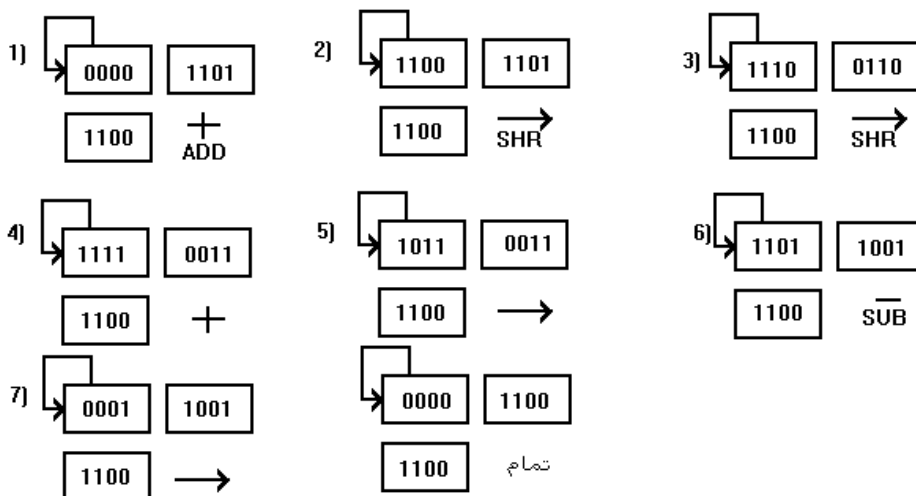
باید به ازای آخرین بیت از ثبات B در صورت یک بودن به جای عمل جمع ، عمل تفریق را انجام دهیم و نیازی به تغییرات اساسی نیست .

حالت چهارم: حالتی که  $X_n$  و  $Y_n$  هر دو منفی باشند. ( $X_n = 1, Y_n = 1$ )

حالت چهارم ترکیبی از موارد دوم و سوم است. با توجه به بحثهای انجام شده می توان اینطور نتیجه گرفت که برای ضرب دو عدد متمم ۲، بایستی بدین صورت عمل کنیم.

همواره به سمت راست ترین بیت ثبات B نگاه می کنیم. اگر این بیت یک باشد محتوای ثبات A را به ثبات C اضافه می کنیم و سپس زوج ثبات B و C را به صورت حسابی (بصورت سخت افزاری) به راست انتقال می دهیم و به ازاء بیت صفر، فقط زوج ثبات B و C را به صورت حسابی به سمت راست شیفٹ می دهیم. به ازای تمامی بیتهای ثبات B این کار را انجام می دهیم. چنانچه آخرین بیت از ثبات B یک باشد به جای عمل جمع، عمل تفریق را به کار می بریم. (در هر لحظه اگر در هنگام انتقال به راست سرریز رخ دهد یک از سمت چپ ثبات C وارد می شود).

مثال: می خواهیم دو عدد -3, -4 را به صورت 2's توسط مدار ضرب کننده ۴ بیتی با هم ضرب نماییم.

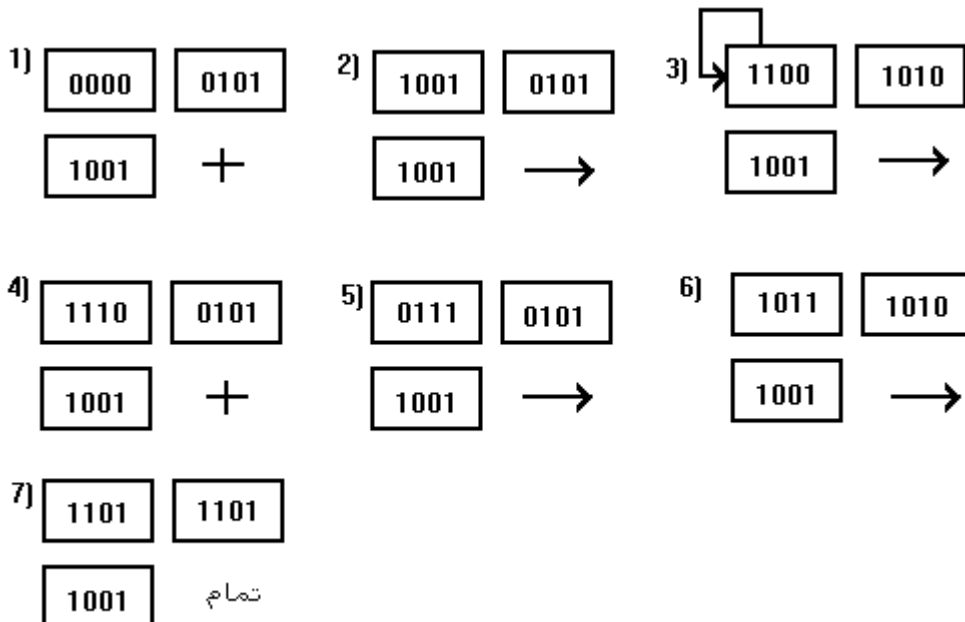


نکته : در مرحله آخر ، چون LSB برابر ۱ است پس عمل تفریق را انجام می دهیم و برای این کار از عدد واقع در ثبات زیرین،  $2^s$  می گیریم و به جای تفریق، عمل جمع را انجام می دهیم .

نکته : در متمم 2 بروز رقم نقلی هیچ گونه اهمیتی ندارد .

مثال : دو عدد A, B را در 4 بیت ضرب نمایید؟

$$\begin{array}{r} -7 \ A \times \\ +5 \ B \\ \hline -35 \end{array}$$



در این مثال در مرحله ی ۵ سرریز رخ داده است بنابراین هنگام انتقال به سمت راست از سمت چپ ثبات C عدد یک وارد می شود .

نکته : اگر در جمع دو عدد مثبت ، حاصل عملیات منفی شود ، در این حالت نیز سرریز رخ داده است که در اینصورت از سمت چپ ثبات C مقدار صفر وارد می شود .

۳-۳: روشهای سریع ضرب

هر چه تعداد بیت های صفر زیاد باشد محاسبه در روشهای قبل آسانتر است . در پیاده سازی ضرب به روش سریع ، به ازای اولین بیت یک ، عمل تفریق و به ازای اولین بیت صفر بعد از مجموعه ایی از یکها ، عمل جمع را انجام می دهیم و به راحتی از روی بقیه یک ها عبور می کنیم .

$$01111 \text{ -----} > 10000 - 00001$$

علت این کار، این است که در روشهای قبلی به ازای هر بیت ، باید هم عمل جمع و هم عمل شیفیت را انجام دهیم . ولی در این روش ما فقط یک بار عمل جمع و یک بار عمل تفریق را انجام می دهیم و به ازای بیت های یک فقط عمل شیفیت به راست را انجام می دهیم. بنابراین عمل ضرب سریعتر از روشهای قبل انجام می شود .

۱-۳-۳: روش ضرب Booth

برای تشخیص دنباله ای از یکها باید ابتدا همه بیتها را چک کنیم تا دنباله ای از یکها را مشخص نماییم این عمل ، عمل پیچیده ای است . لذا شخصی بنام Booth الگوریتمی سخت افزاری را ارائه نمود که به راحتی می توانست این مشکل را حل نماید.

در این روش در هر لحظه به بیت جاری (بیت با ارزش کمتر ) و بیت ماقبل آن توجه می شود . این دو بیت ۴ وضعیت نسبت به یکدیگر دارند که در برخورد با هر یک مطابق الگوی زیر عمل می شود :

بیت ماقبل      بیت جاری

۰

۰

۱- در حالت اول انتقال به راست

۰

۲- در حالت دوم عمل تفریق

۱

۳- در حالت سوم عمل جمع

۱

۰

۴- در حالت چهارم انتقال به راست

۱

۱

چون در هر لحظه به بیت ماقبل نیاز می باشد ، پس یک F/F داریم که بیت ماقبل را در آن نگهداری می کنیم. بیتی که شیفت به راست داده می شود به F/F منتقل می شود (از ثبات B) و هر بار به بیت کم ارزش ثبات B (بیت جاری) و محتوای F/F که بیانگر بیت قبلی است نگاه می کنیم.

**نکته :** در شروع کار محتوای F/F برابر با صفر می باشد .

**مثال :** فرض کنید که عدد  $(00111100)_2$  داخل ثباتهای A و B قرار داشته باشد . با استفاده از الگوریتم Booth آن را تحلیل کنید ؟

### بیت جاری

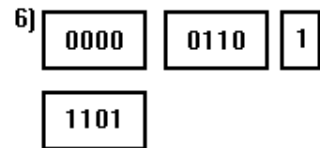
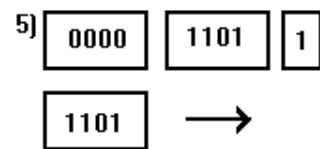
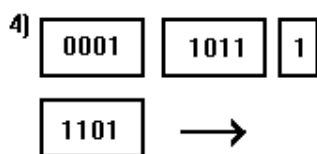
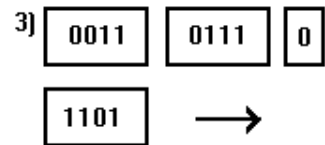
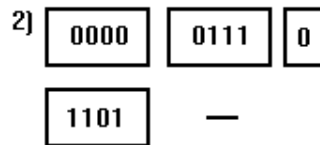
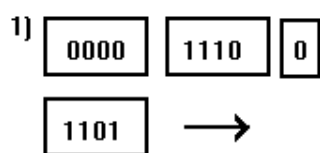
### F/F

- |   |   |   |
|---|---|---|
| ۰ | ۰ | در ابتدای در دنباله ای از صفرها هستیم پس انتقال به راست انجام می شود. |
| ۱ | ۰ | در آغاز دنباله ای از یکها هستیم پس عمل تفریق انجام می شود.            |
| ۱ | ۱ | در دنباله ای از یکها هستیم پس انتقال به راست انجام می شود.            |
| ۱ | ۱ | در دنباله ای از یکها هستیم پس انتقال به راست انجام می شود.            |
| ۱ | ۱ | در دنباله ای از یکها هستیم پس انتقال به راست انجام می شود.            |
| ۰ | ۱ | دنباله ای از یکها پایان یافت پس عمل جمع انجام می شود.                 |

در دنباله‌ایی از صفرها هستیم پس انتقال به راست انجام می شود.

مثال :  $\times -3$  را به روش الگوریتم Booth انجام دهید:

-2



در حل این مثال به روش الگوریتم Booth چون دو عدد ۴ بیتی است ، پس ۴ عمل Shift Right فقط یک عمل تفریق صورت می گیرد ولی با روش قبلی ۲ بار عمل جمع، یکبار عمل تفریق و چهار مرتبه عمل ShR صورت می گیرد .

مثال :  $\times -5$  به روش Booth عمل کنید.

+ 5

$$1) \begin{array}{r} 0000 \\ 1011 \end{array} \begin{array}{r} 0101 \\ - \end{array} \begin{array}{r} 0 \\ \end{array}$$

$$2) \begin{array}{r} 0101 \\ 1011 \end{array} \begin{array}{r} 0101 \\ \rightarrow \end{array} \begin{array}{r} 0 \\ \end{array}$$

$$3) \begin{array}{r} 0010 \\ 1011 \end{array} \begin{array}{r} 1010 \\ + \end{array} \begin{array}{r} 1 \\ \end{array}$$

$$4) \begin{array}{r} 1101 \\ 1011 \end{array} \begin{array}{r} 1010 \\ \rightarrow \end{array} \begin{array}{r} 1 \\ \end{array}$$

$$5) \begin{array}{r} 1110 \\ 1011 \end{array} \begin{array}{r} 1101 \\ - \end{array} \begin{array}{r} 0 \\ \end{array}$$

$$6) \begin{array}{r} 0011 \\ 1011 \end{array} \begin{array}{r} 1101 \\ \rightarrow \end{array} \begin{array}{r} 0 \\ \end{array}$$

$$7) \begin{array}{r} 0001 \\ 1011 \end{array} \begin{array}{r} 1110 \\ + \end{array} \begin{array}{r} 1 \\ \end{array}$$

$$8) \begin{array}{r} 1100 \\ 1011 \end{array} \begin{array}{r} 1110 \\ \rightarrow \end{array} \begin{array}{r} 1 \\ \end{array}$$

$$9) \begin{array}{r} 1110 \\ 1011 \end{array} \begin{array}{r} 0111 \\ 0 \\ \end{array}$$

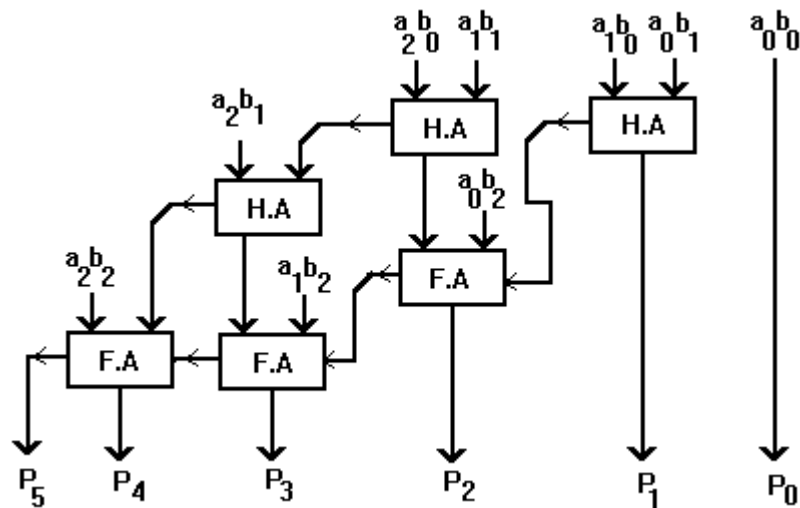
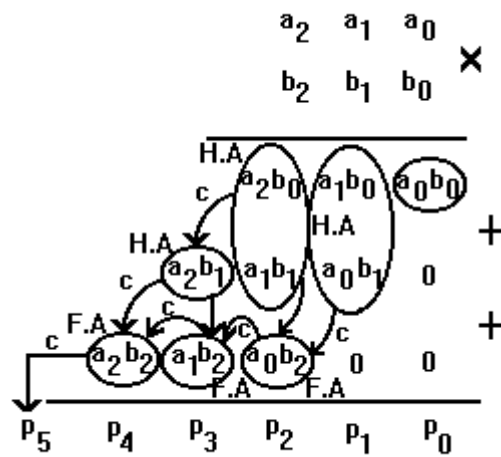
\* عیب این روش این است که اگر یک ها و صفرها بصورت یک در میان در ثبات B قرار گیرند مانند مثال فوق بدترین حالت را داریم . یعنی می بایست به ازای یک ها عمل تفریق و به ازاء صفرها عمل جمع را انجام دهیم .

\* روشهای ضرب سریع دیگری نیز وجود دارد که سرعت انجام کار آنها بیشتر است و برای مثال اگر n صفر متوالی مشاهده شود باید آنرا n شیفت صورت گیرد ، پس این روش سرعت بیشتری دارد . برای پیاده سازی این مدارات ضرب کننده نیاز به واحد کنترل می باشد که این از نظر هزینه به صرفه نیست.



۳-۳-۲: ضرب به کمک مدارات ترکیبی (ضرب آرایه‌ای)

فرض کنید که می‌خواهیم دو عدد ۳ بیتی را در هم ضرب کنیم بنابراین داریم:



(شکل ۳-۲)

نکته: بطور کلی اگر بخواهیم در ضرب آرایه‌ای دو عدد  $n$  بیتی را در هم ضرب کنیم، به  $n$

عدد H.A و  $n^2 - 2n$  عدد F.A نیاز خواهیم داشت.

تمرینات

۱- دو عدد ۱۳- و ۱۵ را به روش قدر مطلق علامت در هم ضرب نمایید و کلیه مراحل را نشان دهید؟

۲- دو عدد ۱۳- و ۵- را به روش متمم ۲ را در هم ضرب نمایید و کلیه مراحل را نشان دهید؟

۳- دو عدد ۷- و ۳ را در ۴ بیت به فرم متمم ۲ نمایش دهید و سپس براساس روش Booth با هم ضرب نمایید. کلیه مراحل را طی عمل ضرب نشان دهید؟

۴- در نظر بگیرید که دو عدد  $A = +6$  و  $B = -7$  را بخواهیم به روش Booth را در هم ضرب نماییم، در هر یک از ۲ مرحله زیر فقط بنویسید که چه تعداد عمل جمع و چه تعداد عمل تفریق لازم است؟ (ثباتها را ۴ بیتی در نظر بگیرید)

الف)  $A \times B$  (A ضرب شونده و B ضرب کننده می باشد).

ب)  $B \times A$  (B ضرب شونده و A ضرب کننده می باشد).

۵- یک ضرب کننده آرایه‌ای طراحی کنید که دو عدد ۴ بیتی را ضرب کند. از گیت‌های AND و جمع کننده های دو دویی استفاده کنید؟

۶- الگوریتمی را به شکل فلوچارت برای عمل ضرب به روش Booth رسم نمایید؟

۷- درستی یا نادرستی عبارات زیر را مشخص نمایید؟

- پیچیدگی ضرب دو عدد، به انتخاب کد جهت نمایش آنها بستگی دارد.

- در یک کامپیوتر که کلیه ی ثباتهای آن n بیتی هستند، در محاسبه ی دو عبارت زیر نمی توان گفت که سرعت محاسبه ی عبارت دوم بیشتر از سرعت محاسبه ی عبارت اول است.

$$(1) M \times K \quad (2) N \times K \quad (3) N < M$$

- در کلیه روشهای ضرب ، عمل ضرب  $n$  ، Shift و حاصل  $2n$  بیت خواهد شد.
- در ضرب دو عدد متمم  $2$  ، علامت دو عدد نیز مانند سایر بیتها پردازش می شود.

#### ۴- مدارات تقسیم کننده

عمل تقسیم هم مانند عمل ضرب الهام گرفته از روش تقسیم دستی است. در عمل تقسیم مقسوم از لحاظ تعداد ارقام معمولاً از مقسوم علیه بزرگتر است. برای کم کردن هزینه ها بهتر است مدار بصورت یکتارچه باشد. بطوریکه اگر مقسوم علیه ما  $n$  بیتی باشد ، باید مقسوم ما در ثبات  $2n$  بیتی ذخیره گردد. پس تا اینجا به  $3$  ثبات  $n$  بیتی نیازمندیم. همچنین ثبات ما باید قابلیت شیفت به چپ (ShL) را داشته باشد. چون ما خودمان در عمل تقسیم دستی هر لحظه یک بیت به جلو می رویم ولی در ثبات باید یک شیفت به سمت چپ داشته باشیم و نیز به یک تفریق کننده در مدار نیاز داریم تا عمل تفریق را انجام دهد. همچنین قبل از عمل تقسیم باید یک مقایسه کننده وجود داشته باشد تا مشخص نماید که مقسوم کوچکتر از مقسوم علیه است یا خیر.

۱-۴: تقسیم به روش مقایسه‌ای

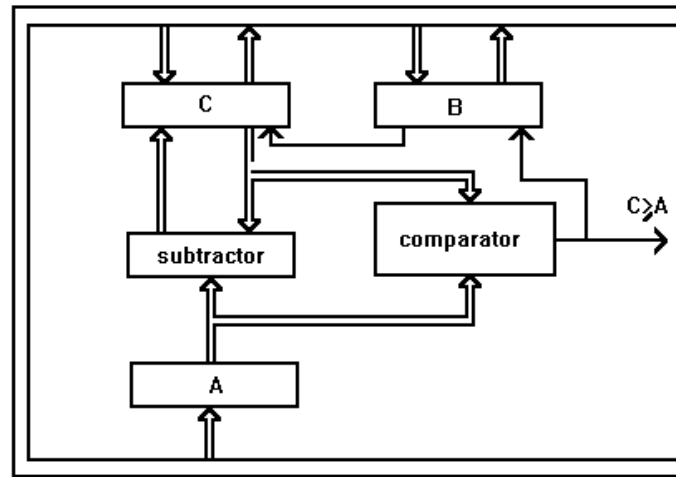
برای عمل تقسیم به روش مقایسه‌ای به موارد زیر نیازمندیم:

۱- به ۳ ثابت  $n$  بیتی

۲- ثابت‌ها قابلیت شیفت به چپ داشته باشند.

۳- مدار تفریق کننده

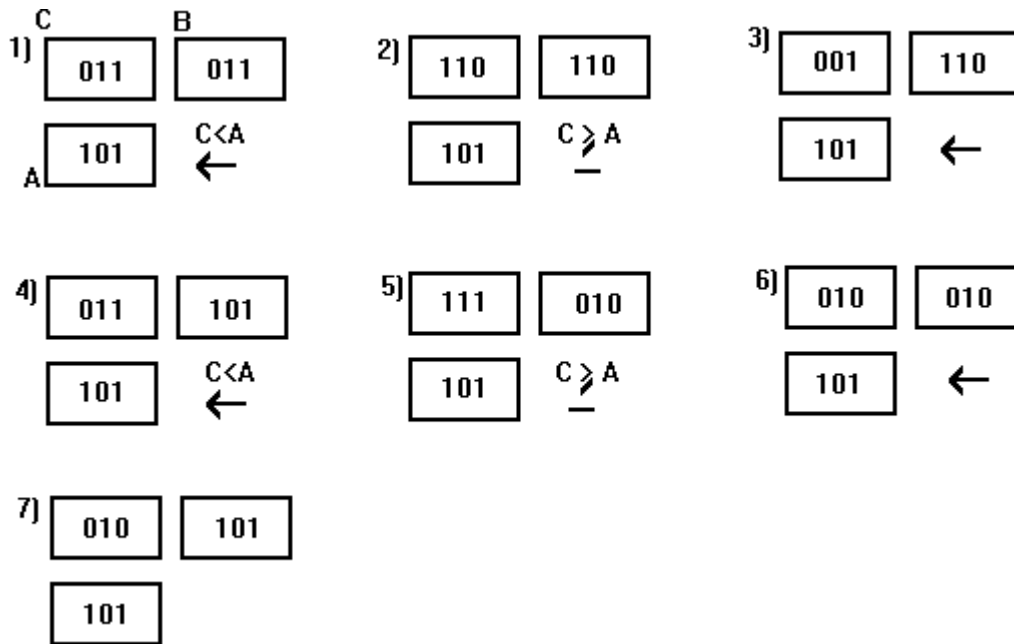
۴- مدار مقایسه کننده



(شکل ۱-۴)

در این روش مقسوم در زوج ثابت  $C$  و  $B$  قرار می‌گیرد و مقسوم علیه در ثابت  $A$  قرار دارد و عمل تفریق از سمت ثابت  $C$  صورت می‌گیرد و بعد از هر بار عمل تفریق نیز یک شیفت به چپ صورت می‌گیرد. البته قبل از عمل تفریق نیز واحد کنترل از طریق مقایسه گرمی فهمد که عدد کوچکتر و قابل تقسیم است، یا نه و یک پیغام به واحد کنترل می‌دهد و واحد کنترل دستور عمل تفریق را صادر می‌کند و خارج قسمت تقسیم در ثابت  $B$  و باقی مانده در ثابت  $C$  قرار می‌گیرد.

مثال: عمل تقسیم  $27 \div 5$  را انجام دهید؟ (ثابت‌ها همگی ۳ بیتی هستند و اعداد نیز بدون علامت هستند.)



**نکته:** پس در تقسیم تعداد ShL برابر  $n + 1$  می باشد که  $n$  تعداد بیتها است و به ازاء آخرین ShL فقط ثبات B، ShL می گردد و ثبات C را ShL نمی دهیم.

**نکته:** اگر در عمل تقسیم اولین مقایسه ما موفقیت آمیز باشد ( یعنی  $C \geq A$  باشد ) نتیجه ی حاصل حتماً سرریز خواهد بود. چون همیشه از تعداد بیتهای ثبات که  $n$  می باشد، عمل ShL یکی بیشتر است، یعنی  $n+1$ ، ShL صورت می گیرد. پس سرریز رخ می دهد. در چنین مواقعی راه حلی نیست زیرا جواب ما از ثبات ما یک بیت بیشتر است و ثبات نمیتواند آنرا نشان دهد.

**نکته:** به علت وجود مقایسه گر هزینه مدار زیاد می شود و معمولاً در مدارات کامپیوتر از این شیوه استفاده نمی شود. اگر  $n$  زیاد باشد هزینه مقایسه گر از هزینه کل مدار بیشتر است.

**نکته:** برای این که مقایسه کننده را از مدار خارج کنیم، از روش دیگری استفاده می کنیم که دو عدد را از هم کم کرده و با وجود رقم نقلی، بزرگتر، کوچکتر و مساوی را تشخیص می دهیم و در واقع از همان تفریق کننده برای عمل مقایسه استفاده می کنیم.

**نکته:** در کامپیوترها برای اینکه هزینه ی کمتری صرف شود، باید ثباتها قابلیت ShL و ShR را داشته باشند و عمل جمع و تفریق توسط یک مدار که هم جمع کننده و هم تفریق کننده است

صورت گیرد. در نتیجه عمل ضرب و تقسیم توسط یک مدار صورت گیرد و در هنگام ضرب، باید جمع کننده و  $ShR$  و در عمل تقسیم باید  $ShL$  و تفریق کننده داشته باشیم.

#### ۴-۲: تقسیم به روش Restoring

در این روش مقایسه گر را از مدار قبلی برداشته و برای مقایسه از تفریق کننده استفاده می نماییم و بدین صورت عمل می کنیم که هر بار جهت مقایسه  $C - A$  می گردد و حاصل به داخل ثبات  $C$  انتقال داده می شود و دو وضعیت پیش می آید.

الف - حاصل مقداری غیر منفی یعنی بزرگتر یا مساوی صفر باشد.

ب- حاصل مقداری منفی باشد یعنی کوچکتر از صفر باشد.

الف: حاصل مقداری غیر منفی (یعنی  $C \geq A$ ) باشد.

لذا می بایست طبق روش گذشته عمل  $C - A$  در ثبات  $C$  انجام گیرد که در اینجا انجام شد. پس از این مرحله کافی است که فقط یک عمل انتقال به چپ ( $ShL$ ) داشته باشیم و یک را در حین انتقال از سمت راست، به ثبات  $B$  وارد کنیم.

ب: حاصل مقداری منفی ( $C < A$ ) باشد.

اگر عملیات منفی باشد، نشان دهنده آن است که  $C < A$  بوده لذا نمی بایست عمل تفریق را انجام دهیم. در نتیجه لازم است مقدار قبلی ثبات  $C$  بازیابی ( $Restore$ ) شود. برای انجام اینکار لازم است محتوای ثبات  $A$  به ثبات  $C$  اضافه گردد. سپس یک واحد، زوج ثبات  $B$  و  $C$  را به سمت چپ انتقال می دهیم و صفر را از سمت راست ثبات  $B$  وارد می کنیم.

نکته: در این روش هزینه کمتر و مدار ساده تر بوده اما سرعت آن از روش قبلی کمتر است . یعنی بار محاسباتی بیشتری دارد و آن هم به خاطر عمل Restore می باشد.

نکته : در این روش در همه ی مراحل یک تفریق داشته و در بعضی مراحل یک جمع داریم پس سرعت تقریباً  $\frac{1}{2}$  برابر سرعت روش قبلی می شود .

#### ۳-۴: تقسیم به روش Non Restoring

وقتی عمل  $C - A$  را انجام دهیم حاصل به ثبات  $C$  می رود، در این حالت اگر حاصل مثبت باشد نیاز به عمل Restore نداریم ولی وقتی حاصل در ثبات  $C$  منفی شود، نیاز به عمل Restoring داریم .

حال این روش را بررسی می کنیم.

$$C-A+A$$

عمل Restoring وقتی منفی شود

سپس یک انتقال به چپ باید بدهیم که معادل دو برابر کردن حاصل عملیات می باشد.

$$(C-A+A)*2$$

در مرحله ی بعد نیز باید مقدار  $A$  از این عبارت کم کنیم .

$$(C-A+A)*2-A$$

تا برای مرحله بعد عمل  $C \geq A$  را بررسی نماییم.

$$\text{----> } (C-A+A)*2-A = 2C-A$$

بجای اعمال قبلی که در این روش گفته شده ، روش معادل زیر را انجام دهیم .

$$(C-A)*2+A=2C-A$$

یعنی اگر حاصل منفی بود ShL داده و در مرحله بعد عمل جمع A صورت می گیرد. پس دو رابطه ۱ و ۲ معادل هم می باشند.

مثال: تقسیم  $27 \div 5$  را به روش Restoring و Non - Restoring انجام دهید.

روش Restoring :

|   |     |     |     |   |  |     |     |     |   |  |     |     |     |   |
|---|-----|-----|-----|---|--|-----|-----|-----|---|--|-----|-----|-----|---|
| 1) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>011</td><td>011</td></tr><tr><td>101</td><td>—</td></tr></table>  | 011 | 011 | 101 | — | 2) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>110</td><td>011</td></tr><tr><td>101</td><td>+</td></tr></table> | 110 | 011 | 101 | + | 3) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>011</td><td>011</td></tr><tr><td>101</td><td>←</td></tr></table> | 011 | 011 | 101 | ← |
| 011   | 011 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101   | —   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 110   | 011 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101   | +   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 011   | 011 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101   | ←   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 4) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>110</td><td>110</td></tr><tr><td>101</td><td>—</td></tr></table>  | 110 | 110 | 101 | — | 5) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>001</td><td>110</td></tr><tr><td>101</td><td>←</td></tr></table> | 001 | 110 | 101 | ← | 6) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>011</td><td>101</td></tr><tr><td>101</td><td>—</td></tr></table> | 011 | 101 | 101 | — |
| 110   | 110 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101   | —   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 001   | 110 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101   | ←   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 011   | 101 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101   | —   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 7) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>110</td><td>101</td></tr><tr><td>101</td><td>+</td></tr></table>  | 110 | 101 | 101 | + | 8) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>011</td><td>101</td></tr><tr><td>101</td><td>←</td></tr></table> | 011 | 101 | 101 | ← | 9) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>111</td><td>010</td></tr><tr><td>101</td><td>—</td></tr></table> | 111 | 010 | 101 | — |
| 110   | 101 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101   | +   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 011   | 101 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101   | ←   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 111   | 010 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101   | —   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 10) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>010</td><td>010</td></tr><tr><td>101</td><td>←</td></tr></table> | 010 | 010 | 101 | ← | 11) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>010</td><td>101</td></tr><tr><td>101</td><td></td></tr></table> | 010 | 101 | 101 |   |  |     |     |     |   |
| 010   | 010 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101   | ←   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 010   | 101 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101   |     |     |     |   |  |     |     |     |   |  |     |     |     |   |

روش Non - Restoring :

|  |     |     |     |   |  |     |     |     |   |  |     |     |     |   |
|--|-----|-----|-----|---|--|-----|-----|-----|---|--|-----|-----|-----|---|
| 1) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>011</td><td>011</td></tr><tr><td>101</td><td>—</td></tr></table> | 011 | 011 | 101 | — | 2) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>110</td><td>011</td></tr><tr><td>101</td><td>←</td></tr></table> | 110 | 011 | 101 | ← | 3) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>100</td><td>110</td></tr><tr><td>101</td><td>+</td></tr></table> | 100 | 110 | 101 | + |
| 011  | 011 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101  | —   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 110  | 011 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101  | ←   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 100  | 110 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101  | +   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 4) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>001</td><td>110</td></tr><tr><td>101</td><td>←</td></tr></table> | 001 | 110 | 101 | ← | 5) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>011</td><td>101</td></tr><tr><td>101</td><td>—</td></tr></table> | 011 | 101 | 101 | — | 6) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>110</td><td>101</td></tr><tr><td>101</td><td>←</td></tr></table> | 110 | 101 | 101 | ← |
| 001  | 110 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101  | ←   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 011  | 101 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101  | —   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 110  | 101 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101  | ←   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 7) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>101</td><td>010</td></tr><tr><td>101</td><td>+</td></tr></table> | 101 | 010 | 101 | + | 8) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>010</td><td>010</td></tr><tr><td>101</td><td>←</td></tr></table> | 010 | 010 | 101 | ← | 9) <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>010</td><td>101</td></tr><tr><td>101</td><td></td></tr></table>  | 010 | 101 | 101 |   |
| 101  | 010 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101  | +   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 010  | 010 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101  | ←   |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 010  | 101 |     |     |   |  |     |     |     |   |  |     |     |     |   |
| 101  |     |     |     |   |  |     |     |     |   |  |     |     |     |   |



تمرینات

- ۱- دو عدد ۱۷ و ۵ را در چهار بیت به فرم بدون علامت نمایش دهید و سپس به کمک روش مقایسه‌ای بر هم تقسیم و کلیه مراحل را نشان دهید؟
  - ۲- دو عدد ۱۳ و ۵ را به روش Restoring بر هم تقسیم نمایید؟
  - ۳- دو عدد ۱۹ و ۵ را به روش Non Restoring بر هم تقسیم نمایید؟
  - ۴- الگوریتمی را به شکل فلوچارت برای عمل تقسیم به روش Restoring دو دویی ممیز - ثابت به دست آورید؟
  - ۵- الگوریتمی را به شکل فلوچارت برای عمل تقسیم به روش Non Restoring دو دویی ممیز- ثابت به دست آورید؟
  - ۶- آیا عبارت زیر درست است؟
- در عمل تقسیم قبل از آنکه محاسبه خاتمه یابد، سر ریز قابل تشخیص بوده و نیاز به ادامی عملیات نیست.

## ۵- اعمال اصلی در اعداد ممیز شناور

### ۱-۵ : عمل جمع در اعداد ممیز شناور

دو عدد صحیح را به راحتی می توان با هم جمع کرد. ولی دو عدد با ممیز شناور را به سادگی نمیتوان یا هم جمع نمود و مکان قرار گرفتن اعداد مهم است .

مثال:

$$\begin{array}{r} 102 \\ + 7 = 109 \end{array}$$

$$102 \times 10^2 + 7 \times 10^{-3} \quad \text{-----} > 10200/000$$

$$\frac{00000}{007} +$$

$$10200/007$$

بصورت کامپیوتری :

در کامپیوتر  $\text{Base} = 2$  در نظر گرفته می شود و می بایست توان دو عدد ممیز شناور با هم برابر شوند برای این منظور دوره موجود است .

یک راه این است که توان عدد بزرگتر را به عدد کوچکتر تبدیل نمود .

E      M       $N_2 = 0.11 \times 10^1, N_1 = 0.101 \times 2^3$       مثال :

$$N_1 \begin{array}{|c|c|} \hline 011 & 101 \\ \hline \end{array}$$

$$N_2 \begin{array}{|c|c|} \hline 001 & 11 \\ \hline \end{array}$$

حال برای اینکه از توان عدد اول دو واحد کم کنیم تا برابر باتوان عدد دوم یعنی یک شود ، کافیسست مانتیس عدد اول را دو بار به سمت چپ شیفت دهیم ، مشاهده می شود که انجام این عملیات باعث تغییر در مقدار عدد اول می شود پس این روش نادرست است .راه حل منطقی این است که توان عدد کوچکتر را به توان عدد بزرگتر تبدیل کنیم .

$$N_1 \begin{array}{|c|c|} \hline 011 & 1010 \\ \hline \end{array}$$

$$N_2 \begin{array}{|c|c|} \hline 011 & 0011 \\ \hline \end{array}$$

\_\_\_\_\_ +

$$N_1 + N_2 \begin{array}{|c|c|} \hline 011 & 1101 \\ \hline \end{array}$$

در این روش به ازای هر بار افزایش توان یک بار محتوای مانتیس را به سمت راست شیفت می دهیم . در واقع به اندازه اختلاف دو توان اعدادی که می خواهند با هم جمع شوند ، مانتیس عدد کوچکتر ShR می دهیم و به همین اندازه به توان اضافه می کنیم .

|     |          |
|-----|----------|
| ۰۰۱ | ۱۱۱۱۱۱۱۱ |
|-----|----------|

**نکته :** اگر عدد ما به صورت زیر باشد و ما بخواهیم برای رسیدن به توان عدد دیگر مانتیس آن را به سمت راست شیفت بدهیم ، در اینصورت دو مقدار یک از سمت راست حذف می گردد در واقع دقت عمل کم می شود و هیچ راهی برای جلوگیری از این کار نیست.

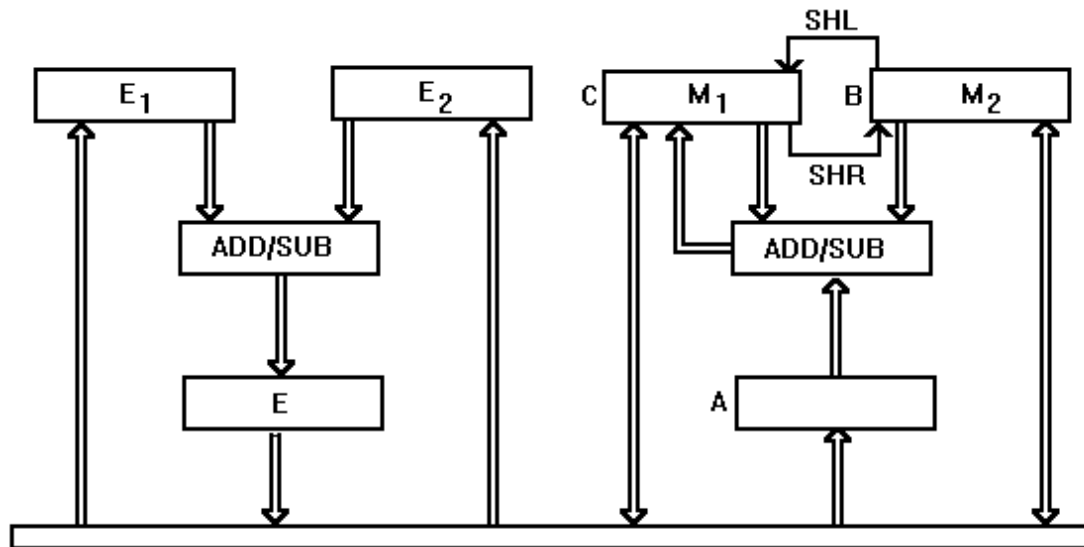
#### ۲-۵ : عمل ضرب در عدد ممیز شناور

از لحاظ الگوریتمی ، الگوریتم ضرب آسانتر از جمع دو عدد ممیز شناور است .

$$102 * 10^2 * 7 * 10^{-3} = 714 * 10^{-1} \quad \text{مثال :}$$

در ضرب دو عدد ممیز شناور کفایت مانتیسها را در هم ضرب نموده و توانها را باهم جمع کنیم.

۳-۵: مدار جمع و تفریق دو عدد ممیز شناور



(شکل ۱-۵)

در ابتدا باید ببینیم که کدامیک از مفسرها کوچکتر است تا آنرا به مفسر بزرگتر تبدیل کنیم. این کار را با تفریق دو مفسر انجام می دهیم ( $E_1 - E_2$ ) و اگر حاصل تفریق منفی بود، یعنی  $E_1$  کوچکتر است. اگر حاصل مثبت بود  $E_2$  کوچکتر است. حال حاصل  $E_1 - E_2$  در  $E$  قرار می گیرد.

به اندازه قدر مطلق  $E$  (یعنی  $|E|$ ) ماننسیس عدد کوچکتر را به سمت راست شیفت می دهد. سپس دو عدد  $M_2, M_1$  نقاط اعشارشان زیرهم قرار می گیرد و برای عمل جمع یا تفریق آماده می باشد. البته توان عدد بزرگتر به منظور توان نهایی حاصل جمع یا تفریق در قسمت توان حاصل عملیات قرار می گیرد.

**نکته:** اگر در قسمت توانها عمل جمع یا تفریق را انجام دهیم و برای قسمت ماننسیسها، بجای عمل جمع و تفریق عمل ضرب و تقسیم صورت گیرد، این مدار می تواند اعمال جمع و تفریق و ضرب

و تقسیم دو عدد ممیز شناور را انجام دهد. البته باید ثابتهای C و B را با قابلیت ShL , ShR داشته باشیم .

تمرینات

۱- توضیح دهید که برای جمع بستن دو عدد Floating Point چه اقداماتی لازم است؟

۲- توضیح دهید که برای تفریق دو عدد Floating Point چه اقداماتی لازم است؟

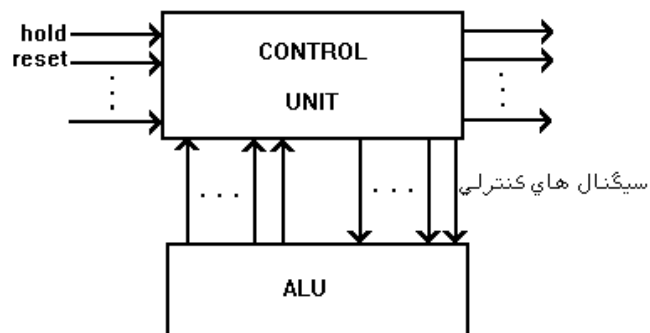
۳- توضیح دهید که برای ضرب دو عدد Floating Point چه اقداماتی لازم است؟

۴- توضیح دهید که برای تقسیم دو عدد Floating Point چه اقداماتی لازم است؟

## فصل سوم - طراحی واحد کنترل و حافظه

## ۱- طراحی واحد کنترل

وظیفه واحد کنترل دریافت دستورات از حافظه و تفسیر آنها و انجام دادن کارهای لازم (فرمانهای لازم) برای اجرای آن دستورات می باشد.



(شکل ۱-۱)

سیگنالهای کنترلی انواع مختلفی دارند :

۱- سیگنالهای کنترلی که از واحد کنترل به ALU می رود .

برای مثال ممکن است دستور ShR در یک ثبات و یا دستور خواندن اطلاعات از Bus و هزاران دستور دیگر باشد.

۲- سیگنالهای کنترلی که از طرف ALU به واحد کنترل می آید.

برای مثال ممکن است در جمع دو عدد رقم نقلی رخ دهد که در این صورت واحد ALU به واحد CU (Control Unit) سیگنال می فرستد.



۳- سیگنالهای کنترلی که از یک ریزپردازنده در کامپیوتر دیگر به واحد کنترل می آیند .

برای مثال عمل Resert کردن یا عمل Hold کردن .

۴- سیگنالهای کنترلی که ممکن از واحد کنترل به یک جای دیگر غیر از Alu فرستاده شود .

برای مثال به یک ریز پردازنده دیگر بروند.

البته سیگنالهای کنترلی دیگری نیز وجود دارند که هر کدام وظیفه مخصوص به خود دارند.

**نکته :** در این درس هدف مطالعه سیگنالهای کنترلی است که بین Alu و CU وجود دارد و در مورد بقیه سیگنالها بحث نمی شود .

بطور کلی طراحی واحد کنترل به دو طریق انجام می گیرد.

۱- روش سیم بندی شده (Hard Wired)

۲- روش ریزبرنامه سازی (Micro Programmed)

۱-۱ : روش سیم بندی شده

در این حالت واحد کنترل همانند یک مدار ترتیبی است که سیگنالهای کنترلی را یکی پس از دیگری تولید می کند . این روش ، سریعترین روش و کم هزینه ترین روش است اما هیچگونه انعطاف پذیری ندارد . یعنی اگر بخواهیم تغییرات جزئی در مدار واحد کنترل دهیم ، لازم است که طراحی مورد نظر از نو انجام شود .

امنیت روش H.W در مقابل Noise کم است چراکه تعداد زیاد سیم در فرکانس های بالا ایجاد Noise می کند.

به سه طریق می توان واحد کنترل رابراساس روش سیم بندی شده طراحی کرد .

۱- State Table Method

۲- Delay Element Method

۳- Sequence Counter Method

### ۱-۱-۱ : State Table Method

در این روش از طراحی واحد کنترل بحث بر سر آن است که طراحی مدار بگونه ای باشد که حداقل فلیپ فلاپها رداشته باشد . به عبارت دیگر این روش از طراحی واحد کنترل نسبت به دو روش دیگر دارای هزینه کمتر وامنیت بیشتری در مقابل Noise می باشد .

البته طراحی واحد کنترل براساس این روش مشکل است و نیاز به بحث مدارات منطقی پیشرفته می باشد .از این روش هنگامی استفاده می شود که واحد کنترل کوچک باشد .

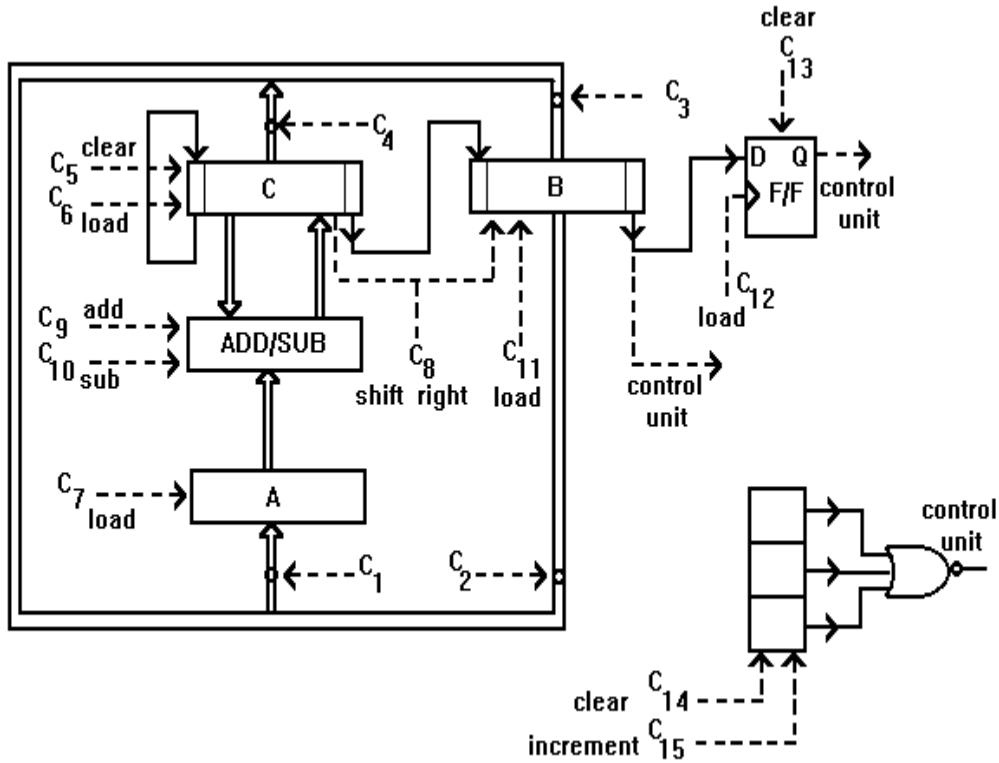
### ۱-۱-۲ : Delay Element Method

برای آشنایی بیشتر با این روش از طراحی مثال زیر را بررسی می نمایم .

**مثال :** طراحی واحد کنترل برای ضرب دو عدد به روش Booth

برای طراحی واحد کنترل ، ابتدا مدار را طراحی می کنیم . سپس سیگنالهای کنترلی ( خطوط نقطه چین ) را مشخص نموده و براساس الگوریتمی پیاده سازی می نمایم . در این

الگوریتم A با B ضرب شده و حاصل در زوج ثبات B : C قرار می گیرد . چون ثبات های A و B هشت بیتی هستند ، برای واحد کنترل به شمارنده ای با ۳ بیت نیاز داریم .



(شکل ۲-۱)

### الگوریتم ضرب به روش Booth

- ۱- اطلاعات موجود در گذرگاه را در ورودی ثبات A آماده می نمایم.
- ۲- مضروب را وارد ثبات A ، فلیپ فلاپ و ثبات C را Clear می نمایم.
- ۳- اطلاعات موجود در گذرگاه را در ورودی ثبات B آماده می کنیم.
- ۴- مضروب ما فیه را وارد ثبات B و شمارنده واحد کنترل را Clear می نمایم.
- ۵- به FF : B(0) توجه شود

C - A -----> C

اگر برابر 10 بودند آنگاه

$$C + A \text{ -----} > C$$

اگر برابر 01 بودند آنگاه

۶- مجموعه  $C : B : FF$  را یک بیت به راست ، بطور حسابی تغییر مکان می دهیم و به شمارنده واحد کنترل یک واحد اضافه می کنیم.

۷- اگر  $Count \leq 7$  بود ، ادامه کار از مرحله 5 انجام شود .

۸- پایان

\* چون اطلاعات در حافظه است ، در یک مرحله زمانی باید اطلاعات را به داخل BUS آورده و در مرحله دیگر اطلاعات را به داخل ثبات A بیاوریم و انجام این دو عمل بطور همزمان ممکن نیست. چون در این صورت اطلاعات نادرستی در A ذخیره خواهد شد .

\* هر چه شماره ها (مراحل الگوریتم ) بیشتر باشد زمان اجرایی بیشتر است ، چون هر شماره زمان خاص خود را دارد و هر چه مراحل کمتر باشد زمان کمتری مصرف می شود . پس مطلوبتر است .

\* ثبات C و F/F می بایست Clear گردند ، که این اعمال می توانند هم زمان اجرا گردند و اگر آنها را در یک سطر دیگر جدای از مرحله دوم بنویسیم زمان بیشتری مصرف می شود . چون این اعمال به هم ربطی ندارند پس می توان آنها را در مرحله دوم قرار دهیم .

\* در مرحله ۳ و ۴ نیز عدد مضروب فیه را به گذرگاه آورده و در مرحله ۴ وارد ثبات B می کنیم. چون ثباتهای ما ۸ بیتی است، پس عملیات باید ۸ بار تکرار شود . پس شمارنده برای شمارش آن نیاز داریم . این شمارندی ۳ بیتی، که برای شمارش ۸ عدد است را در ابتدا باید Clear کنیم .

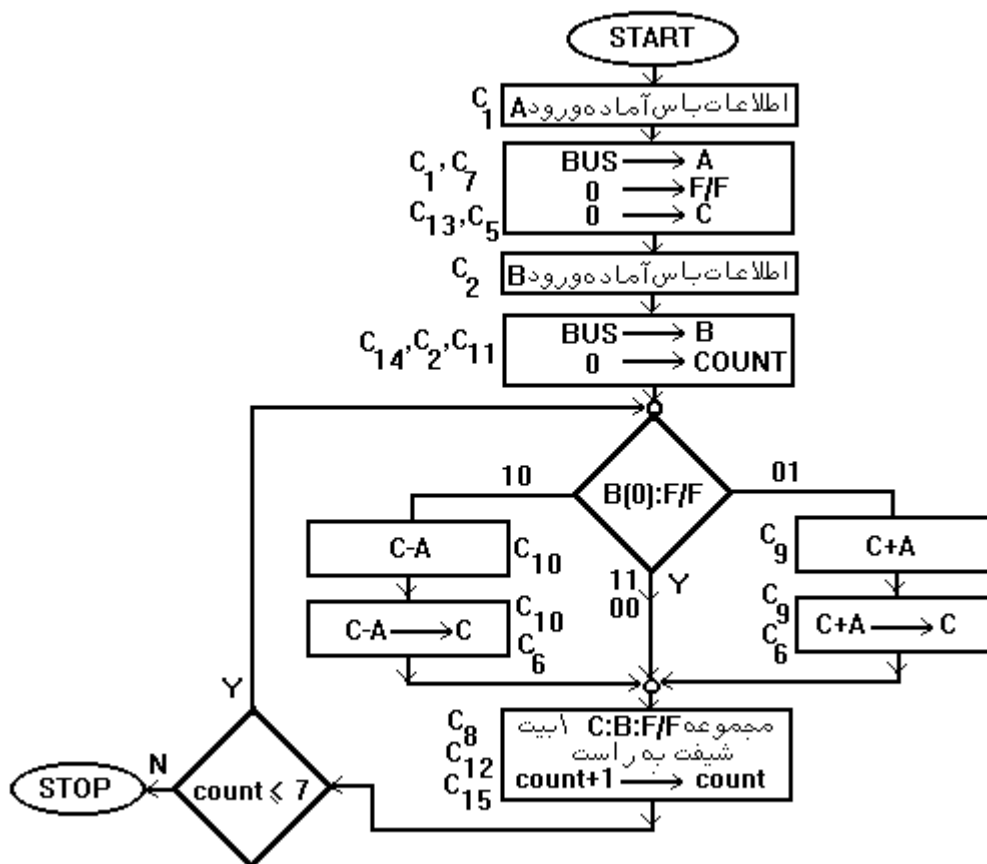
\* در مرحله ۵ چون در واقع عمل مقایسه صورت می گیرد پس شماره به خود نمی گیرد .

\* در مرحله ۶ عمل ShR بصورت حسابی و به اندازه یک بیت صورت می گیرد و چون عمل اضافه کردن یک واحد به شمارنده ، به ShR ربطی ندارد . پس می تواند بصورتی اشتراکی با ShR در یک مرحله صورت گیرد.

\* در مرحله ۷ اگر شمارنده از ۷ گذشته باشد ، به مرحله پایانی می رود و در غیر اینصورت از مرحله ۵ ، مراحل را ادامه می هد.

از این لحظه به بعد تصمیم می گیریم که به روش سیم بندی شده الگوریتم را طراحی کنیم یا روش ریز برنامه سازی، و اگر از روش سیم بندی شده استفاده می نماییم کدامیک از سه روش موجود در این روش را انتخاب می کنیم.

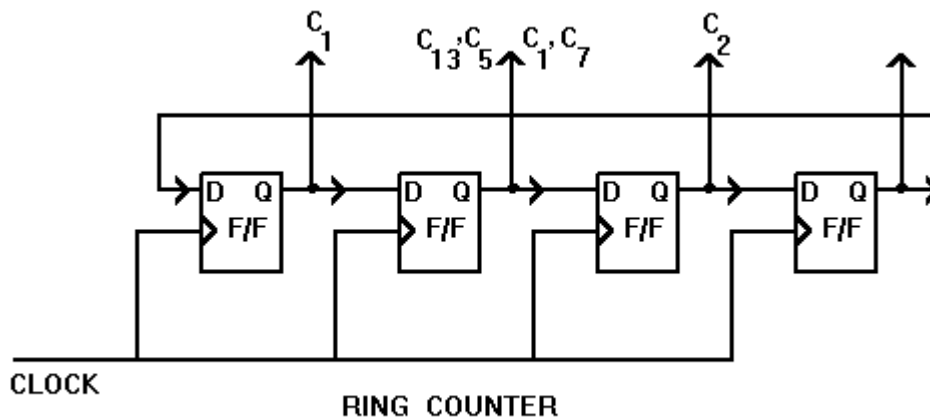
طراحی واحد کنترل الگوریتم Booth به روش Delay Element Method :



(شکل ۱-۳)

برای طی کردن یک بار مدار ، اگر از مسیر مستقیم برویم به ۵ پالس و در غیر این صورت به ۷ پالس زمانی احتیاج داریم . در کنار هر BOX نوشته شده که با کدامیک از سیگنالهای کنترلی فعال می شود . پس از این به بعد نوبت به عمل طراحی رسیده است . ما باید وسیله ای داشته

باشیم که این سیگنالها ( $C_1, \dots, C_{15}$ ) را به ترتیب و در هر لحظه یکی از آنها را فعال کند و اگر این کار را انجام دهیم درواقع واحد کنترل را طراحی کرده ایم. برای اینکار، تعدادی فلیپ فلاپ رادرنظر می گیریم، که همه آنها از نوع فلیپ فلاپ D می باشند و آنها به هم متصل می باشد.



(شکل ۱-۴)

در لحظه اول همه فلیپ فلاپها صفر هستند و در لحظه بعدی یک رابه فلیپ فلاپ اول می دهیم. مشاهده می شود که یک، بصورت چرخشی بین فلیپ فلاپها به گردش در می آید درواقع این همان Ring Counter، در درس مدارات منطقی می باشد. در لحظه اول باید سیگنال  $C_1$  را فعال کند و در لحظه دوم سیگنال  $C_1, C_7, C_{13}, C_5$  که مربوط به BOX دوم در الگوریتم است فعال شود و به همین صورت الی آخر. علت اینکه این روش را Delay Element می نامند به خاطر وجود فلیپ فلاپ می باشد که ایجاد تأخیر می نماید.

برای تبدیل فلورچات به مدار الکتریکی به این نکات توجه کنید:

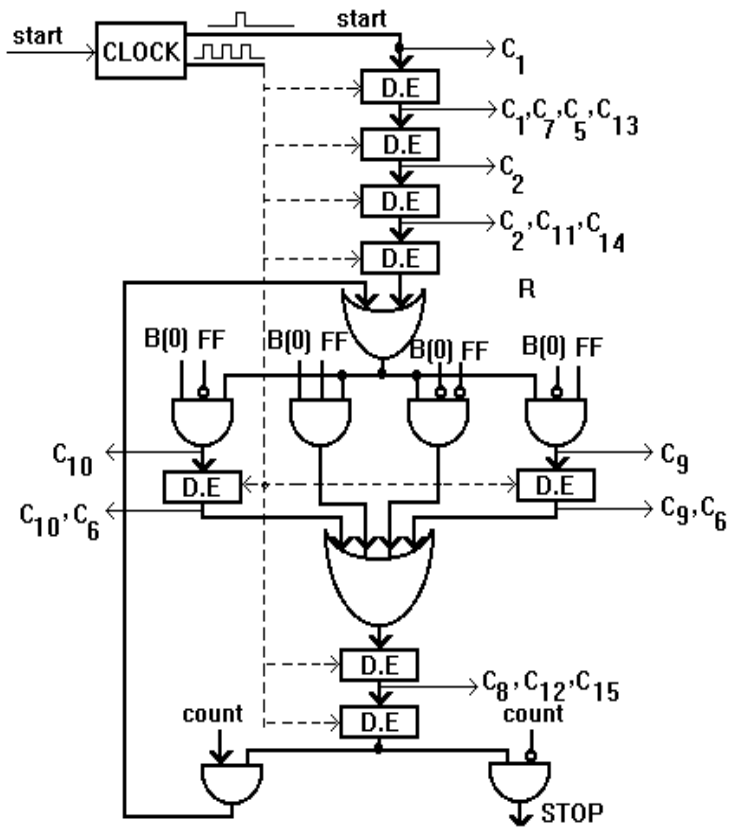
۱- بین هر دو BOX یک Delay Element (D.E) قرار داده و بجای هر BOX یک سیم می گذاریم.

۲- هنگامیکه اطلاعات از چند مسیر وارد یک مسیر می شود باید گیت OR بکار ببریم.

۳- در نقاط تصمیم گیری باید گیت AND بگذاریم.

با رعایت این سه نکته می توان چارت را به صورت یک واحد کنترل پیاده سازی کرد .

\* عمل مقایسه نیز نیاز به زمان (D. E) ندارد.

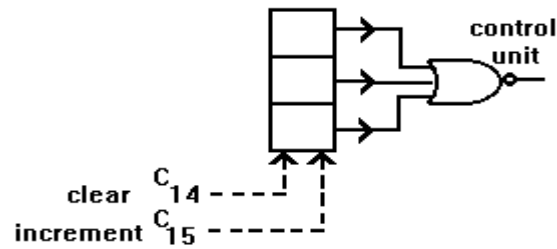


(شکل ۱-۵)

توضیح راجع به نحوه عملکرد شمارنده

در شروع کار پس از رسیدن به مرحله ۷ مقدار Count ، برابر یک می باشد. بنابراین خروجی OR گیت ما یک است، پس حلقه اجرا می شود و باعث قطع حلقه مانمی شود . وقتی که Count برابر ۸ شد ، خروجی گیت OR ما برابر با صفر شده بنابراین از حلقه خارج می شود .

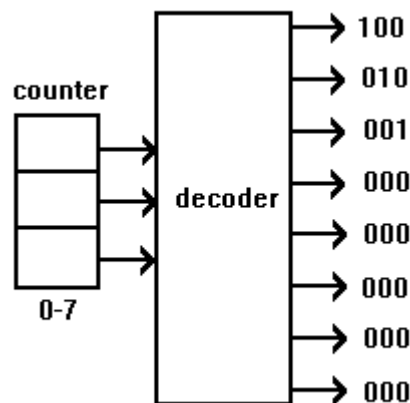
در اینجا نیازی نیست که خروجی های Count به یک NOR گیت برود ، زیرا در مطالب بعدی نیاز به گیت NOR می باشد . پس برای اصلاح عملیات کافی است وقتی که گیت OR را با گیت NOR عوض می کنیم ورودیهای Count در AND گیتهای پایین شکل (۱-۵) را Not می کنیم تا جواب مطلوب حاصل شود ولی در این روش تعداد FF ها زیاد است .



(شکل ۶-۱)

### ۳-۱-۱ Sequence Counter Method :

در این روش سعی می شود که تعداد فلیپ فلاپها (F/F) به نحوی کاسته شود، زیرا در روش Delay Element تعداد فلیپ فلاپها (F/F) زیاد می باشد . در این روش از یک شمارنده معمولی و یک کدگشا (Decoder) بجای شمارنده حلقوی که در روش Delay Element بود ، استفاده می کنیم و در واقع شمارنده عمل شمارش و کد گشا نیز عمل کدگشایی را انجام می دهد. بنابراین هزینه خیلی پایین می آید . با استفاده از Decoder که دائما از ابتدا تا انتها تکرار می شود ، حلقه های مختلفی را طی می کنیم .



(شکل ۷-۱)

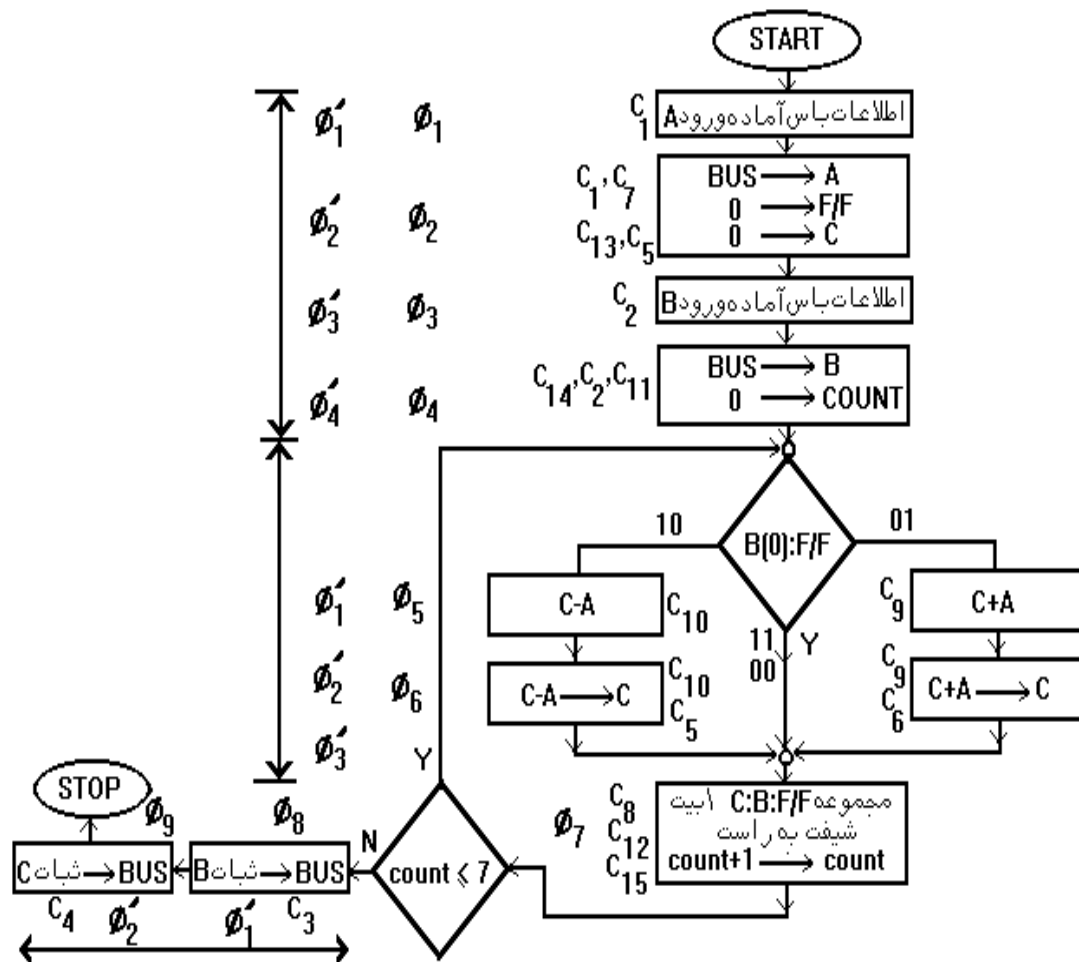
در الگوریتم قبلی یک عمل چندین بار تکرار می شد ، (در یک حلقه قرار می گرفت) و این عمل باعث پیچیدگی مدار در روش S.C (Sequence Counter) می شود و هزینه این مدار



نیز از روش D.E بیشتر می گردد . لذا از الگوریتمی استفاده می کنیم که فقط در دو مرحله با الگوریتم قبلی تفاوت دارد .

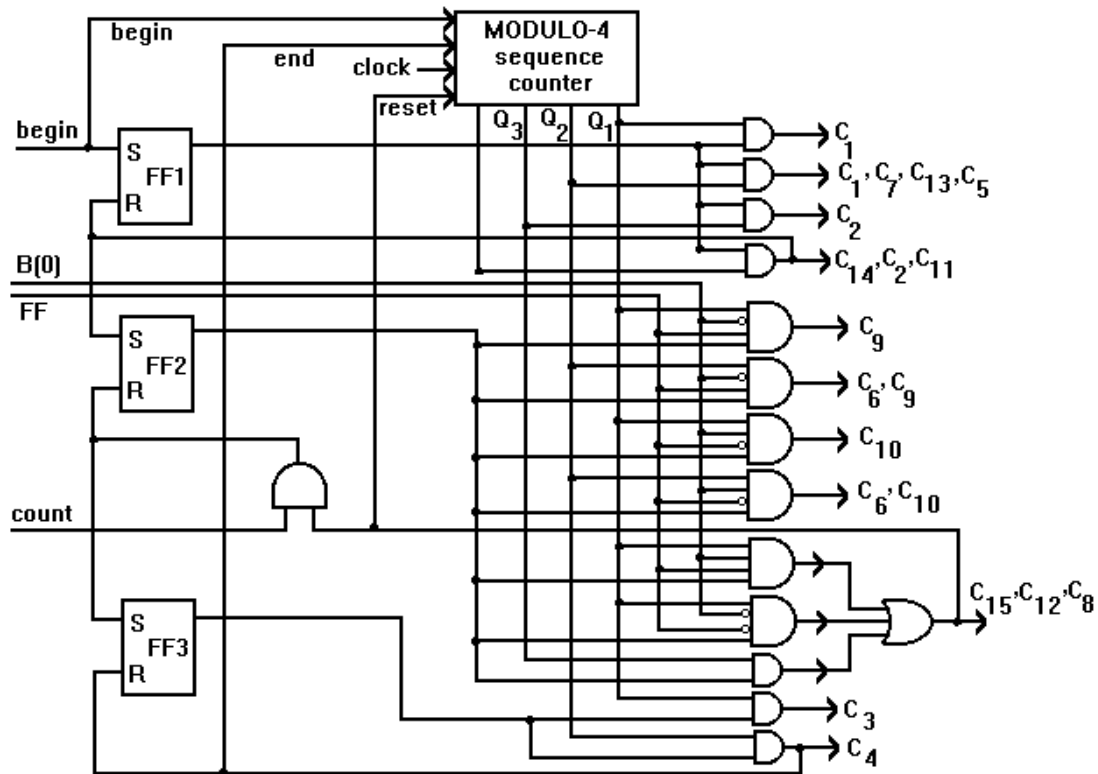
برای درک بهتر روش Sequence Counter مثال زیر را بررسی می نمایم .

مثال : طراحی واحد کنترل الگوریتم Booth به روش Sequence Counter Method



(شکل ۸-۱)

اگر بخواهیم از روش D.E استفاده کنیم باید یک شمارنده حلقوی ۹ بیتی داشته باشیم ولی در روش S.C باید الگوریتم به سه قسمت تقسیم شود . قسمت اول عملیات  $\phi'_1$  تا  $\phi'_4$  و قسمت وسط، از  $\phi'_1$  تا  $\phi'_3$  و قسمت آخر نیز از  $\phi'_1$  تا  $\phi'_2$  را می شمارد . در اینجا حداکثر باید تا ۴ شمارش شود زیرا مرحله دوم که بیشترین زیر مرحله را دارد ، شامل ۴ قسمت می باشد .



(شکل ۱-۹)

حال از یک شمارنده استفاده می کنیم که همه قسمت ها را بشمارد و برای این شمارش ، از شمارنده ای به پیمانه چهار و سه فلیپ فلاپ استفاده می کنیم.

برای اینکه وقتی شمارنده می شمارد متوجه شویم ، عمل شمارش برای کدام قسمت از فلورچات در حال انجام می باشد ، از سه فلیپ فلاپ که هر کدام برای یک قسمت می باشد استفاده می کنیم و عمل شمارش برای هر قسمتی باشد فلیپ فلاپ مربوط به آن Set (یعنی یک) شده و بقیه فلیپ فلاپها صفر می شوند. هنگامی که  $C_4$  فعال شد ، شمارش تمام است و این یعنی عمل ضرب به انتهای خودش رسیده است .

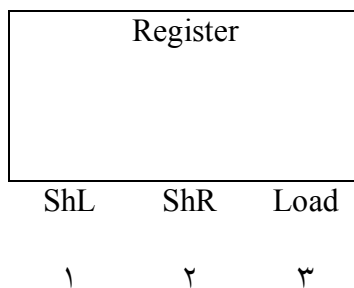
\* وقتی Clock به شمارنده داده می شود قسمت Begin نیز که حالت شروع است ، فعال شده و یک Start از آن نیز به شمارنده می رود.

۲-۱: روش ریز برنامه سازی

این روش که اولین بار در سال ۱۹۵۰ توسط ویلکس (Wilkes) پیاده سازی شد ، روشی است که دارای هزینه بالا ، سرعت کم ،اما از انعطاف پذیری بالایی برخوردار است . این روش بدین صورت انجام می شود که سیگنالهای کنترلی خواننده می شوند و به مقاصد مربوطه هدایت می گردند.

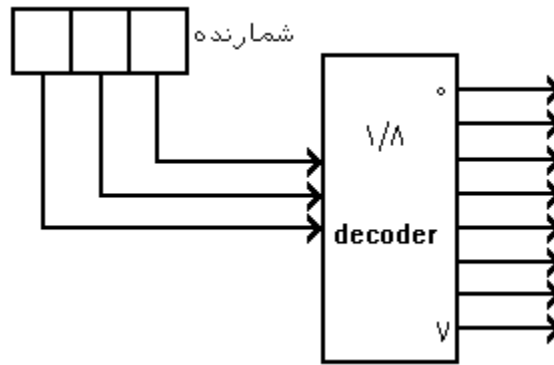
برای درک بهتر این دو روش از طراحی واحد کنترل به مثال زیر توجه کنید .

**مثال :** فرض کنید هر ثبات از سه قسمت تشکیل شده باشد . مثلاً یک پایه ShL و یک پایه ShR و یک پایه Load (بارکردن) داشته باشد ، که اگر محتوای آنها یک باشد ، عمل مورد نظر را انجام دهد و اگر محتوای آنها صفر بود این اعمال را انجام ندهد. حال این پایه ها را شماره گذاری می کنیم و واحد کنترل شماره آنها را معین می کند . برای مثال اگر بخواهد عمل ShR را انجام دهد، شماره ۲ را فعال می کند و به همین صورت برای هر پایه شماره آن را می فرستد.



فرض کنید که بخواهیم اعمالی را طبق شماره های 1,2,6,5,3 انجام دهیم. برای این منظور می توانیم شمارنده‌ایی را طراحی کنیم که به ترتیب اعداد 1,2,6,5,3 را بشمارد .

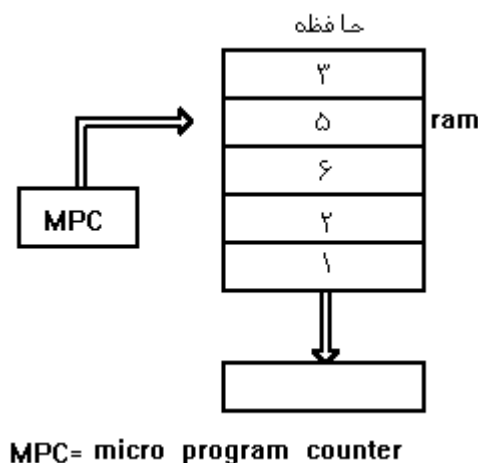
چون بزرگترین عدد که همان عدد 6 می باشد ، در 3 بیت جای می شود . بنابراین سه خروجی از واحد کنترل خارج می شود . این سه خروجی را به کدگشا وصل می کنیم تا وقتی شمارندی ما می شمارد ، کدگشا شماره متناظر با ثبات مورد نظر را فعال نماید .



(شکل ۱-۱۰)

حال اگر اشتباهی در ترتیب شمارنده صورت گرفته باشد ، برای مثال بجای شماره های 1,2,6,5,3 می بایست شماره های 1,3,6,5,2 را وارد می کردیم ، در اینصورت باید یک واحد شمارنده مجزا برای آن طراحی نماییم. (همانطور که در مدار منطقی داشتیم) پس نتیجه می گیریم که روش سیم بندی شده انعطاف پذیر نمی باشد .

ولی در روش ریز برنامه سازی یک حافظه داریم ، که اعداد مورد نظر به ترتیبی که می خواهیم شمارش شوند را در سطرهای حافظه قرار می دهیم . در این حالت اگر بخواهیم اعداد موجود در سطرهای حافظه را عوض کنیم به راحتی می توان جای اعداد را عوض نمود. پس این روش بسیار انعطاف پذیر است.

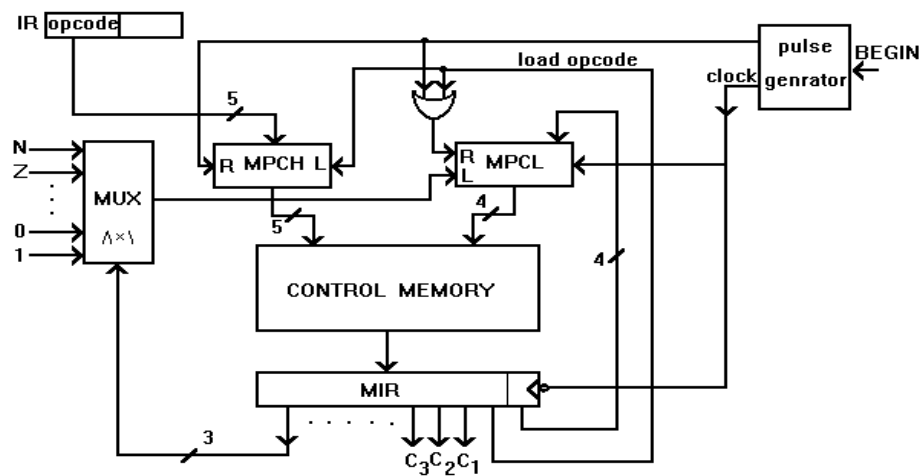


(شکل ۱-۱۱)

نکته : عملاً در کامپیوترها نیز از روش ریز برنامه سازی استفاده می کنند هر چند که پرهزینه و کم سرعت است .

۳-۱ : طراحی واحد کنترل به روش ریز برنامه سازی (Micro Programmed)

حال ببینیم واحد کنترل براساس ریز برنامه سازی چه ساختاری دارد.

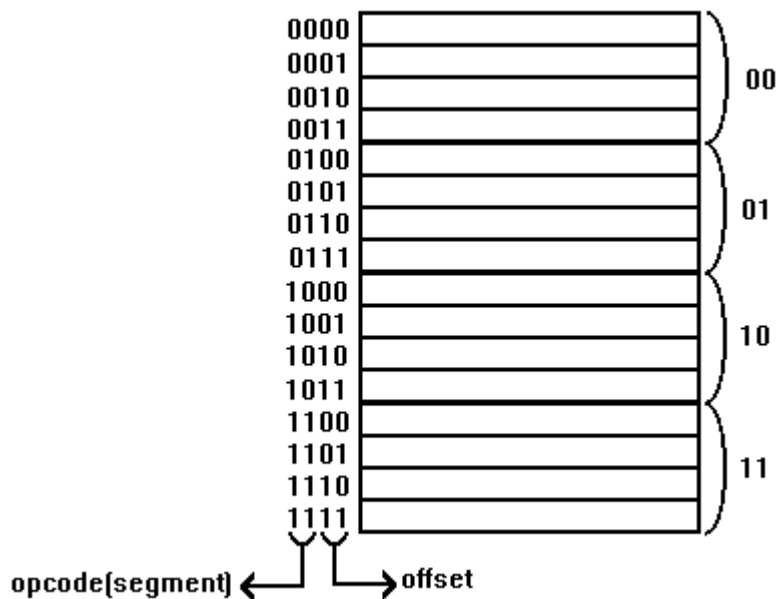


(شکل ۱-۱۲)

در لحظه اول که کامپیوتر را روشن می کنیم از طرف Pulse Generator یک پالس فرستاده می شود تا  $\mu PCL$ ,  $\mu PCH$  را Clear یا Reset کند و به اجبار در سطر اول حافظه قرار خواهد گرفت، و اولین دستوری که در آن ذخیره شده، اجرا می گردد. بنابراین باید در اولین سطر حافظه، برنامه‌ای را که می خواهیم در ابتدا اجرا شود ذخیره می کنیم. سپس در سطر اول حافظه دستورالعمل پرش را قرار می دهیم که برنامه خاصی در آدرس مشخص را در لحظه اول اجرا کند، مثلاً برنامه سیستم عامل را اجرا کند.

در طراحی واحد کنترل براساس ریز برنامه سازی حافظه‌ای وجود دارد که به آن حافظه کنترلی می گوئیم و سیگنالهای کنترلی در داخل این حافظه قرار دارد و حافظه کنترلی به K قسمت

مساوی تقسیم می شود (K برابر است با تعداد دستورالعمل های ماشین) و هر قسمت از این K قسمت ریز دستورات یک دستور ماشین را شامل می شود. با این عمل آدرس های مربوط به هر سطر از حافظه کنترلی را می توان به دو دسته تقسیم نمود. یک آدرس بیانگر قسمت مربوطه از حافظه کنترلی است (آدرس با ارزشتر) و آدرس دیگر مربوط به شماره سطر از آن قسمت می باشد (آدرس کم ارزشتر) که البته در شکل (۱-۱۳) مشخص است.



(شکل ۱-۱۳)

آدرس بخش با ارزشتر در واقع بیانگر Opcode دستورالعمل می باشد. برای اجرای یک دستورالعمل از زبان ماشین Opcode آن دستورالعمل موجود در IR به داخل  $\mu PCH$  منتقل می شود و در همان لحظه  $\mu PCL$ ، Clear می شود این عمل توسط خط Load Opcode انجام می گیرد. با این کار به ابتدای قسمت مربوط به سیگنالهای کنترلی این دستورالعمل از ماشین واقع در حافظه کنترلی مرتبط می شویم، و از آن جا سیگنالها یکی پس از دیگری فعال می گردند. در نوشتن ریز برنامه قطعاً نیاز به پرش (از نوع شرطی یا غیر شرطی) به یک محل خاص از حافظه کنترلی خواهیم داشت. این عمل با انتخاب شرط مربوطه از Mux و مهیا کردن آدرس محل پرش در ورودی  $\mu PCL$  تغییر می یابد. در این حالت در صورتی که شرط مربوطه برقرار باشد خروجی Mux یک و عمل Load آدرس جدید به داخل  $\mu PCL$  انجام می گیرد در غیر این صورت عمل پرش انجام نشده و به  $\mu PCL$  یک واحد اضافه می گردد.



با توجه به ویژگیهای مدار ، حافظه کنترلی بصورت  $40 * 512$  می باشد . بافرض اینکه سخت افزاری که در اختیار داریم به ۳۲ سیگنال کنترلی نیاز داشته باشد و سه بیت هم جهت انتخاب وضعیت (Select Condition) و یک بیت هم جهت Load و چهار بیت هم برای خطی که از  $\mu IR$  می رود که در این صورت داریم :

$$\text{تعداد ستونها} = 32 + 3 + 1 + 4 = 40$$

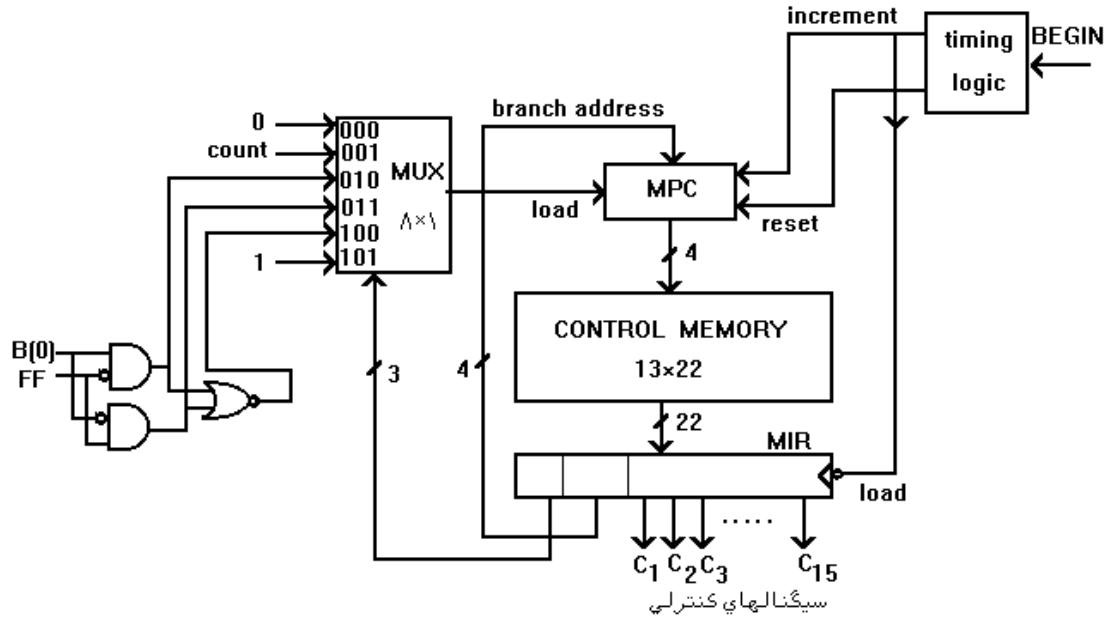
تعداد سطرهای حافظه کنترلی برابر با تعداد خطوط ریز برنامه نوشته شده برای آن سخت افزار مورد نظر می باشد. ( که در اینجا برابر است با ۵۱۲ )

**مثال :** نحوه ساختن واحد کنترل برای الگوریتم ضرب Booth :

چون در اینجا واحد کنترل را فقط برای یک دستورالعمل طراحی می کنیم ، بنابراین از یک ثبات استفاده می نمایم ، پس PC ما به یک قسمت  $\mu PC$  تبدیل می شود و به عنوان نمونه دارای بخش  $\mu PCH$  نمی باشد. ولی اگر بخواهیم برای کامپیوتر پیاده سازی کنیم باید PC دارای دو قسمت باشد .

مدار لازم بصورت بلوک دیاگرام در زیر آمده است :





(شکل ۱-۱۴)

در ابتدا برای تعیین تعداد سطرهای حافظه کنترلی می بایست تعداد سطرهای ریز برنامه نوشته شده را بدست آوریم. که در الگوریتم Booth از سطر ۰ تا سطر ۱۲ می باشد که جمعاً ۱۳ سطر می شود، و برای محاسبه تعداد ستونها ی حافظه کنترلی داریم:

تعداد خطوطی که از  $\mu IR$  به  $\mu PC$  می رود + انتخاب حالت + تعداد سیگنالهای کنترلی = تعداد ستونها

$$\text{تعداد} = 15 + 3 + 4 = 22$$

ستونها

پس ظرفیت حافظه کنترلی در روش Booth برابر است با  $(22 * 13)$ .

برای  $B(0)$  و FF سه حالت مختلف داریم و هنگامی که count به ۸ برسد شرط توقف فراخوانده می شود.

مجموعه سیگنالهای کنترلی:

{c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,c13,c14,c15}

Condition Branch

| C.M.<br>c12 | Select<br>c13 c14 c15 | Address | c1   | c2 | c3 | c4 | c5 | c6 | c7 | c8 | c9 | c10 | c11 |
|-------------|-----------------------|---------|------|----|----|----|----|----|----|----|----|-----|-----|
|             | ...                   | ....    | \    | .  | .  | .  | .  | .  | .  | .  | .  | .   | .   |
|             | ....                  |         |      |    |    |    |    |    |    |    |    |     |     |
|             | ....\                 | ...     | .... | \  | .  | .  | .  | \  | .  | \  | .  | .   | .   |
|             | ....\.                | ...     | .... | .  | \  | .  | .  | .  | .  | .  | .  | .   | .   |
|             |                       |         | .    |    |    |    |    |    |    |    |    |     |     |
|             | ....\)                | ...     | .... | .  | \  | .  | .  | .  | .  | .  | .  | .   | .   |
|             |                       |         | .    |    |    |    |    |    |    |    |    |     |     |
|             | ....\.                | ....    | .... | .  | \  | .  | .  | .  | .  | .  | .  | .   | .   |
|             |                       |         | .    |    |    |    |    |    |    |    |    |     |     |
|             | ....\)                | ....    | .... | .  | \  | .  | .  | .  | .  | .  | .  | .   | .   |
|             |                       |         | .    |    |    |    |    |    |    |    |    |     |     |
|             | ....\.                | ....    | .... | .  | \  | .  | .  | .  | .  | .  | .  | .   | .   |
|             |                       |         | .    |    |    |    |    |    |    |    |    |     |     |
|             | ....\)                | ....    | .... | .  | \  | .  | .  | .  | .  | .  | .  | .   | .   |
|             |                       |         | .    |    |    |    |    |    |    |    |    |     |     |

.

۱۰۱۰    ۰۰۱    ۰۱۰    .   .   .   .   .   .   .   .   ۱   .   .   .   ۱   .   .   ۱

.

۱۰۱۱    ۰۰۰    ۰۰۰    .   .   ۱   .   .   .   .   .   .   .   .   .   .   .   .

.

۱۱۰۰    ۰۰۰    ۰۰۰    .   .   .   ۱   .   .   .   .   .   .   .   .   .   .   .

.



|                  |                   |   |     |
|------------------|-------------------|---|-----|
|                  | PC <----- y       | y | JMP |
| PC <----- y Then | IF Carry Flag = 1 | y | JC  |

۷- شکل کلی طراحی واحد کنترل براساس ریز برنامه سازی برای یک کامپیوتر را رسم نموده و نحوی عملکرد آن را توضیح دهید؟

۸- الف) چهار ثبات ۸ بیتی با خروجی 3-State در اختیار داریم . به کمک یک جمع کننده ۴ بیتی مداری طراحی کنید که بتوان هر یک از دو ثبات خواسته شده را با هم جمع نموده و حاصل را در ثبات دیگری قرار دهد؟

ب) برای مدار قسمت الف) واحد کنترل نوع ریز برنامه سازی را طراحی کنید . نوشتن ریز برنامه لازم نیست؟

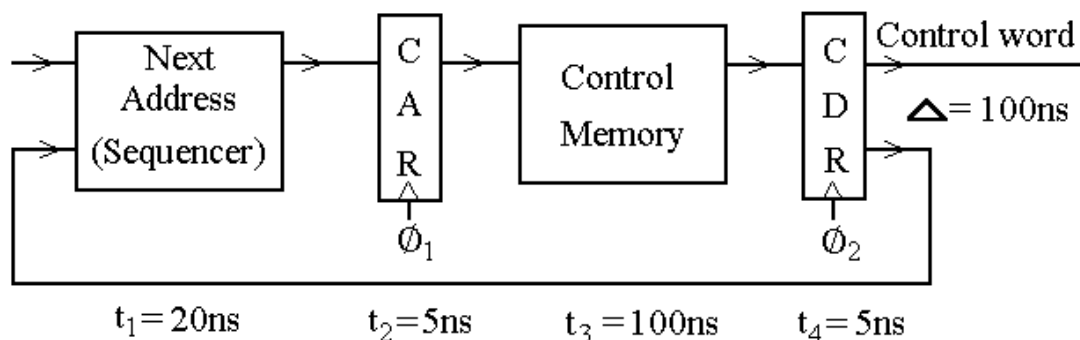
۹- نمودار یک واحد کنترل به روش ریز برنامه سازی نشان داده شده است .  $t$  ها تأخیر در اجزاء و  $\Delta$  تأخیر در اجرای زیرعمل و ( $\mu - op$ ) می باشد .

الف) حداقل پریود کلاک ns ۲۳۰ می باشد .

ب) حداقل پریود کلاک ns ۱۳۰ می باشد .

ج) با حذف CDR حداقل کلاک ns ۲۰۵ می شود .

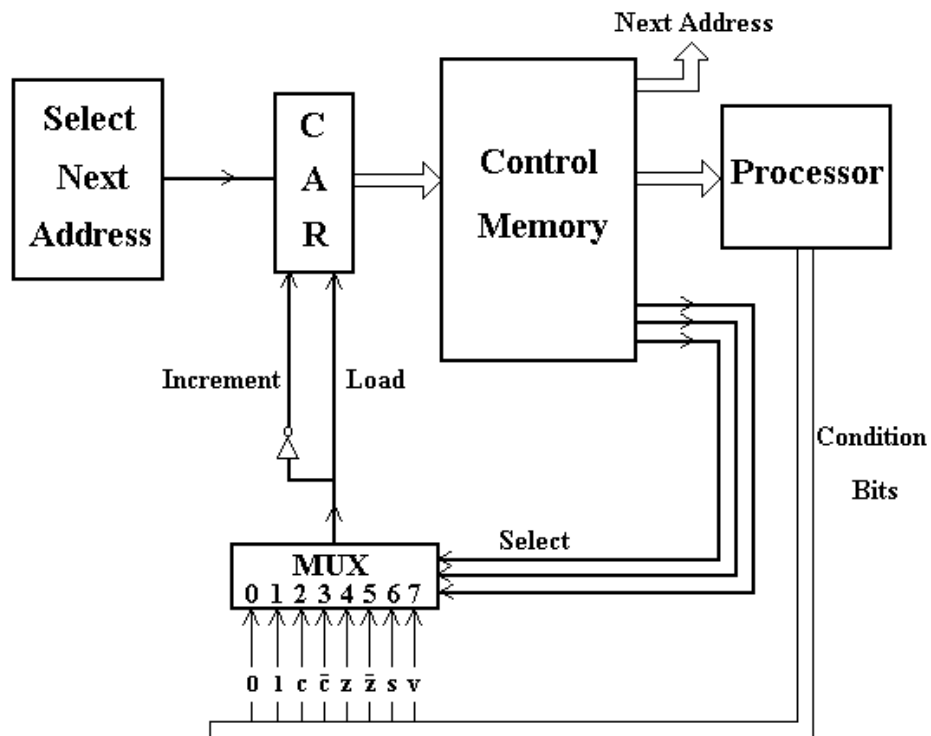
د) با حذف CDR حداقل کلاک ns ۲۲۵ می شود .



کدامیک از گزینه های زیر درست است ؟

- (۱) فقط ب (۲) فقط ج (۳) الف و د (۴) ب و ج

۱۰- شکل زیر بخشی از واحد کنترل زیربرنامه پذیر می باشد . مشخص کنید اگر  $A=36$  و Select و  $(0101)_2$  و  $CAR=20$  و پردازندی دستور R1-R2 را اجرا کرده باشد ، کدام گزاره صحیح است (R1 , R2 بدون علامت فرض شده اند و تفریق به روش مکمل دو انجام می شود .)



IF  $(R_1 \geq R_2)$  Then  $CAR \leftarrow 36$  else  $CAR \leftarrow 21$  (۱)

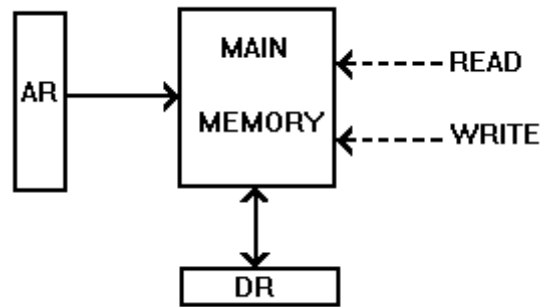
IF  $(R_1 > R_2)$  Then  $CAR \leftarrow 21$  else  $CAR \leftarrow 36$  (۲)

IF  $(R_1 \leq R_2)$  Then  $CAR \leftarrow 21$  else  $CAR \leftarrow 36$  (۳)

IF  $(R_1 < R_2)$  Then  $CAR \leftarrow 36$  else  $CAR \leftarrow 21$  (۴)



## ۲- سازمان حافظه (Memory Organization)



(شکل ۱-۲)

### ۱-۲ : جنس حافظه

بطور کلی تکنولوژی ساخت حافظه به دو دسته زیرتقسیم می شود :

(۱) نیمه رسانا (Semiconductor)

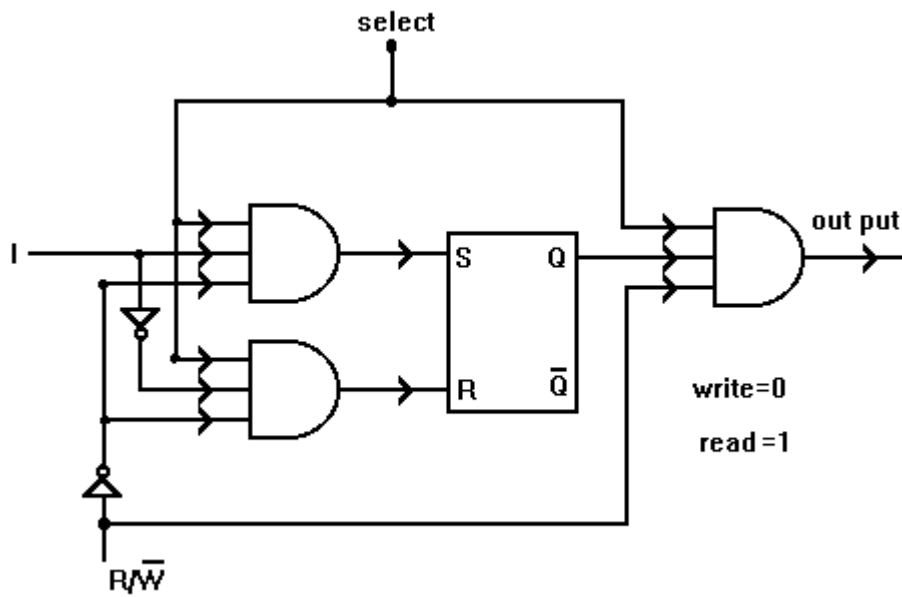
(۲) حلقه های مغناطیسی (Magnetic Core)

### ۱-۱-۲ : حافظه نیمه رسانا (Semiconductor)

این نوع از حافظه با قطع برق محتوای خود را از دست می دهد. اما مخرب در برابر عمل خواندن نیست .

هر سلول از حافظه نوع نیمه رسانا که در کامپیوتر های امروزی همان بیت هاست . هسته ی آن یک فلیپ فلاپ بوده و بقیه برای کنترل آن می باشد .





(شکل ۲-۲)

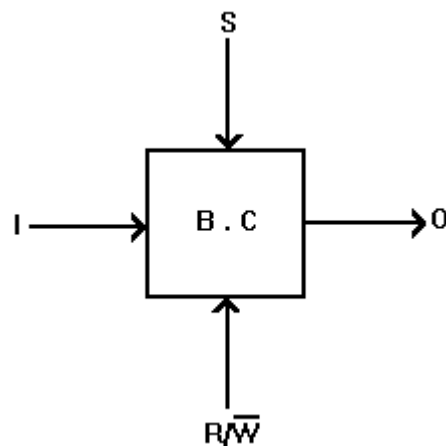
خروجی خود را آزاد می کند .  $Read = 1$

ورودی خود را آزاد می کند .  $Write = 0$

وقتی عمل نوشتن صورت می گیرد که اولاً  $Select = 1$  شود ، ثانیاً برای نوشتن  $W=0$  باشد . وقتی

عمل خواندن انجام می شود که  $Select = 1$  شود ، ثانیاً برای خواندن  $R=1$  باشد .

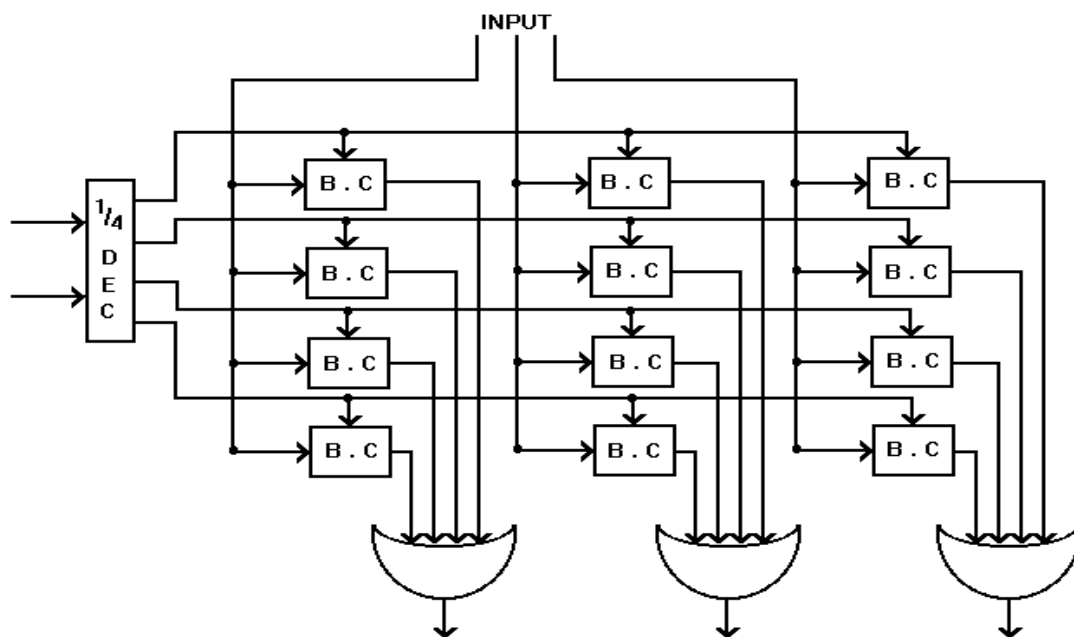
بطور کلی یک سلول از حافظه (Binary cell) را می توان بدین صورت نشان داد.



(شکل ۲-۳)

ساختار داخلی حافظه نوع نیمه رسانا ۳\*۴

\* منظور از ۳\*۴ یعنی اینکه ۴ تا کلمه دارد و تعداد بیت‌های هر کلمه ۳ تا می‌باشد.



(شکل ۲-۴)

\* خروجی سلولها را با هم OR کردیم ، چون نمی‌توانیم آنها را به هم وصل کنیم زیرا باعث سوختن IC می‌شود.

\* در این ساختار اطلاعات به همه سلولها داده می‌شود و از هرکدام که خواستیم اطلاعات مورد نیاز را می‌خوانیم .

\* سیگنال R/W به همه سلولها وصل است و ما آنرا نشان نداده ایم .

هنگامی که می خواهیم مثلا سطر یک را آدرس دهی کنیم ، آدرس آن توسط Decoder اعمال و سطر یک فعال می شود . به این ترتیب که بیت های سطر یک با توجه به اینکه پایه Select آن ( که به Decoder متصل است) فعال است ، فعال می شوند و سطر یک آدرس دهی می گردد.

قیمت RAM به این دلیل بالاست که به ازای هر بیت یک FF می خواهد و اگر تعداد سطرها فقط یک مگا بایت هم باشد ، 20 بیت گیت OR نیاز داریم و مسلما این حافظه به دلیل اینکه باید از FF و گیت OR عبور کند ، تاخیر (Delay) خواهد داشت .

گاهی اوقات که می خواهیم RAM ها را تبدیل کنیم ، می توانیم RAM های ک.چکتر را کنار هم قرار دهیم .

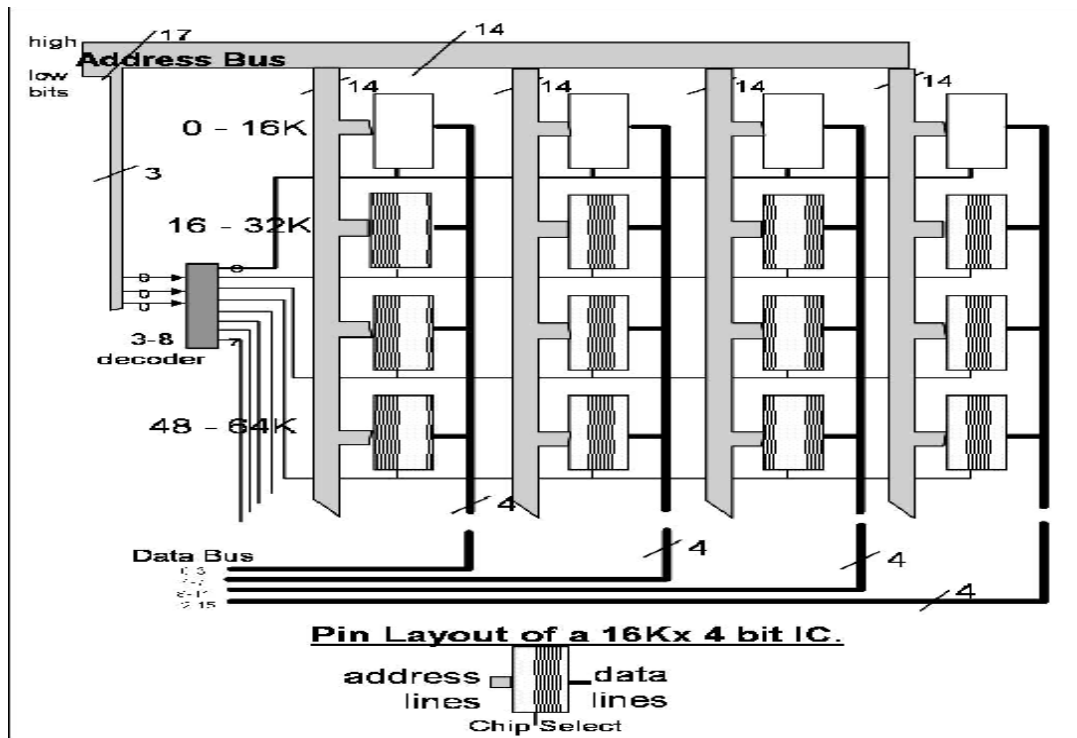
ساختار حافظه زیر را که یک حافظه ی  $128K * 16 \text{ bit}$  است را در نظر بگیرید . این RAM از 20 تا  $16K * 4 \text{ bit}$  RAM ساخته شده است که خود این RAM ها هر کدام از RAM های کوچکتری ساخته شده اند .

برای طراحی باس های داده و آدرس به صورت زیر عمل می کنیم :

باس داده را 4 بیت ، 4 بیت جدا کرده و خطوط کم ارزش تر را به RAM اولی می دهیم . ( کل بیت های باس داده 16 بیت است ) و باس آدرس را که 17 بیت است ، 14 بیت آن را به هر کدام از RAM ها متصل کرده و 3 بیت کم ارزش تر آن را به Decoder متصل می کنیم و مشخص می شود که کدام سطر فعال می شود .

برای جبران کندی حافظه از برگ برگ کردن حافظه استفاده می شود . ( Memory Inter Leaving )

(



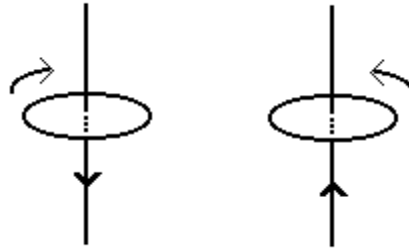
(شکل ۲-۵)

### ۲-۱-۲: حافظه حلقه های مغناطیسی (Magnetic Core)

این نوع از حافظه با قطع برق محتوای خود را حفظ می کند. با خواندن یک سطر محتوای آن سطر از بین می رود، از این جهت در این نوع حافظه به دنبال هر عمل خواندن یک عمل نوشتن انجام می شود و قبل از هر عمل نوشتن در حافظه باید یک عمل خواندن صورت گیرد. زیرا برای نوشتن در یک محل از حافظه باید ابتدا آن محل را Clear کنیم بنابراین یک بار آن محل را می خوانیم تا Clear شود. بدین ترتیب سرعت خواندن و نوشتن اطلاعات کمتر می شود.

سلول حافظه از نوع حلقه مغناطیسی

هر سلول از حافظه نوع حلقه مغناطیسی بنابر قاعده و نسبت به جهت جریان و جهت میدان مغناطیسی حالت صفر یا یک را به خود می گیرد . می دانیم که اگر از یک مسیر بسته میدان مغناطیسی عبور کند ، طبق قانون دست راست جریان در سیم ایجاد می شود .

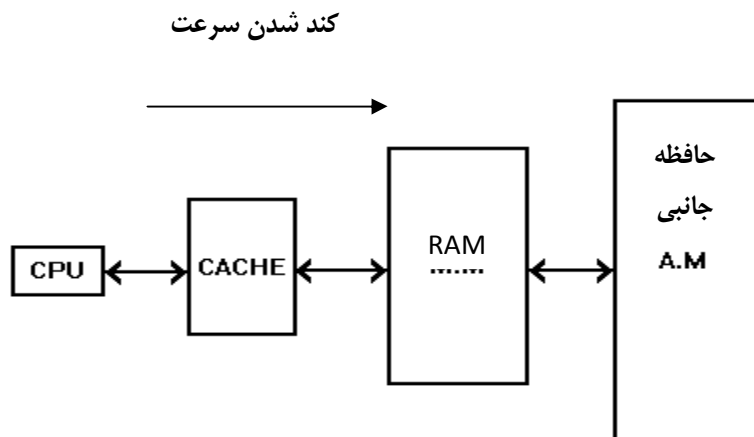


(شکل ۲-۶)

برای خواندن سیم هایی از این حلقه ها عبور می کند که در جهت مشخصی از آن جریانی عبور می دهیم که خود یک میدان مغناطیسی ایجاد می کند . میدان قبلی را با بعدی مقایسه می کنیم . اگر هم جهت باشند تغییری حاصل نمی شود ولی اگر هم جهت نباشند ، جهت میدان قبلی تغییر می کند و بعد از آن چون اطلاعات پاک می شود آن ها را دوباره می نویسیم .

## ۲-۲ : سلسله مراتب ذخیره سازی اطلاعات

شکل زیر اجزای یک سیستم سلسله مراتبی حافظه را نشان می دهد .



(شکل ۲-۷)

چون حجم اطلاعاتی که باید پردازش شود زیاد می باشد . پس ظرفیت حافظه ها هم بالا می رود واز آنجا که CPU ، روز به روز در حال پیشرفت است و سرعتش بیشتر می گردد، اگر بخواهیم با پیشرفت CPU سرعت حافظه را زیاد کنیم ، به علت گرانی حافظه، هزینه زیاد می شود . لذا از Cache استفاده می کنیم که از حافظه اصلی حجمش کمتر بوده ولی سرعتش بیشتر می باشد . در واقع سرعت آن حداکثر، برابر سرعت CPU است . Cache در کامپیوتر های جدید معمولاً دو سطحی است: داخلی و خارجی که Cache داخلی درون CPU قرار دارد . به ازای ظرفیت Cache اطلاعات از حافظه اصلی به داخل Cache می آید. برای مثال اگر ظرفیت Cache ۴ کیلو بایت باشد به اندازه 4KB از حافظی اصلی بر روی Cache می نشیند و Cache نیز با CPU در ارتباط می باشد . به همین خاطر پیشنهاد می شود که در داخل برنامه از حلقه هایی با بدنه طولانی استفاده نشود . زیرا حلقه های کوتاه کاملاً در Cache جای می گیرند ، ولی اگر حلقه طولانی باشد ، باید نیمی از حلقه به Cache بیاید و نیمه دیگر ، دوباره به Cache فراخوانی شود و در نتیجه کارایی Cache پایین می آید . به همین دلیل است که از دستور Goto در برنامه ها کمتر استفاده می شود و یا اصلاً استفاده نمی شود .

فرض کنید که ظرفیت Cache ۱۰۰ سطر و حلقه ی ما به طول ۱۵۰ سطر باشد . ابتدا ۱۰۰ سطر اول درون Cache قرار می گیرد و برای آوردن ۵۰ خط بعدی باید سطرهای قبلی پاک شود . در نتیجه برنامه ناقص خواهد شد و سرعت خیلی پایین می آید .

#### تمرین : برآورد اندازه Cache در کامپیوتر

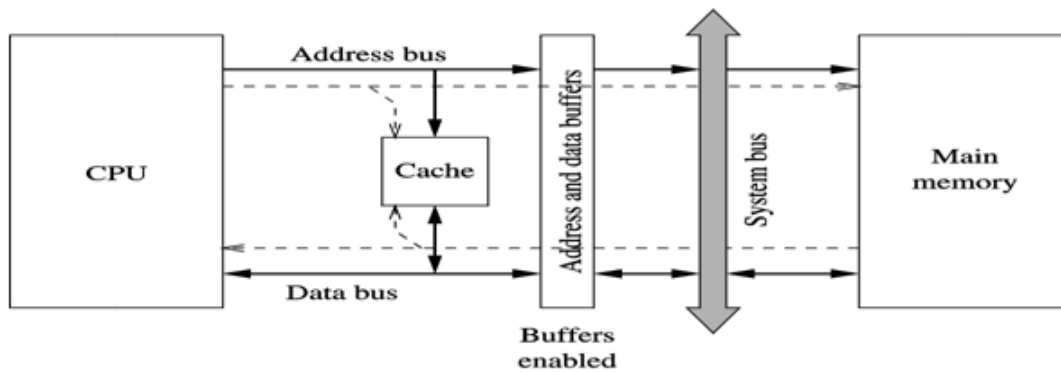
فرض کنید ظرفیت Cache ، 4Kb باشد . ابتدا برنامه ای بنویسید که آرایه ی 1Kb را پردازش کرده و در هر مرحله 1Kb به آن اضافه کرده تا بتوانید میزان تقریبی Cache را بدست آورید و همچنین زمانی که هر پردازش در هر مرحله طول کشیده است را ذخیره و باهم مقایسه کرده و نمودار آن را رسم کنید.

نرخ برخورد (hit ratio) در Cache یعنی احتمال اینکه یک سطر در حافظه باشد، چقدر است و از طریق تقسیم تعداد دفعاتی که محلی که می‌خواهیم و در Cache هست به کل دفعاتی که عمل خواندن صورت می‌گیرد، بدست می‌آید.

هر چه از حلقه‌های بزرگ‌تر از سایز Cache استفاده کنیم، hit ratio پایین می‌آید و در نتیجه باعث پایین آمدن کارایی Cache خواهد شد.

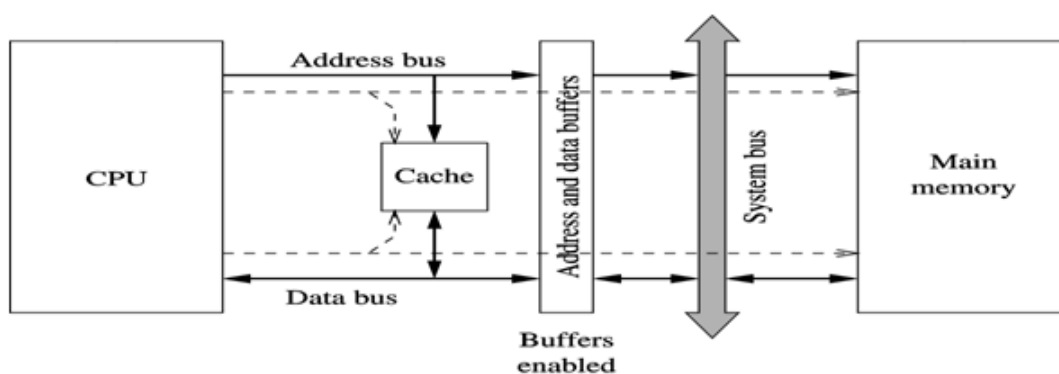
$$\text{Miss ratio} = 1 - \text{hit ratio}$$

هنگامی که CPU می‌خواهد اطاعتی را از حافظه بخواند و hit نیست، اطلاعات همزمان هم به داخل Cache و هم به داخل CPU می‌رود.



(شکل ۲-۸)

و هنگامی که CPU می‌خواهد اطلاعاتی را بنویسد و hit باشد، اطلاعات همزمان هم در Cache و هم در RAM نوشته می‌شود.



(شکل ۲-۹)

تمرین : برای ماتریسی که در اختیار دارید ، نحوه ی پردازش مناسب چگونه است ؟ سطری یا ستونی

برای این کار برنامه ای بنویسید و برای یک ماتریس یک با پردازش سطری را انجام داده و یک بار پردازش ستونی سپس اندازه ی ماتریس را تغییر داده و مدت زمان انجام هر کدام از مراحل را ذخیره کرده و نمودار آنرا رسم نمایید .

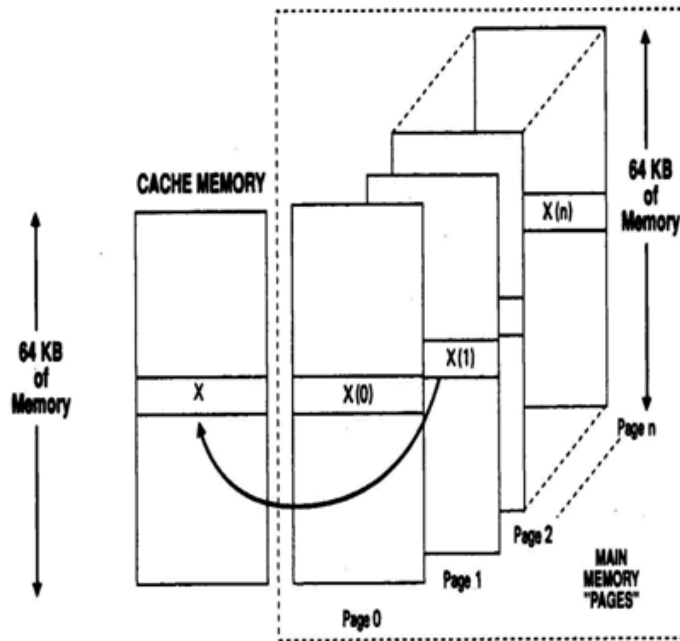
انتقال اطلاعات از RAM به Cache بلاک بلاک است و آدرس آن دو قسمت دارد : شماره بلاک و offset بلاک ( سطر مربوطه در بلاک مشخص شده ) ولی انتقال اطلاعات از CPU به Cache بصورت word است.

۲-۳ : انواع Cache

۲-۳-۱ Direct Mapping

در این نوع از Cache هر سطر از RAM جای مشخصی در Cache دارد. فضای RAM به طولی برابر Cache تقسیم شده و هر سطر در جایی مشخص درج می شود که این روش ساده ترین و کم هزینه ترین است .

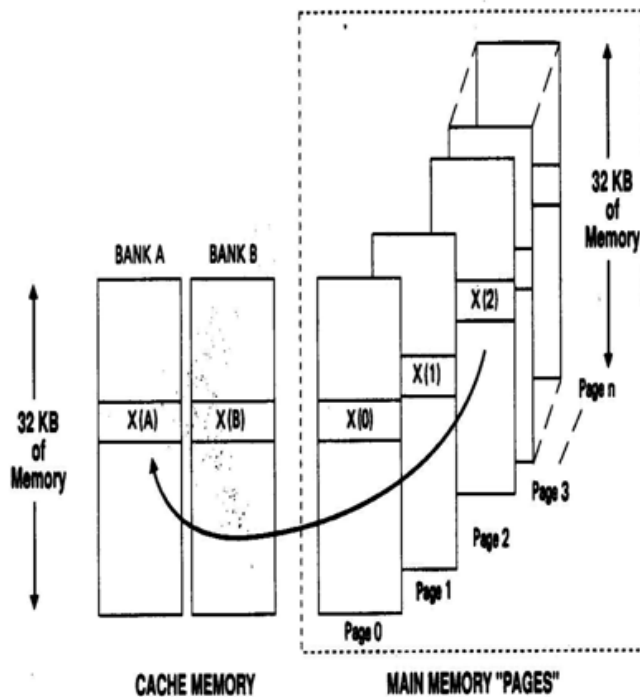




(شکل ۲ - ۱۰)

Set - Associative - ۲ - ۳ - ۲

در این نوع Cache، هر سطر از RAM در تعدادی محل از Cache قابل نشستن است. برای Cache دو تا پشتیبان داریم. یعنی دو آزادی عمل (یا در نیمه چپ یا در نیمه راست)



( شکل ۲-۱۱ )

### ۲-۳-۳ - Associative mapping

در این نوع از Cache، هر سطر از RAM در هر جا از Cache می تواند بنشیند و این روش پر هزینه ترین خواهد بود.

هر سطر از Cache شامل قسمت های Date و tag و valid می باشد که همان اطلاعات درون Cache و tag قسمت آدرس آن است و valid مشخص می کند که آیا این سطر مطابق RAM است یا خیر

مثال : فرض کنید دستور زیر صادر شود.

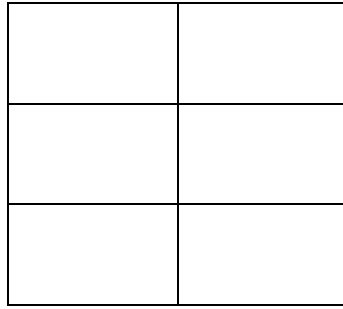
LDA 10

CPU ابتدا به سراغ Cache می رود و اگر دستور مورد نظر در آن وجود داشته باشد، از آن استفاده می کند ولی اگر این دستور در Cache نباشد، از حافظه اصلی آنرا به Cache انتقال می دهد و سپس از آن استفاده می کند (یعنی آنرا اجرا می کند).

### مشکل اول

از کجا بدانیم محتوای خانه 10 که آن را در درون Cache مورد جستجو قرار می دهیم، چیست ؟ برای رفع این مشکل، Cache را به شکل زیر در نظر می گیریم بنابراین در مثال LDA 10 بدنبال شماره ای بنام سطر 10 در داخل ستون شماره سطرهای Cache می گردد .

| شماره سطر | محتوی |
|-----------|-------|
|-----------|-------|



(شکل ۲-۱۲)

### مشکل دوم

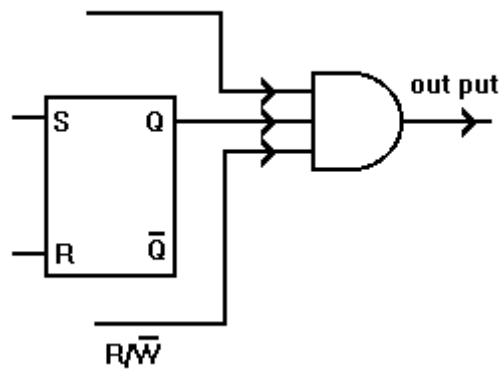
وقتی که گفته می شود سطر 10 را بخوان ، در داخل Cache جستجو می نماید که آیا سطر 10 در Cache وجود دارد یا خیر ؟ برای این عمل جستجو ، باید تمامی حافظه Cache را پیمایش نماید که این کارایی Cache را پایین می آورد و عملیات مورد نظر طولانی می باشد . پس برای رفع این مشکل ساختار حافظه انجمنی را بیان می کنیم . در واقع حافظه Cache کمی امکاناتش از حافظه اصلی بیشتر است.

جستجو به صورت موازی صورت می گیرد یعنی یک پالس کافی است تا به نقطه ی مورد نظر برسیم و این یکی از دلایلی است که قیمت Cache بالاست .

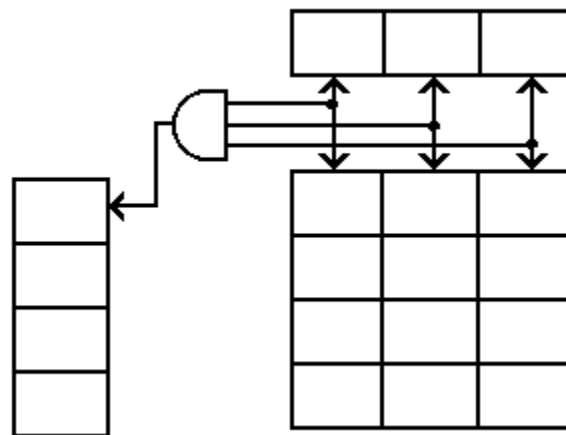
### ۲-۴: حافظه انجمنی

در حافظه انجمنی مقایسه بصورت موازی انجام می شود و ما در این حافظه ها داده (Data) را می دهیم و آدرس آنرا می گیریم ، برخلاف حافظه های معمولی که آدرس را می دهیم و داده مورد نظر را دریافت می داریم .

### ساختار داخلی حافظه انجمنی



(شکل ۲-۱۳)



(شکل ۲-۹)

\* حافظه انجمنی مانند حافظه های معمولی قابلیت دستیابی بصورت تصادفی (Random) را دارد.

قسمتهای متناظر که در شکل (۲-۹) نشان داده شده است با هم یای انحصاری (EX - Clusive NOR) می شوند و حاصل این یاهای انحصاری با هم AND می شود و به ازای هر سطر از حافظه انجمنی یک بیت را در نظر می گیریم ، که حاصل AND شده برای هر سطر در بیت همان سطر قرار می گیرد . واضح است وقتی حاصل بیت انجمنی یک می شود که ، محتوای ثبات با سطر حافظه انجمنی برابر باشد و می تواند چندین بیت از بیتهای تشخیص داده موردنظر در حافظه انجمنی Set (یعنی یک ) باشند . در این حالت این داده در چند سطر از حافظه انجمنی تکرار شده است .

حال این سؤال مطرح می شود که وقتی اطلاعات موردنیاز در داخل Cache نباشد، باید CPU اطلاعات را از داخل حافظه اصلی به داخل CPU بیاورد و سپس آنرا به داخل Cache ببرد، در اینصورت لزوم وجود Cache چیست ؟

در پاسخ باید گفت که اطلاعات به داخل CPU آورده نمی شود . بلکه CPU یک خط کنترلی را فعال کرده و آن وسیله دیگر ، این کار را انجام می دهد . پس این امر موضوع جدیدی را که در فصل آینده بحث خواهد شد ، فراهم می آورد .

تمرینات

- ۱- حافظی RAM از جنس ..... برخلاف ..... با قطع شدن برق محتوای خود را از دست می دهد. سرعت حافظه از جنس ..... کمتر از دیگری است.
- ۲- در ..... برخلاف حافظه های RAM با دادن محتوا ، آدرس محل یا محل های قرار گرفتن آن در حافظه بدست می آید.
- ۳- ساختار داخلی حافظی RAM نوع Semiconductor برای یک حافظی  $4 \times 5$  را نمایش دهید؟
- ۴- مدار داخلی یک سلول از حافظی RAM را رسم نمایید؟
- ۵- جایگاه حافظی Cache ونحوی دسترسی به اطلاعات واقع در حافظی اصلی برای CPU ای که مجهز به حافظی Cache باشد را توضیح دهید؟

### ۳- روشهای متفاوت انتقال اطلاعات بین کامپیوتر و دستگاههای جانبی

برنامه ها و توابع محاسباتی توسط دو مولفه یکی CPU و دیگری حافظه اصلی انجام می پذیرد . اما می دانیم که در هر کامپیوتر ، ما نیازمند به برقراری ارتباط بادنیای کامپیوتر هستیم ، تا بتوانیم اطلاعات را جهت پردازش از خارج بگیریم ویاآنکه اطلاعات پردازش شده را به کاربر نشان بدهیم .از دستگاههای ورودی و خروجی می توان به دیسک خوانها ، چاپگرها و ... اشاره نمود، که آنها را با نام ابزارهای ورودی / خروجی (I/O Device) می شناسیم .

شیوه های مختلفی برای رد و بدل کردن اطلاعات بین I/O Device و کامپیوتر وجود دارد که این شیوه ها عبارتند از :

(۱) Programmed I/O

(۲) Interrupt

(۳) DMA (Direct Memory Access)

#### ۳-۱: برنامه ریزی ورودی / خروجی (programed I/O)

در این روش رد و بدل شدن اطلاعات بین دستگاههای جانبی و حافظه اصلی با دخالت مستقیم CPU انجام می شود و دقیقاً از کانال پردازنده عبور می کند به عبارت دیگر دستگاههای جانبی با حافظه اصلی ارتباط ندارند . برای آنکه اطلاعات از دستگاههای جانبی وارد حافظه شود ، بایستی توسط دستورالعملی به داخل پردازنده انتقال یافته و سپس توسط دستورالعمل دیگری از پردازنده به حافظه اصلی انتقال یابد و چون از کانال پردازنده عبور می کند و اینکه دائماً باید CPU ، I/O ها را چک کند ، دارای سرعت کمی است .

البته در این روش بایستی دقت داشت به اینکه ، در هنگام خواندن اطلاعات توسط CPU از دستگاه جانبی ، آن دستگاه برای چنین کاری مهیا باشد . برای عمل دست دادن ( Hand Shaking) لازم است که CPU مدت زیادی در انتظار به سر برد (تلفن بدون زنگ) . برای رفع این مشکل روش دوم را پیشنهاد می کنند.

### ۲-۳ : وقفه (Intrupt)

در این روش CPU دائماً مشغول اجرای دستورات خود است و چنانچه دستگاه خارجی نیاز به سرویس داشته باشد ، وقفه‌ای به پردازنده ارسال می شود. پردازنده نیز در پایان دستورالعمل جاری به آن وقفه رسیدگی خواهد کرد . دستگاه ها در این روش به یک Encoder متصل هستند و هنگامی که درخواستی ارسال می شود ، مشخص می شود که چه دستگاهی و یا چه آدرسی درخواست سرویس داده است . ( مثل هنگامی که زنگ تلفن به صدا در می آید . ) در پایان هر دستورالعمل وقفه ها چک می شوند . اگر وقفه داشتیم چون مدار ترکیبی است بدون اینکه زمانی تلف شود ، برنامه تغییر مسیر داده و با توجه اولویت بندی Encoder به سرویس لازم رسیدگی می کند .

با استفاده از Intrupt Acknowledge اطلاعات را در CPU ذخیره کرده و بعد از رسیدگی به سرویس وقفه بازمی گردد و چون اطلاعات قبل خود را ذخیره دارد ، به برنامه خودش ادامه می دهد .

دو نوع وقفه داریم : نرم افزاری و سخت افزاری که وقفه های نرم افزاری غیرقابل چشم پوشی هستند ولی با استفاده از گیت AND می توان از وقفه های سخت افزاری چشم پوشی کرد .

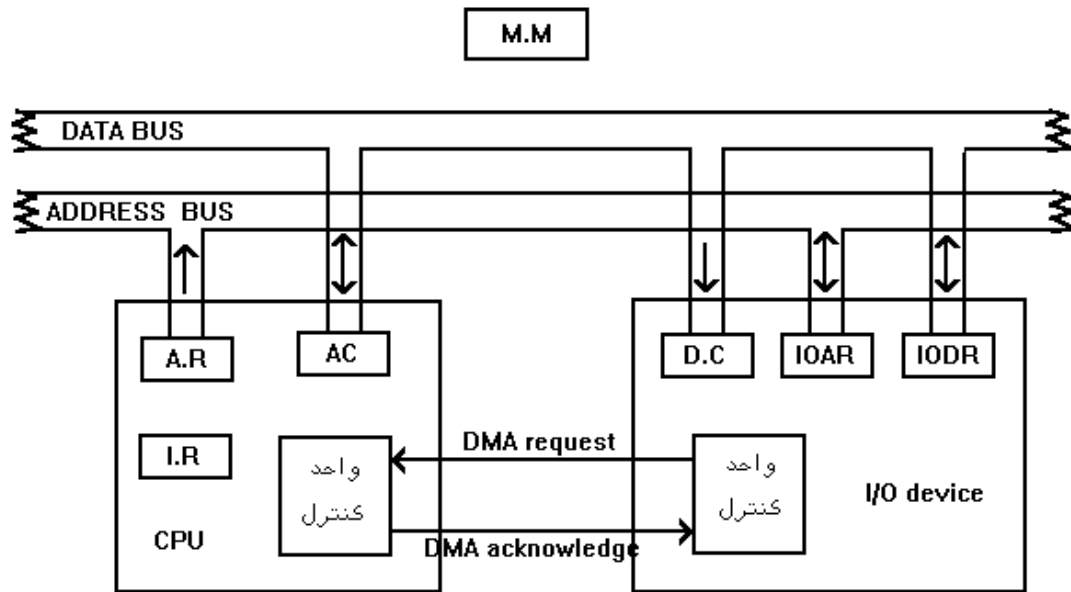
### ۳-۳ : دسترسی مستقیم به حافظه (DMA (Direct Memory Access))



یکی دیگر از معایب روش اول آن است که برای رد و بدل شدن اطلاعات از دستگاه جانبی به داخل حافظه، حتماً بایستی داده از کانال CPU عبور کند، که در حجمهای بالا باعث کندی سرعت می شود. برای رفع چنین مشکلی تکنیک DMA را مطرح می سازند که در این تکنیک دستگاه جانبی بدون دخالت مستقیم CPU بتواند اطلاعات را در داخل حافظه اصلی ذخیره و یا از آن باز یابی کند.

انتقال داده ها بین یک وسیله ذخیره سازی سریع از قبیل دیسک مغناطیسی و حافظه را اغلب سرعت CPU محدود می کند حذف CPU از مسیر انتقال و ایجاد این امکان که دستگاه جانبی مستقیماً گذرگاههای حافظه را کنترل کند، سرعت انتقال را بالا می برد. در حین انتقال به روش DMA، CPU بیکار است و کنترلی بر گذرگاههای حافظه ندارد یک کنترل کننده DMA به منظور اداری امر انتقال به طور مستقیم بین وسیله I/O و حافظه، کنترل گذرگاهها را برعهده می گیرد. در این روش داده ها (Data) بدون آنکه از کانال پردازنده بگذرد به مقصد می رسد، شیوی انجام کار به صورت زیر می باشد.

هنگامی که قرار است داده‌ای بین دستگاه I/O و حافظه اصلی رد و بدل شود، پردازنده آدرسی از حافظه اصلی که دستگاه I/O می تواند با آن آدرس ارتباط برقرار کند را به آن می دهد و سپس پردازنده خود را BUS جدا نموده و ارتباط دستگاه جانبی و حافظه اصلی با توجه به آدرسی که در اختیار دارد به طور مستقیم برقرار می شود. به طور کلی در مدت زمانهایی که پردازنده به حافظه اصلی نیازی نداشته باشد اجازی ارتباط مستقیم (DMA) بین دستگاه جانبی و حافظه اصلی را می دهد. در شکل (۳-۱) نحوی انتقال اطلاعات بین دستگاه جانبی و حافظه اصلی را مشاهده می کنید.

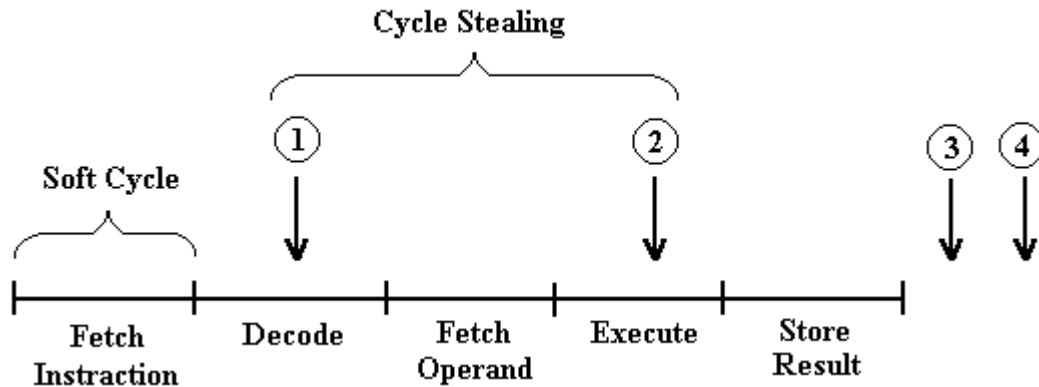


(شکل ۳-۱)

در واقع به DMA در پایان Inst Cycle جاری وزمانی که CPU به حافظه نیازی نداشته باشد اجازه داده می شود تا ارتباط مستقیم خود را برقرار نماید.

آدرس ابتدای اطلاعات مورد نیاز را در IOAR و تا جایی را که می خواهد در DC (یعنی تعداد سطرها را در آن قرار می دهد) که مخفف Data Count است، قرار می دهد. حالا که CPU دیگر اطلاعات را به I/O داد، خود را از I/O جدا می کند و کارهای دیگر خودش را انجام میدهد، حال در وسایل I/O هم AR، هم DR و هم تعداد داده ها را داریم (در DC) پس می تواند با Bus ارتباط برقرار کند و CPU با جداکردن خود از وسایل IO این اجازه ارتباط را می دهد پس در زمانهای مناسب DMA Acknowledge را فعال می کند و خط DMA Acknowledge را فعال نگه می دارد و دستگاه I/O هنگامی که این خط را فعال دید با حافظه اصلی براساس آدرس مشخص شده ارتباط برقرار می کند. هر لحظه که دستگاه I/O یک داده را منتقل می کند یک واحد از DC، کم می شود تا به صفر برسد. هر زمان که DC به صفر رسید سیگنالها به واحد کنترل آن فرستاده نمی شود از این جهت سیگنال DMA Request را غیر فعال می کند و CPU هم سیگنال DMA Acknowledge را غیرفعال می سازد و به این ترتیب ارتباط از بین می رود.

سؤال: در چه زمانی CPU به حافظه اصلی نیاز ندارد؟



( شکل ۳ - ۲ )

در لحظات ۱ و ۲ CPU نیاز به حافظه ندارد، مثلاً در سیستم Unix می توانیم ماکار خود را انجام دهیم و عمل چاپ نیز همزمان انجام شود و یادر هنگام رد و بدل شدن اطلاعات بین بافر نرم افزاری و سخت افزاری نیز همین شرط را داریم . یعنی در لحظاتی که ما به CPU کاری نداریم ، این عمل صورت می گیرد و در واقع از تکنیک DMA استفاده می کند. به وقفه در لحظه ۳ جواب داده می شود یعنی در پایان اجرای دستورالعمل . ولی به DMA در شماره های ۱ و ۲ و ۴ جواب داده می شود یعنی در حین اجرای دستورالعمل و پایان اجرای دستورالعمل CPU به DMA جواب می دهد .

\* بعضی از حلقه های دستورالعمل نیاز به ذخیره (Store) هم دارند ، مثل دستور مثال ۱ ولی در دستور مثال ۲ نیاز به قسمت Store cycle نیست .

ADD NUM , AX

مثال ۱ :

MOV AX , SUM

مثال ۲ :

\* در لحظاتی که CPU نیاز به حافظه اصلی (M.M) ندارد ، DMA Acknowledge می فرستد. ولی در Dos پس از بیکاری CPU به کارهای دیگر رسیدگی می شود .

در درس ذخیره و بازیابی اشاره شد که در بافرینگ مضاعف وقتی یک بافر در حال پردازش است ، بافر بعدی در حال پر شدن است که با تکنیک DMA صورت می گیرد ، CPU از موقعیتهای ۱ و ۲ استفاده می کند. همچنین اگر عمل پردازش بافر ۱ تمام شود ، ولی عمل پر شدن بافر ۲ تمام نشود باید منتظر بمانیم تا بافر ۲ پر شود و سپس عمل پردازش بافر ۲ صورت گیرد که در اینصورت CPU از موقعیت ۴ یعنی CPU Boned استفاده می کند.

CPU Boned یعنی برنامه با CPU درگیر است و مشغول کارهای پردازشی است .

### Cycle Stealing (دزدیدن چرخه)

در هنگام اجرای یک دستور العمل ، دستورالعمل به زیر حلقه هایی (Sub Cycle) تقسیم می شود و در زیرحلقه هایی که دستورالعمل به CPU نیاز ندارد و دستگاههای خارجی از CPU استفاده می کنند را Cycle Stealing می گویند .

تمرینات

۱- در روش ..... قبل از ارسال Data بین CPU و I/O آمادگی I/O برای برقراری این ارتباط توسط CPU تست می شود که به آن اصطلاحاً ..... گفته می شود.

۲- DMA چیست؟ در مورد آن توضیح دهید؟

۳- آیا عبارت زیر درست می باشد؟

- در روش DMA ارتباط دستگاه I/O با حافظه اصلی به طور مستقیم انجام می گیرد اما این ارتباط با مدیریت CPU می باشد. CPU در زمانی که به حافظه نیاز ندارد و تقاضای DMA وجود ندارد خود را از BUS جدا کرده و اجزای ارتباط با حافظه را به I/O می دهد.

۴- آیا عبارت زیر درست می باشد؟

- به تقاضای وقفه در پایان Inst.Cycle توجه می شود، اما به تقاضای DMA علاوه بر آن در طی یک Inst.Cycle نیز توجه می شود.

۵- توضیح دهید که CPU در چه مواقعی به تقاضای Interrupt و DMA پاسخ می دهد؟