اعداد: بدون علامت : $(1010)_2 = 10$ (ارقام از ۳ تا ۰)

علامت دار :

msb ← most significant bit — Lsb

- در سیستم اعداد علامت دار به سمت چپ ترین بیت sign bit می گویند

- عدد بدون علامت a در مبنا r بصورت روبه رو است

$$a = (a_{n-1} a_{n-2} \dots a_0 , a_{-1} a_{-2} \dots a_{-m})_r = \sum_{k=-m}^{n-1} a_k \times r^k$$

(قسمت صحیح: n بیت صحیح، از ۰ شروع می کنیم ← ؛ قسمت اعشار: m اعشار، از ۱- شروع می کنیم →)

$a_k \times r^k$ → a_k ارزش رقم

$0 \le i \le n-1$

$$\text{مقدار} = a_{-m} \times r^{-m} + \dots + a_{-1} \times r^{-1} + a_0 \times r^0 + \dots + a_{n-1} \times r^{n-1}$$

نکته) تعداد اعداد صحیح درون بازه

$a \le x \le b \Rightarrow b-a+1$

$a < x \le b$ و $a \le x < b \Rightarrow b-a$ (۱-)

$a < x < b \Rightarrow b-a-1$ (۲-)

نکته) در مبنا r ارقام از ۰ هستند تا r-1 .

$\text{رقم} \times (\text{مبنا})^{\text{اندیس}}$

تبدیل از مبنا r به مبنا ۱۰ = بدست آوردن مقدار اعداد مبنا r

1- $(753,4 02)_8 = 3\times8^0 + 5\times8^2 + 7\times8^3 + 4\times8^{-1} + 2\times8^{-3}$

نکته) $a^{-m} = \frac{1}{a^m}$

2- $(25C0A,B49)_{16}$ Hex $= A + 12\times16^2 + 5\times16^3 + 2\times16^4 + 11\times16^{-1} + 4\times16^{-2} + 9\times16^{-3}$

A = 10
B = 11
C = 12
D = 13
E = 14
F = 15

3- $(1100111,01)_2 = 2^0 + 2^1 + 2^2 + 2^5 + 2^6 + 2^{-2}$

مثال) مقدار عدد $(0,\overline{35})_7$ در مبنا ۱۰ برابر چند است؟

$$(0,\overline{35})_7 = 3\times7^{-1}+5\times7^{-2}+3\times7^{-3}+5\times7^{-4}+3\times7^{-5}+5\times7^{-6}+\cdots$$

$$3(7^{-1}+7^{-3}+7^{-5}+\cdots)+5\times(7^{-2}+7^{-4}+7^{-6}+\cdots) = 3\times\frac{\frac{1}{7}}{1-\frac{1}{49}}+5\times\frac{\frac{1}{49}}{1-\frac{1}{49}}$$

$$3\times\frac{1\times48}{7\times48}+5\times\frac{1\times48}{48\times48}=\frac{26}{48}$$

یادآوری تصاعد هندسی

تعداد جملات

$$a+ad+ad^2+ad^3+\cdots+ad^n = \text{جمله اول}\times\frac{1-(\text{قدرنسبت})^{n+1}}{1-\text{قدرنسبت}} = a\times\frac{1-d^{n+1}}{1-d}$$

$$\lim_{n\to\infty} a+ad+\cdots+ad^n = \lim_{n\to\infty} a\times\frac{1-d^{n+1}}{1-d} = \frac{a}{1-d} = \frac{\text{جمله اول}}{1-\text{قدرنسبت}}$$

$2+6+18+54+\cdots$
$d=3$

$|d|<1 \rightarrow -1<d<1$

مثال) مقدار عدد $(2,\overline{35})_{10}$ را به صورت کسری درآورید.

$x=(2,\overline{35})$ ← طرفین را در ۱۰۰ ضرب می‌کنیم ← $100x=(235,\overline{35})$ ← ۲ عبارت را از هم کم کنید

$$99x=233 \Rightarrow x=\frac{233}{99}$$

تبدیل از مبنا ۱۰ به سایر مبناها:

قسمت صحیح عدد را متوالیا بر ۲ تقسیم می‌کنیم و قسمت اعشار عدد را متوالیا در ۲ ضرب می‌کنیم و هر بار قسمت صحیح حاصلضرب را بعد از ممیز یادداشت می‌کنیم

مثال) $(3.1,2)_{10} = (?)_8$

۳۰۱	۸	
۲۴	۳۷	۸
۶۱	۳۲	۴
۵۶	۵	
۵		

$(3.1,2)_{10} = (4\overline{85},1463)_8$

$0,2\times8=1,6$
$0,6\times8=4,8$
$0,8\times8=6,4$
$0,4\times8=3,2$
$0,2\times8$

معماری کامپیوتر

رامین رضوی

نکته) مقدار بزرگترین عدد صحیح n رقمی مبنا r برابر است با $r^n - 1$

مثال) بزرگترین عدد صحیح ۳ رقمی مبنا ۸ چه شکلی است و چند است؟ $(VVV)_8 = 8^3 - 1$

محاسبات در مبناهای دیگر

$(256)_7 + (445)_7 = (1034)_7$

$(256)_7 + (445)_7 = (2012)_7$

$(1000)_8 = 1 \times 8^3 \Rightarrow (VVV)_8 = 8^3 - 1$

$(111)_8 = 1 \times 8^0 = 1$

$(100\ldots0)_r = 1 \times r^n$ (n صفر)

$(11\ldots1)_r + 1 = 1 \times r^0 = 1$ (n رقم یک)

$\Rightarrow (11\ldots1)_r = r^n - 1$

نکته) مقدار بزرگترین عدد اعشاری n رقمی مبنا r برابر است با: $1 - r^{-n}$

مثال) بزرگترین عدد اعشاری ۳ رقمی مبنا ۶ چه شکلی است و چه مقداری دارد؟

$(0.555)_6 = 1 - 6^{-3}$

$(0.555)_6 + (0.001)_6 = 1 \times 6^{-3}$

$(1.000)_6 = 1 \times 6^0 = 1$

$\Rightarrow (0.555)_6 = 1 - 6^{-3}$

$(VVV.VV)_8 = ?\quad 8^3 - 1 + 1 - 8^{-2} = 8^3 - 8^{-2}$

نکته) اگر می‌خواهید ببینید عدد n در مبنا ۱۰ در مبنا r چند رقمی است داریم: $\lceil \log_r^{n+1} \rceil$

$9 = 1001$

$\vdots$

$15 = 1111 \Rightarrow \lceil \log_2^{15+1} \rceil = 4$

$4 = (100)_2 \Rightarrow \lceil \log_2^{4+1} \rceil = 3$

$5 = 101$

6

$7 = 111 \Rightarrow \lceil \log_2^{7+1} \rceil = 3$

$8 = 1000 \Rightarrow \lceil \log_2^{8+1} \rceil = 4$

مثال) برای نمایش اعداد ۳۰ رقمی مبنا ۱۰ به مبنا ۲ چند بیت نیاز است؟

بزرگترین عدد ۳۰ رقمی مبنا ۱۰ = $10^{30}-1$ ⟵ $\lceil \log_2^{10^{30}-1+1} \rceil = \lceil 30 \log_2^{10} \rceil = 100$

مثال) مقدار عدد 111111111 (ارقام ۹۸۷۶۵۴۳۲۱۰) چند است؟ 2^9-1

" " 1110111011 " " ? $2^9-1-2^6-2^2$

جمع و تفریق اعداد بی علامت:

مثال) دو عدد ۶ و ۳ را در ۴ بیت باهم جمع کنید

$$6 = 0110$$
$$3 = 0011$$
$$1001 = 9 \quad C=0$$

" " " ۹ و ۸ " " " " " " " "

$$9 = 1001$$
$$8 = 1000$$
$$10001 \quad C=1$$

فلگ سرریز یا overflow:

اگر حاصل در فضا خودش جا نشود سرریز داریم
" " غلط باشد سرریز داریم

- اگر در سیستم نمایش عددمان، عددی را که بدست می آوریم از رنج نمایش خارج باشد overflow رخ میدهد

نکته) به فلگ overflow علامت دار می گویند overflow
" " " بدون علامت " carry

	بدون علامت	علامت دار
1111	15	-1
+0011	3	+3
10010	$18 \Rightarrow C=1$	$2 \Rightarrow V=0$

رنج $-8 \le \dots \le 7$

معماری کامپیوتر

رامین رضوی

نکته ۱) هیچ‌گاه در فلگ carry و overflow همزمان به درد کسی نمی‌خورد، چون یا داریم از اعدادمان بصورت علامت دار استفاده می‌کنیم و یا داریم بصورت بدون علامت استفاده می‌کنیم

اگر محاسبات بدون علامت داریم به C نگاه می‌کنیم و اگر محاسبات علامت دار داریم به V نگاه می‌کنیم

بدون علامت	علامت دار
۹	-۷
۳	+ ۳
۱۲ → C = ۰	-۴ ⇒ V = ۰

۱۰۰۱	
+ ۰۰۱۱	
۱۱۰۰	

نکته ۲) اگر ۲ عدد n بیتی بدون علامت را با هم جمع کنیم و حاصل را در n بیت بریزیم، امکان وقوع سرریز وجود دارد ولی اگر حاصل را در n+1 بیت بریزیم سرریز رخ نمی‌دهد

بی علامت	n بیت
$2^n - 1$	n بیت +
$+ 2^n - 1$	n+1 بیت
$2^{n+1} - 2$	

⇓

$max = 2^{n+1} - 1$

سرریز رخ نمی‌دهد ⇒ $2^{n+1} - 2 < 2^{n+1} - 1$

تفریق دو عدد بدون علامت:

۲ روش برای تفریق ۲ عدد وجود دارد:
- ۱- با تبدیل به جمع
- ۲- بصورت مستقیم و با استفاده از subtractor

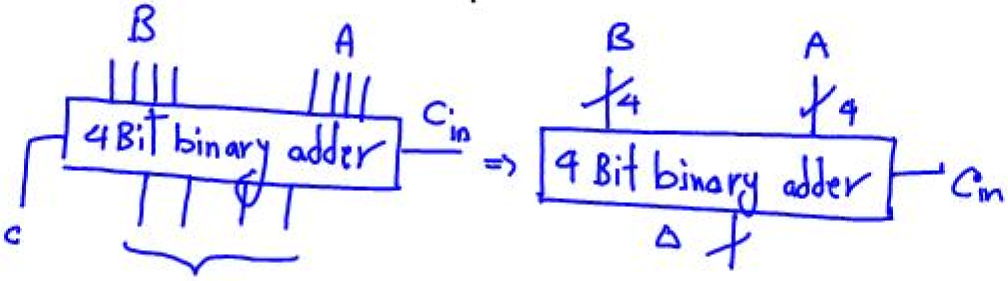
با تبدیل به جمع: $(A)_r - (B)_r = (A)_r +$ مکمل B در r $= (A)_r +$ مکمل B در r-1 + 1

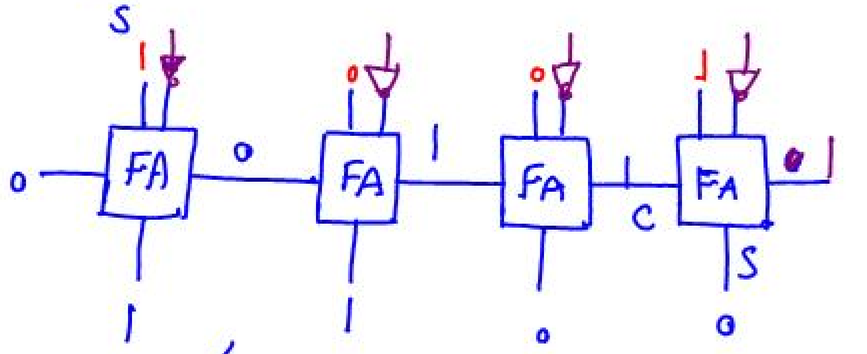
$(A)_{10} - (B)_{10} = (A)_{10} +$ مکمل B در 10 $= (A)_{10} +$ مکمل B در 9 + 1

$(A)_2 - (B)_2 = (A)_2 +$ " " 2 " $= (A)_2 +$ " یک B + 1

A = 1001
B = 0011
00

تحت اجرا





تفریق بدون علامت با تبدیل جمع: عدد اول را با مکمل دو عدد دوم جمع می‌کنیم که در اینصورت اگر

۱) $C=1 \longleftrightarrow A \geq B \longleftrightarrow$ حاصل درست است

۲) $C=0 \longleftrightarrow A < B \longleftrightarrow$ " غلط " و حاصل درست می‌شود مکمل دو

حاصل فعلی یک منفی پشت اش

فلگ Z=0 است.

۲) انجام تفریق بصورت مستقیم (با قرض گرفتن)

۱) $c=0 \longleftrightarrow A \geq B$: حاصل درست است

۲) $c=1 \longleftrightarrow A < B$: حاصل غلط است و حاصل درست، مکمل ۲ حاصل فعلی یک منفی پشت اش است.

```
A = 1011 = 11        A = 0101 = 5
B = 0110 = 6         B = 1001 = 9
---------------      ---------------
    0101 = 5             1100 = 12  →  -0100
                     C=1
```

فلگ Z=0 است (بارو)

نکته) کری در تفریق زمانی 1 می شود که ما از سیستم قرض بگیریم. بنابراین وقتی ما تفریق را بصورت مستقیم انجام می دهیم فلگ B = c نشان دهنده این است که آیا ما از سیستم قرض گرفته ایم یا خیر.

مثال) 12-9 را به ۲ روش گفته شده انجام دهید ($A \geq B$)

روش ۱: با تبدیل به جمع:

```
  1100        1100
- 1001   →  + 0111
           (1)0011 = 3
```

روش ۲: بصورت مستقیم:

```
  1100
- 1001
------
  0011 = 3     B=0
```

مثال) 9-12 را به ۲ روش گفته شده بدست آورید؟ ($A < B$)

روش ۱: با تبدیل به جمع:

```
  1001        1001
- 1100   →  + 0100
           (0)1101 = 13
                ↓
          مکمل ۲ = -0011
```

روش ۲: بصورت مستقیم:

```
  1001
- 1100
------
  1101 = 13     B = C = 1
```

اعداد علامت دار

در حال حاضر با ۳ سیستم ۱- علامت مقدار ۲- مکمل ۱ ۳- مکمل ۲ آشنا خواهیم شد.

معماری کامپیوتر

رامین رضوی

۱ـ علامت مقدار : sign bit فقط علامت عدد را مشخص می‌کند و ارزشی ندارد ، یکی از بدی‌های این سیستم وجود ۲ سمبل برای ۰ است .

0000 = +0
1000 = −0

توجه) در این سه سیستم بیت سمت چپ ۰ نشان دهنده اعداد مثبت و بیت سمت چپ ۱ نشان دهنده اعداد منفی است .

* رنج در ۴ بیت :

min = 1111 = −7 (max)
1110 = −6
⋮
1000 = −0
0000 = +0
0001 = +1
⋮
0111 = +7

۱۵ عدد را نشان میدهد ⇒ رنج ≤ 7−

بدی ۲ :

۱ـ وجود ۲ صفر که از تمامی سمبل ۲ بصورت بهینه استفاده نشده

۲ـ جمع و تفریق در این سیستم کند و زمان گیر است ← تا خیر با ۲

نکته) max و min عدد n بیتی در این سیستم برابر است با : $1-2^{n-1} \le$ رنج $\le (1-2^{n-1})-$

$max = 0\underbrace{11\cdots1}_{n-1} = 2^{n-1}-1$ $\quad min = 1\underbrace{11\cdots1}_{n-1} = -(2^{n-1}-1)$

سیستم مکمل ۱ : در سیستم مکمل ۱ همچنین مکمل ۲ سمت چپ ترین بیت علاوه بر اینکه علامت عدد را نشان میدهد ، مقدار منفی نیز دارد .

min = 1000 = −7
1001 = −6
⋮
1111 = −0
0000 = +0
0001 = +1
⋮
0111 = +7

* رنج در ۴ بیت :

۱۵ عدد را نشان میدهد ⇒ رنج ≤ 7+ ، رنج ≥ 7−

توجه) سیستم مکمل ۱ مشکل دوم بالا را بصورد داده .

نکته) min و max عدد n بیتی سیستم مکمل ۱ برابر است با :

$min = 100\ldots0 = -(2^{n-1}-1)$

$max = 011\ldots1 = 2^{n-1}-1 \Rightarrow -(2^{n-1}-1) \leq \text{رنج} \leq 2^{n-1}-1$

۳- سیستم مکمل ۲ : مزیت‌ها : از تمامی سیستم‌ها استفاده شده و سرعت جمع و تفریق آن از ۲ سیستم دیگر کمتر است

$min = 1000 = -8$

$1001 = -7$

$\vdots$

$1111 = -8+7 = -1$

$0000 = +0$

$0001 = +1$

$\vdots$

$0111 = +7$

* رنج با ۴ بیت :

$\Rightarrow -8 \leq \text{رنج} \leq 7 \Rightarrow$ ۱۶ عدد را نشان میدهد

نکته) min و max عدد با n بیت در این سیستم بصورت زیر است :

$min = 100\ldots0 = -2^{n-1}$

$max = 011\ldots1 = 2^{n-1}-1$

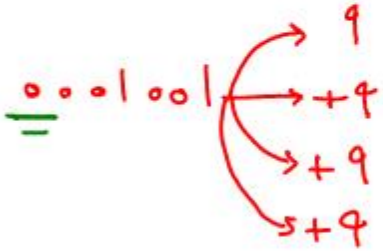
نکته) تفاوت میان این سه سیستم گفته شده فقط بر آن زمانی است که عدد ما منفی باشد و وقتی عدد مثبت است یعنی سمت چپ‌ترین بیت ۰ است، مقدار عدد در این سه سیستم یکسان است

مثال : مقدار ۲ عدد ۱۰۰۱۰۰۱ و ۱۰۰۱۰۰۰ را در ۴ سیستم بدون علامت، مقدار علامت، مکمل ۱ و مکمل ۲ بدست آورید

1001001

- بدون علامت: $2^6+2^3+1 = 73$
- علامت: -9
- مکمل ۱: $-63+9 = -54$
- مکمل ۲: $-64+9 = -55$

مقدار اعداد در این ۴ سیستم گفته شده با توجه به فرمول‌های زیر بدست می‌آید.



$$a=(a_{n-1}a_{n-2}\cdots a_1a_0)_r$$

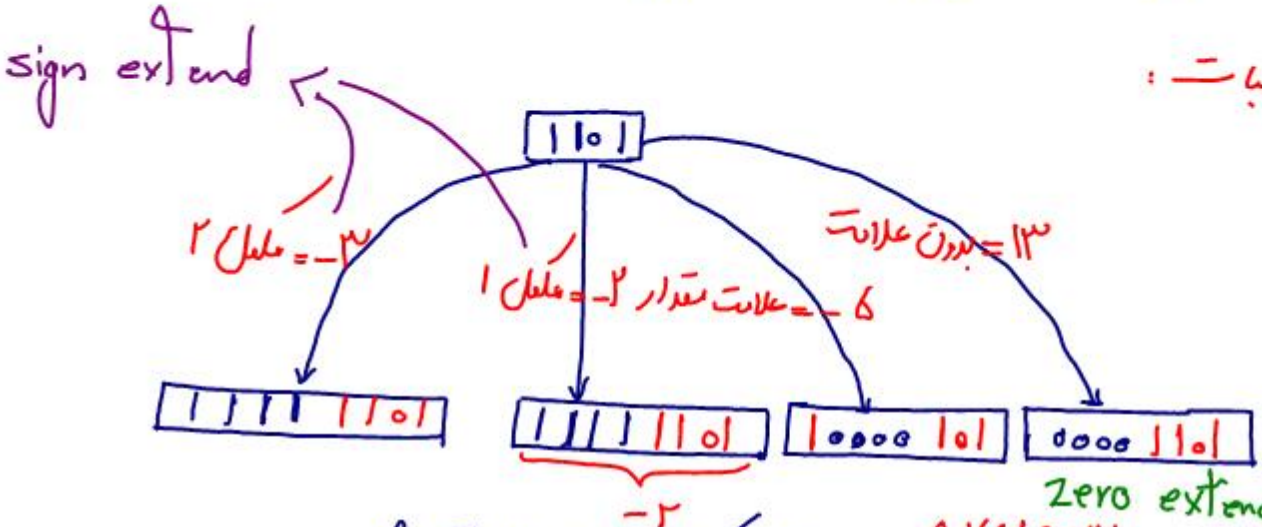
$$\text{بی علامت} = a_{n-1}\times r^{n-1}+a_{n-2}\times r^{n-2}+\cdots+a_0\times r^0$$

$$\text{مقدار علامت} = (-1)^{a_{n-1}}\times(\quad *\quad)$$

$$\text{مکمل ۱} = a_{n-1}\times-(r^{n-1}-1)+(\quad *\quad)$$

$$\text{۲ مکمل} = a_{n-1}\times-r^{n-1}+(\quad *\quad)$$

گسترش یک ثبات:



سوال ۱: مقدار عدد $(11111111)_2$ در سیستم مکمل ۲ چند است؟

$$-2^7+2^6+2^5+\cdots+2^0=-1$$

$$(11111111)_2=-2^0=-1$$

$$(10010)_2=-16+2=-14$$

آشنایی با فلگ‌ها (پرچم)

بیت‌هایی در ثبات‌های cpu هستند که بعد از انجام عملیات محاسباتی نظیر جمع و تفریق و ضرب و ... تنظیم می‌شوند

فلگ C (carry): اگر از سمت چپ‌ترین بیت رقم نقلی خارج شود، این پرچم ۱ می‌شود

Z (Zero): وقتی حاصل عبارتی ۰ بشود این فلگ ۱ می‌شود

S (Sign) یا N (Negative): اگر حاصل منفی باشد این فلگ ۱ می‌شود

O یا V (overflow): اگر حاصل در فضایی که برایش در نظر گرفتیم نگنجد سرریز رخ میدهد.

سیستم مکمل ۲

جمع در سیستم مکمل ۲

مثال) جمع‌های زیر را در ۴ بیت انجام دهید و فلگ‌ها را مشخص کنید

توجه) هنگامی که می‌خواهیم جمع را انجام بدهیم ابتدا تعداد بیت‌های آن ۲ عدد را یکسان می‌کنیم

۰۱۱۱ = +۷	۱۰۱۰ = -۶	۱۰۱۰ = -۶	۱۱۱۰ = -۲
+ ۰۱۱۰ = +۶	+ ۱۰۰۱ = -۷	+ ۱۱۰۰ = -۴	+ ۰۱۱۰ = +۶
۱۱۰۱ = -۳	۱۰۰۱۱ = +۳	۱ ۱۰۱۰ = -۶	۱ ۰۱۰۰
			S
S=1	S=0	S=1	S=0
Z=0	Z=0	Z=0	Z=1
C=0	C=1	C=1	C=1
V=1	V=1	V=0	V=0

با ۴ بیت رنج سیستم مکمل ۲: $-8 \leq$ رنج ≤ 7

روش‌های تشخیص سرریز در سیستم مکمل ۲

۱- چک کردن اینکه آیا حاصل همان چیزی است که ما توقع داریم یا خیر

۲- چک کردن رنج: که اینکه آیا حاصل در رنج ما می‌گنجد یا خیر

۳- در جمع یک عدد + با یک عدد - و یا بالعکس سرریز نداریم زیرا در این حالت حاصل عادی

عددی است بین آنها ، اما اگر جمع ۲ عدد + منفی بشود و یا جمع دو عدد منفی + بشود سرریز داریم .

۴ - اگر رقم نقلی خارج شده از سمت چپ‌ترین بیت با رقم نقلی وارد شده به آن برابر نباشد سرریز داریم .

$V=1 \longleftrightarrow C_n \neq C_{n-1} \Rightarrow V = C_n \oplus C_{n-1}$

a	b	a⊕b	a+b
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1

- XOR تابعی است فرد ، یعنی اگر تعداد یک‌های ورودی فرد باشد خروجی ۱ می‌دهد

$\Rightarrow a.b=0 \longleftrightarrow a \oplus b = a+b$

$M.N=0 \longleftrightarrow M \oplus N = M+N$

$a \oplus b = \bar{a}b + a\bar{b}$

نکته ۳ و ۴ در واقع یکی است :

$a = 0 \cdots$ ← 1

$b = 0 \cdots$

$0\ 1$

$0\ 1 \cdots$

$+\ 1 \cdots$

$1\ 0$

فرمولی کردن ۳)

$C_n \quad C_{n-1} \quad C_{n-2} \quad \cdots \quad C_0$

$a = a_{n-1}\ a_{n-2} \cdots a_0$

$b = b_{n-1}\ b_{n-2} \cdots b_0$

$+$

$S_{n-1}\ S_{n-2} \cdots S_0$

$V=1 \longleftrightarrow a_{n-1}=0, b_{n-1}=0, S_{n-1}=1 \text{ OR } a_{n-1}=1, b_{n-1}=1, S_{n-1}=0$

$V = \underbrace{\overline{a_{n-1}}\ \overline{b_{n-1}}\ S_{n-1}}_{M} \oplus \underbrace{a_{n-1}\ b_{n-1}\ \overline{S_{n-1}}}_{N}$

نکته ۱ در هنگام وقوع سرریز $S_{n-1} = C_{n-1}$ است

$V = \overline{a_{n-1}}\ \overline{b_{n-1}}\ C_{n-1} + a_{n-1}\ b_{n-1}\ \overline{S_{n-1}}$

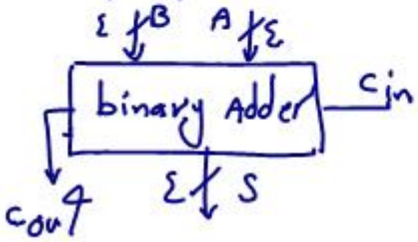
معماری کامپیوتر

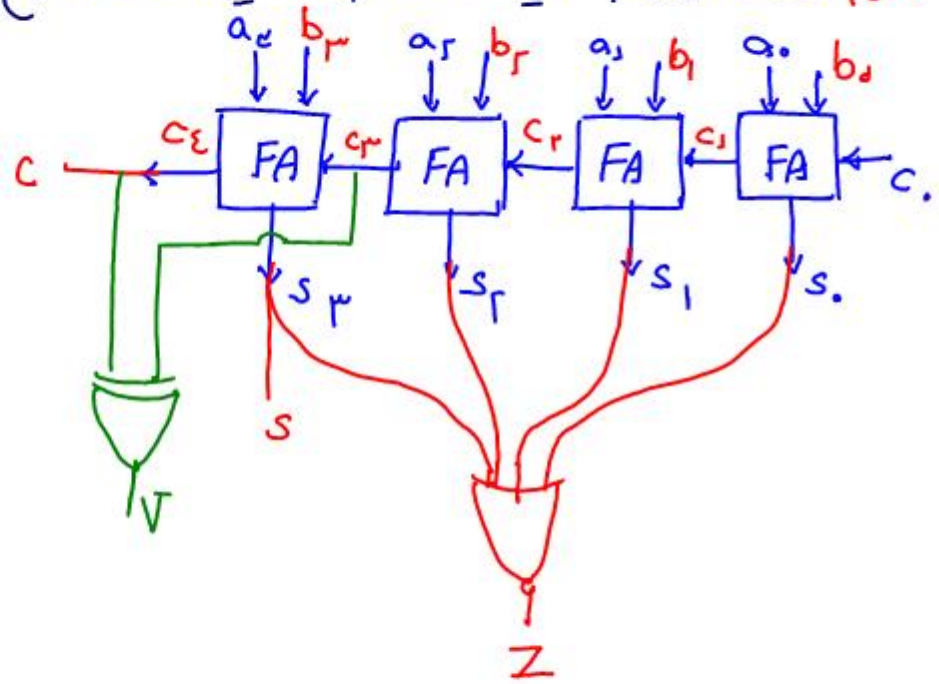
رامین رضوی

مثال) سفت افزاری بسازید که دو عدد چهار بیتی A و B را جمع کند و Eflag را تولید کند

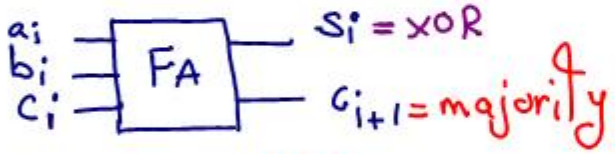
$A = a_3 a_2 a_1 a_0$

$B = b_3 b_2 b_1 b_0$





ساخت FA , HA :



a_i	b_i	c_i	S_i (XOR)	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$S_i(SOP) = \bar{a}_i \bar{b}_i c_i + \bar{a}_i b_i \bar{c}_i + a_i \bar{b}_i \bar{c}_i + a_i b_i c_i$

$c_{i+1} = \bar{a}_i b_i c_i + a_i \bar{b}_i c_i + a_i b_i \bar{c}_i + a_i b_i c_i$

$S_i = a_i \oplus b_i \oplus c_i$ - خروجی XOR زمانی است فرد یعنی اگر تعداد 1 های ورودی فرد باشد خروجی 1 می‌شود.

نکته) $a + a + a + a = a$

$c_{i+1} = \bar{a}bc + a\bar{b}c + ab\bar{c} + abc + abc + abc = ab + ac + bc$

$ab(\bar{c} + c) = ab$

$ac(\bar{b} + b) = ac$

$bc(\bar{a} + a) = bc$

معماری کامپیوتر

رامین رضوی

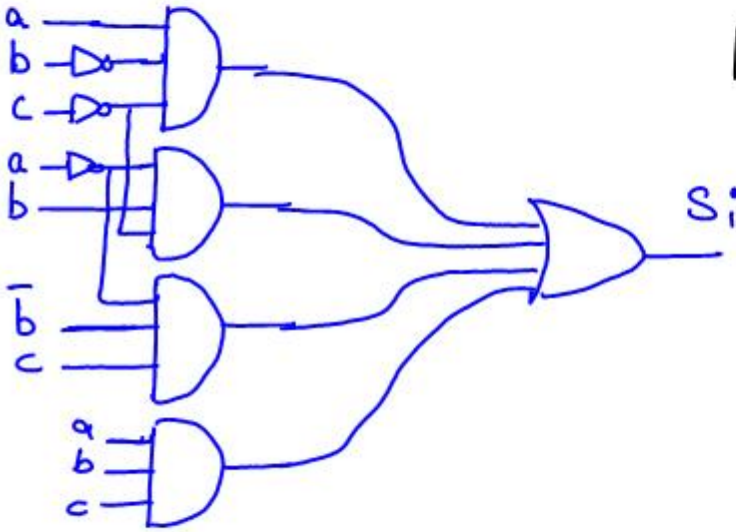
$S_i = a\bar{b}\bar{c} + \bar{a}b\bar{c} + \bar{a}\bar{b}c + abc$

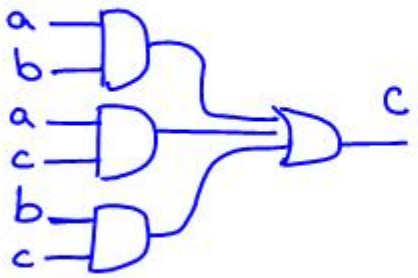
Single rail: تغییرgate not را در اختیار نداریم

Double rail: هم تغیرها و هم نقیض را داریم

- تاخیر گیت ساده را t بگیرید

- تاخیر: single rail = 3t
  double " = 2t



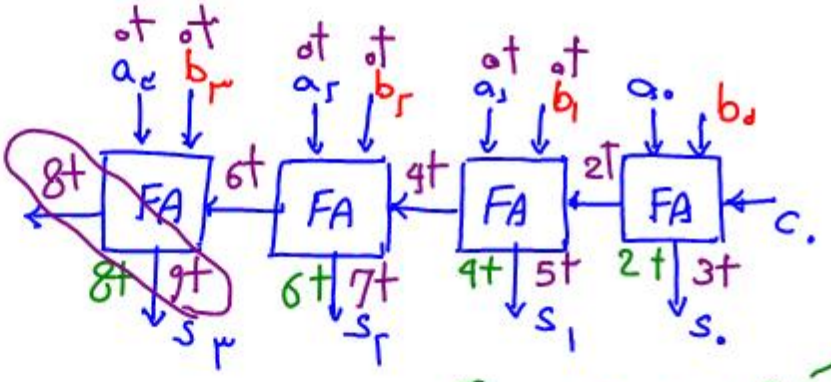


- تاخیر: single rail = 2t
  double " = 2t

- تاخیر جمع کننده ripple چهار بیتی:

single rail = 9t

Double " = 8t



- تاخیر جمع کننده RCA، n بیتی:

single rail = $(n-1)2t + 3t = 2nt + t$

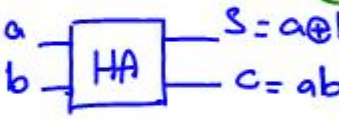
$n \times 2t + t$

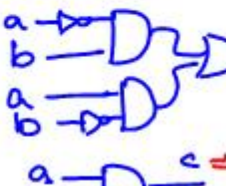
Double rail = $n \times 2t = 2nt$

a	b	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$s = \bar{a}b + a\bar{b}$

$c = ab$





معماری کامپیوتر

رامین رضوی

تفریق در سیستم مکمل ۲:

همیشه به دو صورت می‌توان تفریق کرد ① با تبدیل به جمع (بصورت غیرمستقیم) ② بصورت مستقیم

توجه: الزهدف ما از تفریق مشخص کردن فلگ‌ها است، بطور پیش فرض از روش ۱ استفاده می‌کنیم

و در روش ۱ هم بصورت زیر تفریق را انجام می‌دهیم $A-B = A + \text{مکمل یک } B + 1$

مثال) با ۲ روش گفته شده تفریق کنید و فلگ‌ها را مشخص کنید

(روش ۱)

$$\begin{array}{r} 1111 \\ 1110 \\ +\ 0000 \\ \hline 1110 = -2 \end{array}$$

$S=1,\ C=0,\ V=0,\ Z=0$

(روش ۲)

$$\begin{array}{r} 1 \\ 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ 0 \\ \hline 1 \end{array}$$

$$\begin{array}{r} 10 \\ 0 \\ \hline 10 \end{array} \qquad \begin{array}{r} 10 \\ 0 \\ \hline 10 \end{array}$$

$$\begin{array}{r} 10 \\ 0 \\ \hline 10 \end{array} \qquad \begin{array}{r} 10 \\ 0 \\ \hline 10 \end{array}$$

$$\begin{array}{r} 10000 \\ -\ 1111 \\ \hline 1110 \end{array}$$

$S=1,\ B=C=1,\ V=0,\ Z=0$

مثال) تفریق زیر را با تبدیل به جمع انجام دهید $0000 - 1001$

تبدیل به جمع همان طور که گفتیم

$$\begin{array}{r} 1111 \\ 1001 \\ +\ 1111 \\ \hline 11001 \end{array}$$

$S=1,\ C=1,\ V=0,\ Z=0$

تبدیل به جمع درحالتی که دوی را مستقیم مکمل ۲ کنیم

$$\begin{array}{r} 1001 \\ +\ 0000 \\ \hline 1001 \end{array}$$

$S=1,\ C=0,\ V=0,\ Z=0$

تعیین V وقتی که بصورت مستقیم A را منهای B می‌کنیم:

A	−	B	=	حاصل	
+		−		−	$\Rightarrow V=1$
−		+		+	$\Rightarrow V=1$

نکته) در تفریق دو عدد سیستم مکمل ۲ وقتی A, B هم علامت باشند سربریز نداریم

بررسی وضعیت فلگ ها در سیستم مکمل ۲ پس از انجام عملیات A-B

$A \geq B \longleftrightarrow \begin{matrix} S=0 \\ V=0 \end{matrix}$ OR $\begin{matrix} S=1 \\ V=1 \end{matrix} \longleftrightarrow S=V \longleftrightarrow S \oplus V=0 \longleftrightarrow S \odot V=1$

$S\bar{V}+\bar{S}V=0 \quad SV+\bar{S}\bar{V}=1$

$A<B \longleftrightarrow \begin{matrix} S=1 \\ V=0 \end{matrix}$ OR $\begin{matrix} S=0 \\ V=1 \end{matrix} \longleftrightarrow S \neq V \longleftrightarrow S \oplus V=1 \longleftrightarrow S \odot V=0$

$Z=0$

$A<B \longleftrightarrow (S \oplus V) \cdot \bar{Z}=1$

$A=B \longleftrightarrow Z=1, V=S=0$

$A \neq B \longleftrightarrow Z=0$

مثال) تفریق زیر را در ۴ بیت انجام دهید:

$A \geq B \Leftarrow -6 \leftarrow -1010 \Rightarrow +0101$

$V=0111$

0111

1111

1011

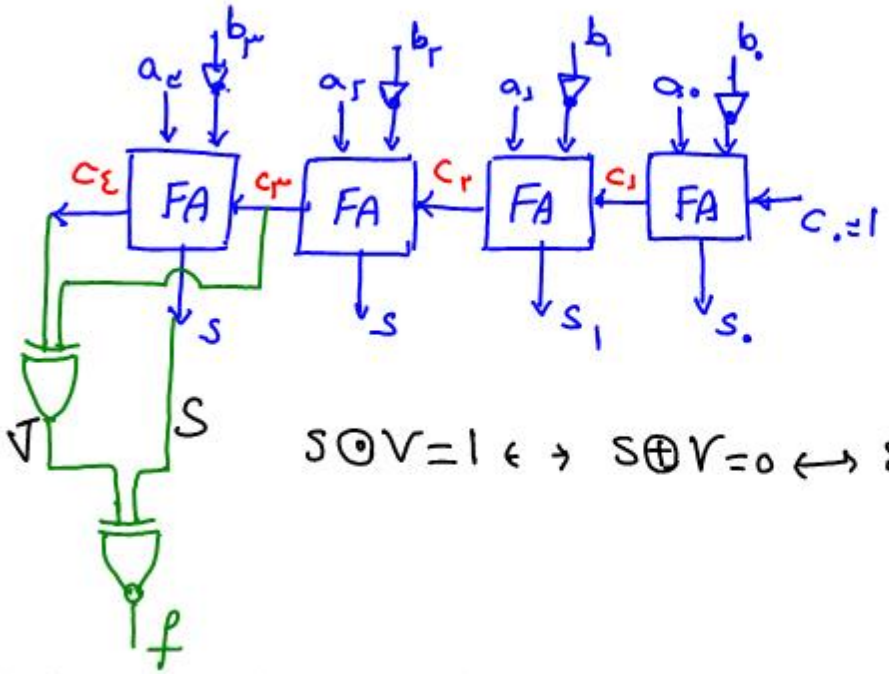
$S=1$

$C=0$

$Z=0$

$V=1$

مثال) در شکل زیر خروجی F یک شده است، کدامیک از گزینه‌های زیر درست است؟



$S \odot V=1 \longleftrightarrow S \oplus V=0 \longleftrightarrow S=V \longleftrightarrow A \geq B$

۱- $A<B$

۲- $A \neq B$

۳- $A>B$

۴- $A \geq B$ ✓

معماری کامپیوتر

رامین رضوی

سیستم مکمل ۱

نکته) در این سیستم مانند سیستم مکمل ۲ ما خاصیت sign extend را داریم

جمع سیستم مکمل ۱ : در جمع سیستم مکمل ۱ برای محاسبه حاصل نهایی همیشه کری را با حاصل، جمع می‌کنند

مثال) با ۴ بیت در سیستم مکمل ۱ جمع‌های زیر را انجام دهید

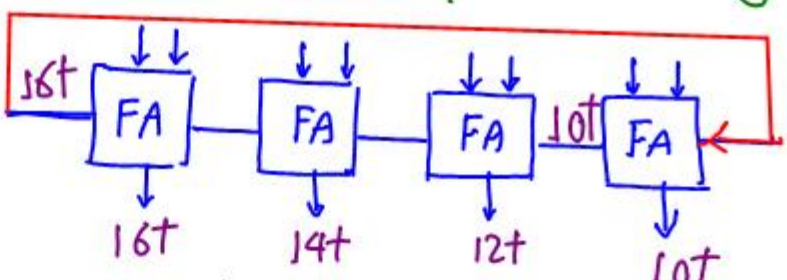
الف) $1001 + 0011$ (−۶ + ۳)

$$\begin{array}{r} 1001 \\ +\ 0011 \\ \hline 1100 = -3 \end{array} \quad C=0+0$$

ب) $1101 + 0110 = +4$ (−۲ + ۶)

$$\begin{array}{r} 1101 \\ +\ 0110 \\ \hline 0011 = +3 \\ +\ \ 1 \\ \hline 0100 = +4 \end{array} \quad C=1$$

سخت افزار جمع کننده n بیتی سیستم مکمل ۱ :



تفریق سیستم مکمل ۱ :

① تفریق غیر مستقیم = مکمل یک $A - B = A + \overline{B}$

- اگر تفریق را با تبدیل به جمع رفتیم، قوانین جمع مکمل ۱ نیز حادث است یعنی کری با حاصل جمع می‌شود - اما اگر تفریق را بصورت مستقیم رفتیم کری هرچه که بود، آن را از حاصل کم می‌کنیم

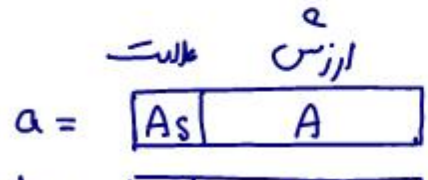
$1001 - 1101$ (−۶ − (−۲))

$$\begin{array}{r} 1001 \\ +\ 0010 \\ \hline 1011 \end{array} \quad C=0+0$$

② $1001 - 1101$

$$\begin{array}{r} 1001 \\ -\ 1101 \\ \hline 1100 = -3 \\ -\ \ 1 \\ \hline 1011 = -4 \end{array} \quad C=1$$

سیستم علامت مقدار :



$a \leftarrow a \pm b$ (op: operation)

الگوریتم جمع و تفریق سیستم مقدار علامت :

$$\begin{array}{r} a \\ b \\ + \\ \hline \end{array} \qquad \begin{array}{r} A \\ +B \end{array}$$

جمع بین علامت

حالت ۱) $A_s \oplus B_s = 0 \leftrightarrow$ b, a هم علامت اند $\Rightarrow A \leftarrow A+B$ ⇒ حاصل غلط: $c=1 \Rightarrow V=1$ ⇒ حاصل درست: $c=0$

$A_s \leftarrow A_s$ یا B_s قدیم (علامت حاصل)

a + b
+ + +
− + −

op = +

حالت ۲) $A_s \oplus B_s = 1 \leftrightarrow$ b, a مختلف علامت هستند ⇒ مانند بالا

a − b
+ − −
− − +

op = −

حالت ۳) $A_s \oplus B_s = 0 \leftrightarrow$ b, a هم علامت اند ⇒ $A \leftarrow A-B$ ⇒ $A_s \leftarrow$ علامت حاصل

با تبدیل به جمع به صورت مستقیم

a − b
+ − +
− − −

op = −

$a < V$ ، $5-7=-2$ ، $7-5=+2$

$A \leftarrow A-B \Rightarrow A \leftarrow A+\bar{B}+1$ (مکمل یک + ۱)

$A_s \leftarrow$ علامت حاصل

$c=1 \leftrightarrow A \geq B$ ⇒ حاصل درست

$c=0 \leftrightarrow A < B$ ⇒ حاصل غلط است و خواهیم داشت: مکمل دو A فعلی $\leftarrow A$ $(A \leftarrow \bar{A}+1)$

$n \geq t$ علامت حاصل فعلی $\leftarrow$ علامت حاصل

حالت ۴) $A_s \oplus B_s = 1 \leftrightarrow$ b, a مختلف العلامه ⇒ مانند حالت ۳

a + b
+ + −
− + +

op = +

مثال ۱) ۲ عدد زیر را در سیستم علامت مقدار با هم جمع کنید

$a = 11001 = -9$ (A)

$b = 01110 = +14$ (B)

$- + + \Rightarrow A \leftarrow A - B$

A_S جدید ← A_S قدیم

تفریق بصورت متمم ۲

$$\begin{array}{r} 1001 \\ -1110 \\ \hline c=1 \quad 1011 \end{array}$$

$\Rightarrow c=1 \Rightarrow B=1 \Rightarrow A<B$

$$\begin{array}{r} 1001 \\ +0010 \\ \hline 1011 \end{array}$$

$c=0 \Rightarrow A<B$

A_S	A
1	1011

$= -11$

مکمل ۲

0	0101

$\Rightarrow +5$

مثال: -3 , $+9$

$9 - 14 = -5$

$14 - 9 = +5$

مثال ۱) ۲ عدد زیر را در سیستم علامت مقدار از هم کم کنید

$1011_0 = -6$ (A)

$-0\,1100 = +12$ (B)

$- - + \quad - + -$

$A \leftarrow A + B$

علامت حاصل ← A_S

$\Rightarrow$

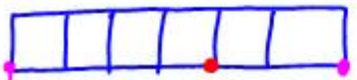
$$\begin{array}{r} 1011_0 \\ 1100 \\ \hline 10010 \end{array}$$

$\Rightarrow c=1 \Rightarrow$ overflow

- برای نمایش اعداد اعشاری ۲ روش وجود دارد ۱- fixed point ۲- floating point

fixed point = قبل از ساخت مدار و هنگام طراحی جای ممیز مشخص می‌شود و بعد از ساخت تغییر محل ممیز نیاز به تغییرات سخت افزاری دارد

فرض کنید یک سیستم اعداد ۶ بیتی ممیز ثابت بصورت زیر داریم



اعداد اعشاری fractional

اعداد صحیح integer

fixed point ای که point آن در سمت الیه سمت راست قرار گیرد fixed point از نوع integer است و ...

نکته) سیستم اعداد صحیح و همچنین سیستم اعداد fractional حالت خاصی از اعداد fixed point هستند.

برگردیم به مثال: با یک سیستم اعداد ۶ بیتی ما ۶۴ عدد را می‌توانیم روی محور اعداد داشته باشیم

$1111,11 = 15,75 = 15\frac{3}{4}$

$0000,10 - 0000,01 = (0000,01)_2 = 1/4 \Rightarrow$

* فاصله بین هر ۲ عدد متوالی $1/4$ است

مثال) مقدار اعداد زیر را در مبنا ۲ بدست آورید

۱) $101,0101 = 5\frac{5}{16}$

۲) $011,1011 = 3\frac{11}{16}$

۳) $110,110110 = 6\frac{91}{128}$

$91 = 64+16+8+3$

معماری کامپیوتر

رامین رضوی

S = دقت : فاصله بین دو عدد متوالی ← در fixed point ثابت است ولی در floating point تغییر میکند

R = رنج یا برد : فاصله بین کوچکترین عدد تا بزرگترین عدد : در مثال قبل رنج مساوی $\frac{3}{4}$ تا $1\frac{1}{4}$ است

overflow = ما یک عدد را در صورتی می توانیم نمایش بدهیم که در رنج قابل نمایشمان باشد و اگر خارج از این رنج باشد می گوییم سرریز رخ داده است.

توجه) هر عددی که در رنج ما باشد، سرریز نیست، شاید ما نتوانیم آن عدد را نشان بدهیم ولی این به معنا سرریز نیست، اگر خود آن عدد را نداشته باشیم در اینصورت با تزدیکترین عدد قابل نمایش به آن نشان اش می دهیم، در اینصورت موقع نمایش اش دچار خطا می شویم

$\hat{n}$ = اعدادی که روی محور داریم

n = عدد مد نظر ما که روی محور نیست

$error = Er = |n - \hat{n}| \leq S/2$

$Er \leq S/2$

نکته) دقت ↑ = کاهش فاصله بین ۲ نقطه متوالی = $S \downarrow$ ← باعث کاهش حد بالا خطا

= ↓ رنج

نکته) به سیستم اعداد fractional سیستم اعداد جزئی نیز می گویند

نکته) کپی کامپیوترها هستند که خیلی خوب انجام می دهند اینها معمولا سیستم اعداد شون fractional است، مزیت سیستم نمایش fractional این است که در ضرب سرریز نمی دهد، اما در جمع ممکن است بدهد : $0/9 + 0/8 = 1/7$

floating point

دو نگاه وجود دارد:

نگاه ۱: یک عدد ممیز شناور از دو ثابت تشکیل شده که یکی عدد بدون ممیز را نمایش می‌دهد و دیگری جای ممیز را نمایش میدهد

مثال) فرض کنید سیستم نمایش اعداد تان ۱۰ بیتی است و ۷ بیت اش جای عدد و ۳ بیت اش برای مشخص کردن جا ممیز باشد

۱	۰	۰

۱	۰	۱	۱	۰	۰	۱

۰ ۱ ۲ ۳ ۴ ۵ ۶ ۷

۱۰۱۱/۰۰۱

نگاه ۲: هر عدد ممیز شناور از ۲ بخش fixed point به m و e درست شده که این این دو بخش باهم یک عدد را نشان میدهد. که e شما یک fixed point از نوع integer است و m شما یک fixed point از جنس fractional است.

S e F = M

		۱	۰	۰		۱	۰	۱	۱	۰	۰	۱

→ ۰/۱۰۱۱۰۰۱

نکته: $M \in [0,1)$

مقدار عدد = $M \times B^{e}$ → exponent (نما)، B: Base

مقدار عدد = $0/1011001 \times 2^{4} = 1011/001$

عدد max = $0/1111111 \times 2^{7} = 1111111 = 127$

عدد min = $0/0000001 \times 2^{0} = 2^{-7} = \frac{1}{128}$

* در این ۲ مثال من e را بدون علامت فرض کردم.

" max = $0/11 \text{——} 1 \times 2^{3} = 111/1111 = 7\frac{15}{16}$

" min = $0/00 \text{——} 1 \times 2^{-4} = 2^{-11}$

* اگر e علامت دار باشد: پیش فرض مکمل ۲: $-4 \leq e \leq 3$

- با ۱۰ بیت می‌توان ۱۰۲۴ عدد نمایش داد، اما این Floating point ها الان به کما تقسیم نمی‌توانند ۱۰۲۴ عدد نمایش بدهند که علت آن وجود نمایش‌های متعدد برای هر عدد است

مثال) می‌خواهیم عدد ۱۱٫۰۰۱ را نشان دهیم، همه نمایش‌های این عدد را نشان دهید

	e	M	
*	۰۱۰	۱۱۰۰۱۰۰	$= ۰٫۱۱۰۰۱ \times ۲^{۲} = ۱۱٫۰۰۱$
	۰۱۱	۰۱۱۰۰۱۰	$= ۰٫۰۱۱۰۰۱ \times ۲^{۳} =$
	۱۰۰	۰۰۱۱۰۰۱	$= ۰٫۰۰۱۱۰۰۱ \times ۲^{۴} =$

مثال) می‌خواهیم عدد ۱۱۱ را نشان دهیم، همه نمایش‌های این عدد را نشان دهید

	e	M	
*	۰۱۱	۱۱۱۰۰۰۰	$= ۰٫۱۱۱ \times ۲^{۳} = ۱۱۱$
	۱۰۰	۰۱۱۱۰۰۰	
	۱۰۱	۰۰۱۱۱۰۰	
	۱۱۰	۰۰۰۱۱۱۰	
	۱۱۱	۰۰۰۰۱۱۱	

۵ نمایش برای عدد ۷ وجود دارد

مثال) عدد ۰ چند نمایش دارد در اینجا؟

۸ نمایش دارد، همه بیت‌های مانتیس را ۰ می‌کنیم و سپس با e بازی می‌کنیم

* برای حل این مشکل از روش Normalized floating point استفاده کردند

اتلاف نمایش‌ها

- برای رفع مشکل وجود داشتن حالت‌های متعدد قرارداد کردند که از همه نمایش‌های مختلف یک عدد فقط یکی‌اش را قبول کنند و آن کسی است که با ارزش‌ترین رقم مانتیس اش ۱ باشد.

معماری کامپیوتر

رامین رضوی

$(0/421)_4$ → 01 10 11

$(0/\frac{1}{2})_4$

$0/1_2$

* رنج ماشین نرمال به این شکل در مبنا ۲ چقدر است؟

$0/1000 = 0/1_M$

$M \in [0/5, 1)$

مثال) با فرض اینکه سیستم نمایش اعداد ما ۱۶ بیتی باشد و ما ۵ بیتی سیستم مکمل ۲ باشد و ماشین نرمال باشد عدد زیر در این سیستم چگونه ذخیره می‌شود؟ (این را دو تکمیلی بگیرید)

$-2/3 = -10/0\overline{1001} = -0/100\overline{1001} \times 2^{+2}$

$0/3 \times 2 = 0/6$

$0/6 \times 2 = 1/2$

$0/2 \times 2 = 0/4$

$0/4 \times 2 = 0/8$

$0/8 \times 2 = 1/6$

s	e	M
1	00010	1001001100

$-0/1001001100 \times 2^{2}$

* نکته) یه را نمی‌توان با این قرار داد نشان داد

* در مثال بالا ۰ را با عدد روبه رو نمایش می دهیم:

$min = 0/1 \times 2^{-16} = 2^{-17}$

* نرمال IEEE اصلی دوگام دارد نه تک گام

۱- از بین همه نمایش ها ارزهایی را قبول داریم که سمت چپ ترین رقم آن غیر ۰ باشد

۲- در floating point نرمال بر ارزش ترین بیت ماشین هیچ وقت نمایش داده نمی‌شود و مقدارش ۱ است.

$-0/100\overline{1001} \times 2^{+2} = -1/00\overline{1001} \times 2^{+1}$

s	e	M
1	00001	0010011001

$\Rightarrow -1/0010011001 \times 2^{+1}$

نتیجه) در IEEE استاندارد داریم. | S | E | M | $= (-1)^S \times 1/M \times B^E$

سوال) در این سیستم تر دقیق ترین عدد به ۰ کیست؟ | ۰ | ۱۰۰۰۰ | ۰۰...۰ | $\Rightarrow 1/0...0 \times 2^{16}$

$= 2^{-16}$

$M=10$ و $e=5$ و $S=1$ و تمام شکل ۲ و $B=2$

$-16 \leq e \leq +18$

مشکل: ۰ را نداریم

۱- ۰ را با نزدیکترین عددی که داریم نمایش بدهیم

۲- درسته که ۰ را نداریم و درسته که ۰ را می توان مثل هر عدد دیگری با تردیکترین عدد به آن نشان داد ولی بهتر است در سیستم نمایش اعداد ما اعداد خیلی پرکاربرد را داشته باشیم ← برای این منظور IEEE قرار گرد که اگر تمام بیت های عدد ما ۰ بود این نمایش برای ۰ باشد

در این صورت ما ۱ را نخواهیم داشت | ۰ | ۰۰۰۰۰ | ۰۰...۰ | $= 1/00...0 \times 2^0 = 1$

و این خوب نیست چون ۱ نیز جزء اعداد پرکاربرد است.

حال برای رفع این مشکل میان دار سیستم اعداد ممیز شناور نرمال شده بایاس شده استفاده می کنند.

سیستم بایاس: سیستم اعداد علامت دار بایاس شده سیستمی است که اعداد علامت دار را به شکل بدون علامت ذخیره می کند

$bias = 2^{\text{تعداد بیت}} - 1$

$bias = 2^5 - 1 = 31$

n یک عدد n بیتی بدون علامت است: $0 \leq n \leq 31$

y " " " " علامت دار " : $-16 \leq y \leq 15 \xrightarrow{+16} 0 \leq y+16 \leq 31$

معماری کامپیوتر

رامین رضوی

مثال) فرض کنید ۳ بیت دارید و عدد ۳- را در سیستم بایاس نشان دهید

$Bias = 2^{3-1} = 4 \Rightarrow -3+4 = +1$

0	0	1	$= 1-4 = -3$

اعداد ۳ بیتی سیستم بایاس | اعداد ۳ بیتی مکمل ۲

011 = +3	⟶	111 = 7-4 = +3
010 = +2	⟶	110
001 = +1		101
000 = 0		100
111 = -1		011
110 = -2		010
101 = -3		001
100 = -4		000 = 0-4 = -4

* در سیستم بایاس هرکسی که هیکل اش گنده تره واقعا گنده تره
* سیستم بایاس شکل اعداد را مرتب می کنه

مثال) عدد ۱۲- را در سیستم بایاس نمایش دهید

* حداقل تعداد بیت برای نمایش ۱۲- در سیستم مکمل ۲، ۵ بیت است = 10100 = -12

جواب:
$-12+16 = +4 \Rightarrow 00100$
$10100 = 00100$

نکته) در سیستم بایاس نیز ما بیت علامت داریم، و بیت علامت اش به اینصورت است که اگر MSB صفر باشد عدد منفی است و اگر ۱ باشد عدد مثبت است

سوال) وقتی که e در سیستم بایاس است و نمایش تمام ۰ که آن را ۰ گرفتیم و اصل چه کسی بوده است. Base=2 و s=1 bit و e=5 bit و m=10 bit

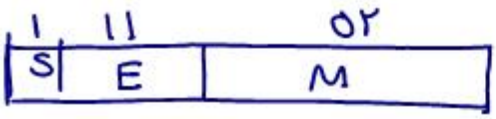
s	e	m
0	00000	00 ⋯ 0

$= 1.00\cdots0 \times 2^{-16} = 2^{-16}$

نکته: IEEE استاندارد Base=۱۰ است.

استاندارد IEEE 754 : این استاندارد ۲ نوع فرمت تعریف کرده

۱) دقت معمولی (single precision) = ۳۲ بیتی



۲) " مضاعف (Double precision) = ۶۴ بیتی

$$(-1)^S \times 1,M \times B^{E-bias}$$

بایاس در اینجا $2^{e-1}-1$

دقت معمولی = $2^7-1=127$

مضاعف = $2^{10}-1=1023$

مثال) عدد $-2\frac{1}{6}$ را با دقت معمولی IEEE 754 نمایش دهید

$-10,00\overline{01} = -1,00\overline{01} \times 2^{+1}$ ⇒ (+127)

S	E	M
1	10000000	00101010101010...

$\frac{1}{6} \times 2 = \frac{2}{6} = \frac{1}{3}$

$\frac{1}{3} \times 2 = \frac{2}{3}$

$\frac{2}{3} \times 2 = 1\frac{1}{3}$

$\frac{1}{3}$

سوال: با فرض اینکه سیستم نمایش اعداد ما در ماشین ۱۶ بیتی باشد و نما ۵ بیتی بایاس شده باشد و ماشین نرمال به شکل زیر باشد، min و max عدد مثبت را مشخص کنید؟ (Base=۲)

S E M

Min: 0 00000 100—0 $= (0,1)_2 \times 2^{-1} \times 2^{-16} = 2^{-17}$

0 00000 00—01 $= (0,00—01)_2 \times 2^{-1} \times 2^{-16} = 2^{-26}$

گام بعدی دقت ماشین نرمال است:

max: 0 11111 11—1 $= (0,11—1)_2 \times 2^{15} = (1-2^{-10}) \times 2^{15}$

قبل max: 0 11111 11—10

0 11111 00—001 $= (0,00—01)_2 \times 2^{15} = 2^{-10} \times 2^{15} = 2^{+5}$

$10-1=9$

* برای یافتن کمترین دقت ماشین مکزیمم عدد و قبلی آن را پیدا کنید و آنها را از هم کم کنید

برای یافتن بیشترین دقت باید min عدد و عدد بعدی آن را از هم کم کنید

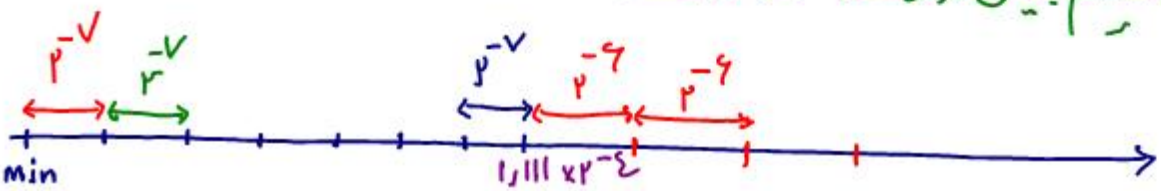
min عدد بعدی: 0 00000 100—01

- عدد min: 0 00000 100—0

0 00000 0—001 $= (0,000—01)_2 \times 2^{-16} = 2^{-26}$

مثال) فرض کنید ماشینی ۳ بیت و نما نیز ۳ بیت باشد و base=۲ ؟ (چون جایی دیگر گفته نشده ماشین را از نوع IEEE در نظر می‌گیریم)

رنج ۳ بیت نما در سیستم بایاس از ۴- تا ۳+ است.



عدد min = $1,000 \times 2^{-4}$) 2^{-7}

min یکی جدا از = $1,001 \times 2^{-4}$) 2^{-7}

min دو تا جدا از = $1,010 \times 2^{-4}$

$1,111 \times 2^{-4} \longrightarrow 0,1111 \times 2^{-3}$

$1,000 \times 2^{-3}$

$1,001 \times 2^{-3}$

$\Rightarrow 0,1111 \times 2^{-3}$

$0,001 \times 2^{-3} = 2^{-6} = 2^{-3} \times 2^{-3}$

نکته) ما در floating point دقت مناسب برای هر کاربردی (هر عددی) را داریم و در واقع تفاوت floating point نسبت به fixed point این است که دقت اعداد کوچک مناسب اعداد کوچک است و دقت اعداد بزرگ مناسب اعداد بزرگ است

نکته) اگر ما اعداد خیلی کوچک و در همان حال اعداد خیلی بزرگ را نیاز داریم (اگر بازه ای که نیاز داریم وسیع است) از floating point استفاده می‌کنیم در غیر اینصورت از fixed point استفاده می‌کنیم

معماری کامپیوتر

رامین رضوی

جمع و تفریق مسیر شناور

$0/M_1 \times 2^{E_1} \pm 0/M_2 \times 2^{E_2}$

ابتدا نماها را با هم مقایسه می‌کنیم، سپس نما کوچکتر را بزرگتر می‌کنیم که در نتیجه این کار مانتیس اش به سمت راست شیفت پیدا می‌کند، با این کار بستگی به این که مانتیس چه بالایی ممکن است کمی دقت را از دست بدهیم. سپس یکی از نماها را می‌نویسیم و بعد مانتیس ها را با هم جمع یا از هم تفریق می‌کنیم، حال اگر مانتیس حاصل نرمال نبود نرمال اش می‌کنیم

مثال) جمع زیر را انجام دهید

S	E	M
0	1001	11011 $= 0/11011 \times 2^{1001}$
+ 0	1010	11100 $= 0/11100 \times 2^{1010} \rightarrow 0/11000 \times 2^{1001}$

↳ با ارزش ترین رقم مانتیس از بین رفت

وقتی نما کوچکتر را بزرگ می‌کنیم $\Rightarrow 0/01101 \times 2^{1010}$ نما کوچکتر را بزرگ کنیم

شروع می‌کنیم و از بیت هایی که رقم ارزش مانتیس دور می‌ریزیم

$$\begin{array}{r} 0/01101 \times 2^{1010} \\ + \quad 0/11100 \times 2^{1010} \\ \hline 1/01001 \times 2^{1010} \end{array} \longrightarrow 0/10100 \times 2^{1011} = 0/10100 \times 2^{+3}$$

↳ مانو به این می‌گوییم سرریز در مانتیس

مثال) تفریق زیر را انجام دهید

مانو به این می‌گوییم زیر ریز در مانتیس

$$\begin{array}{r} 0/11101 \times 2^{0111} \\ - \quad 0/11010 \times 2^{0111} \\ \hline 0/00011 \times 2^{0111} \end{array} \longrightarrow 0/11000 \times 2^{0100} = (0/11)_2 \times 2^{-4}$$

معماری کامپیوتر

رامین رضوی

ضرب مبنیز شناور

$$0/M_1 \times 2^{E_1} \times 0/M_2 \times 2^{E_2} = 0/M_1 \times 0/M_2 \times 2^{E_1+E_2-bias}$$

$$0/1000 \times 2^{1001} \times 0/1111 \times 2^{0110} = 0/01111000 \times 2^{1001+0110=1111}$$

$$= 0/01111000 \times 2^{0111} = 0/1111000 \times 2^{0110} = 0/1111 \times 2^{0110}$$

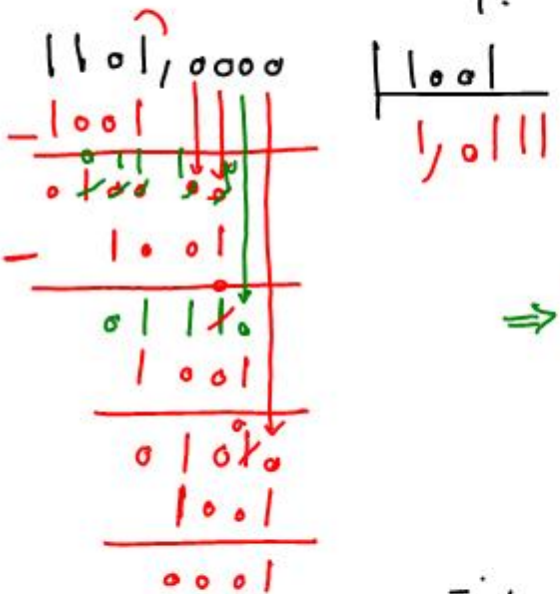
تقسیم مبنیز شناور

$$0/M_1 \times 2^{E_1} \div 0/M_2 \times 2^{E_2} = \frac{0/M_1}{0/M_2} \times 2^{E_1-E_2+bias}$$

$$0/1101 \times 2^{1100} \div 0/1001 \times 2^{1010} \qquad +2 \text{ (bias)}$$

$$\frac{0/1101}{0/1001} \times 2^{0010+8(1000)} = \frac{0/1101}{0/1001} \times 2^{1010} = \frac{1101}{1001} \times 2^{1010}$$

نکته) در کامپیوتر همیشه تقسیم $2n$ بیت به n بیت انجام می‌شود



$$\Rightarrow 1/0111 \times 2^{1010} \Rightarrow 0/10111 \times 2^{1011} = \frac{11}{16} \times 2^3$$

نکات

۱- در جمع و تفریق مبنیز شناور امکان سرریز و زیر ریز در ماشین وجود دارد.

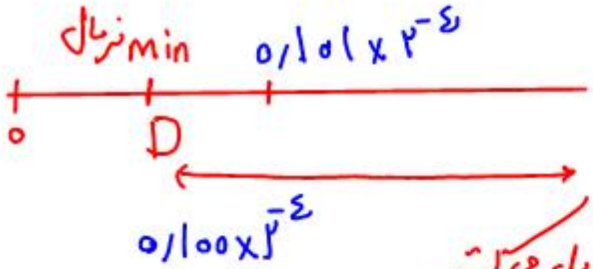
۲- در ضرب مبنیز شناور ⟵ زیر ریز در ماشین وجود دارد ⟵ سرریز رخ نمیدهد

۳- در تقسیم ⟵ سرریز ⟵ زیر ریز

۴- در همه عملیات مبنیز شناور امکان وقوع سرریز و زیر ریز در نرم‌افزار وجود دارد

- سرریز در توان یعنی توان حاصل از max توان قابل ذخیره در سیستم بیشتر باشد، این سرریز واقعی است و در اینجا V=1 می‌شود ← سرریز در توان قابل درست کردن نیست.

- زیر ریز در توان یعنی توان حاصل از min توان قابل ذخیره در سیستم کمتر باشد



۱- برخی این عدد را 0 می‌گیرند

۲- برخی می‌آیند و آن توان را به بیاس خارج نشون

مانتیس از حالت نرمال به min توان قابل نمایش تبدیل می‌کنند

(فرض کن رنج نما $3 \geq$ نما ≥ -4 باشد)

مثال: $0/11001 \times 2^{-6}$

$\downarrow$

$0/0011 \times 2^{-4}$

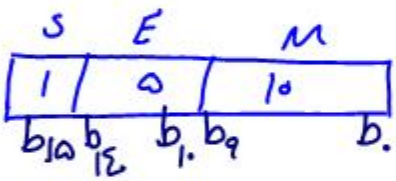
سوال) در عمل تقسیم میزبنشاور کدام یک از خطا ها امکان رخ دادن دارد؟

۱- فقط خطا overflow

۲- " " underflow

۳- خطای overflow و underflow

۴- فقط خطا underflow اگر اعداد علامت مخالف هم داشته باشند



بیان اعداد میزبنشاور بصورت ریاضی

$$\text{مقدار عدد} = (-1)^{b_{15}} \times \left(\sum_{i=0}^{9} b_i \times 2^{i-10} \right) \times 2^{E-16}$$

$$E = \sum_{i=10}^{14} b_i \times 2^{i-10}$$

مهندسی کامپیوتر ۸۲ : در یک کامپیوتر با قابلیت پردازش اعداد صحیح ۵ بیتی، عدد ۳۲ بیتی

مقدار عدد $= 2^{S}\left(\frac{1}{2}-b_{21}\right)\left(1+\sum_{i=0}^{24} 2^{i-24} b_i\right)$: b_0 ـــ b_{21} مشخص کننده مقدار زیر است

و S برابر است با $S=-64+\sum_{i=24}^{30} 2^{i-24} b_i$

حال مقدار بزرگترین و کوچکترین عدد مثبت قابل نمایش در این سیستم را بدست آورید ؟

min+ ——— max+

2^{-65} و $(1-2^{-25})2^{63}$

$S=-64+2^0\times b_{24}+2^1\times b_{25}+\cdots+2^6\times b_{30}$

$$S\begin{cases} max \Rightarrow b_i=1 \Rightarrow -64+2^0+\cdots+2^6=-64+127=+63 \\ min \Rightarrow b_i=0 \Rightarrow -64 \end{cases}$$

$M=2^{-24}\times b_0+2^{-23}\times b_1+\cdots+2^{-1}\times b_{23}$

$$M\begin{cases} max \Rightarrow b_i=1 \Rightarrow 2^{-24}+2^{-23}+\cdots+2^{-1}=1-2^{-24} \\ min \Rightarrow b_i=0 \Rightarrow 0 \Rightarrow M_{min}=0 \end{cases}$$

مقدار عددی max $= 2^{+63}\times\frac{1}{2}\times(1+1-2^{-24})=2^{+63}(1-2^{-25})$

min " " $= 2^{-64}\times\frac{1}{2}\times(1+0)=2^{-65}$

ضرب دو عدد در هم: الگوریتم های ضرب به دو دسته ترکیبی و ترتیبی تقسیم می‌شوند

توجه) در بعضی مراجع به ضرب ترکیبی، ضرب آرایه‌ای نیز می‌گویند

ضرب ترکیبی

$a = 1101 = ۱۳$ می‌دانیم $۱۳ \times ۱۱ = ۱۴۳$

$b = 1011 = ۱۱$

$$(۷۵۲)_{10} \times (۲۳۵)_{10} = ۵\times۱۰^{۰} + ۳\times۱۰^{۱} + ۲\times۱۰^{۲} \Rightarrow ۷۵۲\times۵\times۱۰^{۰} + ۷۵۲\times۳\times۱۰^{۱} + ۷۵۲\times۲\times۱۰^{۲}$$

۳۷۶۰
.......۰
.....۰۰

ضرب‌شونده = مضروب: 1101 → multiplicand

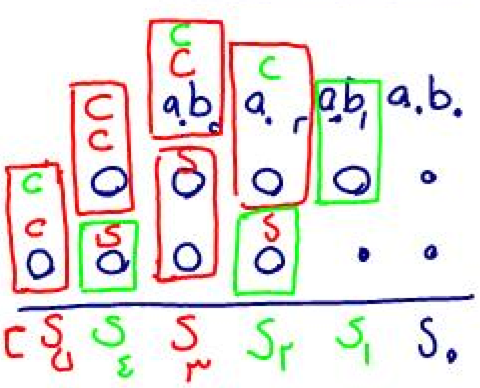
ضرب‌کننده = مضروب‌فیه: multiplier

$$1101 \times 1011 = 2^0 + 2^1 + 2^3 \Rightarrow 1101\times2^0 + 1101\times2^1 + 1101\times2^3$$

```
      1101
     1101
    0000
 + 1101
 ---------
  10001111 = ۱۴۳
```

حاصلضرب های جزئی = partial product

$b_3b_2b_1b_0 \times a_2a_1a_0$



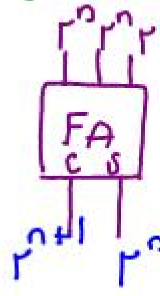
FA = قرمز = ۵

HA = سبز = ۳

AND = ۱۲

ساخت ضرب کننده با استفاده از FA و HA

```
 543210
   1110
  10111
+ 01101
 ------
   1000
```



* در ضرب ترتیبی عددی که تعداد بیت های کمتر است را معمولاً ضرب کننده قرار میدهند

m bit
× n bit
m+n bit ⇒ min فضایی که نیاز داریم تا سرریز رخ ندهد

AND = m × n

Half Adder = min{m, n}

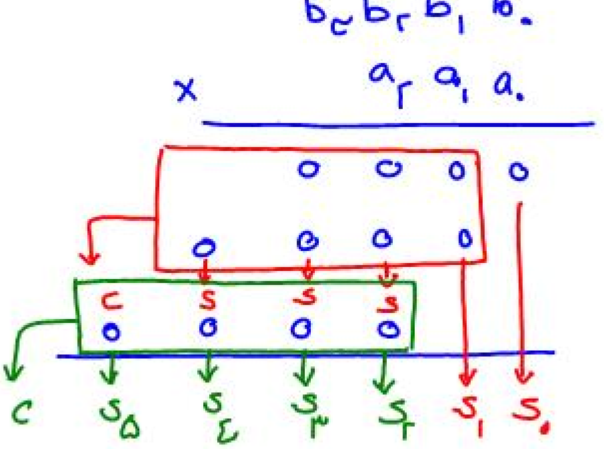
Full Adder = m × n − (m + n)

nbit × n bit

n^2 — الزاماً FA — n^2

n — استفاده کنیم → 0

$n^2 - 2n$ — $n^2 - n$



طراحی با n bit binary adder:

m bit
n bit

AND = m × n

Row = n تا

n−1 جمع کننده m بیتی

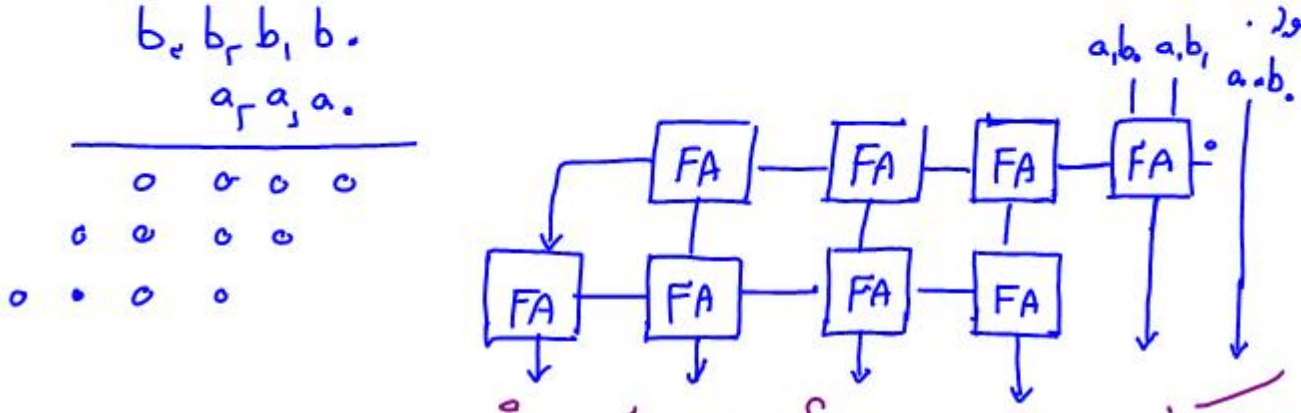
n bit
× m

m−1 جمع کننده n بیتی

بهترین مدل ضرب آرایه ای از لحاظ تاخیر

وقتی است که ما برای جمع partial product ها از CSA (carry save adder) استفاده کنیم. CSA یک روش خاص جمع حاصلضرب های جزئی است، در جمع به روش CSA هر FA کری اش را به جای اینکه به فرد چپی خودش بدهد به فردی که در جنوب غربی اش است میدهد. در این روش در ضرب m × n، n ردیف داریم و در هر ردیف m−1 تا FA داریم بنابراین در کل n × (m−1) تا FA داریم. البته FA سطر اول می توانند HA نیز باشند

قبلا بصورت زیر بود:



- باید هوشمندانه تر عمل کنیم: این کار با جمع به روش CSA انجام می شود.

در این روش FA ها کری شان را به FA جنوب غربی شان میدهند (البته بجز FA های سطر آخر که آنها به اجبار ripple اند) با این کار حاصل فرقی نمی کند ولی این کار باعث می شود FA ها داخل یک سطر بتوانند موازی کار کنند و بهم وابسته نباشند

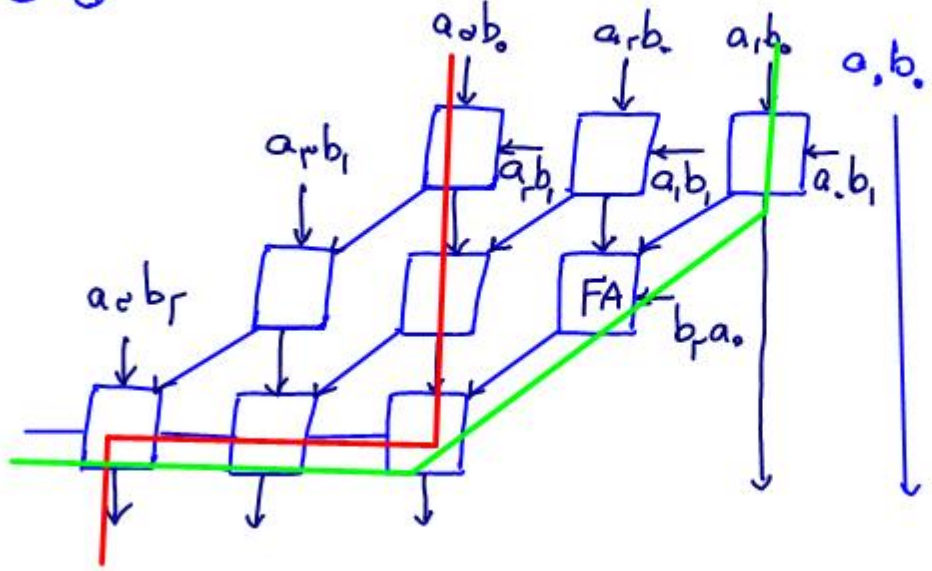
* در بدست آوردن تاصیر مهم است که $m>n$ / $m<n$

$A = a_3a_2a_1a_0$

$B = b_2b_1b_0$

مثال: $m>n$

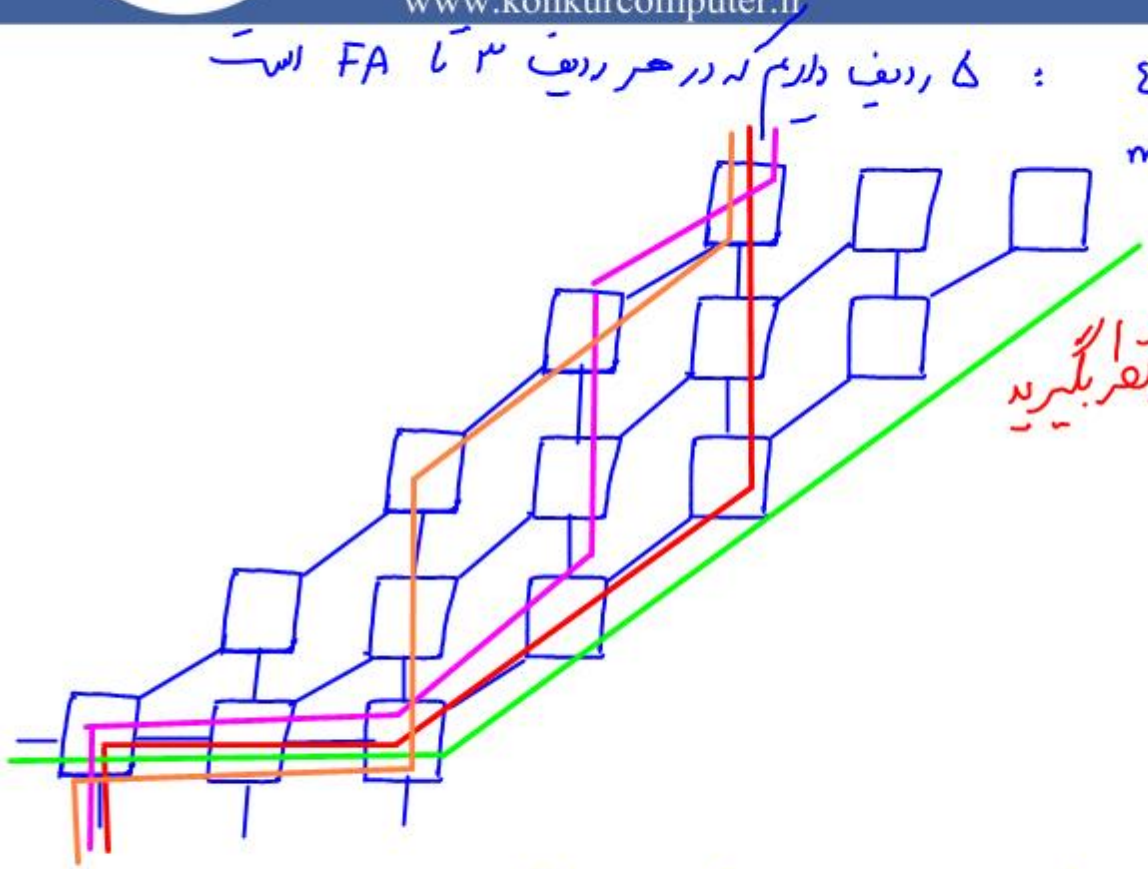
۳ ردیف داریم که در هر ردیف ۴ تا FA داریم



$t_s \geq t_c$
$t_c \geq t_s$

مثال - ضرب ۴×۵ : ۵ ردیف داریم که در هر ردیف ۳ تا FA است

$m \leq n$



$t_s \geq t_c$ ، مسیر بحرانی را در نظر بگیرید

$t_c \geq t_s$

$t_{AND} + n t_{sum} + (m-2) t_c$: $t_s \geq t_c$

$t_{AND} + (n-1+m-1) t_c$: $t_c \geq t_s$

$\Big\} m > n$

$t_{AND} + (m-1) t_s + (n-1) t_c$: $t_s \geq t_c$

$t_{AND} + (n-1+m-1) t_c$: $t_c \geq t_s$

$\Big\} m \leq n$

- وقتی ما قرار است تعداد زیادی جمع انجام دهیم از CSA استفاده می‌کنیم

- CSA در مدل پیاده سازی دارد
  - ترکیبی ← درخت والاس
  - ترتیبی

۲۵۴	۲۵۴
۷۲۶	۷۲۶
۸۷۶	۹۸۰
۲۴۵	۸۷۶
۱۷۸	۳۲۴
	۱۸۱۷

```
S = 0   (sum=0)
for( i=0 to 100)
    S = S + A[i];  ( S+= A[i];)
```

- در جمع قبلی هر ستون نمی توانست مستقلا جمع بزند و منتظر کری جمع کننده سمت راستی اش است

اما در CSA می گیم چون جمع جابجایی دارد، کری حاصل از جمع هر ستون را در همان مرحله از جمع استفاده نکن و کری را نگه دار و در جمع بعدی ازش استفاده کن

دستی، ستی

۲۵۴	۲۵۴
۷۲۶	۷۲۶
۸۵۷	۹۸۰
۵۲۴	۸۵۷
۱۷۸	۱۸۱۷
	۳۲۴

CSA

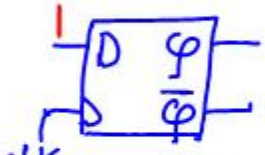
	۰	۰	۱	۰
	۲۵۴			
	۷۲۶			
	۹۷۰			
	۸۳۷			
	۷۱۷			

	۱	۱	۰	۰
+		۷	۱	۷
	۱	۸	۱	۷

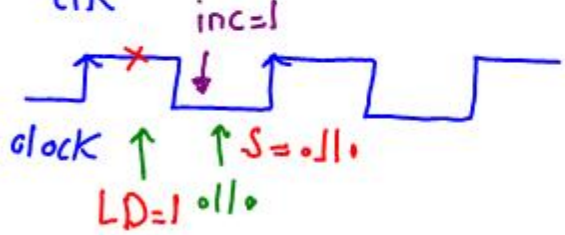
- قبل از اینکه مدار ترتیبی CSA را بکشیم، کمی در مورد مدارات ترتیبی صحبت می کنیم

مدارات
- ترکیبی : مدارات ترکیبی کلاک ندارند و وقتی ورودی می آید بعد از یک تاخیری خروجی
- ترتیبی ایجاد می شود، اما مدارات ترتیبی برای انجام کارشون به کلاک نیاز دارند

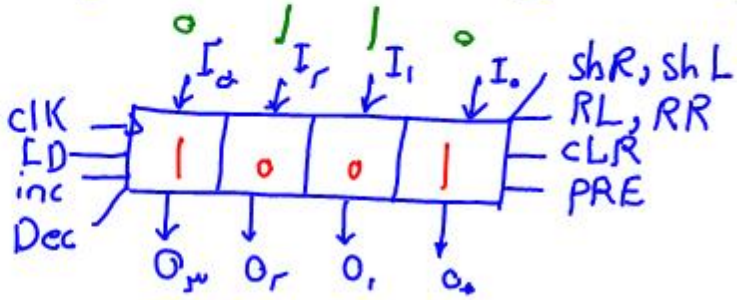
- FF کلاک می خورد و هرچه با FF ساخته می شود کلاک می خورد

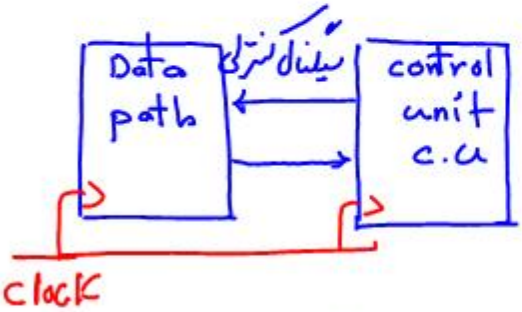


D	Q*
1	1
0	0



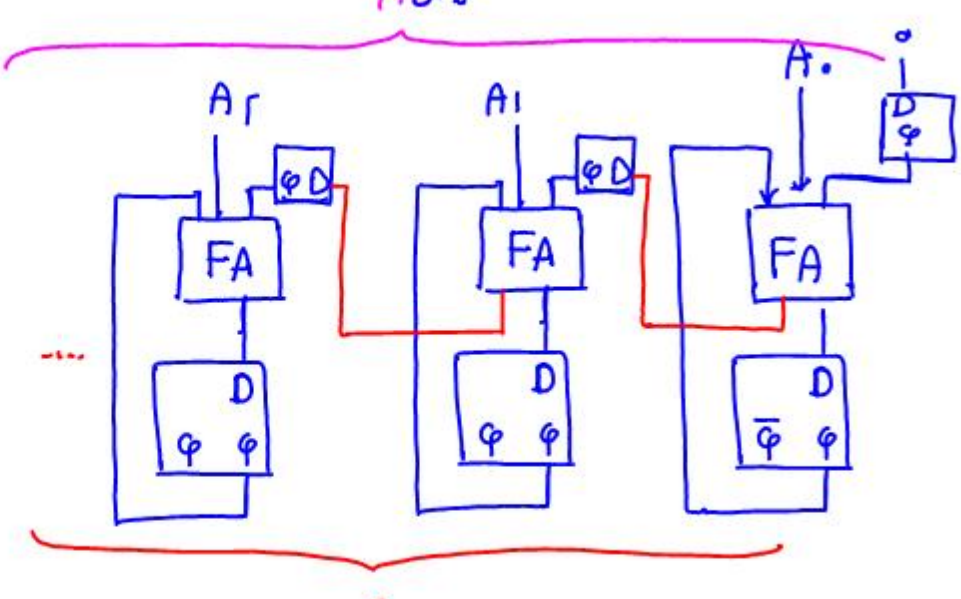
- یک ثبات ۴ بیتی از ۴ تا D-FF تشکیل شده



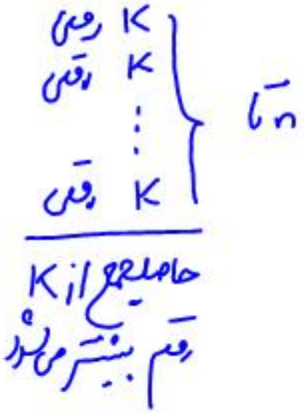


- پایه‌های کنترلی یا با کلاک سنکرون هستند یا با کلاک آسنکرون هستند، وقتی یک پایه کنترلی آسنکرون است هرزمانی که سیگنال اش برسد در همان لحظه تاثیر خودش را می‌گذارد

- PRE و CLR بصورت پیش‌فرض آسنکرون هستند و بقیه پایه‌ها سنکرون هستند



- می‌خواهیم n عدد K رقمی را با هم جمع بزنیم



سوال: اگر جمع کننده تا K بیتی بایشه جمع n عدد چند کلاک طول می‌کشه؟

در هر کلاک یک عدد جدید وارد می‌کنیم، بنابراین n کلاک برای وارد کردن n عدد نیاز داریم، حال بعد از وارد کردن n عدد ممکن است در FF ها مقدار ۱ وجود داشته باشد برای اینکه این مقدارها در جمع شرکت کنند بعد از اینکه n عدد وارد شد شروع می‌کنیم به وارد کردن ۰ و آن قدر ۰ وارد می‌کنیم تا همه FF صفر بشوند، چون K طبقه داریم K-۱ بار باید ۰ وارد کنیم و کلاک بزنیم، پس تعداد کل کلاک ها می‌شود $n+K-2$

معماری کامپیوتر

رامین رضوی

پیاده سازی های ترتیبی
- جمع متوالی
- shift & Add
  - معمولی (شیفت رجیستر معمولی)
  - بشکه ای (Barrel shifter)
  - بوث با sR معمولی
  - بوث با Barrel shifter

جمع متوالی: دو عدد A و B را که می خواهد در هم ضرب کند، ابتدا عدد کوچکتر را مشخص می کند و سپس به تعداد عدد کوچکتر عدد بزرگتر را با خودش جمع می کند و هر بار یک واحد از عدد کوچکتر کم می کند تا به یک برسد

مثلاً اگر دو عدد ۸ بیتی بصورت بدبه رو داشته باشیم

$17 \times 2^{c2} = \underbrace{2^{c2} + 2^{c2} + \dots + 2^{c2}}_{17 \text{ بار}}$

توجه) برای اینکه پیچیدگی های الگوریتم ها بیان را درست کوییم باید بدانیم عمل های کدام در الگوریتم های ضرب چیست و باید آنها را بشماریم، در الگوریتم های ضرب تعداد عمل های شیفت و تعداد عمل جمع و تعداد مکان اعمال کلیدی دخیل ما هستند

مرتبه الگوریتم جمع متوالی

اگر اعداد n بیتی باشند مرتبه الگوریتم جمع متوالی $O(2^n)$ است. $2^n - 1 \geq$ تعداد اعداد n بیتی ≥ 0

حال مقدار یک عدد رندم بطور متوسط برابر $\frac{2^n - 1}{2}$ است، بنابراین بطور متوسط تعداد عمل جمع برابر $\frac{2^n - 1}{2}$ خواهد بود، که مرتبه آن $O(2^n)$ است.

تفاوت تعداد عمل شیفت با تعداد بیت شیفت: در یک ثبات ۸ بیتی عدد زیر را ۳ بیت به چپ شیفت دهید

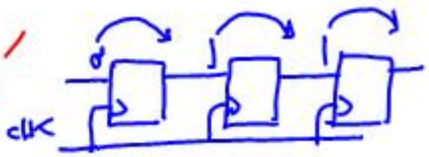
$10110 \rightarrow 10110000$

نکته) اگر ما ۳ بیت شیفت قبل را در یک لحظه انجام بدهیم ۱ عمل شیفت داریم و ۳ بیت را شیفت داده‌ایم ولی اگر تک بیت تک بیت شیفت دهیم تعداد عمل شیفت با تعداد بیت شیفت یکسان می‌شود

$R_1 \leftarrow R_2 \gg R_3 = constant$

↓ شیفت به راست

shift { یک بیت یک بیت شیفت میدهد = شیفت رجیستر معمولی
یکجا بین شیفت میدهد = " " بسته‌ای



ضرب معمولی shift & Add :

```
  1011
x 1001
------
  1011
 0000
0000
1011
------
```

$\bigcirc + \bigcirc + \bigcirc + \bigcirc$

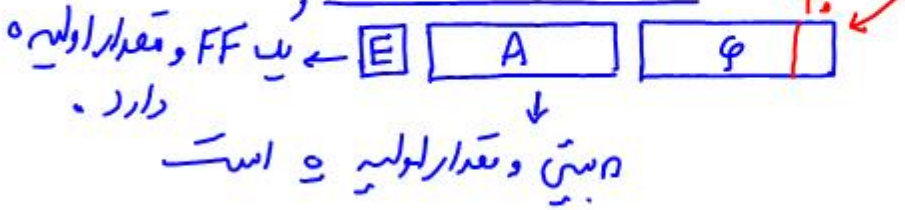
4 partial product ⇒ 3 Add ⇒ بهینه
4 Add ⇒ غیر بهینه

shift & Add

نکته) اگر تعداد بیت‌های ضرب کننده n بیت باشد تعداد عمل جمع در معمولی n-1 تاست و تعداد عمل شیفت به تعداد بیت‌های ضرب کننده یعنی n تاست

$A \cdot \varphi \leftarrow B \times \varphi$

B [n bit]
× φ [n bit]



* در نهایت قسمت کم ارزش حاصل در ضرب کننده قرار می‌گیرد

← یک FF و مقدار اولیه ۰ دارد.

↓ n بیتی و مقدار اولیه ۰ است

- به تعداد بیت های φ (n بار) اعمال زیر را انجام می دهیم:
هر بار بیت راست φ را نگاه می کنیم حال دو حالت پیش می آید

۱- اگر یک باشد: $EA \leftarrow A+B$ و سپس shR EA φ

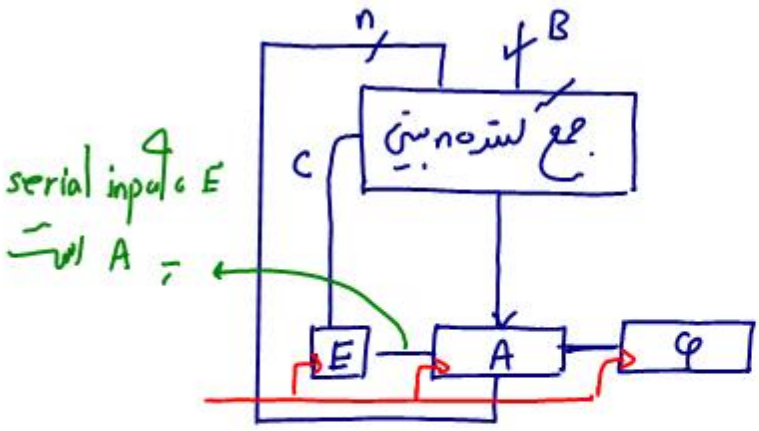
۲- " صفر " : فقط shR EA φ

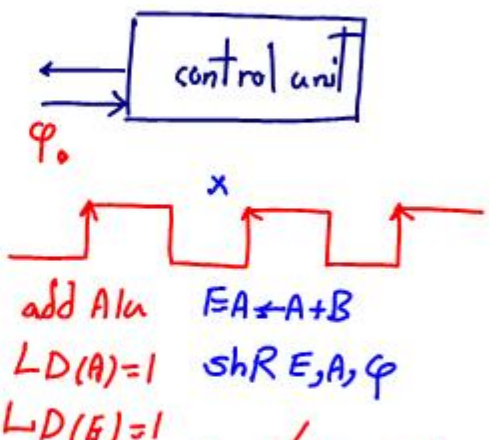
$B = 1101 = 13$
$\varphi = 1011 = 11$

$13 \times 11 = 143$

$A = 0000, E = 0$

	E	A	φ	φ_0
۱) $EA \leftarrow A+B$	0	1101	1011	
shR EA φ	0	0110	1101	
۲) $EA \leftarrow A+B$	1	0011	1101	
shR EA φ	0	1001	1110	
۳) shR EA φ	0	0100	1111	
۴) $EA \leftarrow A+B$	1	0001	1111	
shR EA φ	0	1000	1111	$= 128 + 15 = 143$





* در اینجا تعداد کلاک ها برابر است مجموع تعداد بیت های ضرب کننده (sh داریم) بعلاوه تعداد بیت های ضرب کننده

الگوریتم Shift & Add با شیفت بسکداری :

چون بطور متوسط بین از حاصلضرب های جزئی ۰ هستند ایده این روش آن است که برای افزایش سرعت اصلا ppهای ۰ را تشکیل ندهیم ، در اینصورت هم تعداد عملیات جمع و هم تعداد شیفت ها کم می شود

نکته) در این روش دیگر نمی توانیم از SR معمولی استفاده کنیم زیرا SR معمولی در هر کلاک یک بیت را شیفت میدهد و باید از Barrel shifter استفاده کنیم . در shift & Add بسکداری می توانیم در یک پالس ساعت چند شیفت بدهیم

* توجه) در اینجا اگر بخواهیم از SR معمولی استفاده کنیم حذف ppهای صفر برای ما gain ای نخواهد داشت

نکته) تعداد جمع کننده در این روش به تعداد 1های ضرب کننده کمتر است و تعداد عمل شیفت به تعداد بیت های ضرب کننده خواهد بود . مرتبه این الگوریتم O(n) است ولی سرعت اش از الگوریتم قبلی بیشتر است .

ضرب بوث

ایده اش این است که می گوید ما نه تنها از روی دسته ۰های متوالی بپریم بلکه از روی دسته 1های متوالی ضرب کننده نیز بپریم .

A
xB 1006 1110011

* برای نمایش اعداد علامت دار سیستم دیگری وجود دارد به نام sign digit این سیستم می گوید به جای اینکه برای یک عدد بیت علامت بگذاریم ارقام علامت دار

معماری کامپیوتر

رابین رضوی

ارقام ما علامت دار باشند، مثلاً در مبنا ۱۰ مجموعه ارقام ما بصورت زیر است

۱۹ رقم داریم ⇐ $\{-9, -8, \dots, -1, 0, 1, 2, \dots, +9\}$ = مجموعه ارقام

مثال = $(7\bar{8}\bar{1})_{10} = (-1)\times10^0 + (-8)\times10^1 + 7\times10^2$

- نکته: برای نمایش اعداد مبنا ۲ داشتن ۲ نوع رقم کفایت می‌کند، اما در سیستم SD ما ۲-۱ رقم داریم، بنابراین می‌گویند این سیستم دارای افزونگی است و این افزونگی باعث می‌شود هر عدد نمایش‌های متعددی داشته باشد، اما در اینجا همه نمایش‌های متعدد استفاده می‌شود.

$25 = 3\bar{5} = 1\bar{7}\bar{5} = \dots$

ارقام در مبنا r = $\{-(r-1), -(r-2), \dots, -1, 0, 1, \dots, (r-1)\}$

" " " ۲ = $\{-1, 0, 1\}$

مثال = $11 = 10\bar{1} = 1\bar{1}1 = 1\bar{1}\bar{1}1 = \dots$

- $\underline{1}$ رقم در مبنا دو در SD می‌تواند ۱ یا ۱- یا ۰ باشد بنابراین برای ذخیره ۱ رقم در سیستم SD به ۲ بیت نیاز است.

$A = \_\_\_\_\_\_$

$\times\ B = 1000\ 11100\ 11$

یک عدد باینری را می‌خواهم ببرم تو سیستم SD

$111 = 8 - 1 = 1000 - 0001$

$100\bar{1}$

$1000\ 11100\ 11$ → عدد علامت دار باشد

$\bar{1}001\ 00\bar{1}01\ 0\bar{1}$

$0100 \Rightarrow 4 = 100 =$ بی‌علامت $\quad -4 = 100 =$ علامت دار

$1\bar{1}00 \qquad \bar{1}00$

A . . . .
x B 1000111110011

$\Rightarrow$ A . . . .
x B = $\bar{1}001000\bar{1}01001\bar{1}$

* Booth گفت نیایم و اعداد را ببریم تو سیستم SD و بیایم ضرب کننده را دو بیت دو بیت بپیمایش کنیم

$\bar{1}000\overset{+}{1}111\bar{1}00\overset{+}{1}11\bar{1}$ 0

$\bar{1}001000\bar{1}01001\bar{1}$

اگر صعود دیدم ($0\bar{1}$) یعنی منفی

نزول به (10) به +

اگر تغییر بیتی ندیدم

مثال:

-3

1101

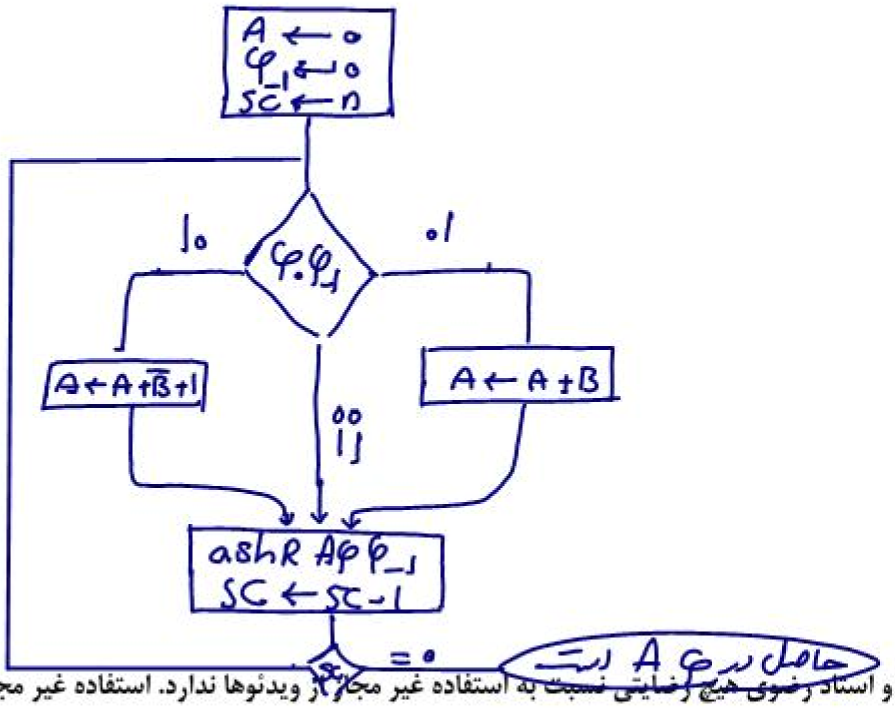
$\bar{1}\overset{+}{1}0\bar{1}$ 0 $\quad -2^5+2^2-2^3 = \bar{1}10\bar{1}$

00000011
111101
+00011
100001111

حاصل در ۸ بیت = ۱۸

* وقتی ۲ عدد n بیتی علامت دار را در هم ضرب می‌کنید در اکثر اوقات حاصل در $2n-1$ بیت جا می‌شود و فقط درحالتیکه دو عدد min منفی باشند ما $2n$ بیت نیاز داریم

B
x Q 0
A: QQ_{-1}



نکته) در روش Shift & Add های به روش بوث با Barrel shifter تعداد عمل جمع (با فرض اینکه عمل تفریق با استفاده از جمع انجام میشود) برابر است با تعداد مرزهای ۰ و ۱ ضرب کننده یکی کمتر (برابر است با تعداد ۰۱ ها بعلاوه ۱۰ ها در ضرب کننده یکی کمتر) و تعداد عمل شیفت هم برابر تعداد مرزهای ۰ و ۱ ضرب کننده است.

توجه ۱: در اکثر کامپیوترها عمل تفریق بوسیله جمع انجام میشود

* توجه ۲: یادتان نرود که سمت راست ضرب کننده یک ۰ در اول مساوی بذارید

* " ۳: اگر ضرب کننده بی علامت بود و بیت سمت چپ اش 1 بود، یک ۰ سمت چپ آن 1 قرار دهید

مثال - اینکه بوث با Barrel بهتر است یا Shift & Add بشکه‌ای به اینکه ضرب کننده چطور باشد بستگی دارد

ضرب کننده = 10101010

shift بشکه‌ای = ۳ جمع و ۴ شیفت

Booth = ۶ " و ۷ "

نکته) نکته بالا به تعداد تغییر بیت های ضرب کننده بستگی دارد

نکته) اگر در ضرب بوث در ضرب کننده یک ها دو تایی باشند بوث بشکه‌ای با shift & add بشکه‌ای یکسان است

ضرب کننده = 000110001101100

shift & Add بشکه‌ای = ۵ جمع و ۶ شیفت

بوث = ۵ " و ۶ "

نکته) ضرب بوث برای ضرب کننده هایی که یک های متوالی بیش از دو تایی باشند تاثیر مثبتی دارد

ضرب ترتیبی سیستم علامت مقدار

کافی است علامت دو عدد را XOR کنیم که علامت حاصل به دست آید، سپس علامت ۲ عدد را کنار می‌گذاریم و ارزش ۶ را با روش‌هایی که تا کنون گفته شده بصورت بی علامت ضرب می‌کنیم

ضرب سیستم مکمل ۲

ضرب در این روش سه تفاوت با ضرب بی علامت دارد

۱- آخرین حاصلضرب جزئی در صورت وجود تفریق می‌شود، نه جمع

۲- در هنگام جمع حاصلضرب‌های جزئی، اگر نیاز به گسترش pp بود، ما sign extend می‌کنیم

۳- در سیستم مکمل ۲ اگر کری به وجود بیاید ما آن را دور می‌ریزیم

n bit
x n bit
۲n بیت فضا می‌خواهیم

۳ ۲ ۱ ۰
-۵ | ۱ ۱ ۰ ۱ = -۳
x ۱۱ | ۱ ۰ ۱ ۱ = $(2^3 - 2^2 + 2^1 + 2^0)$

```
  1 1
  1 0 1 1
+ 1 1 1 1 1 0 1 1
+ 1 1 1 1 1 0 1 0
- 1 1 1 0 1 0 0 0
-----------------
1 1 1 1 0 0 0 0 = -16
```

سوال: فرض کنید سیستم مکمل ۲ است و ۶ بیتی است، حال بیان ضرب Bx6 بهترین گزینه کدام است؟

۱- ضرب را با گسترش علامت انجام دهیم و حاصلضرب‌های جزئی را به جز آخرین حاصلضرب جزئی جمع کنیم و آخری را تفریق کنیم

۲- ۶ را مکمل ۲ کنیم، سپس ضرب را با گسترش علامت انجام دهیم و حاصلضرب‌های جزئی را جمع کنیم و در نهایت حاصل را مکمل ۲ کنیم

معماری کامپیوتر

رامین رضوی

۳- B و ۴ را تکمیل ۲ کنیم سپس ضرب را با گسترش علامت انجام دهیم و حاصلضرب های جزئی را جمع کنیم

۴- هر سه گزینه

گزینه ۳

B و ۴ را تکمیل ۲ کن و با گسترش علامت ضرب را انجام بده.

۲n بیت را تکمیل دو می کنیم

گزینه ۲

B nbit
× ۴ nbit

۲n bit

اگر ۴ مثبت بود تکمیل ۲ اش کن و بعد... و در نهایت حاصل را تکمیل دو کن

۳n bit را تکمیل ۲ می کنیم

ترتیب اینها:

۱ > ۳ > ۲

بررسی شیفت:

منطقی: شیفت منطقی مخصوص اعداد بی علامت است. در این شیفت ما از چپ یا راست ثابت ۰ وارد می کنیم

شیفت به راست:

۸ = ۱۰۰۰ —sR→ ۰۱۰۰ = ۴

۵ = ۰۱۰۱ —sR→ ۰۰۱۰ = ۲

$\lfloor \frac{9}{2} \rfloor$

« « چپ:

۱۰ = ۱۰۱۰ —sL→ ۰۱۰۰ = ۴

۳ = ۰۰۱۱ —sL→ ۰۱۱۰ = ۶

نکته) شیفت به چپ همیشه معادل ضرب در دو است

و فقط هنگامیکه سرریز رخ ندهد این گفته درست است

$A = a_{n-1} a_{n-2} \cdots a_0$

حسابی: شیفت حسابی مخصص اعداد علامت دار است. در شیفت به چپ حسابی از سمت راست عدد ۰ وارد می شود و در شیفت به راست حسابی ما sign bit را وارد می کنیم.

$V = a_{n-1}$

↻ ۰۰۱۱ —ashR→ ۰۰۰۱ = ۱

-۶ = ↻ ۱۰۱۰ —ashR→ ۱۱۰۱ = -۳

شیفت به راست حسابی

$\lfloor \frac{a}{2} \rfloor$

معماری کامپیوتر

رامین رضوی

شیفت به چپ حسابی:

$A = a_{n-1} a_{n-2} \dots a_0$

$V = a_{n-1} \oplus a_{n-2}$

$+3 = 0011$	ashL →	$0110 = +6$ ✓
$-3 = 1101$	→	$1010 = -6$ ✓
$+6 = 0110$	→	$1100 = -4$ ✗
$-7 = 1001$	→	$0010 = +2$ ✗

ضرب ترتیبی سیستم مکمل ۲

```
   B
 x φ
-----
  A : φ
```

$B = 1101 = -3$

$\varphi = 1011 = -5$

		A	φ	
1) $\varphi_0 = 1$:	$A \leftarrow A+B$	C 1101	1011	↙ φ_0
	ashR Aφ	1110	1101	
		+ 1101		
2) $\varphi_0 = 1$:	$A \leftarrow A+B$	C 1011	1101	
	ashR Aφ	1101	1110	↙ φ_0
3) $\varphi_0 = 0$	ashR Aφ	1110	1111	↙ φ_0
		- 1101		
4) $\varphi_0 = 1$	$A \leftarrow A-B$	0001	1111	
	ashR Aφ	0000	1111	$= 15$

برای اعداد علامت دار

سوال - اگر ضرب کننده n بیتی مخالف ۰ باشد حداقل و حداکثر تعداد جمع و تفریق چندتا است؟

نکته) تعداد ضرب کننده هایی که حداقل جمع و تفریق را دارند n تاست.

1000
1100
1110
1111

⇒ حداقل جمع = ۰ ، تفریق = ۱

نکته) احتمال آنکه یک ضرب کننده n بیتی حداقل تعداد جمع و تفریق را داشته باشد $\frac{n}{2^n}$

n زوج = $\overline{1}0\overline{1}0$ ⇐ ۲- و ۱+
$0\overline{1}0\overline{1}$ ⇐ ۲- و ۲+

n فرد = $\overline{1}0\overline{1}0\overline{1}$ ⇐ ۳- و ۲+
$0\overline{1}0\overline{1}0$ ⇐ ۲- و ۲+

* حداکثر جمع یا تفریق چند تاست؟ $n = \lfloor n/2 \rfloor + \lceil n/2 \rceil$

تفریق = $\lceil n/2 \rceil$

جمع : $\lfloor n/2 \rfloor$

* در ضرب ترتیبی بوث

تعداد جمع ها = به تعداد ۱۰ ها

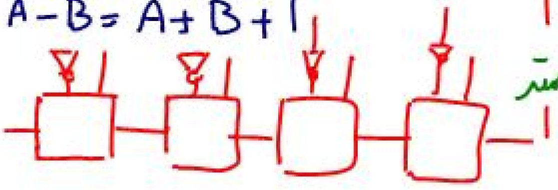
" تفریق " = " " ۱۰

" شیفت = به تعداد ۰۱ ها بعلاوه ۱۰ ها در صورتیکه از Barrel استفاده شود

به تعداد بیت های ضرب کننده : " " shR معمولی استفاده شود

در ضرب ترتیبی بوث چند جمع و چند عمل مکمل گیری داریم؟

$A - B = A + \overline{B} + 1$



تعداد جمع ها = بهینه : تعداد ۰۱ ها بعلاوه ۱۰ ها یکی کمتر

غیربهینه : جمع ۰۱ ها و ۱۰ ها

مکمل گیری : به تعداد ۱۰ ها

2n bit | n bit / n bit

تقسیم (بدون علامت):

Div cn : DnAn | cn → An, Dn

Dn n (n) | cn n — n بیت 1 — n+1 بیت

```
 n bit
 1000 1001 | 1101
-1101        1010
 01000
 -1101
 00111
```

$Dn \geq Cn$

نکته: اگر n بیت با ارزش مقسوم از مقسوم علیه بیشتر باشد حتما سرریز داریم

از اینجا به بعد به ازای هر کدام از بیت‌های مقسوم یک بیت در خارج قسمت می‌داریم

$$10011101 \div 1000 \Rightarrow 1\ldots$$

- overflow در تقسیم زمانی رخ میدهد که خارج قسمت در فضایی که بهش اختصاص دادیم نگنجد

$$1110010 \div 0011 \Rightarrow 1\ldots \text{ (۶ بیت، ۷ بیت)} \Rightarrow V=1$$

$$10011101 \div 1000 = 10011$$

```
 1001|1101 | 1000
-1000      |------
 ----      | 10011
   1110
  -1000
   ----
    1101
   -1000
    ----
     101
```

تست) می‌خواهیم 5n بیت را به 2n بیت تقسیم کنیم و خارج قسمت را در 4n بیت بگذاریم کدام سرریز رخ میدهد

۱- اگر 2n بیت با ارزش مقسوم از مقسوم علیه بزرگتر مساوی باشد

۲- " " n " " " " " " " " " ✓

۳- " " 4n " " " " " " " " "

۴- هیچکدام

$$5n \div 2n \to 4n \Rightarrow n\,4n \div 2n \to 4n$$

نکته: همیشه به اندازه خارج قسمت تا از سمت راست مقسوم جدا کنید

سوال: در تقسیم $n \div m \to K$ کی خطا رخ میدهد؟

$$(n-K)\,K \div m \to 1\,K$$

$$n-K \geq m \Rightarrow V=1$$

نکته)

$$m \div n \to m-n$$

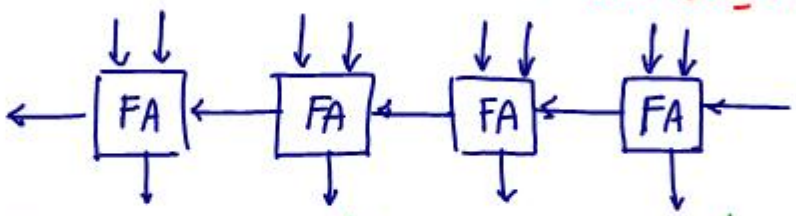
$$\overline{n}$$

جمع کننده ها:

جمع کننده موج گونه (ripple) : (RCA) ripple carry Adder
(cpa) carry propagation adder

شکل اش: هر FA ای به کری FA سمت راستی اش وابسته است

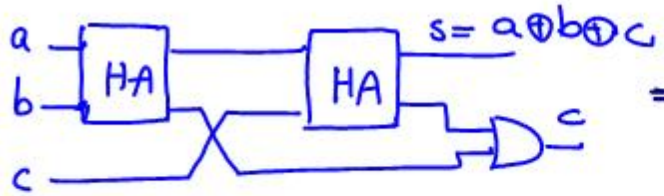
جمع کننده RCA ۱۶ بیتی ← تاخیر 32t ← اگر تعداد بیت ها n بیت باشد تاخیر 2nt است، پس تاخیر از $\Theta(n)$ است.

مدار ۸ سطحی است: 

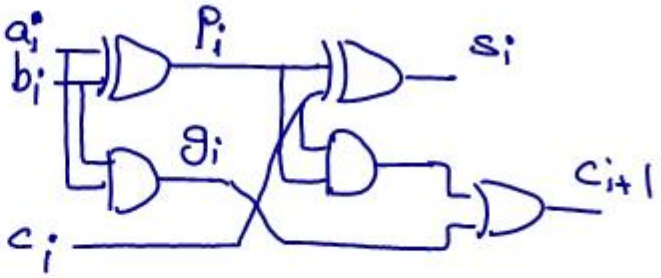
مشکل RCA این است که هر FA باید منتظر کری FA سمت راست خود باقی بماند؛ جمع کننده CLA عرضه اهد این وابستگی را از بین ببرد.

جمع کننده با پیش بینی رقم نقلی: carry look ahead Adder

$C_4\ C_3\ C_2\ C_1\ C_0$
$A = a_3 a_2 a_1 a_0$
$B = b_3 b_2 b_1 b_0$



⇒



① $P_i = a_i \oplus b_i$
$g_i = a_i b_i$

② $c_{i+1} = g_i + P_i c_i$

③ $S_i = P_i \oplus c_i$

$c_{i+1} = g_i + P_i c_i$

* همیشه برای تولید c_i ها به P_i ها و g_i ها و c_0 نیاز داریم

$i=0 \Rightarrow c_1 = g_0 + P_0 c_0$

$i=1 \Rightarrow c_2 = g_1 + P_1 c_1 \Rightarrow c_2 = g_1 + P_1 g_0 + P_1 P_0 c_0$

$i=2 \Rightarrow c_3 = g_2 + P_2 c_2 \Rightarrow c_3 = g_2 + P_2 g_1 + P_2 P_1 g_0 + P_2 P_1 P_0 c_0$

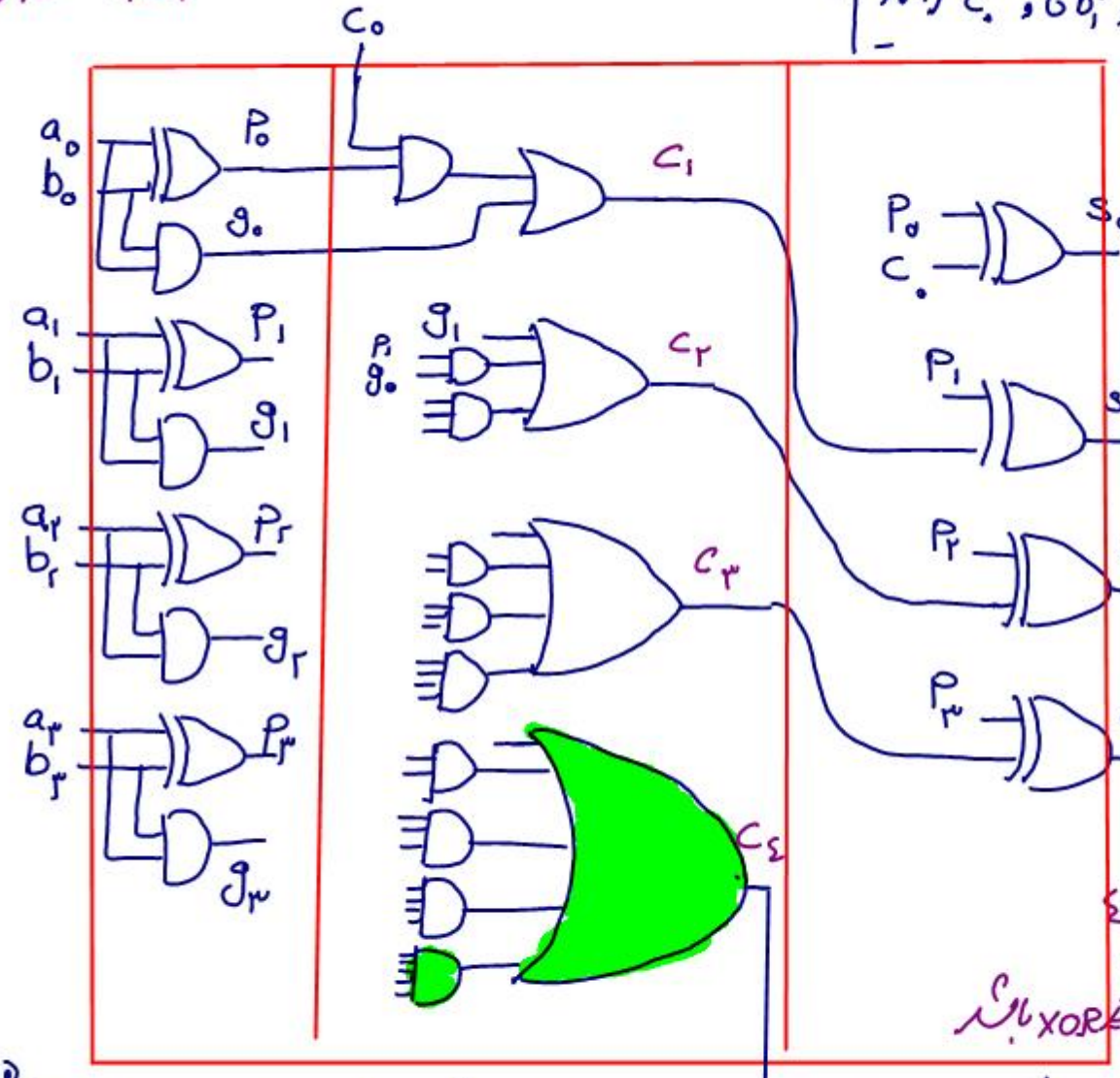
$i=3 \Rightarrow c_4 = g_3 + P_3 c_3 \Rightarrow c_4 = g_3 + P_3 g_2 + P_3 P_2 g_1 + P_3 P_2 P_1 g_0 + P_3 P_2 P_1 P_0 c_0$

① $P_i = a_i \oplus b_i$
$g_i = a_i b_i$

③ $S_i = P_i \oplus c_i$

* S_i برای آنکه محاسبه شود به c_i ها نیاز دارد

* ما در ابتدا g_i و b_i و c_0 را داریم.



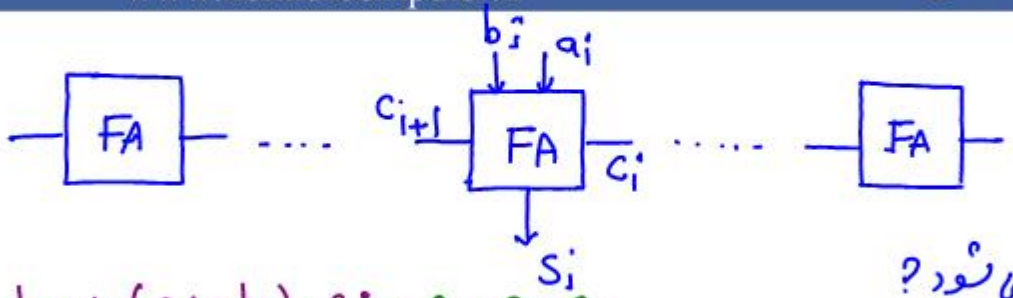
اگر XOR=۲t باشد

$4t \leq t_{CLA} \leq 6t$

اگر XOR=۴t باشد

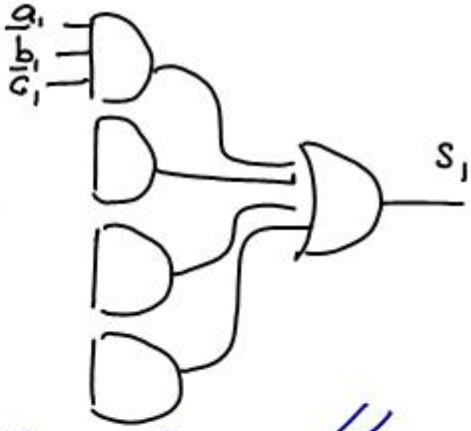
نکته) اگر با این روش جمع کننده n بیتی بسازیم تاخیر عوض نمی‌شود ولی AND و OR ای می‌خواهیم

n+1 ورودی



c_{i+1} کی 1 می شود؟

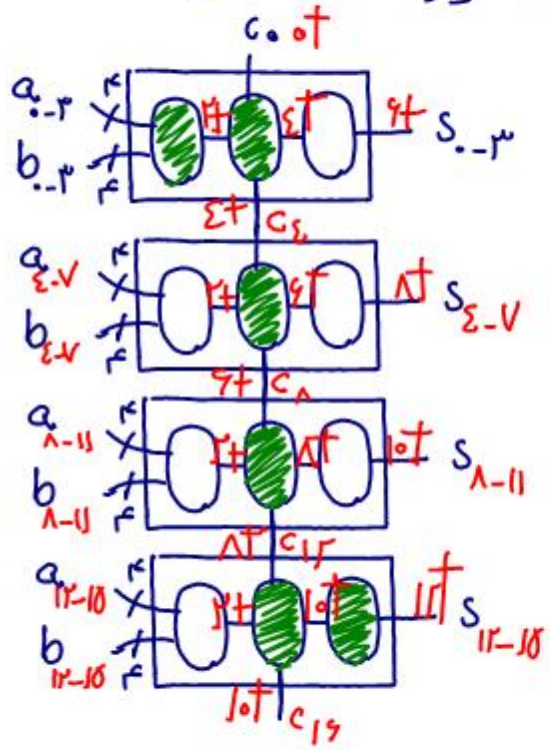
$$c_{i+1} = \underbrace{a_i b_i}_{g_i} + \underbrace{(a_i + b_i)}_{P_i} \cdot c_i = g_i + P_i c_i$$



$$S_i = a_i \bar{b}_i \bar{c}_i + \bar{a}_i b_i \bar{c}_i + \bar{a}_i \bar{b}_i c_i + a_i b_i c_i$$

نکته ۱ تاخیر تولید S_i در این روش ۵t است.

سوال - باگک CLA ی ۴ بیتی دو عدد ۱۶ بیتی را جمع کنید و تاخیرش را حساب کنید ($t_{XOR}=2t$)



نکته ۱ باگک n تا جمع کننده با پیش بینی رقم نقلی ۴ بیتی یک جمع کننده ۴n بیتی تاخیرش برابر

$(n+2)\times 2t$

۲ + ۶ = ۸

0010 + 0110 = 1000 = ۸ } 4 bit binary adder

۴ + ۸ = ۱۲ → 0001 0010

0100 + 1000 = 1100 = ۱۲
+ 0110
10010

A → BCD رقم
+ B → " "
حاصل

۰ ≤ حاصل ≤ ۹ ⟵ باینری = دهدهی

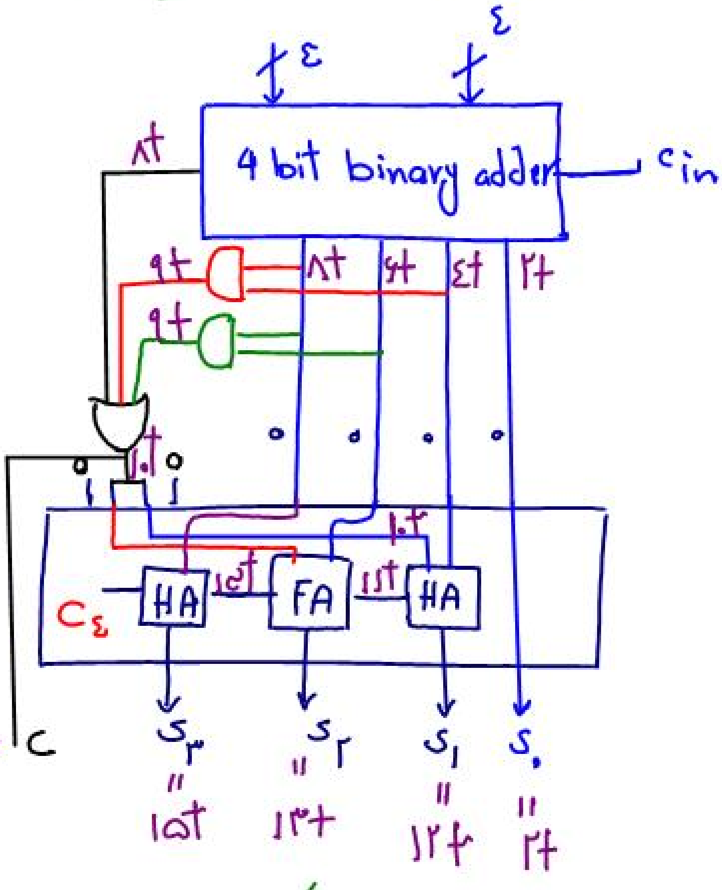
۱۰ ≤ حاصل ≤ ۱۹ ⟵ دهدهی = باینری + ۶

مجاز (۰-۹)

0110
0111
1000
1001
1010 *
1011 *
1100 *
1101 *
1110 * *
1111 * *
10000 *
10001 *
10010 *
10011 *

نکته) C_4 وقتی یک می‌شود که ۱۰ ≤ حاصل ≤ ۱۵ باشد و وقتی حاصل بین ۱۶ و ۱۹ باشد، C_4 صفر میدهد.

ما دنبال کسی هستیم برای کری Bcd Adder که وقتی حاصل بین ۱۰ تا ۱۹ هست ۱ بدهد



نکته) به جای HA آخر در جمع کننده دوم می‌توانیم فقط یک XOR بذاریم

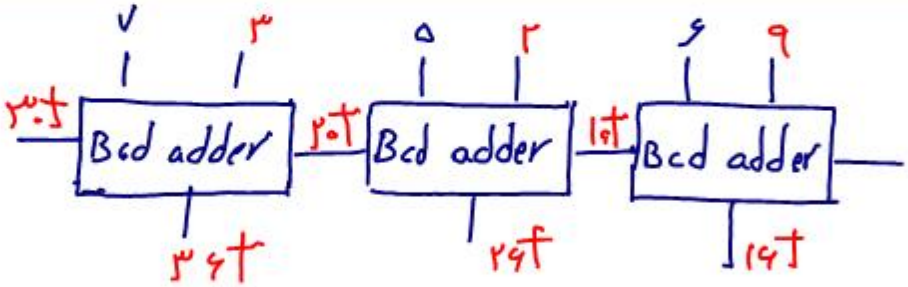
نکته) اگر تاخیر FA را 2t در نظر بگیریم و تاخیر گیت ساده را t فرض کنیم، Bcd Adder 15t تاخیر دارد و تاخیر تولید رقم دهگان اش (کری Bcd Adder) برابر 10t است.

سوال: به کمک سه تا Bcd Adder جمع ۲ عدد سه رقمی زیر در چه زمانی انجام می‌شود

معماری کامپیوتر

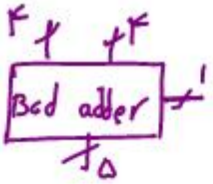
رامین رضوی

۷۵۶
+۳۲۹
۱۰۸۵



نکته) اگر n تا Bcd adder را باهم ripple کنیم تاخیر برابر: $(n-1)\times 10t + 16t$

سوال: Bcd adder چندتا ورودی غیرمجاز دارد؟



راه ۱: ورودی‌های غیرمجاز = ورودی‌ها − کل ورودی‌های مجاز = $512 - 10\times10\times2 = 312$

راه ۲: ورودی‌های غیرمجاز = $2\times$(A غیرمجاز و B غیرمجاز + A غیرمجاز و B مجاز + A مجاز و B غیرمجاز)

$(6\times10 + 10\times6 + 6\times6)\times2 = 312$

سوال: Bcd adder چند خروجی غیرمجاز دارد؟

غیرمجاز = کل − مجاز

$= 32 - 20 = 12$

تفریق کننده BCD: $(A)_2 - (B)_2 = (A)_2 + B$ مکمل ۲ $= (A)_2 + B$ مکمل ۱ $+ 1$

$(A)_{10} - (B)_{10} = (A)_{10} +$ مکمل نه $B + 1$

مکمل ۹ یک عدد BCD

۱- با ۶ جمع کن و سپس not کن

۲- ابتدا not کنید و سپس با ۱۰ جمع کنید

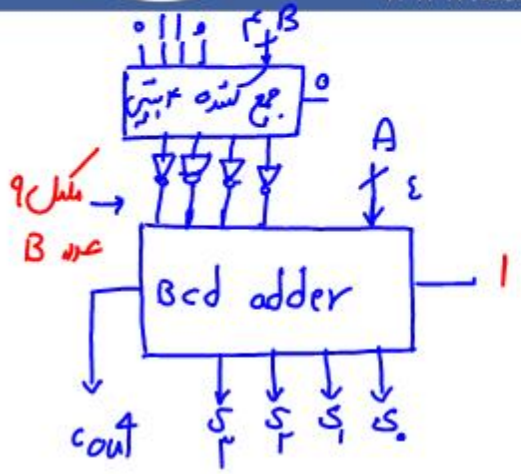
$5 = 0101 \xrightarrow{+6} 1011 \xrightarrow{\text{not}} 0100 = 4$

$5 = 0101 \rightarrow 1010 \xrightarrow{+10} 0100 = 4$

$1010 + 1010$

معماری کامپیوتر

رامین رضوی



ساخت Alu : (واحد محاسبه و منطق)

یکی از روش‌های ساخت Alu روشی است بنام برش بیت (Bit slice). در این روش برای ساخت یک Alu n بیتی ابتدا یک Alu یک بیتی ساخته می‌شود، سپس n تا از آن را بهم می‌چسبانند

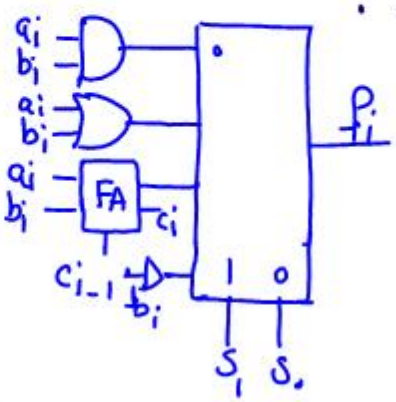
مثال - با روش Bit slice یک Alu چهاربیتی بسازید که عملیات not, ADD, OR, AND را انجام بدهد.

$A = a_3 a_2 a_1 a_0$

$B = b_3 b_2 b_1 b_0$

$f_3\ f_2\ f_1\ f_0$

S_1	S_0	operation
0	0	A AND B
0	1	A OR B
1	0	A + B
1	1	not (B)



فصل دوم: RTL (Register Transfer language) زبان انتقال ثبات

RTL زبانی است برای توصیف سخت افزار، یعنی برای ساخت سخت افزار ابتدا باید آن را به زبان RTL که نوعی اش کنید. RTL در سطح ثبات عمل می‌کند و در سطح گیت و ترانزیستور وارد نمی‌شود

همه دستورات RTL به صورت زیر هستند:

ریزدستور: بخش کنترل (سیگنال‌های کنترلی)

ریزعمل (Micro operation): عملی است که در 1 لبه کلاک انجام می‌شود

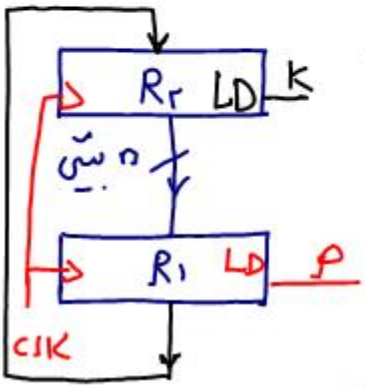
ریزدستور (micro instruction): به یک تعدادی ریزعمل که در 1 لبه کلاک انجام می‌شود

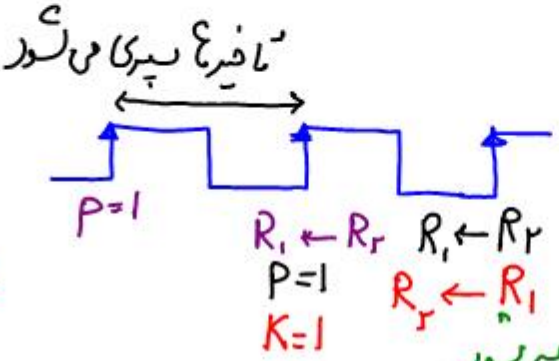
معماری کامپیوتر

رامین رضوی

مثال) دستور زیر را سنتز کنید

$P: R_1 \leftarrow R_2$





نکات:

۱- اگر می‌خواهیم در یک لبه یک زیر عملی انجام شود باید در لبه قبلی سیگنال های کنترلی لازم برای انجام شدن آن کار فراهم شود

۲- سیگنال های کنترلی بین دو لبه کلاک فعال هستند و بعد از آن از بین می‌روند

۳- در یک لبه کلاک ۲ نوع اتفاق ممکن است رخ بدهد

۱- زیر عملیاتی که سیگنال هایشان در لبه قبلی مهیا شده است اجرا شوند

۲- سیگنال هایی تولید شوند که در لبه بعدی زیر عملیاتی انجام شوند

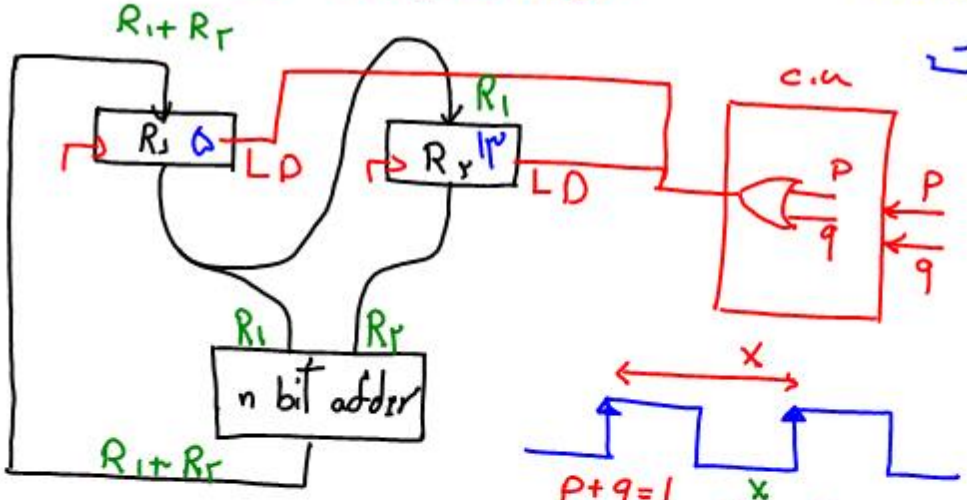
واحد کنترل | Data path

$P+Q: R_1 \leftarrow R_1 + R_2$ و $R_2 \leftarrow R_1$

OR — ADD — همزمان در ۱ لبه کلاک

مثال) RTL روبه رو را سنتز کنید

* یک ثبات همیشه مقدار درون اش درون پایه های خروجی اش است



$R_2 \leftarrow R_1$

$R_1 \leftarrow R_1 + R_2$

مثال) RTL روبه رو را سنتز کنید

$PQ: R_1 \leftarrow R_1 + R_2$ و $R_1 \leftarrow R_2$

غلط است . خطای conflict دارد

معماری کامپیوتر

رامین رضوی

نکته) در زبان RTL وقتی کلاک می‌آید، هر زیردستوری که شرطش همیا باشد فارق از آنکه در کجا کد نوشته شده است اجرا می‌شود

C_0 : —
C_1 : —
⋮
C_{1000} : —

سوال: RTL زیر را ساده‌تر کنید.

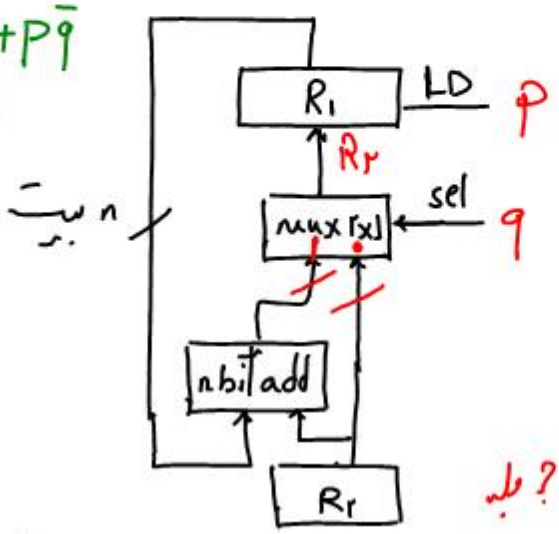
$P9: R_1 \leftarrow R_1 + R_2$
$P: R_1 \leftarrow R_2$

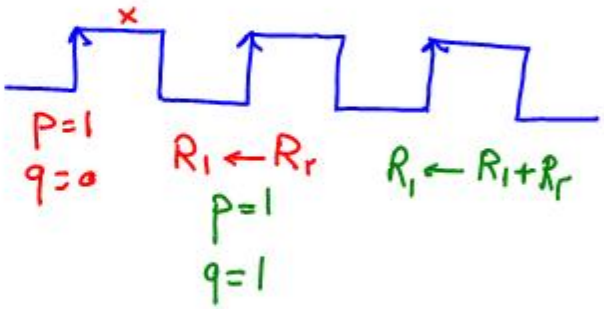
وقتی هر دو P و 9 یک باشند هر ۲ خط می‌خواهند اجرا شوند که این باعث می‌شود conflict رخ بدهد

سوال: RTL روبه رو را ساده‌تر کنید؟

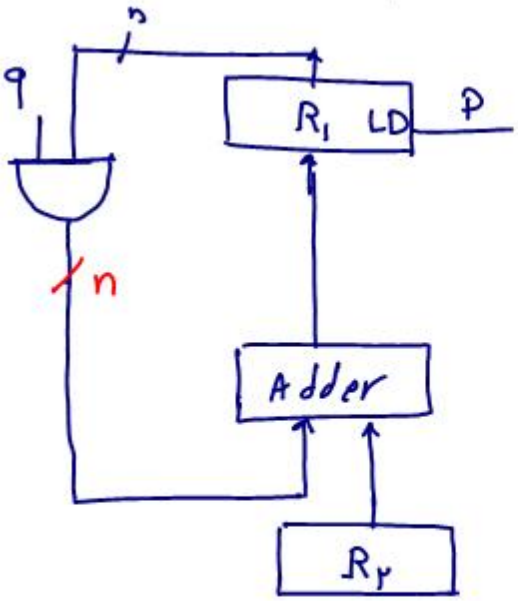
$P9: R_1 \leftarrow R_1 + R_2$
$P\bar{9}: R_1 \leftarrow R_2$

$LD(R_1) = P9 + P\bar{9}$
$= P(9 + \bar{9}) = P$





آیا می‌شود RTL بالا را مدل دیگری ساده‌تر کرد؟ بله



یک mux 2x1 از دو تا and و یک or تشکیل شده. بنابراین در بالا n تا and و 2n تا or داریم در صورتی که در پایین n تا and داریم بنابراین ساخت افزار بالایی بیشتر است.

معماری کامپیوتر

رامین رضوی

مثال - سفت افزار بسازید که عملیات زیر را به ترتیب و بی نهایت بار انجام بدهد

$T_0 : R_1 \leftarrow R_1 + R_2$
$T_1 : R_2 \leftarrow R_3$
$T_2 : R_1 \leftarrow R_2 + R_3$
$T_3 : R_2 \leftarrow R_1 + R_3$

۱ ۰ ۰ ۰
۰ ۱ ۰ ۰
۰ ۰ ۱ ۰
۰ ۰ ۰ ۱

- سیگنال های زمانی به صورت هستند که در یک لحظه از زمان فقط یکی از آنها مقدار دارند و بقیه ۰ هستند

۲ روش کلی برای تولید سیگنال زمانی داریم:

۱- با استفاده از شمارنده ها
  ۱- با استفاده از شمارنده n بیتی و یک دیکدر
  ۲- با ... ... حلقوی
  ۳- ... ... جانسون
۲- با ... ... فلیپ فلاپ ها

(۲)

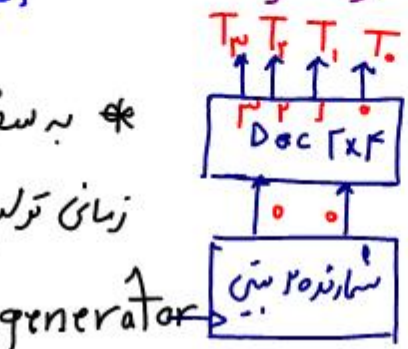
P_1 : ۰ ——— ۱ ، $P_1 = 0$ و $P_2 = 1$
P_2 : ——— ، $P_2 = 0$ و $P_3 = 1$
P_3 : ——— ، $P_3 = 0$ و $P_4 = 1$

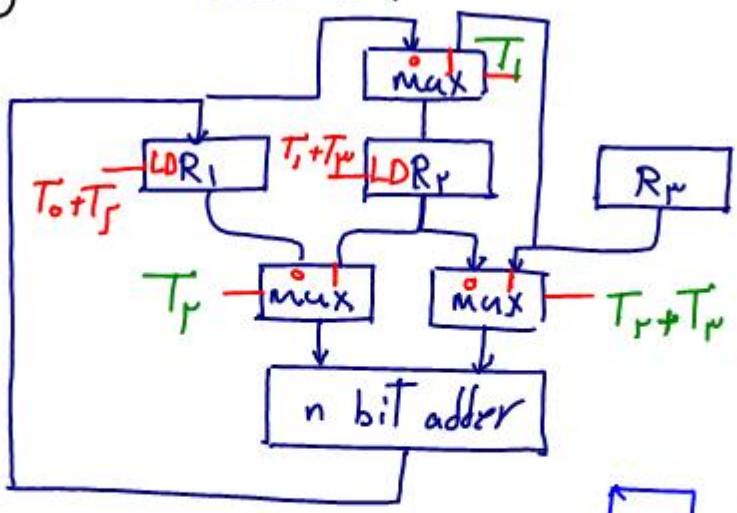
۱- تولید سیگنال زمانی با استفاده از شمارنده n بیتی و دیکدر

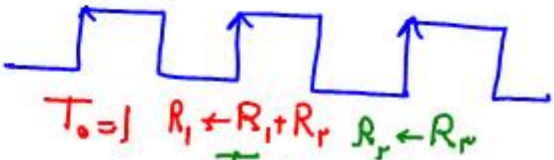
* شمارنده ها با استفاده از فلیپ فلاپ ها ساخته می شوند و FF ها نیز کلاک می خورند

* به سفت افزارهایی که سیگنال زمانی تولید می کنند TSG می گویند

TSG = Time signal generator







MI : سیگنال های کنترلی

- تمامی خط های RTL بصورت زیر است :

MI : به مجموعه ای از RT ها میگویند

RT (Register transfer) assignment & RT ها هستند که دارای ۳ ویژگی هستند

۱- متصدر assignment یک ثبات باشد

۲- در ۱ کلاک انجام شود

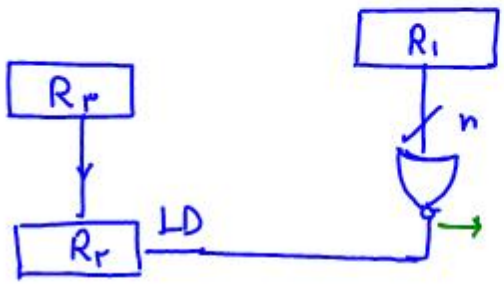
۳- عملی باشد (pragmatics) : صفت اقرار انجام آن کار وجود داشته باشد

مثال - RTL روبه رو را شرح کنید :

if $R_1=0$ then $R_2 \leftarrow R_3$

* if $OR(R_1)=0$ then $R_2 \leftarrow R_3$

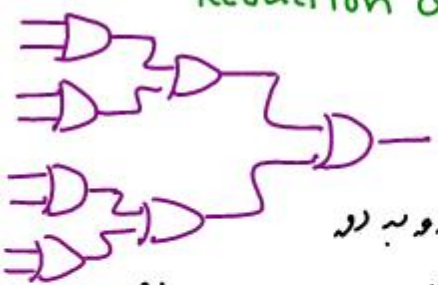
- صفر بودن یا نبودن را گیت NOR تشخیص می دهد



if $(R_1 \neq 0)$ then $R_2 \leftarrow R_3$ ⇒

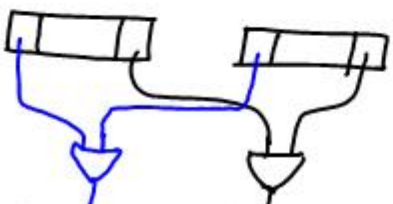
- مخالف صفر بودن با OR تشخیص داده می شود

Reduction OR



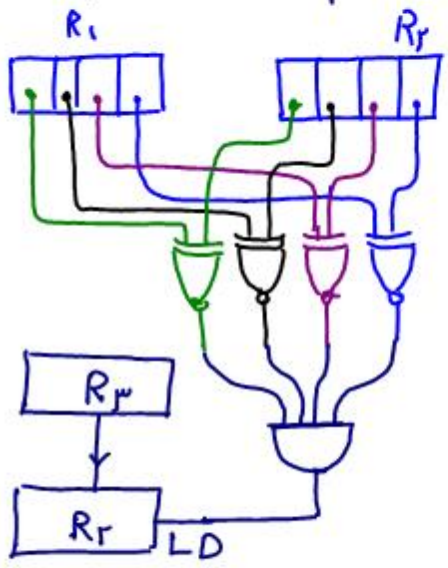
* ما از هر گیتی دو نوع داریم ، مثلاً در مورد OR ما هم Bitwise OR داریم و هم Reduction OR

Bitwise OR-۱ : دو داده n بیتی را میگیرد و بیت های متناظر را دو به دو باهم OR می کند و یک خروجی n بیتی ایجاد می کند ، این OR را معمولاً با نماد $R_1 \vee R_2$ نشان می دهند

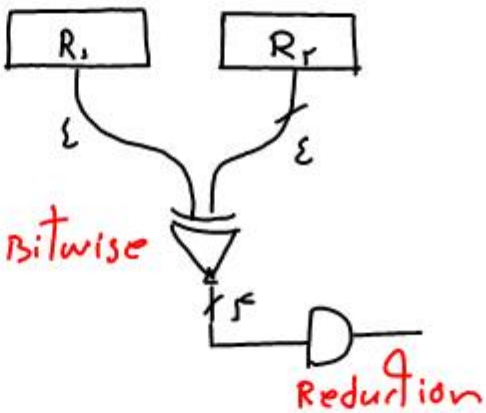


مثال -

if ($R_1 == R_2$) then $R_2 \leftarrow R_3$



برای توضیح این سوال فرض می‌کنیم دو ثبات چهار بیتی اند



نکته) زبان های HDL (hardware description language) برای توصیف سخت افزار استفاده می‌شوند، زبان های HDL زیادی داریم که به طور کلی زبان های LSI (large scale integration) (مدارهایی با حدود ۱۰۰۰ گیت) که بخش کوچکی از HDL هستند را یاد می‌گیریم.

SSI ≃ ۱۰ گیت
MSI ≃ " ۱۰۰
LSI ≃ " ۱۰۰۰
VLSI ≃ " ۱۰۰۰۰

* سه زبان LSI داریم: ۱) RTL
۲) ASM
۳) mp (micro programing)

نکته - mp, ASM نسخه‌های تعمیم یافته RTL هستند

نکته - زبان RTL دارای خصوصیات زیر است

۱- هم روند است: زبان هایی که گاه‌ها خطوط برنامه را جابجا کنیم برنامه عوض نشود

۲- flow ندارد

۳- در RTL هم می‌توانیم اجرا سری داشته باشیم و هم اجرا موازی - مثلا اجرا سری را می‌توانیم با استفاده از فلگ ها ایجاد کنیم. موازات در سخت افزار مهم است که وجود داشته باشد

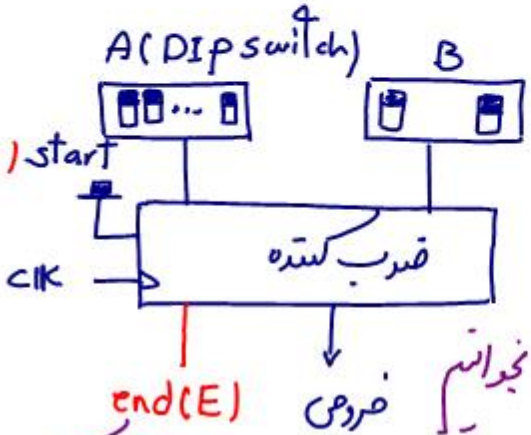
معماری کامپیوتر

رامین رضوی

نکته ـ هر سیستمی را که با RTL پیاده سازی کنید (توصیف کنید) حتما سیستم شما داراں کلاک خواهد شد، چون شما هر چیزی که با RTL پیاده سازی کنید، حتما در آن سیستم ثبات خواهید داشت اما ثبات هم که نیاز به کلاک دارند، پس سیستم هایی که با RTL پیاده سازی می شوند ترتیبی هستند

نکته ـ با RTL ما نمی توانیم عملیاتی مثل sin، $\sqrt{}$، ضرب و تقسیم و ... را انجام دهیم، زیرا این عملیات پیچیده هستند و در یک کلاک انجام نمی شوند.

مثال جامع: برنامه ای (الگوریتمی) به زبان RTL برای ضرب دو عدد n بیتی به روش جمع متوالی بنویسید و سپس Data path (DP) و واحد کنترل (CU) را بصورت کامل طراحی کنید



ـ یک ملیپلکس نلاپ (فلگ) E در نظر می گیریم که هر زمان که جواب نهایی آماده شد 1 شود، به ۲ دلیل از این E استفاده می کنیم

۱ـ زودتر از زمانی که حاصلضرب آماده شده بیاییم و E را به غلط نخوانیم

۲ـ زمان محاسبه حاصلضرب لزوما ثابت نیست و با توجه به ورودیهای A و B می تواند تغییر کند

الگوریتم جمع متوالی: به اینصورت است که ما به تعداد متغیر کوچک، متغیر بزرگ را با خودش جمع می زنیم

به همین دلیل در الگوریتم جمع متوالی ابتدا اعداد را با هم مقایسه می کنند. فرض کنید A > B

بعد از مقایسه تعداد کوچکتر در R_1 قرار دارد، حال می آییم به ازای هر باری که یکی از R_1 کم می کنیم R_2 را با خودش جمع می کنیم و این کار را تا زمانیکه R_1 به ۰ برسد ادامه می دهیم.

S: $R_1 \leftarrow A$, $R_2 \leftarrow B$, $E \leftarrow 0$, $R_3 \leftarrow 0$

G(R_1, R_2): $R_1 \leftarrow R_2$ و $R_2 \leftarrow R_1$

۱- $\overline{F_1}$ S: $R_1 \leftarrow A$, $R_2 \leftarrow B$, $E \leftarrow 0$, $R_3 \leftarrow 0$, $F_1 \leftarrow 1$

۲- F_1 G(R_1, R_2): $R_1 \leftarrow R_2$, $R_2 \leftarrow R_1$

۳- F_2 oR(R_1): $R_3 \leftarrow R_2 + R_3$, $R_1 \leftarrow R_1 - 1$

۴- $\overline{F_2}$ F_1: $F_2 \leftarrow 1$

۵- F_2 $\overline{oR(R_1)}$: $F_1 \leftarrow 0$, $F_2 \leftarrow 0$, $E \leftarrow 1$, $R_4 \leftarrow R_3$

حال می‌خواهیم این که را استند کنیم

سفت افزار ما از دو قسمت تشکیل شده

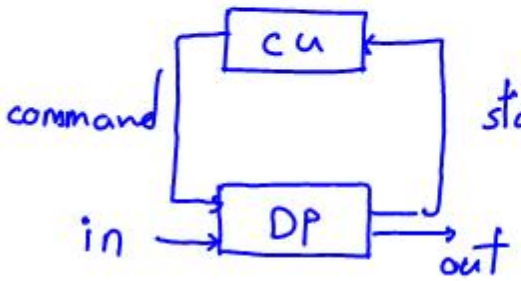
۱- Data path (Dp)

۲- control unit (cu) : سفت‌ور واحد کنترل خود ۲ مرحله دارد

۱- پیاده سازی sequencer (یا Flag سازی)

۲- ایجاد سیگنال های کنترلی (command)

- Dp مسئول انجام اعمالی مانند جمع و تفریق و مقایسه و انتقال و ... است و ترتیب انجام کارها را نمی داند و این واحد کنترل است که با ارسال سیگنال هایی بنام command در هر پالس ساعت ترتیب انجام کارهایی که Dp باید انجام بدهد را مشخص می‌کند، حال برای اینکه cu بتواند توالی کارها را بدرستی مشخص کند نیاز دارد که از وضعیت برخی از داده ها در Dp اطلاع داشته باشد، حال اسم این سیگنال هایی که وضعیت مقادیر درون Dp را به cu اطلاع میدهد status است، بنابراین شکل زیر را داریم،



- ورودی هایی که وارد سیستم می‌شوند همیشه وارد Dp می‌شوند و داده هایی هم که بعنوان خروجی از سیستم خارج می‌شوند همیشه از Dp خارج می‌شوند

* در یک گذر RTL تمام RT ها سمت راست دو نقطه به غیر از RT هایی مربوط به فلگ ها مربوط به DP هستند و در سمت چپ دو نقطه هم هرجا که محاسباتی انجام می‌شود مربوط به DP است.

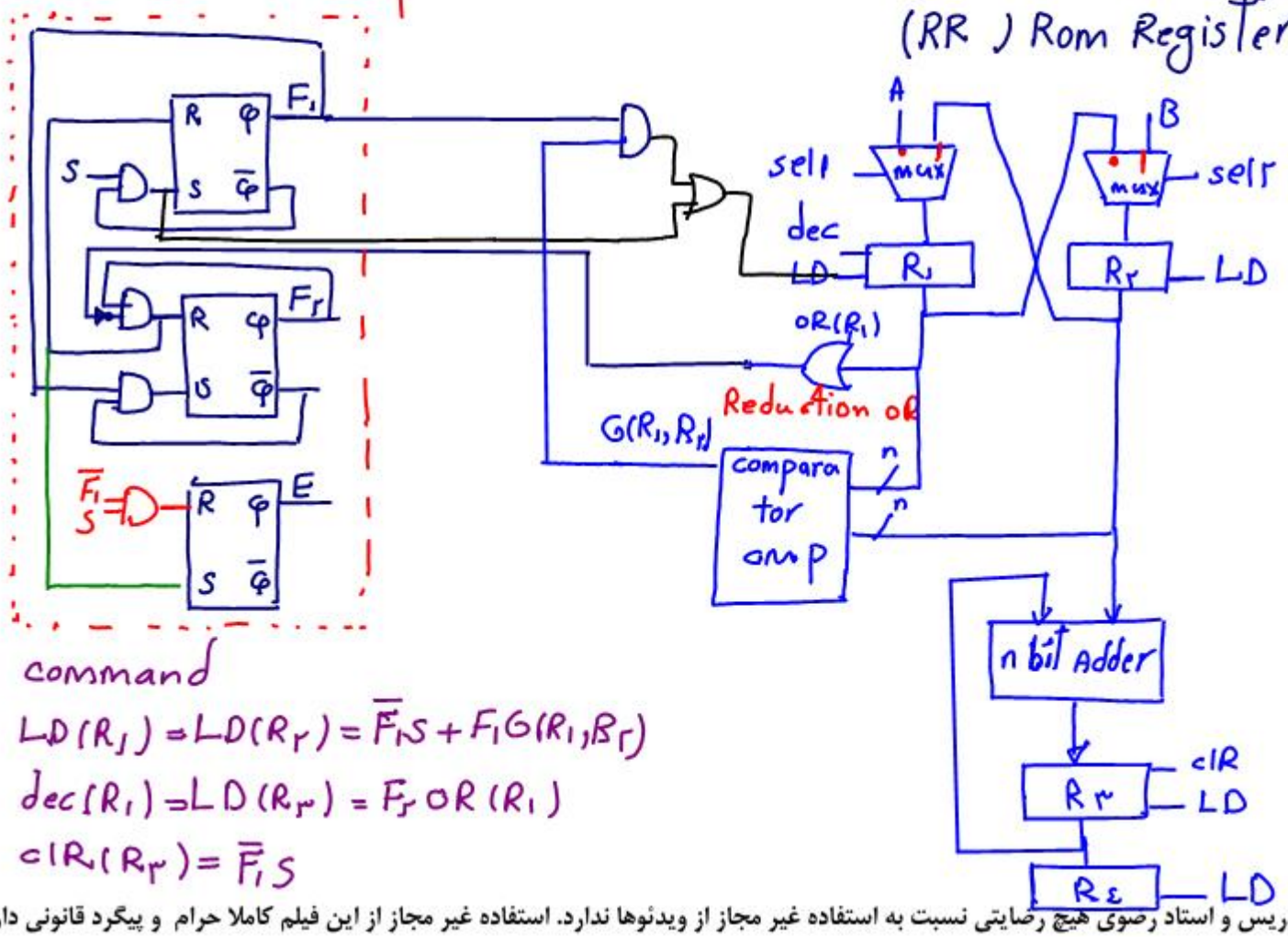
- ابتدا می‌رویم سراغ بستر DP و بعد می‌رویم سراغ بستر CU، برای بستر DP از هرجای RTL که دوست داریم می‌توانیم شروع کنیم، برای پیاده‌سازی CU هم ۳ روش زیر وجود دارد که ما در اینجا فعلا روش اول را بررسی می‌کنیم

۱- پیاده‌سازی با فلیپ فلاپ‌های R-S

۲- روش one-hot (Direct)

۳- RR (Rom Register)

R	S	φ^*
0	0	φ
0	1	1
1	0	0
1	1	غیرمجاز



command

$LD(R_1) = LD(R_2) = \bar{F}_1 S + F_1 G(R_1, R_2)$

$dec(R_1) = LD(R_3) = F_2 OR(R_1)$

$clR_4(R_4) = \bar{F}_1 S$

معماری کامپیوتر

رامین رضوی

- تقسیم پیاده سازی واحد کنترل ۲ مرحله دارد

۱- سافت sequencer ← با استفاده از فلیپ فلاپ های R-S پیاده سازی کردیم

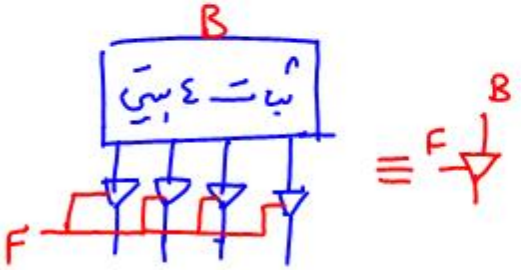
۲- ایجاد سیگنال های command

در این روش به ازای هر فلگ یک فلیپ فلاپ R-S گرفتیم

۱- هر وقت فلگی برابر ۱ شد، سیگنال کنترلی مربوط به آن خط را به R می دهیم

۲- " " " " " " " " " S " "

تعریف BUS = به سیم هایی (اتصالاتی) که دارای چند تا درایور هستند باس می گویند و درست است که باس یک اتصال Multi-Driver است ولی در هر لحظه از زمان یک تنها درایور باس را درایو می کند، اگر این اتفاق نیفتد conflict رخ میدهد



- don't care ها در یک مدار خروجی هایی هستند که چه ۰ بدهیم چه ۱ بدهیم چه X بدهیم روی کارکرد مدار تاثیری ندارد، حالا این موضوع باعث می شود که سیگنال هایی که گاهی اوقات می توانند don't care باشند (مثل sel 1) را بتوان بصورت های مختلفی بدست آورد، چون یک طراح ممکن است don't care ها را ۱ بگیرد، دیگری ۰ بگیرد و به این جواب منحصر به فرد نداریم، حال چون مقدار sel 1 مانند بقیه همیشه مهم نیست و فقط زمانیکه در R_1 نوری حرکت می گیرد مهم است، پس گاهی اوقات sel 1 می توانند don't care باشند پس برای طراحی باید

۱: $\overline{F_1}S$; sel 1 = ۰

۲: $F_1 G(R_1, R_2)$; sel 1 = ۱

۳: $F_2 OR(R_1)$; sel 1 = X

۴: $\overline{F_2} F_1$; sel 1 = X

۵: $F_2 \overline{OR(R_1)}$ = sel 1 = X

* ما باید sel 1 را جوری بدست بیاوریم (بنویسیم) که هر وقت $F_1 G(R_1, R_2)$ یک شود sel 1 = ۱ شود و هر وقت $\overline{F_1}S$ یک شد، sel 1 = ۰ شود، با توجه به این نکته برای sel 1 عبارت های زیر را می توان نوشت

$sel\ 1 = F_1 G(R_1, R_2) = F_1 + \overline{S} = F_1$

نکته) DP بیش از ۹۵ درصد مدارات را (گاهی اوقات بیش از ۹۹ درصد) اشغال می‌کند و واحد کنترل در بهترین حالت ۵ درصد مدارات را اشغال می‌کند به همین علت در معماری کامپیوتر به زبان ساده کردن CU نیستند، چون gain خاصی برایمان ندارد و اصلا دلیل نمیشه componentlevel را DP طراحی می‌کنند و نه gate level همین بزرگ بودن بسیار زیاد DP .

نکته) CU برای تولید سیگنال‌های command از داده‌های status و همین طور sequencer (۲) (۱) استفاده می‌کند

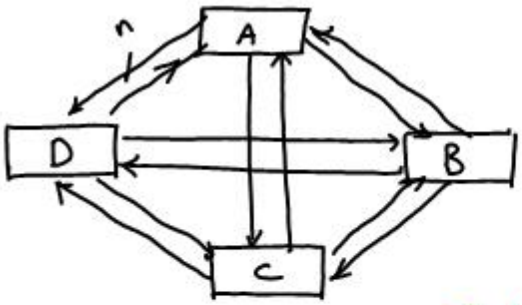
طراحی باس: برای طراحی باس یا مسیر یا گذرگاه بین معماران ثبات ۲ روش کلی وجود دارد

۱- point to point (نقطه به نقطه یا باس اختصاصی)

۲- common bas (باس مشترک)

۱- p2p : به ازای هر دو ثبات یک مسیر رفت و برگشت قرار می‌دهیم

سوال - برای ۴ ثبات ۵ بیتی به روش p2p باس طراحی کنید



مزیت‌های p2p:

۱- چندین انتقال همزمان را می‌توان انجام بدهد

۲- point of failure ندارد. تحمل پذیری اش در برابر خرابی بالاست

معایب p2p:

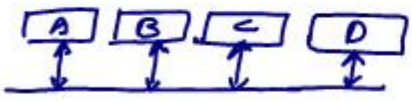
سخت افزار زیادی مصرف می‌کند

فرض کنید که m ثبات n بیتی داشته باشیم در اینصورت $m(m-1)$ مسیر مختلف n بیتی داریم

$$m-1+m-2+\dots+1=\frac{m(m-1)}{2}\times 2$$

و یکسری هم مالتی پلکسر وجود دارد.

طراحی باس مشترک: همان طور که از اسم اش پیداست، در اینجا رجیسترها دیگر بصورت مستقیم با هم ارتباط ندارند، بلکه یک مسیر مشترکی وجود دارد که رجیسترها با آن مسیر ارتباط دارند



- طراحی باس مشترک به دو روش انجام می‌شود

۱- با استفاده از MUX

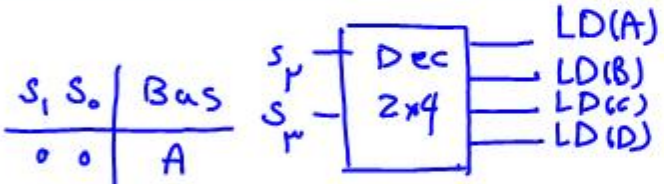
۲- به بافرهای سه حالته

معماری کامپیوتر

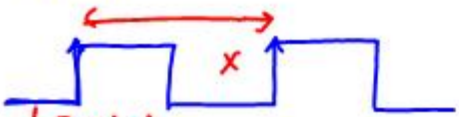
رامین رضوی

۱ـ با استفاده از مالتی پلکسر

مثال ـ برای ۴ ثبات ۲ بیتی باس مشترک بسازید

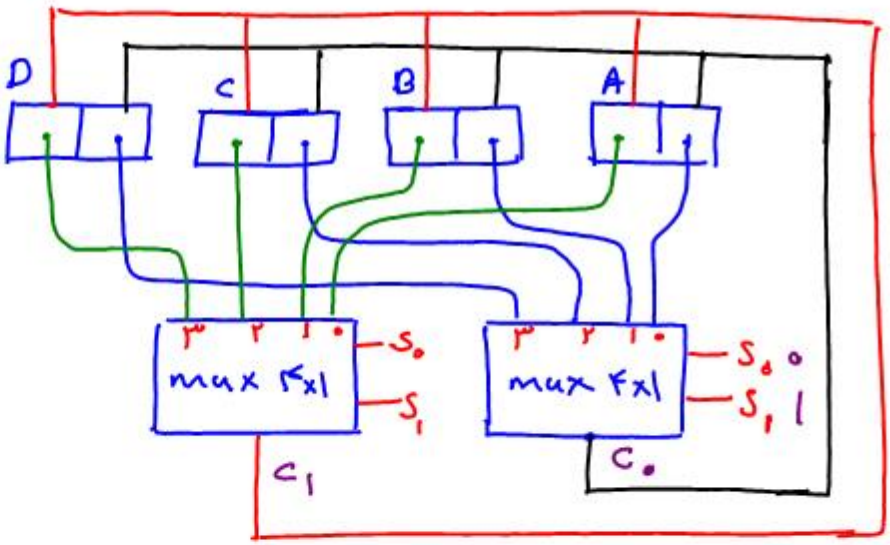


S_1 S_0	Bus
0 0	A
0 1	B
1 0	C
1 1	D



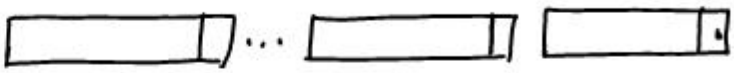
LD(A)=1
LD(D)=1
$S_2S_0=10$

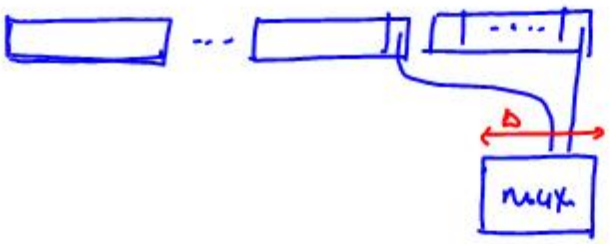
A ← C
D ← C



S_3S_2	
0 0	LD(A
0 1	LD(B)
1 0	LD(C)
1 1	LD(D)

سوال ـ برای ۷ ثبات ۱۵ بیتی به چند مالتی پلکسر چندتایی نیاز است؟





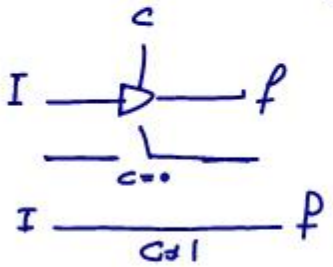
* کاما مالتی پلکسر ۸x1

سوال ـ برای ۵ ثبات ۸ بیتی سوال بالا را حل کنید

۸ mux ۸x1 + ۱ dec ۳x۸

نکته ـ برای m ثبات n بیتی ۱ dec $\lceil \lg m \rceil \times 2^{\lceil \lg m \rceil}$ ، n mux $2^{\lceil \lg m \rceil}$ x1

۲ـ با استفاده از بافرهای سه حالته.



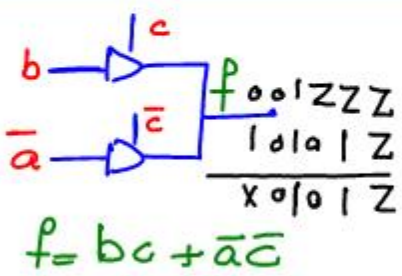
C	f
0	Hz
1	I

$x f = Ic$

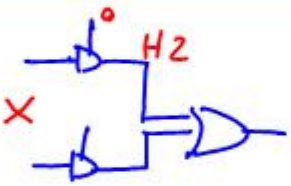
ـ اتصال خروجی بافرهای سه حالته منطق OR را به وجود می آورد البته در صورتیکه در هر لحظه از زمان یکی از آن خروجی ها مقدار داشته باشد و سایر خروجی ها Hz باشد

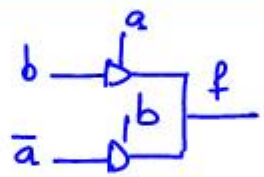
معماری کامپیوتر

رامین رضوی

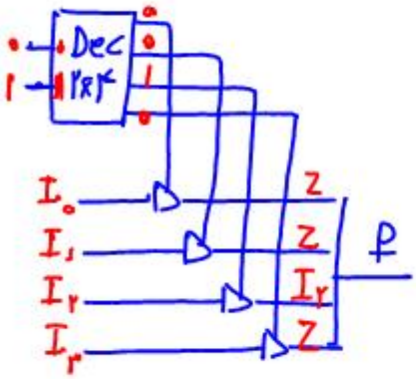


$f = bc + \bar{a}\bar{c}$

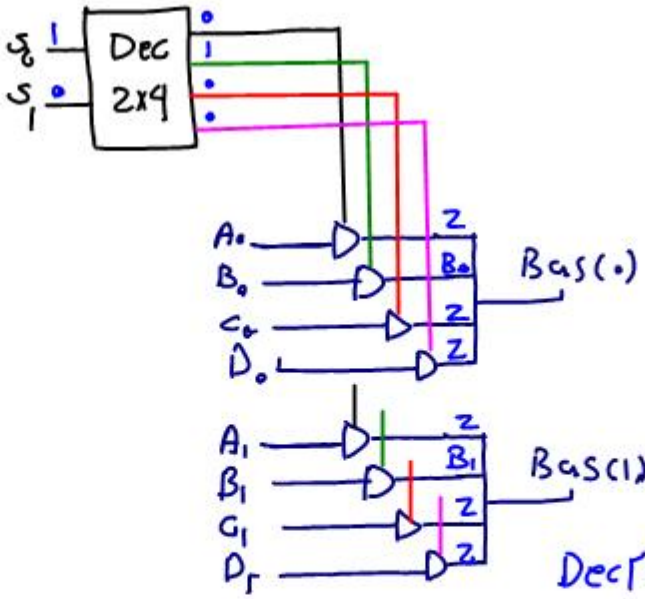




a	b	f
0	0	Hz
0	1	1
1	0	0
1	1	X



سوال - برای 4 ثبات دوبیتی با استفاده از بافرهای سه حالته باس مشترک بسازید؟

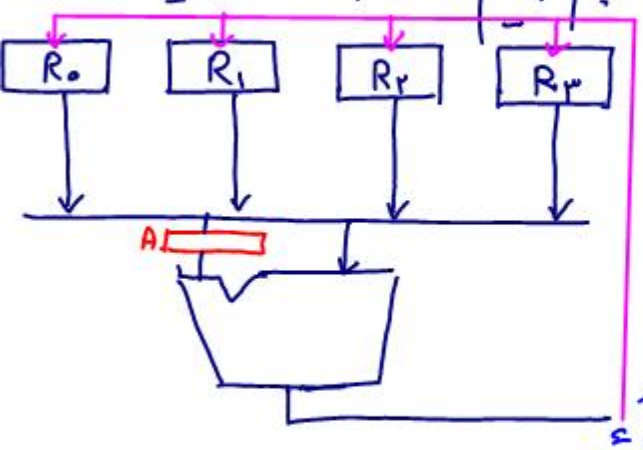


S_1	S_0	Bus
0	0	A
0	1	B
1	0	C
1	1	D

- ۲ خروجی نامیده می‌شود
- ۸ تا بافر داریم بعلاوه یک Dec 2x4
- ۲ دسته بافر ۴ تایی

سوال - ۳۱ ثبات ۱۷ بیتی داریم : ۱۷ دسته ۳۱ تایی و یک Dec 5x32

سوال - در شکل زیر می‌خواهیم عملیات $R_1 \leftarrow R_2 + R_1$ را انجام بدهیم، حال به سوالات زیر پاسخ دهید



۱- زمان انجام این عملیات چقدر است

۲- تعداد کلاک لازم برای این عمل چقدر است

۳- پریود کلاک چقدر است

اگر: الف) هیچ رجیستری پشت پایه‌های ALU نباشد

ب) پشت یکی از پایه‌های ALU یک رجیستر باشد

ج) " هردو " ALU " "

د - پشت هر دو پایه ALU و جلو پایه خروجی آن یک رجیستر باشد

- فرض کنید تاخیر بایس ۱۰ns و تاخیر جمع کننده ۱۵ns است، تاخیر سیم ها را هم ناچیز بگیرید

الف - هیچ رجیستری پشت ALU نباشد : در اینصورت باید از دو باس مشترک استفاده کنیم

در این حالت به ۱ کلاک نیاز داریم

یک کلاک : $R_1 \leftarrow R_1 + R_2$

$T = 25ns$

زمان انجام این عملیات = 25ns

ب - پشت یکی از پایه های ALU یک رجیستر باشد

۱- $A \leftarrow R_1$: 10ns
۲- $R_1 \leftarrow A + R_2$: 25ns

$T = 25ns$

زمان اجرا = $2 \times 25ns = 50ns$

ج) پشت هر دو پایه ALU ثبات وجود داشته باشد

۱- $A \leftarrow R_1$: 10ns
۲- $B \leftarrow R_2$: 10ns
۳- $R_1 \leftarrow A+B$: 15ns

$T = 15ns$

زمان اجرا = $3 \times 15ns = 45ns$

د)

۱- $A \leftarrow R_1$: 10ns
۲- $B \leftarrow R_2$: 10ns
۳- $C \leftarrow A+B$: 15ns
۴- $R_1 \leftarrow C$: ناچیز

$T = 15ns$

زمان اجرا = $4 \times 15ns = 60ns$

معماری کامپیوتر

رامین رضوی

معماری پردازنده ها

کامپیوترها به طور کلی به ۲ دسته تقسیم می شوند

۱- کامپیوترهای چند منظوره (multi perpose) (پردازنده) : کامپیوتری که نرم افزار قبول و اجرا کند

۲- " Dedicated : اینها کامپیوترهایی هستند که برای انجام یک کار خاص ساخته شده اند.

نکته) اکثر سیستم های embedded ، Dedicated هستند ، البته ماشین های نخفته چند منظوره نیز داریم

- پردازنده ها فقط نرم افزارهایی را قبول می کنند که به زبان ماشین نوشته شده باشند ، زبان ماشین نیز رشته ای از ۰ و ۱ است ، این رشته را به یکسری دسته تقسیم می کنند و به هر دسته یک instruction یا macro instruction می گویند

ADD R_1, R_2 ← اسمبلر ← 00110100 00010 11010

نکته) اسمبلر هیچ خاصیتی در برنامه شما نمی کند ولی کامپایلر خاصیت می کند

توجه) micro instruction را با macro instruction اشتباه نگیرید micro instruction یا زیر دستور مجموعه ای از RT های ساده ای بود که در ۱ پالس کلاک اجرا می شد.

توجه) گرچه instruction خالی نیستند به احتمال زیاد منظور درست دستور (macro inst) است و منظورشان دستورات RTL ، Asm ، micro program نیست

نکته) زبان ماشین از یک پردازنده به پردازنده دیگر فرق می کند ، حتی زبان ماشین پردازنده های یک خانواده ممکن متفاوت باشد : مثلاً زبان ماشین ARM7 با ARM9 متفاوت است

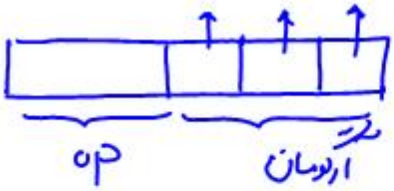
معماری کامپیوتر

رامین رضوی

- هر inst ـ macro شامل دو بخش است:
  1ـ op (opcode = operation code)
  2ـ Argument (آرگومان)

توجه) البته هر بخش می تواند به زیربخش هایی تقسیم شود

- op مشخص می کند که آن mi چه کاری می خواهد انجام بدهد
- آرگومان مشخص می کند که داده هایی را که می خواهیم روی آنها عملیات انجام بدهیم از کجا بیاوریم و حاصل عملیات را در کجا بگذاریم.



معماری پردازنده

برخلاف آنچه که عموم فکر می کنند معماری پردازنده به معنای شکل سخت افزاری اش نیست و معماری پردازنده همان مجموعه اطلاعاتی که (آن دانشی که) برنامه نویس زبان ماشین از یک پردازنده به واسطه زبان ماشین بلد بودن اش بدست می آورد. هر معلومات دیگری که برنامه نویس به واسطه دانش خودش از آن پردازنده بداند جزء معماری آن پردازنده محسوب نمی شود

توجه: اینکه یک پردازنده چه سخت افزار و چه قطعاتی دارد و آنها چگونه بهم وصل شده اند و ... هیچ کدام جزء معماری پردازنده محسوب نمی شوند و به اینها زیر معماری می گویند (micro architecture)

وقتی RT یا Asm یا micro program را به کار می بریم انگار زیر معماری اش را داریم

چه مواردی جزء معماری پردازنده محسوب می شوند

1ـ چه دستورات زبان ماشینی داریم
2ـ opcode دستورات چیست
3ـ قالب دستور (instruction format): هر دستور از چه میدان هایی تشکیل شده

۴- طول دستورات (طول macro inst)، طول op، طول آرگومان و طول هر میدان

۵- Data type

۶- چه مدهای آدرس دهی داریم: مد آدرس دهی نحوه تفسیر آرگومان را مشخص می‌کند و این op است که مد آدرس دهی را مشخص می‌کند

ADD R1,R2 11000 001 010
ADD R1,23 00001
ADD R1,[100]

۷- چه ثبات‌های قابل مشاهده‌ای وجود دارد.

ثبات‌ها در هر پردازنده ای به ۲ دسته قابل مشاهده و غیرقابل مشاهده تقسیم می‌شود

visible register: ثبات‌هایی که برنامه نویس زبان ماشین در برنامه نویسی زبان ماشین از آنها استفاده می‌کند.

invisible reg: ثبات‌هایی که داخل پردازنده هستند اما برنامه نویس زبان ماشین به واسطه زبان ماشین بلد بودن لزوما از آنها آگاه نیست، این‌ها هم زیر معماری آن پردازنده هستند

۸- انواع کلاس کال دستور: در پردازنده ها دستورات به کلاس کال مختلف تقسیم می‌شوند

۱- کلاس دستورات محاسباتی و منطقی

۲- " " دسترسی به حافظه: اینها دستوراتی هستند که سراغ RAM می‌روند مثل load store

۳- " " I/O: دستوراتی که برای دسترسی به وسایل جانبی استفاده می‌شود

۴- " " کنترلی: دستوراتی که flow برنامه را کنترل می‌کنه، که شامل دستورات پرش مشروط یا پرش غیر مشروط

ثلثه- کامپیوترهای Dedicated معماری پردازنده ندارند ولی زیر معماری دارند

معماری کامپیوتر

رامین رضوی

انواع معماری پردازنده ها : انواع زیادی وجود دارد ولی ۴ نوع را در درس معماری می خوانیم

* این دسته بندی بر اساس اینکه دستورات محاسباتی آن کامپیوتر چند عملوند است انجام می شود

۱ـ کامپیوترهای بدون عملوند (صفر عملوند ، صفر آدرسه ، بدون آدرس ، کامپیوترهای پشته ای)

۲ـ " " تک عملوند (تک آدرسه یا کامپیوترهای بتنی بر انباره (Ac) یا کامپیوترهای انباره ای)

۳ـ " " دو " (دو آدرسه یا CISC) : مثل 8086

۴ـ " " " (" " " RISC) : مثل پردازنده SPARC

معماری پردازنده مقیاس پذیر = scalable processor architecture

۱ـ پردازنده های بدون عملوند (پشته ای) : در کامپیوترهای پشته ای ثبات های عمومی (ثبات های عمومی زیر مجموعه ای از ثبات ها visible هستند که برای کارهای محاسباتی استفاده می شوند) باهم دیگر تشکیل پشته می دهند ! در کامپیوترهای پشته ای این ثبات ها هیچ اسمی ندارند)

ADD → ۲ تعر باالی پشته را pop می کند ، آن دو را باهم جمع می کند وحاصل را در پشته push می کند

SUB →

push [۲۰۰] = از آدرس ۲۰۰ حافظه می خواند و داده را باالی پشته push می کند

مثال - برنامه ای برای ماشین های پشته ای بنویسید که عبارت روبه رو را محاسبه کند

X = (A+B) / (C-D)

* برای انجام این کار ابتدا باید عبارت ریاضی را به فرم پسوندی (معکوس لهستانی RPN) (Reverse polish notation) تبدیل کنیم سپس از سمت چپ عبارت تبدیل شده شروع می کنیم و اگر عملوند دیدیم آن را push می کنیم و اگر عملگر دیدیم ، عملگر را می نویسیم

معماری کامپیوتر

رامین رضوی

ابتدا باید عبارت را اولویت گذاری کنیم

$X = (A+B) / (C-D)$

۱ ۲ ۳

۱- ( ) : از چپ به راست

۲- منفی و مثبت : " " " " : عملگرهای تک عملوندی اند + A + B -

۳- توان : اولویت اش از راست به چپ است .

۴- ضرب و تقسیم : " " چپ " راست "

۵- جمع و تفریق : " " " " " "

$postfix = ((AB)^+ (CD)^-)/$

$= AB + CD - /$

$prefix = /(+(AB)\ -(CD))$

$= / + AB - CD$

ex: $(A*B-C)/D*E+F-G$

۱ ۲ ۳ ۴ ۵

$= ((((((AB)*C)^- D)/E)*F)^+ G)^-$

$postfix = AB*C-D/E*F+G-$

ex: $(A*B-C)/D\wedge E+F\wedge G$

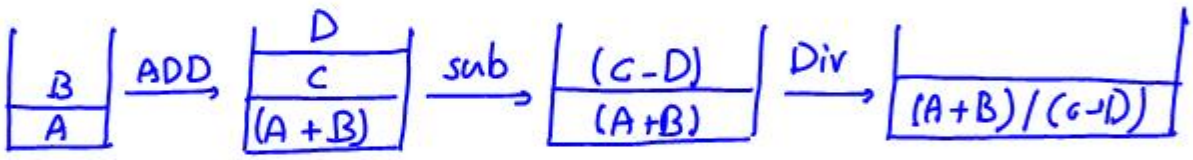
۱ ۳ ۲ ۴ ۵

$= (((((AB)*C)^- (DE)\wedge)/(FG)\wedge)^+$

$postfix = AB*C-DE\wedge/FG\wedge+$

$X = (A+B)/(C-D) \Rightarrow postfix = AB+CD-/$

```
push A
push B
ADD
push C
push D
sub
Div
pop [X]
```



* حداکثر پشته ان با اندازه ۳ می خواهیم
* چندتا push داریم = به تعداد عملوندها بیان است .

معماری کامپیوتر

رامین رضوی

توجه) در کامپیوترهای پشته ای ثبات های عمومی اصلی ندارند چون ما نمی توانیم در زبان اسمبلی یا زبان ماشین از آنها نام ببریم، بنابراین اینجا در زبان ماشین هم کدی ندارند

ADD

push [۲۳] : * همان طور که گفتیم در کامپیوترهای پشته ای فقط دستورات محاسباتی عملوند ندارند و دستورات غیر محاسباتی مانند دستورات دسترسی به حافظه می توانند عملوند داشته باشند

از آدرس ۲۳ حافظه می خوانند و در بالای پشته قرار میدهد

نکته) مزیت ماشین های بدون فیلد آرگومان آن است که دستورات محاسباتی در آنها فقط opcode دارد و بنابراین وقتی به زبان ماشین تبدیل می شوند حجم کمی می گیرند، بنابراین برنامه هایی که دستورات محاسباتی شان زیاد است سایز فایل خروجی شان کم می شود - بالای ۹۰ درصد کامپیوترهایی که در کاربردهای embedded استفاده می شود (در mp3 player، ماشین ظرف شویی، تجهیزات نظامی و ...) RAM ندارند و بنابراین نباید حجم برنامه شان زیاد شود و بنابراین از ماشین های پشته ای در این کاربردها استفاده می شود.

۲- ماشین های تک فیلد آدرسی یا ماشین های مبتنی بر Ac یا ماشین های انباره ای (z80)

در این ماشین ها ثبات های عمومی اسم دارند، در ماشین های تک عملوندی ثباتی وجود دارد که در تمامی دستورات این ماشین ها مورد استفاده قرار می گیرد، ما انسان ها به این ثبات در صحبت هایمان Ac (ثبات انباره) می گوییم ولی در اسمبلی (زبان ماشین) این ثبات اسمی ندارد

ADD ۲۳ : Ac ← ۲۳ + Ac

LD [۲۵] : Ac ← [۲۵]

LD B : Ac ← B

STR B : B ← Ac

STR [۲۵] : [۲۵] ← Ac

معماری کامپیوتر

رامین رضوی

مثال - برنامه اسمبلی ای برای ماشین خاص انباره ای بنویسید که عبارت زیر را محاسبه کند؟

$x = (A+B)/(C-D)$ فرض کنید در ابتدا A و B و C و D و x در حافظه هستند

* اگر دستوری مانند sub B = Ac ← Ac-B آن گاه بهتر است برنامه option → ←

را از راست به چپ بنویسیم، چون در اینصورت تعداد دستورات کمتر می شود

از راست به چپ

1- LD C ; Ac ← C
2- sub D ; Ac ← Ac - D ; Ac = C-D
3- STR x ; x ← Ac
4- LD A ; Ac ← A
5- ADD B ; Ac ← Ac + B ; Ac = A+B
6- div x ; Ac ← Ac/x ; Ac = (A+B)/(C-D)
7- STR x ; x ← Ac

* به ماشین 2 عملیه آرگومانی CISC هم می گویند و به ماشین های 3 عملیه آرگومانی RISC می گویند

نکته در ماشین های 2 یا 3 عملیه آرگومانی در اکثر اوقات می گویند که حداکثر فقط یکی از آرگومان ها مجاز است حافظه باشد، چون حافظه کندترین جای سیستم است.

CISC = complex instruction set computer : یعنی کامپیوتری که مجموعه دستورات زبان ماشین اش خیلی زیاد است

RISC = Reduce " " "

- در ابتدا فقط کامپیترهای CISC بودند و سپس کامپیترهای RISC آمدند

- در CISC دستورات محاسباتی دو عملوندی است مثلا در 8086 داریم ADD Ax,Bx و در RISC دستورات محاسباتی سه عملوندی هستند (ADD R_1, R_2, R_3)

معماری کامپیوتر

رامین رضوی

* ایده در RISC این بود که پردازنده شون خیلی ساده باشد که در اینصورت خیلی از دستورات را پشتیبانی نخواهند داشت ولی همان دستورات کم کامل باشد، در نتیجه ظرفیت خیلی زیاد خالی می ماند که از این ظرفیت برای افزایش سرعت استفاده کردند با ایده هایی مثل pipeline، شل بزرگ کردن کش و ...

فرق اصلی RISC و CISC : CISC با افزایش ظرفیت دستورات اسمبلی را زیاد می کند ولی ریسک پردازنده را ساده می کند و حدود ۱۰ درصد از فضا را صرف پیاده سازی پردازنده می کند

فرق ها RISC و CISC

۱- سایز کش در ریسک بیشتر است از CISC

۲- ریسک از پایپلاین استفاده می کند، اما در CISC چون پردازنده هم فضا را گرفته بود جایی برای گذاشتن پایپلاین نداشت

۳- تعداد ثبات های عمومی در ریسک بیشتر است (در حدود چند صد تاست در ریسک در صورتی که در سیسک در حدود چند ده تاست)

۴- در ریسک چون پردازنده خیلی از کارها را بصورت مستقیم انجام نمی دهد بنابراین کارهای کامپایلر بیشتر شده.

۵- پریود کلاک در ریسک کمتر هست ← فرکانس کار ریسک ها بالاتر است

نکته) سخت افزار پیچیده تر ← طبقات بیشتر ← مسیر بحرانی بزرگتر ← پریود کلاک بزرگتر

۶- تعداد کلاک های ریسک کمتر است ← چون دستورات ساده اند ← در ۱ پالس معمولا انجام می شوند

۷- طول دستورات زبان ماشین در RISC ثابت است ← چون اگر طول متغیر باشد پردازنده برای مفسیدن اش نیاز به سخت افزار اضافه دارد

۸- متوسط تعداد کلاک مورد نیاز برای اجرا یک دستور = cpI = clock per instruction

cpI در CISC از RISC بیشتر است . cpI در RISC ها تقریبا ۱ است اما cpI در CISC حدود ۲۰ و ... است

۹- زمان اجرا یک دستور در RISC کمتر است زمان اجرا یک دستور = cpI × T

۱۰- طول برنامه در RISC بیشتر است

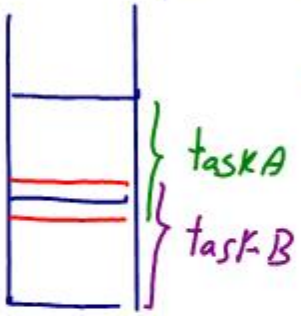
۱۱- تعداد دستورات زبان اسمبلی در CISC از RISC بیشتر است

۱۲- مدهای آدرس دهی در CISC بیشتر است .

۱۳- ساختن کامپایلر در RISC سخت تر است

۱۴- معمولا طول op در RISC ها کمتر است

۱۵- در کامپیوترهای RISC پنجره ثبات داریم ولی در CISC ها چنین چیزی نداریم



* در ماشین های RISC برای context switching با استفاده از ثبات های عمومی خیلی زیادی که وجود دارد صورت می گیرد و به هر task پنجره ثبات اختصاص خودش را می دهند و علاوه بر این بین task ها یی که با هم نیاز اطلاعات می کنند چند ثبات مشترک می گذارند .

* خیلی از ثبات ها در RISC ها مشترک اند ولی خیلی از ثبات ها در RISC ها اختصاصی اند در نتیجه وقتی بین تسک ها سوئیچ می شود نیازی نیست که مقدار این ثبات ها را بریزیم و در حافظه ذخیره کنیم

۱۶- طول دستورات ثابت باعث می‌شود بیت‌های unused در برخی از دستورات زبان ماشین ظاهر بشود و همین طور به کل حافظه دسترسی مستقیم نداریم در رجیسترها

انواع مدهای آدرس دهی رایج

۱- بلافصل، بلاواسطه (immediate) : خود عددی که قرار است روی آن محاسبه انجام بشود بخشی آرگومان است.

ADD R_1, ۲۳ ; $R_1 \leftarrow R_1 + ۲۳$

↑
مد ثباتی R_1

۲- ثبات (Register) : داده در ثبات است و ما کد آن ثبات را بعنوان بخشی از آرگومان داریم

۳- ضمنی (implicit) : داده شما در یک جایی قرار دارد و ما درون دستورمان نمی‌گوییم آن داده کجاست ولی پردازنده خودش می‌داند آن داده کجاست

ADD B ; $AC \leftarrow AC + B$

در این مثال دو تا مد آدرس دهی وجود دارد، مد آدرس دهی B ثباتی است و مد آدرس دهی Ac ضمنی است. Ac بصورت ضمنی آدرس دهی شده است

مثال ۲، این دستور دو تا آرگومان دارد، یکی اندازه است که ضمنی است و یکی ۱۰۰ است که بلافصل است.

load ۱۰۰ ; $AC \leftarrow 100$

نکته) در ماشین های پشته بر، Ac در همه دستورات محاسباتی شرکت می‌کند، برای همین ثبات Ac همراه با Alu با هم پیاده سازی می‌شوند. این مورد را بعدا مشاهده می‌کنید

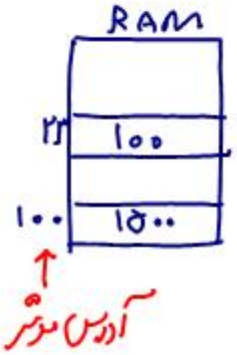
۴- مستقیم (Direct) : داده در حافظه (RAM) است و در دستور آدرس آن خانه حافظه که دنیای ما در آن قرار دارد را قرار می‌دهیم

مثال -

ADD Ax, [۲۳] ; Ax ← Ax + m[۲۳]

۵ - غیر مستقیم (indirect)

ADD R1, [[۲۳]] ; R1 ← R1 + m[m[۲۳]]



R1 = ۵۰

آدرس، آدرس داده است

توجه) این مد در کامپایلر پردازنده ها استفاده نمی شود

چون این مد دو بار سراغ حافظه می رود و حافظه کند ترین

موجود سیستم است ← بنابراین گفتند بیاید مدی را بسازیم که هم سریعتر باشد و

هم به pointer ها بدهد

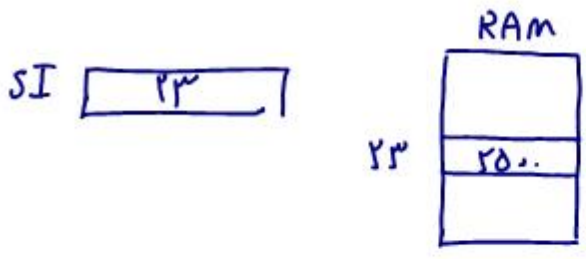
نکته) آن قدر حافظه بد است که در برنامه نویسی مفهومی وجود دارد به نام recomputation، که به این معناست که ما قبلا داده ای را محاسبه کردیم و در حافظه است ولی چون حافظه خیلی کندتر از این هست که مجددا محاسبه اش کنیم، می رویم و دوباره محاسبه اش می کنیم

۶ - غیر مستقیم ثباتی (Register indirect)

ADD Ax, [SI] ; Ax ← Ax + m[SI]

SI یکی ثبات است و دسترسی به آن بسیار سریع است

حلول می کند



- در اینجا SI یک پوینتر است که دارد به خانه ای از حافظه که داده ما درون آن است point می کند

- مدهای دیگری داریم که نسخه های متفاوتی از Register indirect هستند

۷- Base indirect : در اسمبلی به ۲ شکل از این مد استفاده می شود ولی که زبان ماشین

هر دو یکسان است.

ADD Ax, ۲۳[SI] ⇒ ADD Ax, Base[index]

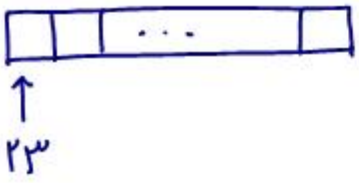
ADD Ax, [۲۳+SI]

آدرس مؤثر = آدرس پایه + فیلد آدرس

ثابت (Base) ← ثبات پایه

تغییر (index) ← فیلد آدرس

Base index



* این روش برای پیاده سازی آرایه ها استفاده می‌شود به این شکل که آدرس شروع آرایه را در Base نگه دارند

۸- Register-Base-index

ADD Ax, ۲۳[Bp,SI]

ADD Ax, [۲۳+Bp+SI]

۹- Auto decrement و Auto increment: خود افزاینده و خود کاهنده

هرکدام از این مدها دو تا وزن دارند ۱- post increment ۲- pre increment

۱. Auto increment

pre increment

ADD R1, ++[R2]

R2 ← R2+1

R1 ← R1+m[R2]

post increment

ADD R1, [R2]++

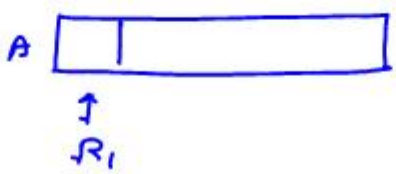
R1 ← R1+m[R2]

R2 ← R2+1

توجه) این مد نیز به درد کار با آرایه ها می‌خورد، این مد به درد دسترسی به خانه‌های توالی حافظه می‌خورد

مثلاً می‌خواهیم اعداد درون یک آرایه را جمع بزنیم

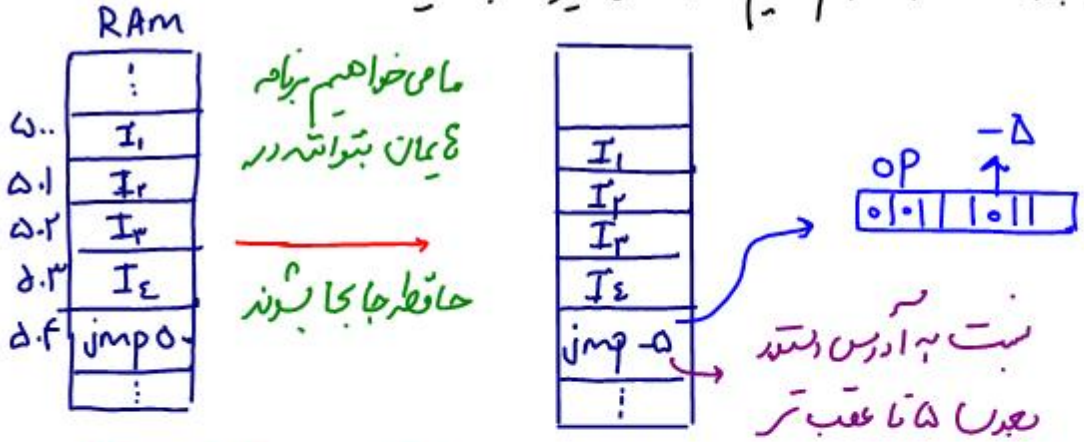
```
Ac=0;
for(i=1 to 100)
    ADD [R1]++;
```



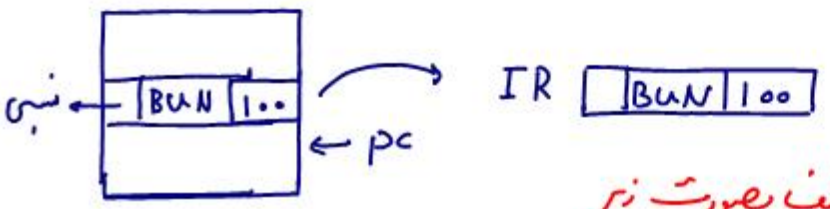
در پایان جمع اعضا آرایه در Ac است

۱۰- مد آدرس دهی نسبی (Relative)

مدهای آدرس دهی که تا الان بررسی کردیم همگی برای آدرس دهی داده استفاده می شدند ولی مد آدرس دهی Relative برای آدرس دهی دستورات استفاده می شود، در دستورات شرطی نیاز داریم که بتوانیم آدرس دهی مقصد را بصورت نسبی انجام دهیم، به شکل زیر توجه کنید



آدرس مؤثر = آدرس فیلد آدرس + pc



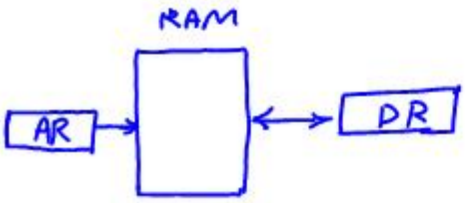
- در برخی از کتاب ها نماد اسمبلی مدهای مختلف بصورت زیر معرفی شده است.

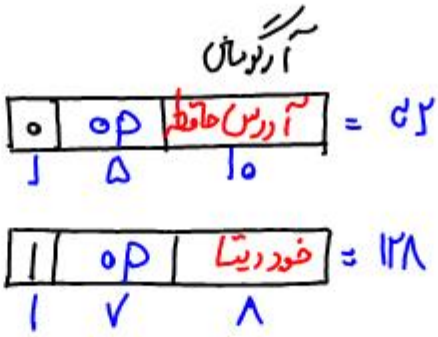
نام مد	نمادش
مستقیم	ADD ۲۰۰
غیرمستقیم	ADD @۲۰۰
ثباتی	ADD R_1
ثباتی غیرمستقیم	ADD [R_1] OR ADD (R_1)
آنی = بلافصل	ADD #۲۰۰ OR ADD i۲۰۰
نسبی	ADD $۲۰۰

معماری کامپیوتر

رامین رضوی

تست، مهندسی کامپیوتر ۸۶ : در یک کامپیوتر یک آدرس دو شیوه نشانی‌دهی مستقیم و بلافاصله (immediate) استفاده می‌شود. طول ثبات ACC هشت بیتی است، طول ثبات IR شانزده بیت و طول ثبات MAR ده بیت می‌باشد، حداکثر تعداد دستورات ماشین چند عدد می‌تواند باشد؟

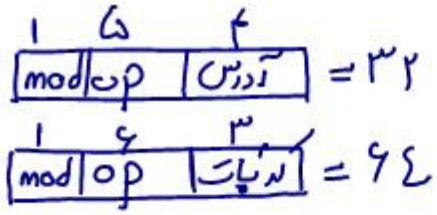




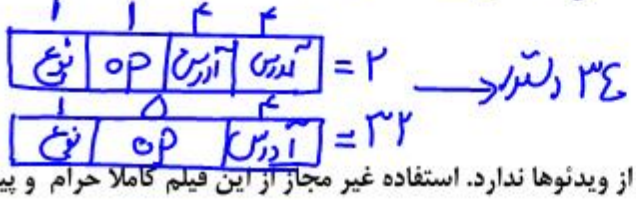
۱- ۶۴
۲- ۱۶۰
۳- ۲۵۶
۴- ۳۱۲

- وقتی گفته ACC هشت بیتی است یعنی در این کامپیوتر داده ۸ بیتی هستند

تست - در یک ماشین حافظه ۱۶×۱۰ است، دستورات یک قیلد آرگومانی هستند، دستورات یک کلمه‌ای هستند، دو تا مد داریم در این ماشین، مستقیم حافظه‌ای و مستقیم ثباتی، ۸ تا ثبات در این ماشین وجود دارد، این ماشین حداکثر چند دستور می‌تواند داشته باشد؟



مثال - یک ماشین حافظه‌اش ۱۶×۱۰ است، دستورات این ماشین یک کلمه‌ای اند، ۲ نوع دستور در این ماشین وجود دارد، دستورات یک قیلد آدرسی، دستورات ۲ قیلد آدرسی و مد آدرس دهی وجود ندارد، قیلدی وجود دارد که نوع دستور را مشخص می‌کند، این ماشین حداکثر چند دستور دارد؟

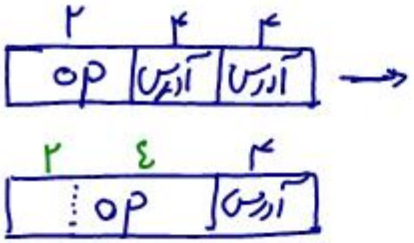


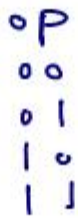
→ ۳۴ دستور

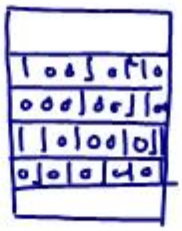
معماری کامپیوتر

رامین رضوی

مثال - در یک ماشین حافظه 16×10 است، دستورات یک کلمه‌ای هستند، دو نوع دستور داریم یک فیلد آدرسی و دو فیلد آدرسی ولی هیچ فیلدی نداریم که نوع دستور را مشخص کند، اگر این ماشین ۱ تا ۲ دستور ۲ فیلد آدرسی داشته باشد چند تا دستور تک فیلد آدرسی دارد؟ ۳ ۴







* اگر n دستور ۲ فیلد آدرسی داشته باشیم چند دستور تک فیلد آدرسی داریم: $(4-n) \times 2^4$

فرض ۱: $1 + 3 \times 16 = 49$

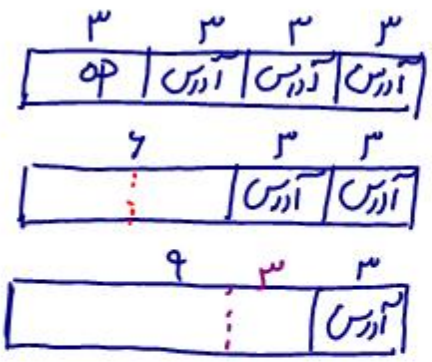
فرض ۲: $2 + 2 \times 16 = 34$

۳ ن: $3 + 1 \times 16 = 19$

۴ ن: $4 + 0 = 4$

مثال - ماشینی داریم که ۳ نوع دستور دارد، ۳ فیلد آدرسی، ۲ فیلد آدرسی و یک فیلد آدرسی، n تا دستور ۳ فیلد آدرسی داریم، m تا دستور دو فیلد آدرسی داریم، چند تا دستور تک فیلد آدرسی می‌توان داشت؟

در این ماشین طول IR را ۱۲ بیت و طول AR را ۳ بیت بگیرید



۱ - تعداد حالات باقیمانده از opcode دستورات ۳ فیلد آدرسی برای تشخیص دستورات ۲ فیلد آدرسی

۲ - تعداد کل دستورات ۲ فیلد آدرسی که می‌تواند وجود داشته باشد

۳ - تعداد حالات باقیمانده از opcode دستورات ۲ فیلد آدرسی برای تشخیص دستورات تک فیلد آدرسی

$$\left(\left(\underbrace{(2^3 - n)}_{1} \times 2^3\right) - m\right) \times 2^3$$

معماری کامپیوتر

رامین رضوی

نکته) در اکثر ماشین های RISC هیچ دستوری بجز load و store نمی‌تواند عملوند حافظه‌ای داشته باشد

نمونه واقعی از این ماشین ها، ماشین MIPS است

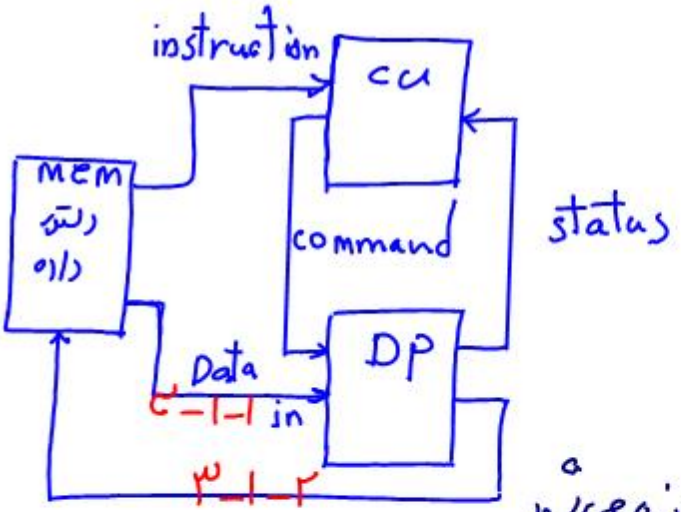
مثال - برنامه‌ای برای ماشین های RISC بنویسید که عبارت زیر را محاسبه کند $X=(A+B)/(C-D)$

(با فرض گفته شده در نکته بالا) (X و A و B و C و D همگی در حافظه‌اند)

```
LD  R1,A
LD  R2,B
LD  R3,C
LD  R4,D
ADD R1,R1,R2
sub R3,R3,R4
Div R1,R1,R3
STR X,R1
```

- ما می‌خواهیم پردازنده بسازیم، بنابراین باید که RTL برایش بنویسیم، پس ما باید مشخص کنیم که RTL چه الگوریتمی را می‌خواهیم بنویسیم.
حال کار پردازنده اجرا برنامه است، حال یک پردازنده که می‌خواهد یک برنامه (برنامه) را اجرا بکند طبق یک الگوریتمی این کار را انجام میدهد که اسم این الگوریتم (Nuemann) است

بررسی دقیق الگوریتم نیومن



این الگوریتم شش گام دارد.

نکته) تقریبا یک دستوری که می‌خواهد اجرا شود همه شش مرحله را طی می‌کند

۱- fetch : یک دستور از حافظه خوانده می‌شود و به واحد کنترل منتقل می‌شود و معمولا در شتابی بنام IR نگه‌داشته می‌شود

توجه) به خواندن داده از حافظه fetch نمی‌گویند

معماری کامپیوتر

رامین رضوی

۲- Decode : تولید سیگنال های command توسط واحد کنترل با استفاده از

۱- سیگنال های status

۲- sequencer

۳- دستوری که fetch شده و در ثبات IR است

۲-۱ : operand fetch : خواندن داده از ثبات های visible می گوییم OF را

ما داریم عدد ۱۰ را از ثبات IR می خوانیم

مثال = ADD R_1, R_2

ADD $R_1, 10$

نکته) در کامپیوتر نیومن نمی توان همزمان هم محاسبات انجام داد و هم به حافظه دسترسی داشت

چنین دستوری نداریم چون این دستور هم جمع انجام میدهد و هم می خواهد سراغ حافظه برود

ADD $R_1, [23]$ ×

۳- execute : انجام محاسبات روی داده هایی که قبلا OF شده است

۳-۱ : memory access : نیومن معتقد بود که اگر دستورات محاسباتی باشند ما execute داریم و گفته دستورات حافظه دار باشند ما MA داریم و این دو با هم اتفاق نمی افتند

memory access یعنی خواندن داده از حافظه (۳-۱-۱) یا نوشتن داده در حافظه (۳-۱-۲) ← فقط ۱ مورد رخ می دهد

۴- write back : نوشتن نتیجه دستور در ثبات (ثبات های قابل مشاهده)

توجه) از حافظه خواندن و در حافظه نوشتن OF و WB نیست

نکته - همه دستورات Fetch و Decode را دارند

سوال - در هریک از دستورات زیر مشخص کنید که از چه مراحلی از الگوریتم نیومن استفاده می کنند

	FA	Dec	oF	exe	MA	WB
ADD R_1, R_2	✓	✓	✓	✓	X	✓*
LD $R_1, [R_2]$	✓	✓	X	X	✓	✓
STR $R_1, [R_2]$	✓	✓	✓	X	✓	X
cmp R_1, R_2	✓	✓	✓	✓	X	✓*

* وقتی حاصل در R_1 نوشته می‌شود همزمان نیز فلگ ها در psw ، WB می‌شود ← اسم این هم WB می‌شود چرا psw نیز یک ثبات قابل مشاهده توسط برنامه‌نویس زبان ماشین است

سوال) دستور cmp چیکار می‌کند ؟ cmp عینا مانند sub است و تنها تفاوت sub با cmp در این است که sub در psw و R_1 می‌نویسد ولی cmp فقط در psw ، WB می‌کند و R_1 را تغییری نمی دهد

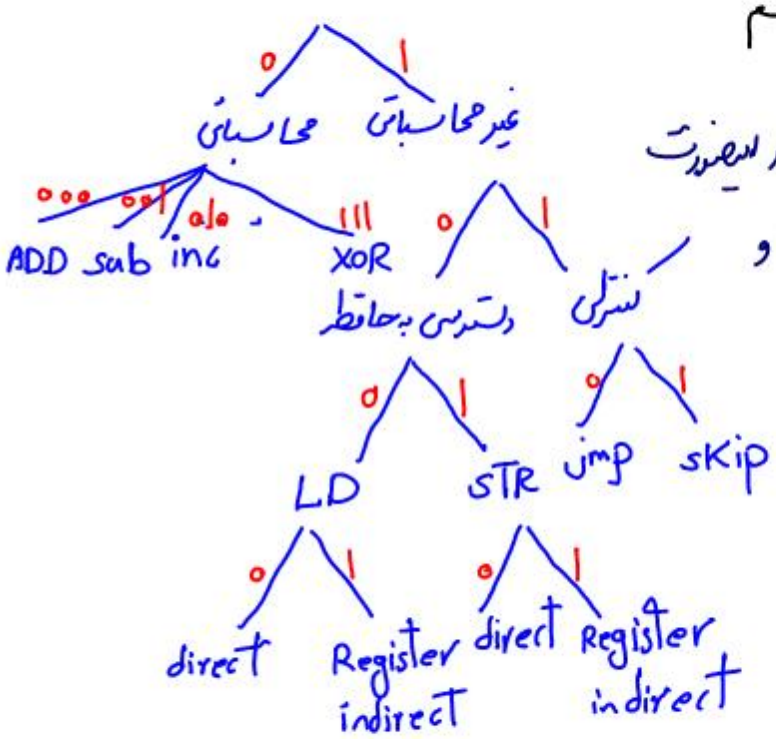
- برای طراحی یک پردازنده باید علاوه بر اینکه الگوریتمی که پردازنده بر اساس آن کار می‌کند را بدانیم ، معماری پردازنده یا همان ISA (instruction set architecture) پردازنده را بدانیم . و ما با دانستن زبان ماشین یک پردازنده متوجه معماری آن پردازنده اش می‌شویم

می خواهیم پردازنده خودمان را بسازیم ، لذا بر این ما مشخص می‌کنیم که چه نوع دستوراتی داشته باشیم ، طول دستورات چقدر باشند ، چند قیلد آدرسی باشند هر دستور و ...

معماری کامپیوتر

رامین رضوی

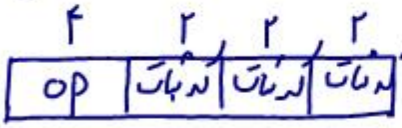
می‌خواهم ۲ کلاس دستور بصورت زیر داشته باشم



- فرض می‌کنیم طول دستور ۱۰ بیت باشد. در اینصورت در دستورات محاسباتی و LD و STR ۶ بیت و در دستورات jmp و skip، ۷ بیت از IR باقی می‌ماند

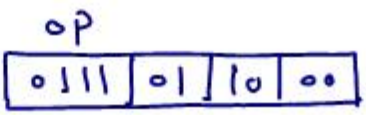
- فرض می‌کنیم در این کامپیوتر ۴ ثبات عمومی داریم و اسم اینها را R_0 تا R_3 و کد اینها را ۰۰ تا ۱۱ می‌گذاریم.

- ما دوست داریم ۸ دستور محاسباتی داشته باشیم و می‌خواهیم دستورات محاسباتی ۳ آدرسه باشند و این دستورات فقط دارای مد آدرس دهی ثباتی باشند



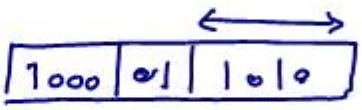
مثال - دستور زیر را اسمبل اش کنید؟ معادل زبان ماشین اش را بنویسید

XOR R_1, R_2, R_0 : $R_1 \leftarrow R_2 \oplus R_0$ ⇒



دستورات دسترسی به حافظه: دستورات دسترسی به حافظه‌مان می‌خواهیم مثلا بصورت زیر باشد

LD $R_1, [10]$: $R_1 \leftarrow M[10]$ ⇒



سوال مهم: حافظه در این ماشین چند در چند است؟ حافظه ۱۰۲۴ × ۱۶ بیت دارد می‌تواند هر طولی داشته باشد، منتهی ما فقط می‌توانیم به ۱۶ خانه اول این حافظه بصورت Direct دسترسی داشته باشیم، حال اگر خانه‌های دیگر حافظه را می‌خواهید باید بصورت indirect بروید سراغ شون

دستور دسترسی به حافظه غیرمستقیم: LD R1, [R2] : R1 ← M[R2]

زبان ماشین: | 1001 | 01 | 10 | xx |

↓ unused

دستورات کنترلی:

jmp: برای مشخص کردن دستور پرش فقط ۳ بیت سمت چپ دستور کافی است، بنابراین از ۷ بیت باقی مانده برای آدرس استفاده می‌کنیم، آدرسمان را نیز relative در نظر می‌گیریم به این علت که دوست داریم برنامه‌هایمان Relocatable باشند، می‌دانیم که آدرس relative باید علامت دار باشد، بنابراین مقدار صحیح آدرس در این دستورات می‌تواند در بازه ۶۴- تا ۶۳+ باشد، بنابراین max فاصله پرش ۶۴ خانه عقب‌تر و ۶۳ خانه جلوتر است.

پرش
- شرطی: skip ← Branch
- غیرشرطی: go to، jmp

skip: دستور skip یک نوع Branch ای است که فقط شرط دارند و آدرس مقصد ندارند حال اگر شرط جلو skip درست باشد از روی دستور بعدی می‌پریم و دستور بعدش اجرا می‌شود ولی اگر شرط جلو skip غلط باشد دستور بعدی skip اجرا می‌شود، دستور skip بعنوان شرط یکی از فلگ‌ها status را چک می‌کند.

معماری کامپیوتر

رامین رضوی

نکته - در کامپیوترهای Dedicated سیگنال‌های status مودر مختلفی می‌توانند بالا و پستی به کامپیوتری که ما می‌سازیم دارد، اما در کامپیوترهای multi perpose سیگنال‌های status همگی سیگنال‌های شعوری هستند و تقریبا در همه پردازنده‌ها یکسان هستند، این سیگنال‌ها همان فلگ‌های sign, carry, zero, overflow و ... حال هرکدام از این‌ها یکی از خانه‌های ثباتی بنام psw است (The program status word)، این ثبات در DP قرار دارد و سیگنال‌ها، وضعیت از psw خارج می‌شوند و می‌روند توی cu. ثبات psw جزء ثبات‌های visible است بنابراین برنامه نویس زبان ماشین از وجود این ثبات آگاه است و از فلگ‌های درون این ثبات استفاده می‌کند

برو و فلگ گیری ثبات psw را ببین و اگر 1 بود بپر به L1 و گرنه برو نپر : jc L1

* حال skip چگونه هریک از بیت‌های status را چک می‌کند؟

اگر بخواهیم 4 فلگ c یا z یا v یا s را در شرط skip بررسی کنیم، 5 بیت در نظر می‌گیریم که 4 بیت اول مشخص می‌کند که چه فلگی را می‌خواهیم چک کنیم، بیت 5ام مشخص می‌کند که اگر آن فلگ چی باشد می‌خواهیم بپریم

مثال -

czvs r

skip 0100 0 : اگر z=0 باشد دستور بعدی skip می‌شود

skip 0100 1 : " " " " " z=1 "

α_x : تعیین کننده این است که آیا x باید چک بشود یا خیر و اگر $\alpha_x=1$ باشد یعنی می‌خواهیم x را چک کنیم و می‌خواهیم اگر x برابر r باشد بپرد

$\alpha_c c+\alpha_z z+\alpha_v v+\alpha_s s==r$

- هر وقت این تساوی برقرار بود skip می‌پرد

skip z==0; $\xrightarrow{\text{السمبلر}}$ 111 0100 0 xx

unused

نکته) ما وقتی می خواهیم که RTL بزنیم از TSG استفاده می کنیم به ۲ علت

۱. تعداد فلیپ فلاپ ها به نحو قابل توجهی کاهش پیدا می کند

۲_ احتمال اشتباه بسیار پایین می آید

- TSG درحقیقت یک شمارنده است که clear آشکرون دارد، اما نحوه شمارش اش با شمارنده های باینری متفاوت است. وقتی ما سیگنال clear را اعمال می کنیم، TSG ریست می شود و T_0 = ۱ می شود و سایر T_i ها ۰ می شود

- لازم نیست حتما تعداد سیگنال های TSG توانی از ۲ باشد.

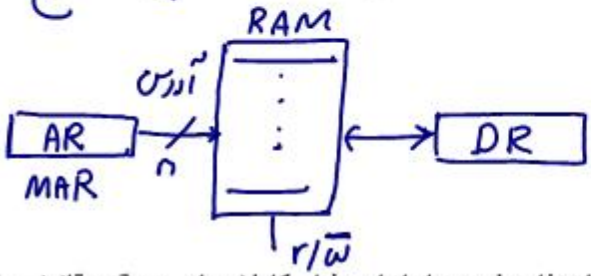
- ما با استفاده از TSG ترتیب ایجاد می کنیم بدون اینکه بخواهیم درگیر فلیپ فلاپ ها بشویم، اما دقت کنید که TSG ما را بطور کامل از فلیپ فلاپ ها بی نیاز نمی کند.

- TSG بخشی از sequencer است.

نکته - زبان ماشین flow دارد، بنابراین یک ثبات نیاز داریم که flow را برای ما نگه دارد، ما باید بدانیم در هر لحظه از زبان چه دستوری از برنامه زبان ماشین که در حافظه است باید اجرا شود. برای نگهداری flow از ثبات pc یا IP (instruction pointer) استفاده می شود.

RAM

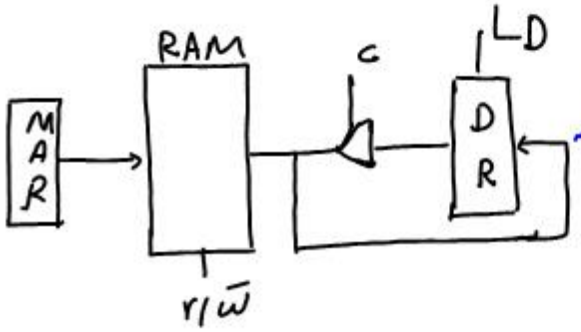
حافظه کلاک ندارد (البته حافظه هایی داریم که کلاک بخورد، اما حافظه ما در کامپیوتر نیومن از این نوع است) اما تاخیر دارد و تاخیرش هم بسیار زیاد. از دید cpu و DP که کلاک دارند وقتی می خواهند بروند سراغ حافظه باید به اندازه مثلا ۳ هزار تا کلاک خودشون معطل بشوند



معماری کامپیوتر

رامین رضوی

- در تمام زبانی که داریم در حافظه می نویسیم یا از حافظه می خوانیم باید پایه‌ها آن داده و آدرس و کنترلی ثابت باشد، در توشتن اگر در وسط کار پایه‌ای عوض شود کل حافظه آسیب می بیند، اما در هنگام خواندن فقط همان خانه خراب می شود و کل حافظه آسیب نمی بیند.



- فرض کنید تاخیر حافظه 120ns باشد و پریود کلاک 50ns باشد
- آدرس حافظه ثابت به AR وصل باشد
- دیتاش هم به DR

Fetch:

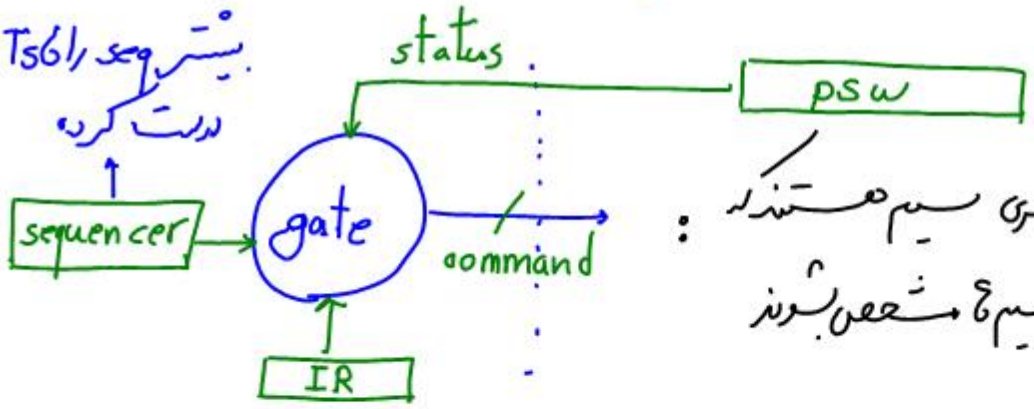
T_0: $AR \leftarrow pc$ ، $cL \leftarrow 0$

T_1: $DR \leftarrow M[AR]$ ، $pc \leftarrow pc+1$

T_2: $IR \leftarrow DR$

در گام دوم باید دقیق کنم،

ساختار واحد کنترل در کامپیوترها چند مقدوره بصورت زیر است



سیگنال‌های command یکسری سیم هستند که در مرحله دقیق باید مقادیر این سیم‌ها مشخص شوند

control unit

Data path

- زبان RTL دقیقا زبانی است که مقادیر اینکه سیم‌ها چه باشند را بصورت مستقیم مشخص می کند و اینکه مقادیر سیم‌ها چه هستند را بصورت غیر مستقیم مشخص می کند، بنابراین RTL برای دیگه نمی تواند خطی

داشته باشد .

* پس از آنکه در T_4 ، DR به IR منتقل می‌شود ، در T_5 دیکد انجام شده است و سیگنال‌های command مقدارشان تعیین شده است

دستورات محاسباتی

دستور موقع fetch وارد ثبات IR می‌شود و ثبات IR هم ۱۶ بیتی است ، حال بیت‌های این ثبات را اسم‌گذاری می‌کنیم از IR[0] تا IR[9] . حال برای اینکه ببینیم آیا دستور محاسباتی هست یا نه کافی است ببینیم IR[9] نگاه کنیم

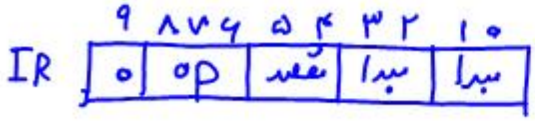
$\overline{IR[9]}\ T_5$: کار محاسباتی مربوطه انجام شود

قرار داد : ما گفتیم در این ماشین ۴ تا ثبات عمومی داریم ، قرار داد می‌کنیم که ثبات‌های عمومی را بصورت زیر نامگذاری کنیم :

$R_0 = R[0,0]$
$R_1 = R[0,1]$
$R_2 = R[1,0]$
$R_3 = R[1,1]$

XOR R_3, R_2, R_1 : $R_3 \leftarrow R_1 \oplus R_2$

می‌خواهیم بعنوان مثال RTL مربوط به دستور ADD را بنویسیم . فرض کنید op دستور ADD بصورت 0000 باشد



$\overline{IR[9]}\ T_5 : Tmp \leftarrow R[IR[3], IR[2]]$ OF

$\overline{IR[9]}\ \overline{IR[8]}\ \overline{IR[7]}\ \overline{IR[6]}\ T_6 : R[IR[0], IR[1]] \leftarrow Tmp + R[IR[1], IR[0]]$, OF

D0 | WB | exe

$psw \leftarrow$ Alu flags, $cL \leftarrow 1$; *

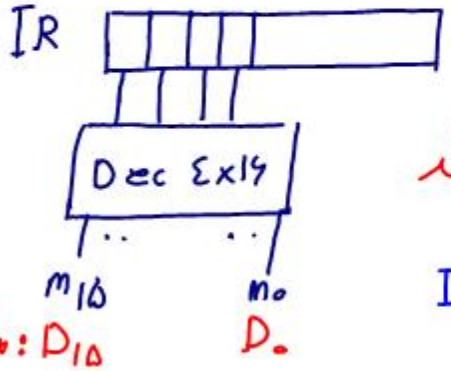
نکته) وقتی Tmp نمی‌گذاریم یک باس به جای ۲ باس خواهیم داشت ولی بها این کار را با ۱ کلاک بیشتر می‌دهیم

معماری کامپیوتر

رامین رضوی

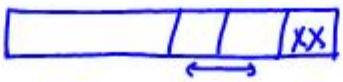
نکته) Alu ها میتونه یک خروجی داشته که سیگنال های وضعیت c و z و ... را مشخص میکنند که اسم این ها را Alu flags میگذارند. Alu flags ها را حتما باید در psw بریزیم چون دستور بعدی ممکن است بخواهد از psw استفاده کند

نکته) وقتی که RTL پردازنده را بنویسیم، مشاهده میکنید که بیشترشان چون خیلی در قسمت شرط ها ظاهر شده اند و خیلی پرکاربرد اند، به همین دلیل قسمت opcode ثبات IR را به یک دیکدر می دهند و سپس در قسمت شروط از خروجی های دیکدر استفاده می کنند، حال انتظاری نوشتن که RTL ساده تر است.



مثال: فرض کنید دستور inc نیز داریم و چون اش برابر ۰۱۰۰ باشد RTL اش را بنویسید

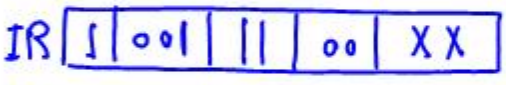
Inc R_1, R_2; $R_1 \leftarrow R_2 + 1$



$\overline{IR[9]}\,\overline{IR[8]}\,IR[7]\,\overline{IR[6]}\,T_6 : R[IR[5..3]], IR[2..0]] \leftarrow Tmp+1, psw \leftarrow Alu\ flags, cL \leftarrow 1$

دستورات دسترسی به حافظه

Load غیرمستقیم: LD $R_3, [R_0]$: از خانه ای از حافظه که آدرس اش در R_0 است می خواند و در R_3 می ریزد



$D_9 T_5$: $AR \leftarrow R[IR[4..3], IR[2..0]]$: oF

$D_9 T_8$: $DR \leftarrow m[AR]$: mA

$D_9 T_9$: $R[IR[5..3], IR[2..0]] \leftarrow DR, cL \leftarrow 1;$

wB

مثال، دستور store مستقیم: STR R_2, [13]; از ثبات R_2 می‌خواند و در خانه ۱۳ حافظه می‌گذارد

۴	۲	۴
1010	10	1101

Zero extention

$D_1.T_5$: AR ← IR[۳..۰], DR ← R[IR[۵], IR[۴]]

$D_1.T_8$: CL ← 1

M[AR] ← DR ✗ در RTL ما حق نداریم به غیر ثبات به جای دیگری مقدار بدهیم.

دستورات کنترلی

دستور jmp: jmp −۲۳

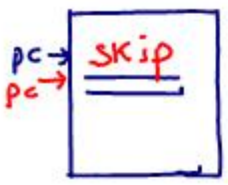
110	۷ بیت

آدرس جدید

$IR[9]\,IR[8]\,\overline{IR[7]}\,T_5$: pc ← pc + IR[۶..۰], CL ← 1

$(D_{12}+D_{13})T_5$: ——

دستور skip: skip 0100 1



111	α_6	α_5	α_4	α_3	r	xx

$IR[9]\,IR[8]\,IR[7]\,[IR[6]\cdot 6 + IR[5]\cdot 5 + IR[4]\cdot 4 + IR[3]\cdot 3 \odot IR[1]]\,T_5$: pc ← pc + 1

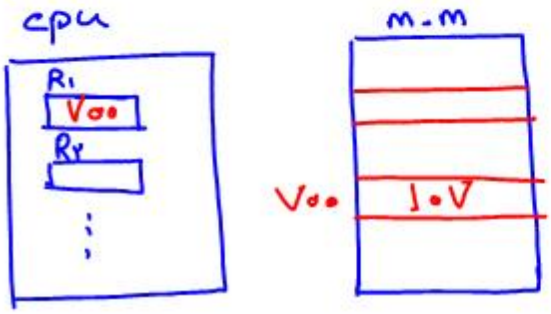
چه skip شود و چه skip نشود می‌خواهیم CL را 1 کنیم: T_5 : CL ← 1 //

۶) مد ثباتی غیر مستقیم

ثبات R_1

تباتی غیر مستقیم	ADD	۱۲۳

$Ac \leftarrow Ac + 10V$



نماد	مد آدرس دهی
ADD ۲۰۰	مستقیم
ADD @۲۰۰	غیرمستقیم
ADD R_1	ثباتی
ADD (R_2), ADD $[R_1]$	ثباتی غیرمستقیم
ADD #۲۰۰, ADD i۲۰۰	آنی
ADD $۲۰۰	نسبی
ADD $(R_1)+$	خود افزاینده (پس افزاینده

۷) خود کاهنده، خود افزاینده : مانند ثباتی غیر مستقیم اند ، با این تفاوت که مجدار ، یا قبل از یافتن عملوند ثبات افزایش یا کاهش می یابد

ADD $(R_1)+$

$Ac \leftarrow Ac + m[R_1]$

$R_1 \leftarrow R_1 + 1$ ← می تواند هر چیزی باشد

این مد به درد کار روی آرایه می خورد

۸) مد نسبی (Relative)

معماری کامپیوتر

رامین رضوی

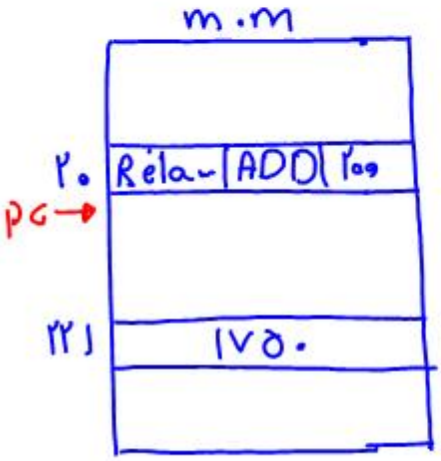
مدهایی که در ادامه می بینیم همشون بصورت زیر هستند

یک چیزی + فیلد آدرس = آدرس موثر

مد بی

آدرس موثر = فیلد آدرس + pc

AC = ۱۲۵



۱: . . . .
۲: . . . .
۲۰: ADD $۲۰۰
۲۱: . . . .
۲۲۱: ۱۷۵۰

در اثر اجرای خط ۲۰: AC ← AC + ۱۷۵۰

۹) مد شاخص دار: displacement

آدرس موثر = فیلد آدرس + ثبات شاخص (XR یا IX)

فیلد آدرس: ثابت
ثبات شاخص: متغیر

به درد کار با آرایه می خورد. که فیلد آدرس آدرس شروع آرایه باشد و ثبات شاخص اندیس آرایه باشد

۱۰) ثبات پایه:

آدرس موثر = فیلد آدرس + ثبات پایه

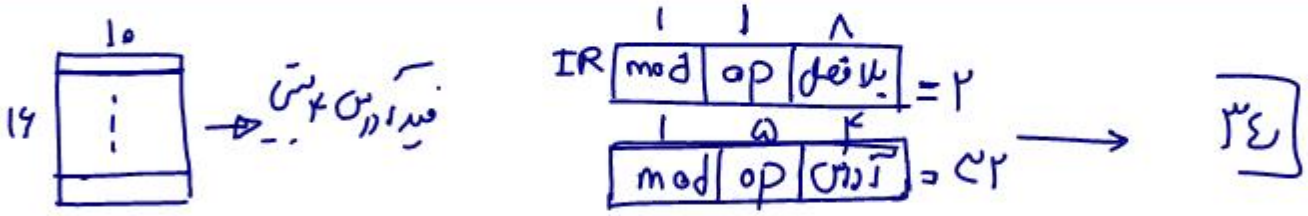
فیلد آدرس: متغیر
ثبات پایه: ثابت

این مد به درد برنامه نویس سیستم می خورد، برنامه نویس سیستم با این مد می تواند حافظه را بخش بندی کند (سگمنت درست کند). به این صورت که فیلد آدرس آفست باشد و فیلد ثبات پایه آدرس شروع سگمنت باشد.

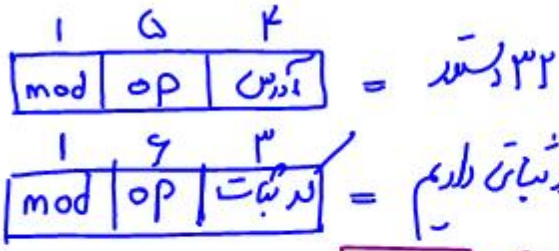
تست) در یک ماشین حافظه ۱۶×۱۰ است، دستورات ۱ فیلد آدرسی هستند، همچنین دستورات ۱ کلمه ای اند، دو مد در این ماشین وجود دارد: ۱- مد بلافصل ۲- مد مستقیم، AC هشت بیتی است حداکثر چند دستور در این ماشین داریم؟

معماری کامپیوتر

رامین رضوی



مثال) در یک ماشین حافظه ۱۶x۱۰ است، دستورات یک کلمه‌ای اند، ۸ تا ثبات وجود دارد، دو تا مد داریم، مستقیم حافظه‌ای و مستقیم ثباتی، این ماشین چند تا دستور دارد؟



ماشین پایه مانو

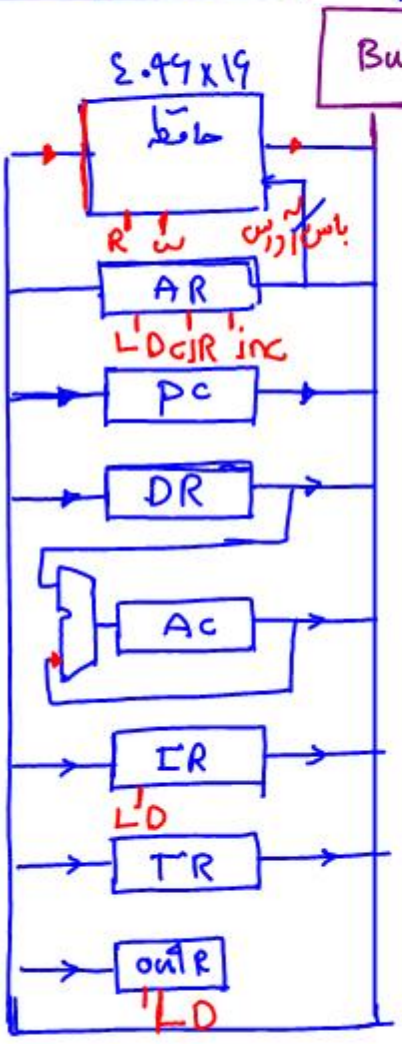
Data path این ماشین مطابق شکل زیر است.

- در این ماشین تنها کسی که می‌تواند به حافظه آدرس دهد ثبت AR است

- ۷ ثبات ۱۶ بیتی

16 Mux 8x1

$S_2 S_1 S_0$	Bus
0 0 0	nothing
0 0 1	AR
0 1 0	PC
0 1 1	DR
1 0 0	AC
1 0 1	IR
1 1 0	TR
1 1 1	mem



معماری کامپیوتر

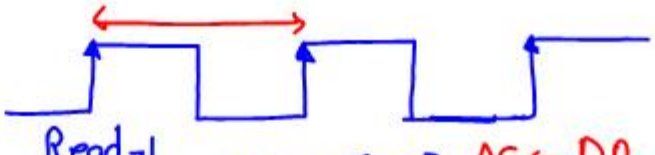
رامین رضوی

سوال - می خواهیم کلمه ای از حافظه که آدرس آن در AR است را بخوانیم و آن را در Ac قرار دهیم، چند کلاک نیاز است و چند سیگنال کنترل را باید فعال کنیم؟

$Ac \leftarrow M[AR]$

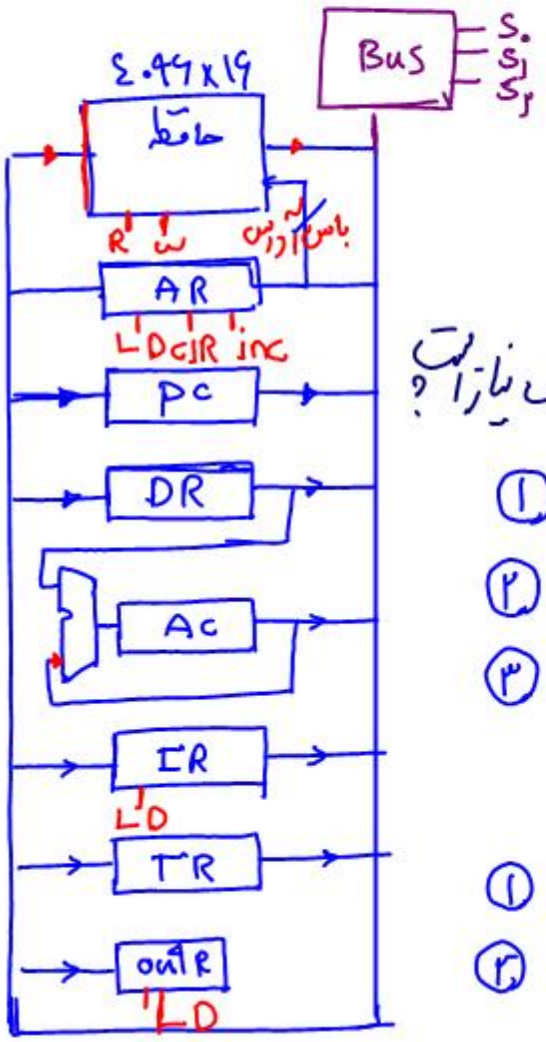
① $DR \leftarrow M[AR]$

② $Ac \leftarrow DR$



سیگنال هایی لازم برای اینکه ALU، DR را از خودش عبور بدهد مشخص می کنیم

مثال) برای انجام عملیات $IR \leftarrow m[pc]$ چند کلاک نیاز است



① $AR \leftarrow pc$

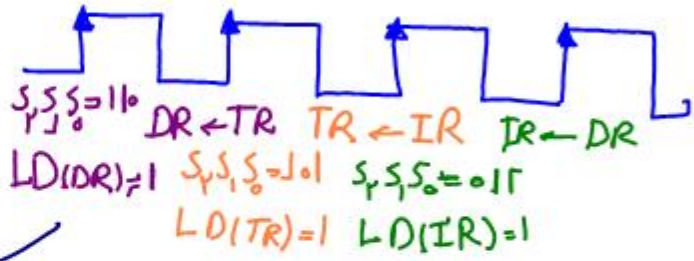
② $IR \leftarrow m[AR]$

مثال) برای جابجا کردن مقادیر ثبات های IR و TR چند کلاک نیاز است؟

① $DR \leftarrow TR$

② $TR \leftarrow IR$

③ $IR \leftarrow DR$



مثال) مقادیر دو ثبات IR و DR را جابجا کنید

① $DR \leftarrow IR$, $Ac \leftarrow DR$

② $IR \leftarrow Ac$

مثال) دو ثبات Ac و DR برای تعویض شان به چند کلاک نیاز دارند؟

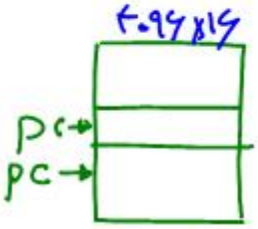
① $Ac \leftarrow DR, DR \leftarrow Ac$

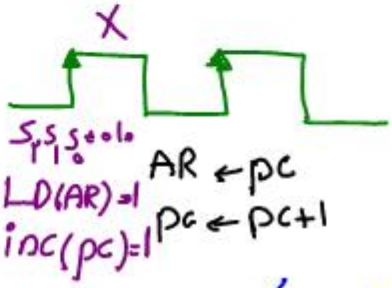
سیکل دستور: سیکل دستور یعنی مراحل اجرا دستور (سیکل خن نیومن نیز می گویند)

در این ماشین سیکل اجرا دستور بصورت زیر است: ۱) fetch ۲) دیکد ۳) بدست آوردن آدرس موثر ۴) اجرا

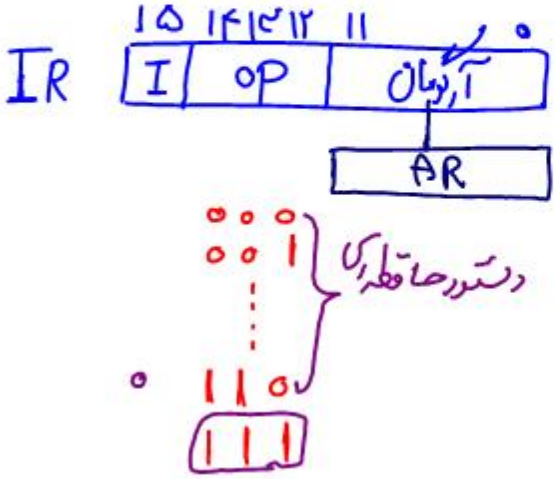
fetch (واکشی دستور): همیشه اولین قدم اجرا یک دستور در هر ماشین fetch است.

$IR \leftarrow m[pc]$
$pc \leftarrow pc+1$ → توان ای از ۲ →
① $AR \leftarrow pc$, $pc \leftarrow pc+1$
② $IR \leftarrow m[AR]$,



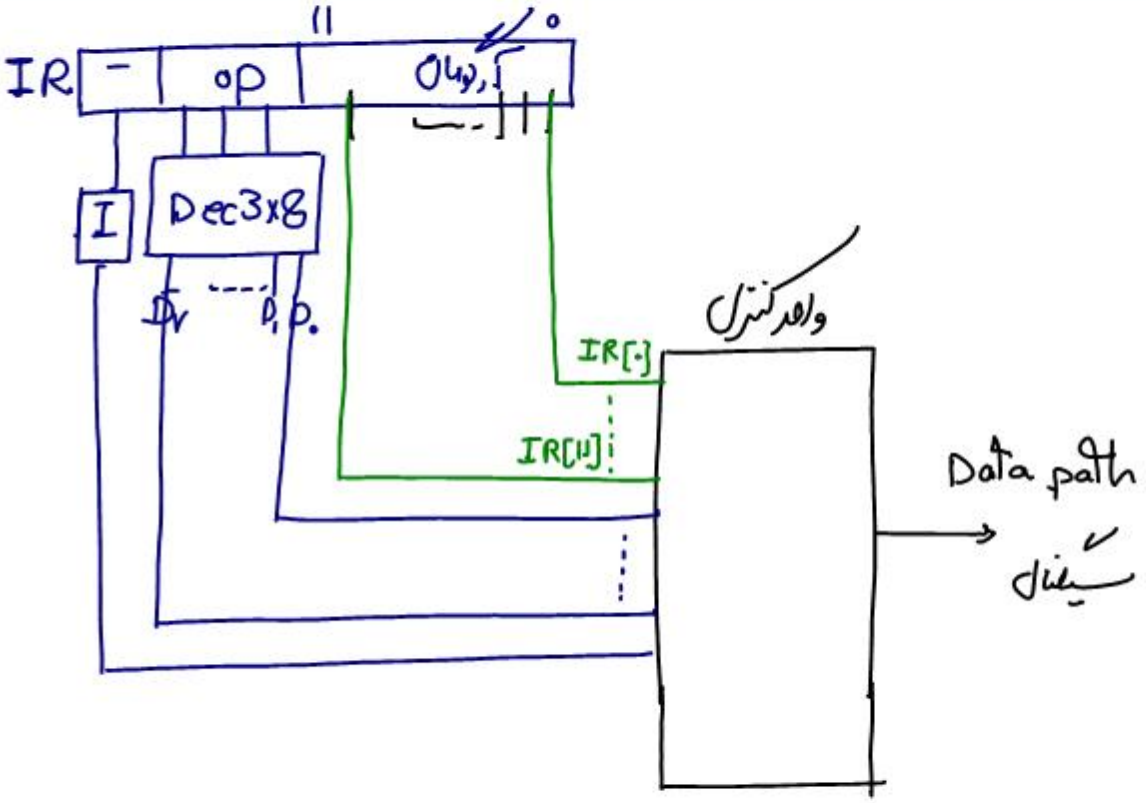


دیکد یعنی ترجمه دستور، برای اینکه بتوانیم دستور را ترجمه کنیم باید فرمت ثبات دستور العمل را بدانیم، فرمت ثبات دستور العمل در هر ماشینی مشخص است، و در این ماشین بصورت زیر است.



* ما در این ماشین ۳ نوع دستور داریم

۱ـ دستورات حافظه ای
۲ـ " ثباتی
۳ـ " I/O



- در این ماشین سه کلاس دستور وجود دارد، حافظه‌ای، ثباتی، I/O . حافظه‌ای ها op شون مخالف 111 است ($D_7 \neq 1$) بنابراین 7 تا دستور حافظه‌ای در این ماشین وجود دارد، در دستورات حافظه‌ای آدرسهان آدرس است و I مشخص می‌کند که آدرس دهی مستقیم است یا غیرمستقیم اگر $I=0$ باشد فیلد آدرس، آدرس دیتا است و اگر $I=1$ باشد فیلد آدرس، آدرس، آدرس دیتا است اگر op=111 باشد ($D_7=1$) با توجه به I مشخص می‌شود که دستور ثباتی است یا I/O است که در هر 2 صورت فیلد آدرسهان نقش op را بازی می‌کند. اگر $I=0$ باشد دستور ثباتی است و اگر $I=1$ باشد دستور I/O است. ما در این ماشین 12 تا دستور ثباتی و 7 تا دستور I/O داریم، بنابراین op دستورات ثباتی و I/O را بصورت 111 تا صفر و یکی در نظر I مشخص می‌کنیم تا دیکد راحت‌تر شود. (تا سفت اختراع و زمان دیکد بهتر شود)

۳) بدست آوردن آدرس موثر : این مرحله فقط در دستورات حافظه‌ای غیرمستقیم کاربرد دارد

$AR \leftarrow m[AR]$

ما دوست داریم قبل از اینکه وارد مرحله اجرا دستور بشویم آدرس موثرمان در AR باشد

۴) اجرا

سیکل دستور یک سیکل ترتیبی است . بنابراین باید سیگنال زمانی تولید کنیم و در این ماشین سیگنال‌های زمانی با مدار مقابل تولید می‌شوند.

می‌خواهیم سیگنال اجرا دستور را بنویسیم

fetch:

T_0 : $AR \leftarrow PC$

T_1 : $IR \leftarrow m[AR]$, $PC \leftarrow PC+1$

Decode:

T_2 : $I \leftarrow IR(15)$, $D_V \leftarrow D_0 \leftarrow Dec\ IR(12,13,14)$, $AR \leftarrow IR(0-11)$

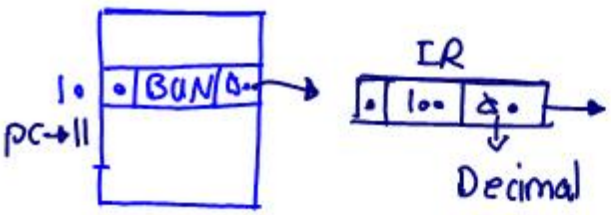
T_3 :

- $D_V=0$, $I=0$ → دستور حافظه‌ای مستقیم → nothing : $T_3 \bar{D}_V \bar{I}$: nothing
- $D_V=0$, $I=1$ → " غیرمستقیم → $AR \leftarrow m[AR]$: $T_3 \bar{D}_V I$: $AR \leftarrow m[AR]$
- $D_V=1$, $I=0$ → دستور ثباتی → اجرا دستور ثباتی : $T_3 D_V \bar{I}$: exe
- $D_V=1$, $I=1$ → I/O → I/O " " : $T_3 D_V I$: exe

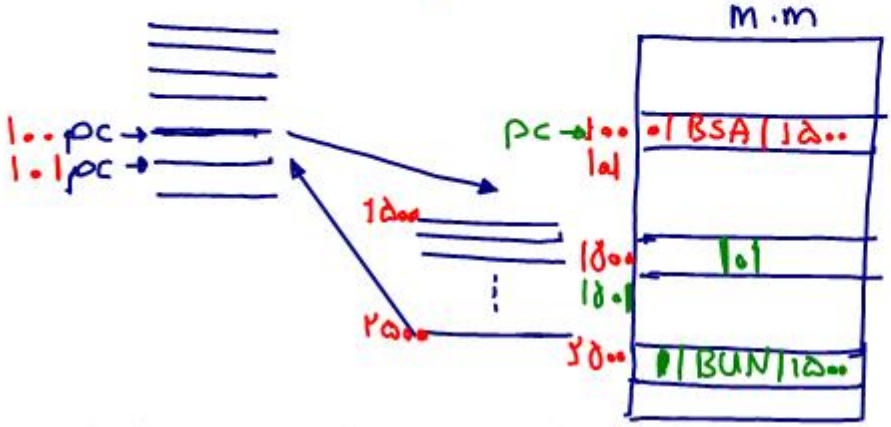
T_4 : در این ماشین هر وقت T_4 تولید نشد یعنی موقع اجرا دستور حافظه‌ای است

دستورات حافظه‌ای: همان طور که قبلتر گفتیم ۷ تا دستور حافظه‌ای داریم =

	op	نام دستور	توضیحات	
۱)	۰۰۰	AND	$AC \leftarrow AC \wedge m[AR]$ ⇒	$D_0T_4: DR \leftarrow m[AR]$ $D_0T_5: AC \leftarrow DR \wedge AC, SC \leftarrow 0$
۲)	۰۰۱	ADD	$AC \leftarrow AC + m[AR], E \leftarrow C_{out}$:	$D_1T_4: DR \leftarrow m[AR]$ $D_1T_5: AC \leftarrow AC + DR, SC \leftarrow 0$ $, E \leftarrow C_{out}$
۳)	۰۱۰	LDA	$AC \leftarrow m[AR]$: (load to Ac) :	$D_2T_4: DR \leftarrow m[AR]$ $D_2T_5: AC \leftarrow DR, SC \leftarrow 0$
۴)	۰۱۱	STA	$m[AR] \leftarrow AC$: (store Ac) :	$D_3T_4: m[AR] \leftarrow AC, SC \leftarrow 0;$
۵)	۱۰۰	BUN	$pc \leftarrow AR$: (Branch unconditional) :	$D_4T_4: pc \leftarrow AR$ $, SC \leftarrow 0$



۶) ۱۰۱ BSA → $M[AR] \leftarrow pc$, $pc \leftarrow AR+1$ Branch and save return address



$D_5T_5: m[AR] \leftarrow pc, AR \leftarrow AR+1$

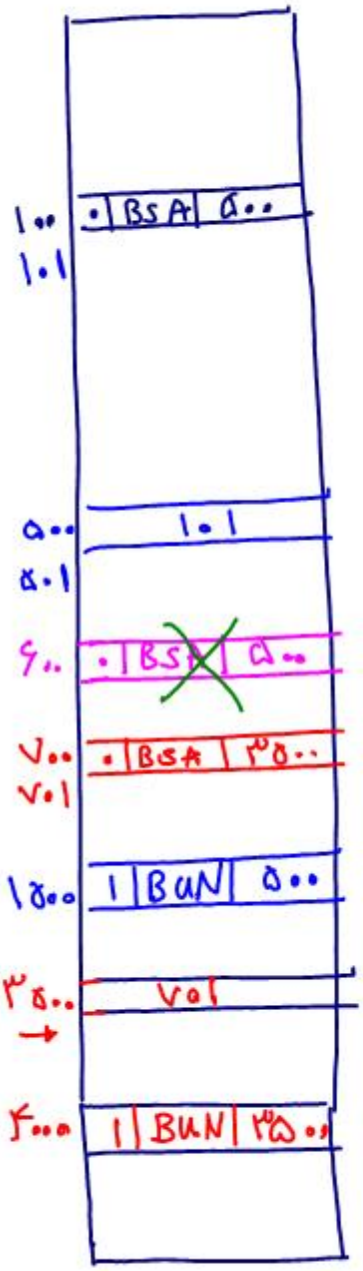
$D_5T_5: pc \leftarrow AR, SC \leftarrow 0$

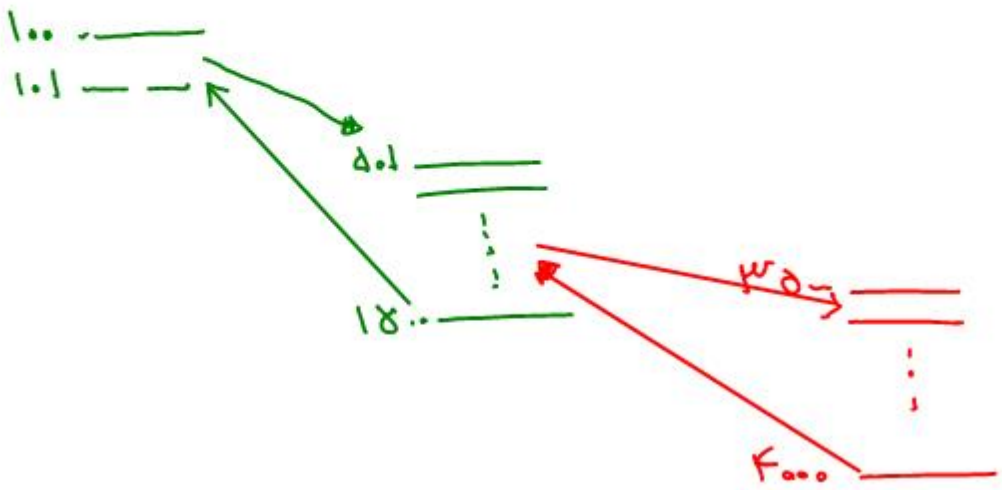
op	نام دستور		
101	BSA	→ $M[AR] \leftarrow PC$	Branch and save return add.
		$PC \leftarrow AR+1$	

$D_5T_4: M[AR] \leftarrow PC,\ AR \leftarrow AR+1$

$D_5T_5: PC \leftarrow AR,\ SC \leftarrow 0$

110 ISZ



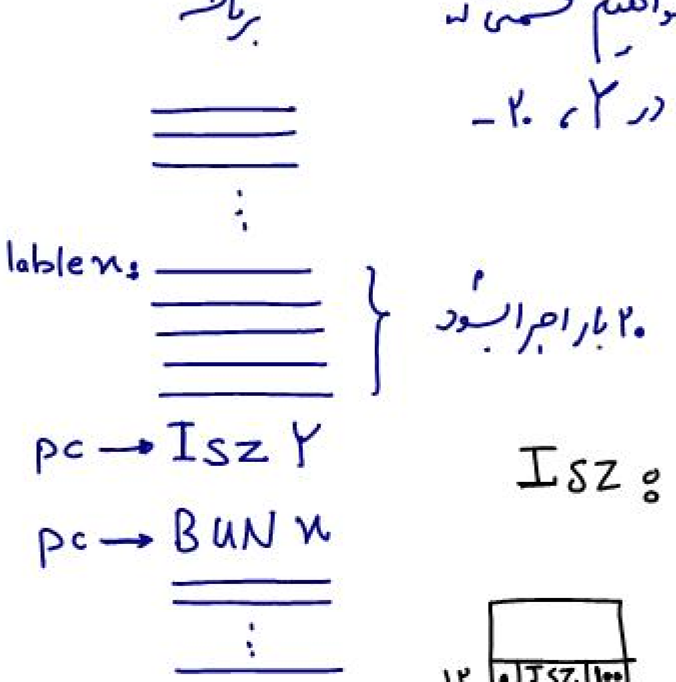


نکته) در این ماشین آدرس برگشت قبل از زیر برنامه ذخیره می‌شود، بنابراین امکان اینکه یک زیر برنامه خودش را فراخوانی کند وجود ندارد، ولی یک زیر برنامه در این ماشین می‌تواند زیر برنامه‌های دیگر را فراخوانی کند

نام دستور

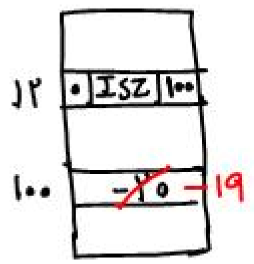
ISZ increment and skip if zero

- این دستور به درد ایجاد حلقه می‌خورد ؛ مثلاً در کد متقابل می‌خواهیم قسمتی که مشخص کردیم ۲۰ بار اجرا بشود ، برای این منظور ابتدا در Y ، ۲۰- می‌ریزیم.



$ISZ: m[AR] \leftarrow m[AR]+1$

$\text{if}(m[AR]==0)\ \text{then}\ PC \leftarrow PC+1$



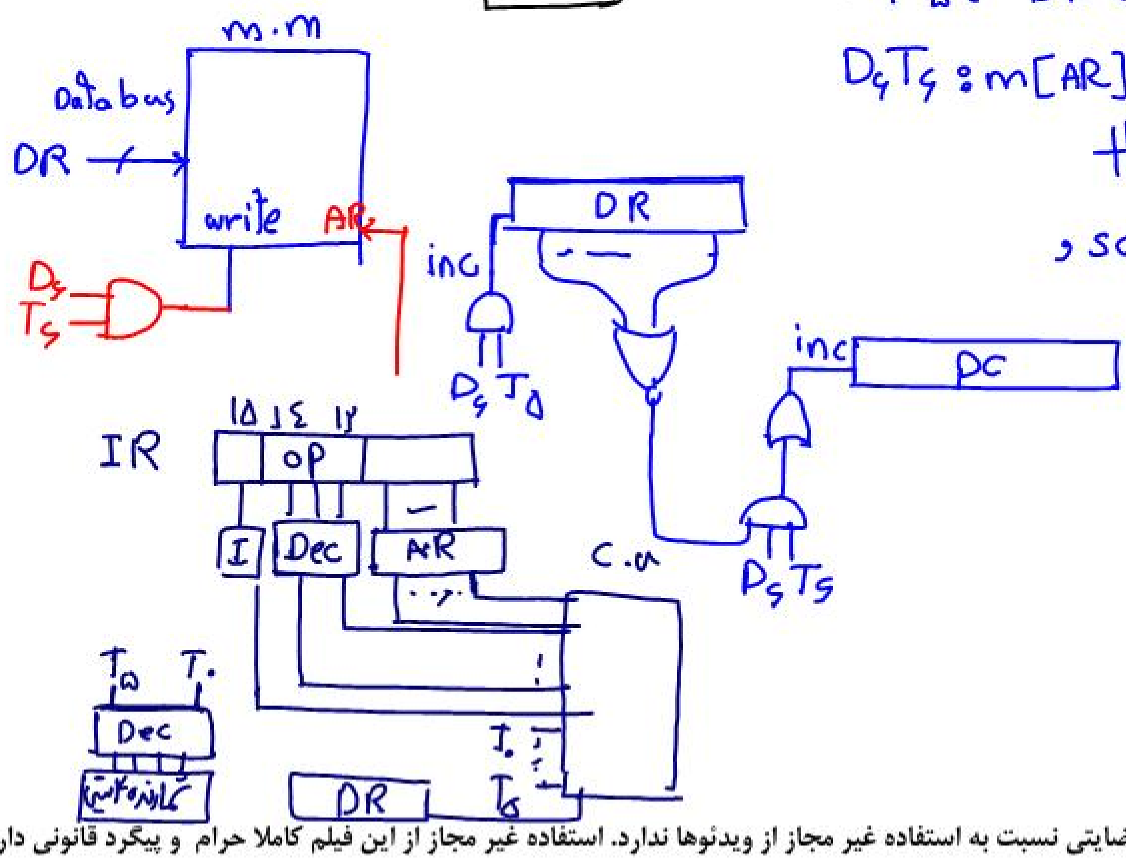
$D_6T_4: DR \leftarrow m[AR]$

$D_6T_5: DR \leftarrow DR+1;$

$D_6T_6: m[AR] \leftarrow DR, \text{if}(DR==0)$

$\text{then}\ PC \leftarrow PC+1$

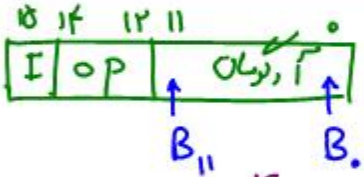
$, SC \leftarrow 0$



معماری کامپیوتر

رامین رضوی

دستورات ثبات:



(OP=111, I=1)

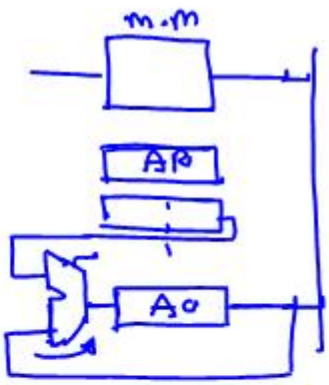
CLA $D_7 \bar{I} T_3$ (r) B_{11} : AC ← 0, SC ← 0;

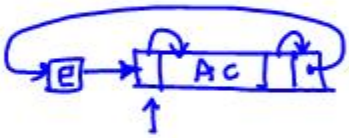
CLE rB_{10} : E ← 0, SC ← 0;

CMA : complement AC : rB_9 : AC ← $\overline{AC}$

CME : " E : rB_8 : E ← $\bar{E}$

CIR : circulate right : rB_7 : AC(15) ← E, AC ← shr AC, E ← AC(0)





CIL: rB_6 : "

INC: rB_5 : AC ← AC+1

SPA: skip if positive AC ⇒ اگر AC مثبت است از روی دستور بعدی بپر

rB_4 : if AC(15)=0 then pc ← pc+1

SNA: skip if Negative Ac : rB_3 : if AC(15)=1 then pc ← pc+1

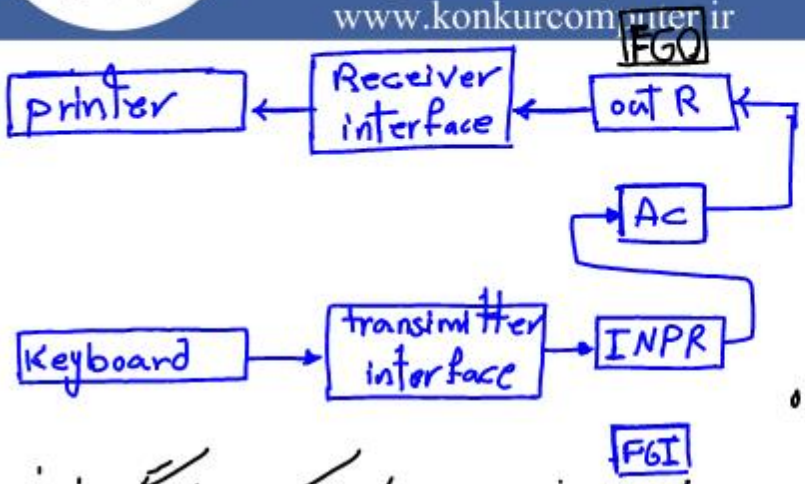
SZA: skip if Ac zero : rB_2 : if (AC==0) then pc ← pc+1

SZE: " " E " : rB_1 : " (E==0) " "

دستورات I/O :

معماری کامپیوتر

رامین رضوی



FGI فلگ (پرچم) ورودی است که مقدار اولیه اش صفر است، و بودن FGI بدان معناست که دیوایس INPR کسی نیست، در اینصورت اگر از ورودی (صفحه کلید) یک کاراکتر وارد شود، آن گاه وارد INPR می شود و FGI یک می شود (وقتی FGI یک می شود، هرچه کاربر کلید بزند دیگر پذیرفته نمی شود) یک بودن FGI بدین معناست که پردازنده می تواند به ورودی سرویس بدهد، یعنی کاراکتر inpR را به Ac منتقل کند، FGO فلگ خروجی است و مقدار اولیه اش 1 است، یک بودن FGO به این معناست که دیوایس outR کاراکتری نیست، و در اینصورت پردازنده می تواند به خروجی سرویس بدهد، سرویس دادن به خروجی به این معناست که پردازنده می تواند اگر کاراکتری در Ac است آن را به outR منتقل کند، اگر این کار انجام شود FGO را 1 می کنیم، حال هروقت کاراکتر دیوایس outR از طریق interface به خروجی منتقل شد FGO را 1 می کنیم.

نتیجه) یک بودن FGI و FGO به این معناست که ورودی یا خروجی آماده است و پردازنده می تواند سرویس بدهد.

۳ روش وجود دارد که پردازنده بیاید و I/O را چک کند

۱- pooling (سرکشی) ۲) interrupt ۳) DMA

pooling این است که پردازنده می وضعیت I/O را چک کند، این روش خوب نیست و روش خوب

وقفه است. وقفه این است که پردازنده کار خودش را انجام بده، بعد هرکسی که باهاش کار داشت، یک سیگنال بدهد، آن وقت اگر پردازنده با این وقفه موافقت کرد، می رود و به وقفه پاسخ می دهد، پاسخ دادن به وقفه به این صورت است که یک کدی برایش اجرا می شود، (به این سرویس وقفه اجرا می شود ISR (interrupt service routine

ماشین ما نیز از وقفه استفاده می کند، ماشین ما برای اینکه بخواهد وقفه را بپذیرد ۳ شرط دارد:

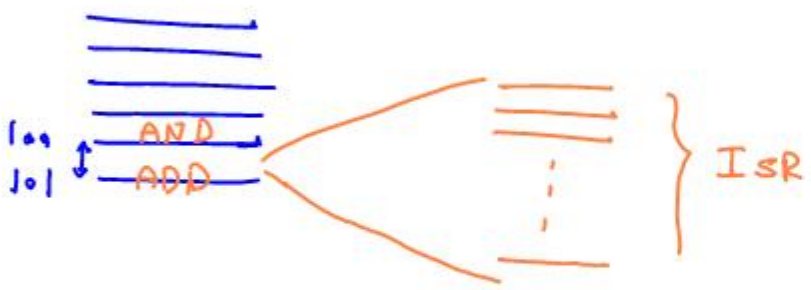
۱- ماشین در سیکل fetch و Dec نباشد

۲- وقفه ناتوان نشده باشد (IEN یک باشد یعنی اگر وقفه بیاید می خواهم اعمال اش کنم)

۳- درخواستی وجود داشته باشد ← FGI یا FGO یک باشد ← چون در این ماشین وقفه‌های ما فقط ورودی و خروجی است.

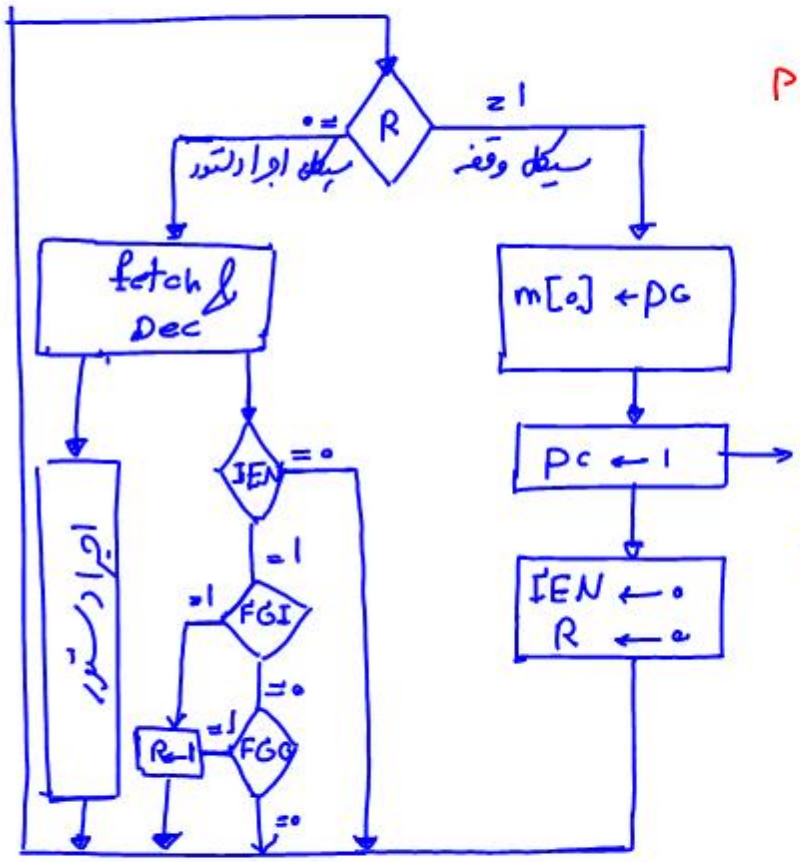
ریزعمل پذیرش وقفه: (RTL پذیرش وقفه)

$T_0'T_1'T_2' \cdot IEN \cdot (FGI + FGO) : R \leftarrow 1$



معماری کامپیوتر

رامین رضوی



IEN=1

در خانه 1 حافظه نوشته می‌شود آدرس ISR

$R'T_0$: AR ← pc

$R'T_1$: IR ← m[AR], pc ← pc+1

$R'T_2$: I ← IR(15), $D_7 - D_0$ ← dec IR(12,13,14), AR ← IR(0-11)

RTL سیکل وقفه:

RT_0 : AR ← 0, TR ← pc

RT_1 : m[AR] ← ~~pc~~ TR, pc ← 0

RT_2 : pc ← pc+1; R ← 0, IEN ← 0, SC ← 0

دستورات IO به شکل زیر است: $(op=111 \Rightarrow D_7=1, I=1, T_4=1)$

$\overline{p}$

out: D_7IT_4 B_{11} : output character outR ← AC(0-7), FGO ← 0

skI : skip on input flag : pB_9 : if (FGI=1) then pc ← pc+1

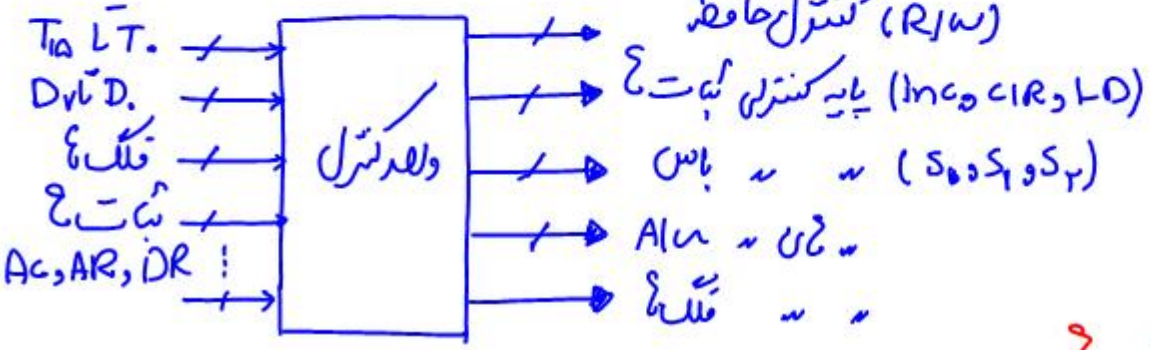
INP pB_{11} : AC(0-7) ← INPR , FGI ← 0

R: SKI
BUN R

- ماشین‌های دیجیتال از ۲ قسمت data path و واحد کنترل تشکیل می‌شود data path را بررسی کردیم، حال می‌خواهیم واحد کنترل را طراحی کنیم:

طراحی واحد کنترل به روش Hardwire :

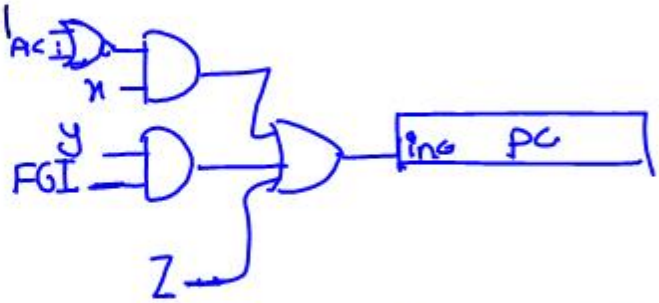
وظیفه واحد کنترل تولید سیگنال‌های کنترلی است، sequenc اجرا دستورات را واحد کنترل مشخص می‌کند



ساخت پایه‌های کنترلی ثبات‌ها

سوال) فرض کنید RTL ال بصورت روبرو داریم، پایه inc pc را طراحی کنید

R: if (AC=0) then pc ← pc+1
Y: if (FGI=1) " "
Z: pc ← pc+1



سوال) تمامی پایه‌های pc در ماشین پایه را بسازید:

$R'T_1$: $pc \leftarrow pc+1$;

RT_1 : $pc \leftarrow 0$

RT_2 : $pc \leftarrow pc+1$

D_5T_5 : $pc \leftarrow AR$

D_4T_4 : $pc \leftarrow AR$

D_6T_6 : $m[AR] \leftarrow DR$, if $(DR=0)$ then $pc \leftarrow pc+1$

rB_7 : if $(AC(15)=0)$ then $pc \leftarrow pc+1$

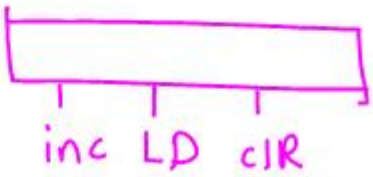
rB_6 : " " $=1$) " "

rB_5 : " $(AC=0)$ " "

rB_4 : " $(E=0)$ " "

pB_3 : " $(FGO=1)$ " "

pB_2 : " $(FGI=1)$ " "



$clR(pc) = RT_1$

$LD(pc) = D_5T_5 + D_4T_4$

$inc(pc) = R'T_1 + RT_2 + D_6T_6\,M[AR] + D_7I'T_4B_7\,\overline{AC(15)} + \cdots$

ساخت پایه‌ای read و write حافظه

۱- ساخت پایه Read: تمام جملاتی که سمت راست شون $m[AR]$ است را لیست می‌کنیم

۲- " " write: " " چپ " " "

write = $RT_1 + D_2T_2 + D_5T_5 + D_6T_6$

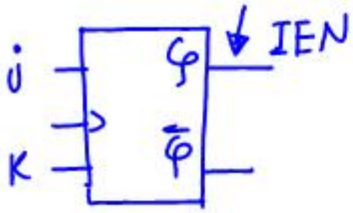
Read = $R'T_1 + D'_7IT_0 + (D_0 + D_1 + D_2 + D_3)T_4$

$R'T_1 : IR \leftarrow m[AR]$

$D_{...}T_2 : DR \leftarrow m[AR]$

طراحی چاپه‌ها (کنترل فلگ‌ها)

هر فلگی را می‌توانیم با یک فلیپ فلاپ JK بسازیم، این ماشین ۷ تا فلگ دارد، بنابراین این ماشین ۷ تا فلیپ فلاپ دارد.



j	K	φ*
0	0	φ
0	1	0
1	0	1
1	1	φ̄

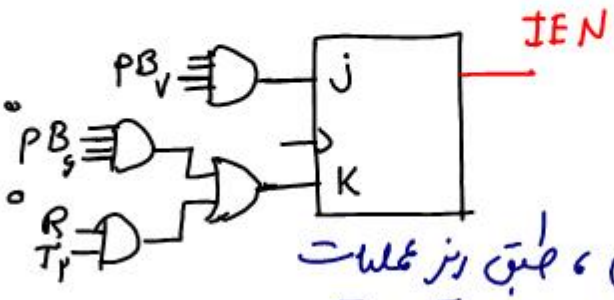
DVITd

↑

pBV: IEN ← 1

pBφ: IEN ← 0

RT۲: IEN ← 0



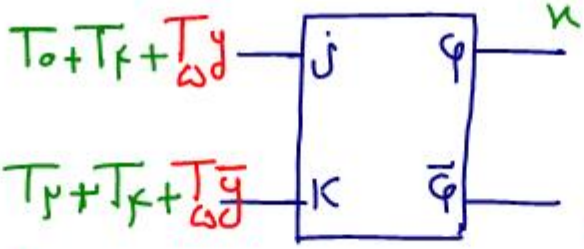
مثال) با فلیپ فلاپ JK، طبق زیر عملیات زیر فلگ x را بسازید:

T۰: x ← 1

T۲: x ← 0

T۴: x ← x̄

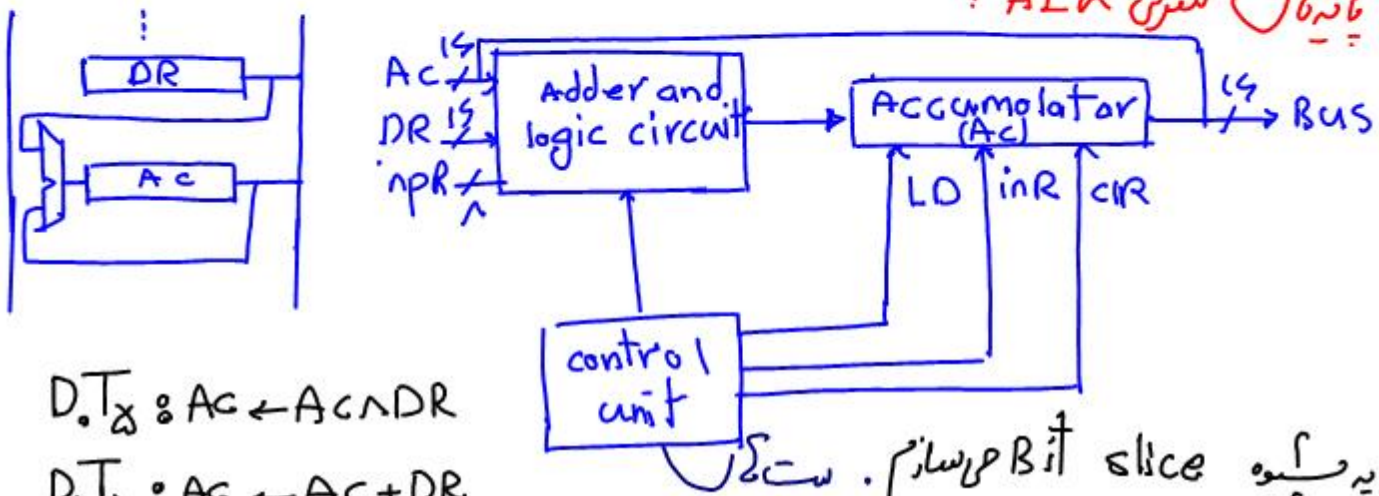
T۵: x ← y



* هروقت خواستی چیزی و به فلیپ فلاپ JK وارد کنی خودش را بده به بالایی و not اش را بده به پایینی.

x
0 → j=0 و K=1
1 → j=1 و K=0

طراحی یاده‌ای نمونه‌ی ALU :



$D_5T_5 : AC \leftarrow AC \wedge DR$

$D_6T_5 : AC \leftarrow AC + DR$

$D_7T_5 : AC \leftarrow DR$

$pB_{11} : AC(0) \leftarrow INPR$

$rB_9 : AC \leftarrow \overline{AC}$

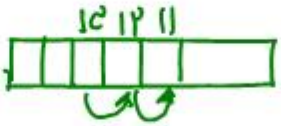
$rB_{11} : AC \leftarrow 0$

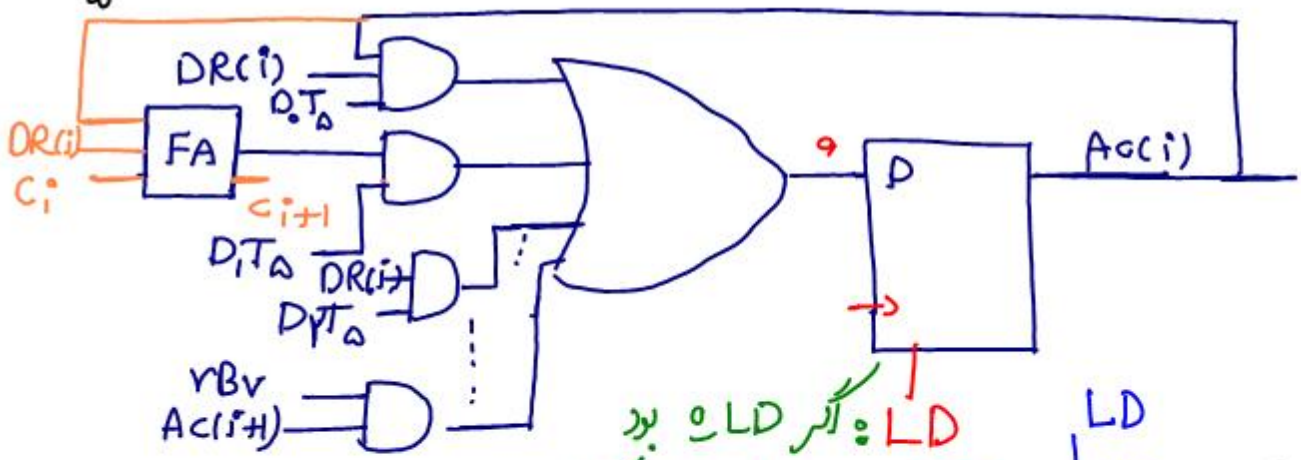
$rB_7 : AC \leftarrow shr\ AC, AC(15) \leftarrow E$

$rB_6 : AC \leftarrow shL\ AC, AC(0) \leftarrow E$

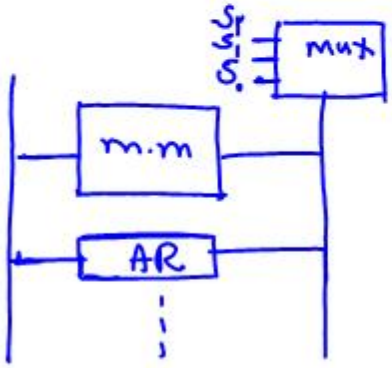
$rB_5 : AC \leftarrow AC + 1$

AC را به صورت Bit slice می‌سازیم. یعنی هر بیت AC را جدا جدا می‌سازیم، مثلاً می‌خواهیم بیت i ام AC را بسازیم





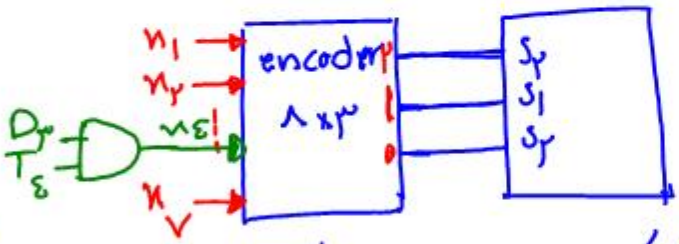
طراحی باید های کنترل باس



$S_2S_1S_0$	روی باس میره
0 = 0 0 0	Nothing
1 = 0 0 1	AR
2 = 0 1 0	PC
3 = 0 1 1	DR
4 = 1 0 0	AC
5 = 1 0 1	IR
6 = 1 1 0	TR
7 = 1 1 1	m.m

S_2 کی ها باید 1 شود ؟ زمانیکه mem یا TR یا IR یا AC روی باس می آید .

$$S_2 = \overbrace{D_3T_4}^{n_4} + \overbrace{R'T_2}^{n_5} + \overbrace{RT_1}^{n_6} + \overbrace{read}^{n_7}$$



کار با پشته

پشته ساختمان داده ای است LIFO : آخرین کسی که وارد می شود ، اولین کسی است که خارج می شود

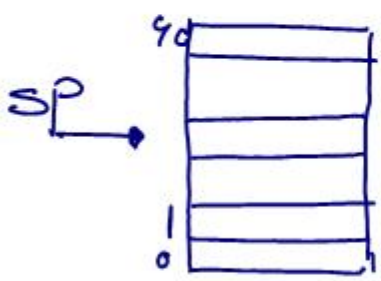
برای ساخت پشته ۲ راه وجود دارد : ۱ـ قسمتی از حافظه اصلی را پشته می کنیم (پشته حافظه ای)

۲ـ از یک حافظه مجزا به عنوان پشته استفاده می کنیم

(register stack : پشته ثبات)

یک sp در مورد پشته ثباتی است

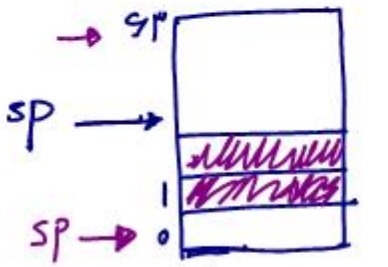
- می خواهیم از یک حافظه ۶۴ کلمه ای بعنوان پشته استفاده کنیم.



- stack ال بارم که sp به خانه سر بالای پشته اشاره می کند و شماره گذاری خانه ها از پایین به بالاست .

معماری کامپیوتر

رامین رضوی



* خانه ۰ آخرین خانه‌ای است که پر می‌شود

```
push Ac
if(Full=0){
    sp ← sp+1;
    m[sp] ← Ac
    if(sp=0) Full←1;
    EMTY←0;
```

```
pop Ac
if(EMTY=0)
{
    Ac ← m[sp]
    sp ← sp-1;
    if(sp=0) EMTY←1
    Full←0
```

- ما دوتا flag داریم : Full : ۱ بودنش به این معناست که پشته پر است . EMTY : ۱ بودنش به این معناست که پشته خالی است .

توجه) در پشته‌ای که ما می‌سازیم sp می‌تواند ۱ واحد کم و زیاد شود ، اما می‌دانیم که ممکن است sp ۲ یا ۴ یا ... کم و زیاد شود .

توجه) در پشته‌ای که الان ساختیم sp به خانه پر بالای پشته اشاره می‌کرد ، می‌توان پشته را جوری ساخت که sp به خانه خالی بالای پشته اشاره کند . همچنین شماره‌گذاری خانه‌ها می‌تواند از بالا به پایین باشد

تست) فرض کنید در یک ماشین sp به خانه پر اشاره می‌کند و موقع push دو واحد کم می‌شود اگر مقدار فعلی sp ، F000H باشد ، بعد از دستورات زیر ، bp چند است؟

```
push sp
pop bp
```

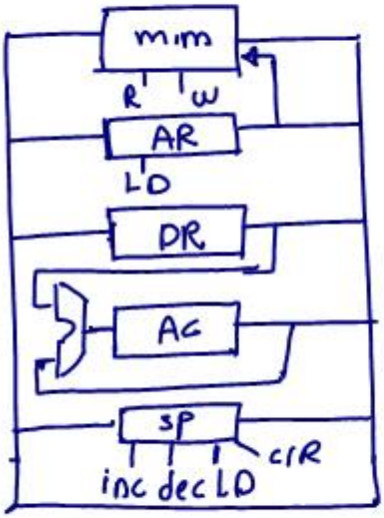
EFFF
F000
- 2
EFFE

bp=EFFE

معماری کامپیوتر

رامین رضوی

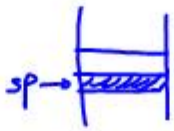
مثال) در شکل زیر فرض کنید sp به خانه پر اشاره می‌کند، push Ac و pop Ac هر کدام چند کلاک می‌خواهد؟



push Ac

sp ← sp+1

m[sp] ← Ac



1- sp ← sp+1

2- AR ← SP

3- m[AR] ← Ac

pop Ac

Ac ← m[SP]

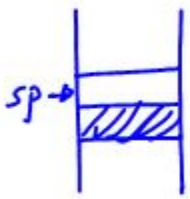
sp ← sp-1

1- AR ← sp

2- DR ← m[AR]

3- Ac ← DR, sp ← sp-1

سوال) اگر sp به خانه خالی بالا پشته اشاره کند سوال قبل چه می‌شود



push Ac

m[sp] ← Ac

sp ← sp+1

1- AR ← sp

2- m[sp] ← Ac, sp ← sp+1

pop Ac

sp ← sp-1

Ac ← m[sp]

1) sp ← sp-1

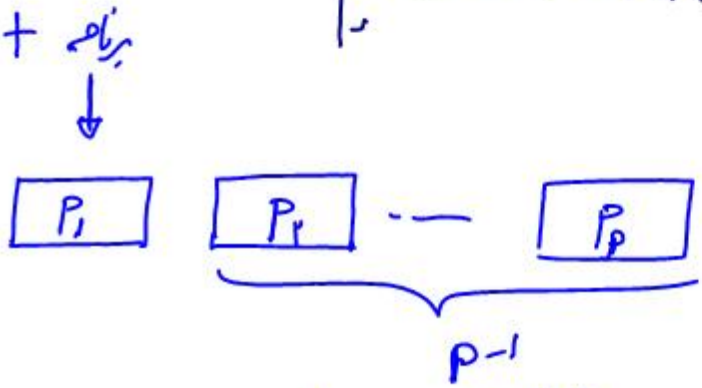
2) AR ← sp

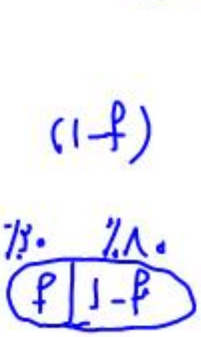
3) DR ← m[AR]

4) Ac ← DR

فصل موازات و پایپ لاین

سوال) فرض کنید برنامه ای روی یک پردازنده اجرا شده، زمان اجرایش + شده، حال همین برنامه را روی p پردازنده مشابه (همگون) تقسیم کرده ایم، در حالت ایده آل، انتظار داریم که زمان اجرایمان $\frac{1}{p}$ بشود، اما این حالت ایده آل هیچ وقت پیش نمی آید، یک علت این امر این است، که معمولا دستوراتی در برنامه ها وجود دارند که بهم وابسته هستند و نمی توانی آنها را از هم جدا شون کرد و حتما باید یک پردازنده آنها را اجرا کند، در اصل چنین دستوراتی در برنامه ها را معمولا با f نشان می دهند (f عددی است بین ۰ و ۱ که برخی از آن مولفه ترتیبی نیز می گوییم)، حال با وجود این f می خواهیم زمان اجرا روی p پردازنده را محاسبه کنیم





مدل ۱) $\max\left(ft, \frac{(1-f)t}{p-1}\right)$: قانون آمدال

مدل ۲) $ft + \frac{(1-f)t}{p}$ → زمان اجرا برنامه روی p پردازنده

- سرعت p پردازنده به سرعت ۱ پردازنده را محاسبه کنید

$$\text{speed up} = \frac{\text{سرعت } p \text{ پردازنده}}{\text{سرعت ۱ پردازنده}} = \frac{\text{زمان اجرا روی ۱ پردازنده}}{\text{زمان اجرا روی } p \text{ پردازنده}} = \frac{t}{\left(ft + \frac{1-f}{p}\right)t} = \frac{1}{f + \frac{1-f}{p}}$$

معماری کامپیوتر

رامین رضوی

$f = 0/1$

$\text{Max speedup} = \frac{1}{f} = 10$ ← ناجیه بشریت

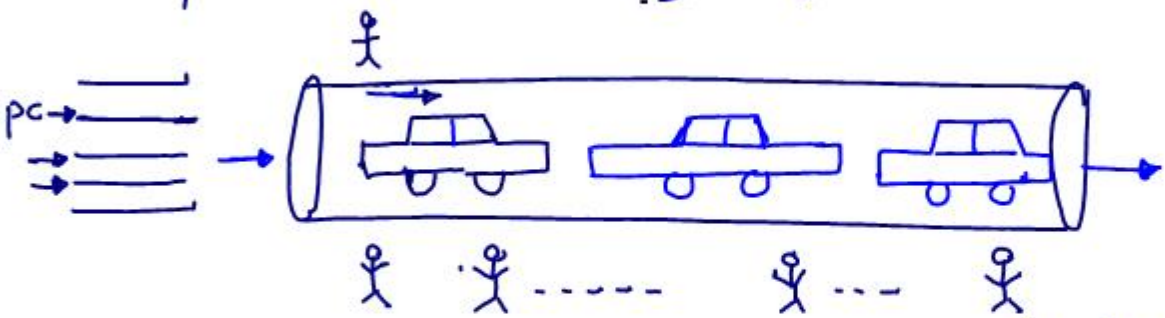
$p \to \infty$

$$\text{efficiency} = \frac{\text{speed up}}{\text{number of processor}}$$

کارایی

پایپ لاین

پایپ لاین روشی است که بالک آن می‌توان عملیات ترتیبی را بصورت شبه موازی انجام داد.



اگر سیکل دستور ۴ مرحله داشته باشد و هر مرحله ۱ کلاک طول بکشد و n دستور داشته باشیم:

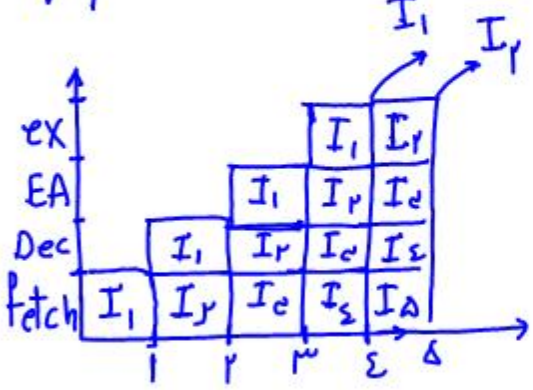
زمان اجرا no pip $= 4nT$

pip زمان $= (3+n)T$

$$\text{speedup} = \frac{S_{pip}}{S_{no\ pip}} = \frac{\text{no pip زمان}}{\text{pip زمان}} = \frac{4nT}{(4+n)T}$$

$\text{Max speedup} = 4$

$n \to \infty$



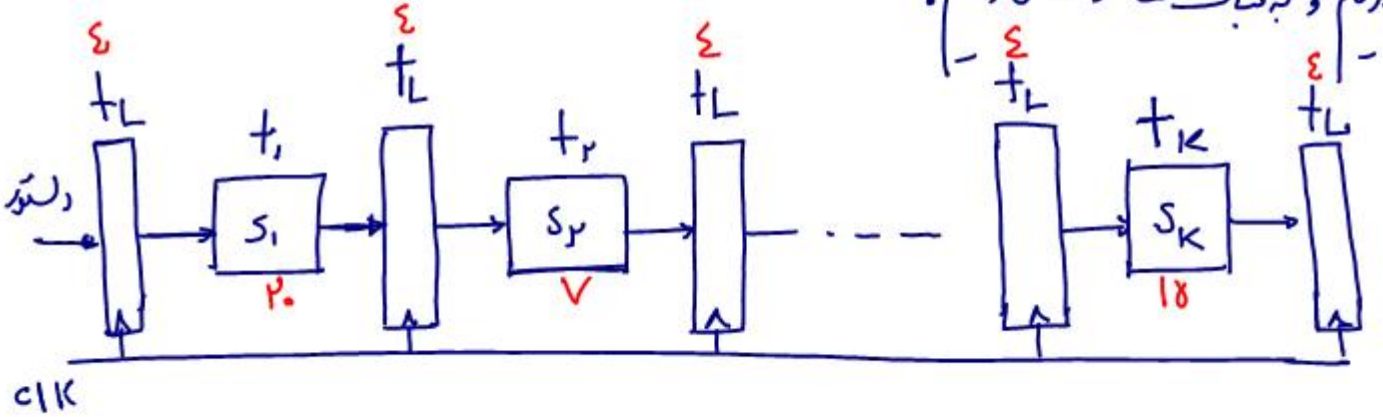
Multi clock cycle diagram نمایش

معماری کامپیوتر

رامین رضوی

بررسی یک پایپ K بندی

از آنجایی که تاخیر بندها تروما یکسان نیست، در بین بندها برای سنکرون کردن بندها ثبات می‌گذاریم و به ثبات‌ها کلاک می‌زنیم.



$$T_{clk} = Max(t_1, t_2, \dots, t_K) + t_L$$

زمان اجرا n دستور با پایپ‌لاین:

$$\text{زمان اجرا} = KT + (n-1)T = (K+n-1)T$$

دستور اول باید تمام مراحل را طی کند، ولی به بعد هر دستور با یک کلاک میخته بیرون (اجرا می‌شود)

زمان اجرا n دستور بدون پایپ: (پس دیگه به ثبات‌ها هم نیازی نداریم)

$$n \times (t_1 + t_2 + \dots + t_K)$$

بایکی فرضیات زمان اجرا n دستور بدون مزاح:

فرضیات: ۱- تاخیر بندها یکسان باشد ۲- تاخیر لچ‌ها (ثبات‌ها) ناچیز باشد

$$T = Max(t_1, t_2, \dots, t_K) = t_1 = t_2 = \dots = t_K$$

زمان اجرا $= nKT$

با فرض گفته شده speed up رابطه آوریم.

$$\text{speed up} = \frac{S_{pip}}{S_{no\ pip}} = \frac{\text{زمان no pip}}{\text{زمان pip}} = \frac{nKT}{KT+(n-1)T} = \frac{nK}{K+n-1}$$

$$\text{Max speedup} = K \Rightarrow \lim_{n\to\infty}$$

خط لوله‌ای که K بند دارد حداکثر سرعت را می‌تواند K برابر کند

$$\text{throughput} = \text{توان عملیاتی (گذردهی)} = \text{تعداد دستورات انجام شده در واحد زمان} = \frac{\text{تعداد دستور}}{\text{زمان}} = \frac{n}{KT+(n-1)T}$$

$$\lim_{n\to\infty} \text{max thro} = \frac{1}{T} = f$$

فرکانس 100Hz: در 1 ثانیه 100 آپارات می‌بینی

$$CPI = \text{clock per inst.}^{\text{avg}} = \text{میانگین تعداد کلاک به ازای هر دستور} = \frac{\text{تعداد کلاک‌ها}}{n} =$$

$$= \frac{K+n-1}{n} \Rightarrow \lim_{n\to\infty} \text{max } CPI = 1$$

هر دستور بطور میانگین: 1 کلاک نیاز دارد

حافظه ها

حافظه خیلی کند است، بنابراین عملاً تعیین کننده سرعت کامپیوتر حافظه است، به همین علت به حافظه ها گویند memory wall.

مثال: فرض کنید زمان اجرای یک برنامه 1 ساعت است و cpu برای انجام محاسبات 20 دقیقه زمان نیاز دارد و زمان دسترسی به حافظه در این برنامه 40 دقیقه است

برای افزایش سرعت به cpu یا 10 برابر سرعت جایگزین می کنیم ← زمان اجرا برنامه = 42 دقیقه

سلسله مراتب حافظه

حافظه را می توان با توجه به پارامترهای مختلف دسته بندی کرد. حال اگر با توجه به سرعت و اندازه حافظه ها را دسته بندی کنیم خواهیم داشت.

دسته بندی حافظه ها می تواند از دیدگاه های مختلفی باشد

1- location: در چه مکانی نسبت به cpu قرار گرفته است

1- on chip (داخل cpu): reg، برخی از کش ها (control memory، (cache L1

2- internal (داخل سیستم): cache L2 or main memory

3- external: کارد دیسک، حافظه نانویه

2- capacity

3- unit of transfer

4- Access method: 1- ترتیبی (sequential) 2- Direct (در دیسک استفاده می شود)

3- Random: به هر location ها با یک تایم مساوی دسترسی پیدا می کنیم کش و main memory

۵- کارایی: می‌توانیم access time را بعنوان معیار کارایی در نظر بگیریم

۶- physical type: ۱- نیمه‌هادی: RAM (DRAM & SRAM)

۲- مغناطیسی: Disk & Type

۳- حافظه‌های مقاومتی

۴- optical: CD & DVD

۷- physical characteristics

۱- volatility: ماندگاری حافظه، حافظه‌ای را volatile گوییم که اگر باتری را قطع کنیم اطلاعات روش باقی نماند

۲- erasable:

۳- power consuption

۴- Decay: از دست رفتن خاصیت حافظه: در حافظه‌های مغناطیسی دوری سطح رو از خودرگی و فرار می‌شوند

۸- organization

ما چه دوست داریم: ما دوست داریم یک حافظه بسیار بزرگ و بسیار سریع داشته باشیم

نکته - هرچه حافظه بزرگتر بشود کندتر می‌شود

- حافظه‌ای که بعنوان main memory در کامپیوترهای ما استفاده می‌شوند معمولا از تکنولوژی RAM هستند، هر سل (cell) رم می‌تواند 1 بیت را در خودش ذخیره کند، ما دو مدل RAM داریم

معماری کامپیوتر

رامین رضوی

RAM
- SRAM : سریعتر، گران تر، ظرفیت کمتر
- DRAM : کندتر، ارزان تر، به زمان

حداقل ۱۰۰ برابر سریعتر است SRAM : حدود ۱۰۰۰ برابر سریعتر از DRAM است.

memory technology
- static RAM : ۰/۵ns - ۲/۵ns, $2000 - $5000 per GB
- Dynamic RAM : ۵۰ns - ۷۰ns, $20 - 75$ per GB
- magnetic disk : ۵ms - ۲۰ms, $0.2 - $2 per GB

↓

۱ میلیون بار کندتر است از DRAM

DRAM : عناصرش در دو بعد سطر و ستون قرار می‌گیرد و برای دسترسی ابتدا سطر و بعد ستون انتخاب می‌شوند، هر بیت اطلاعات توسط یک خازن و یک Tr ذخیره می‌شود، باری که روی آن خازن است تعیین می‌کند که آن اطلاعات ۰ است یا ۱. خازن‌ها نشتی دارند و باید که روی خازن‌ها هست به خاطر نشتی دشارژ می‌شوند و برای همین ما مجبوریم هر ۱۰ تا ۱۰۰ میلی ثانیه یکبار مقدار دیتا خازن‌ها را رفرش کنیم

SRAM = در SRAM برای نگهداری هر بیت از ۶ عدد Tr استفاده می‌شود (که ۴ تا از Tr ها برای نگهداری اطلاعات و دو تای دیگر نقش کنترل کننده برای خواندن یا نوشتن را دارند) به مجموع این ۶ ترانزیستور فلیپ فلاپ می‌گویند و یک FF نشتی ندارد بنابراین نیازی هم به رفرش کردن ندارد

نکته) هر دو فرارند یعنی با قطع برق دیتاشون پاک می‌شود (volatile). مثلاً حافظه flash non-volatile است.

نکته) حافظه دیسک ماندگار است، پس ما همه چیز را روی دیسک نگهداری می‌کنیم و هر چیزی را که لازم داشتیم را لود می‌کنیم میاریم در حافظه‌های سریعتر

معماری کامپیوتر

رامین رضوی

۲ مدل SRAM داریم
- قابل مشاهده : برنامه نویس زبان ماشین می‌تواند آن را کنترل کند
- غیرقابل مشاهده : سخت افزار کنترل آن می‌کند

کش یک SRAM کوچک غیرقابل مشاهده از دید برنامه نویس زبان ماشین است، بنابراین کش جز معماری پردازنده نیست

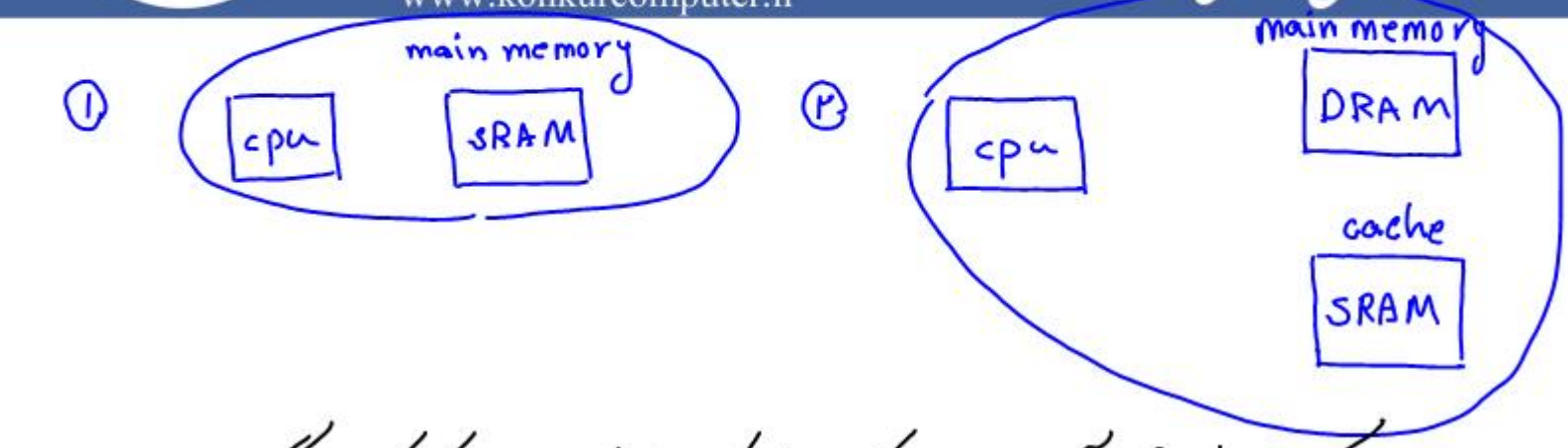
- یک DRAM chip می‌تواند یک access time بین ۳۰ns - ۵۰ns داشته باشد و کل دسترسی به حافظه از زمانی که cpu یک آدرس تولید می‌کند تا زمانیکه دیتایش را دریافت می‌کند حدود ۱۰۰ns است. برابر این زمان طول می‌کشد حال سرعت cpu ها کسری از نانو ثانیه است و بنابراین یک دستور که operand هایش در حافظه است باید ۲ الی ۳ هزار سیکل منتظر بماند. حال حدود ۳/۱ دستورات، دستورات load و store هستند که این امر وابستگی شدیدی را به حافظه نشان می‌دهد حال اگر ما بخواهیم به ازای این ۳/۱ دستورات هر بار این penalty را بدهیم کارایی مان به شدت افت پیدا می‌کند

- ۲ روش برای افزایش سرعت حافظه یاد می‌گیریم
  ۱- استفاده از کش
  ۲- استفاده از ایده بزرگ‌تر کردن حافظه

فعلا سراغ ایده ۱ می‌رویم.

- اگر کامپیوتر RAM
  ① کمی بخواهد : کلا SRAM می‌گذاریم و مشکل memory wall هم نداریم (۵۱۲K)
  ② زیادی بخواهد : یک SRAM کوچک و یک DRAM بزرگ می‌ذاریم

چرا SRAM بزرگ نمی‌ذاریم
۱- خیلی گران می‌شود، هم خیلی فضا می‌گیرد
۲- وقتی بزرگ‌اش می‌کنیم دیگه کند می‌شود



- حال سوالی که در اینجا پیش می آید این است که چطور یک حافظه SRAM کوچک میانگین زمان دسترسی به حافظه را به شدت افزایش میدهد ؟

locality of refrences

محلی بودن دسترسی ها این نکته را به ما میدهد که می شود با نگهداشتن بخش کوچکی از اطلاعات در یک حافظه سریع ، سرعت نهایی دسترسی به حافظه را بالا برد ، این همان چیزی بود که در نهایت باعث به وجود آمدن ایده کش شد. محلی بودن مراجعات دو جنبه دارد

۱- spatial locality : اگر به x رجوع کردن به احتمال بالا به آدرس های مجاورش نیز رجوع میکنیم مثل دستورات پشت سر هم در برنامه ها

۲- temporal locality : اگر به آدرس x رجوع کردی به احتمال زیاد در آینده نزدیک مجددا به x رجوع خواهی کرد مثل دستورات درون حلقه های for

* پس اگر cpu خانه ای را دست نزده براساس LOR زمانی همان خانه را در کش قرار میدهیم ، همچنین براساس LOR مکانی خانه های مجاور آن خانه را نیز در کش قرار می دهیم

نکته - به این کار که شما براساس LOR مکانی می روید و خانه هایی که cpu دست نزده را در کش می گذارید prefetch کردن می گویند . کلا به آوردن داده در کش قبل از آنکه cpu آن را بخواهد prefetch می گویند

معماری کامپیوتر

رامین رضوی

نکته) رفتار پردازنده (ویژگی پردازنده این است که) locality مکانی و زمانی دارد پس از این ویژگی پردازنده استفاده می‌کند

نکته) با توجه به اصل locality هروقت که miss رخ دهد ما باید یک بلاک از حافظه اصلی به کش وارد کنیم که این باعث می‌شود نه تنها خود آن word در کش برود بلکه word های اطراف اش نیز در کش برود

- هر خانه حافظه یک word است و هر خانه کش یک بلاک است و هر بلاک از توانی از دو تا word درست شده . حال شما وقتی در یک حافظه چیزی بنویسید یا از آن حافظه چیزی بخوانید باید به اندازه سایز یک خانه آن حافظه بخوانید یا بنویسید ، حال وقتی cpu آدرس یک word را تولید می‌کند و آن word در کش نیست باید آن word را وارد کش کنیم اما کش که word قبول نمی‌کند ، پس مجبوریم بلاک ای که آن word متعلق به آن است را به کش بدهیم ، اینطوری LOR مکانی و زمانی در نظر گرفته می‌شود

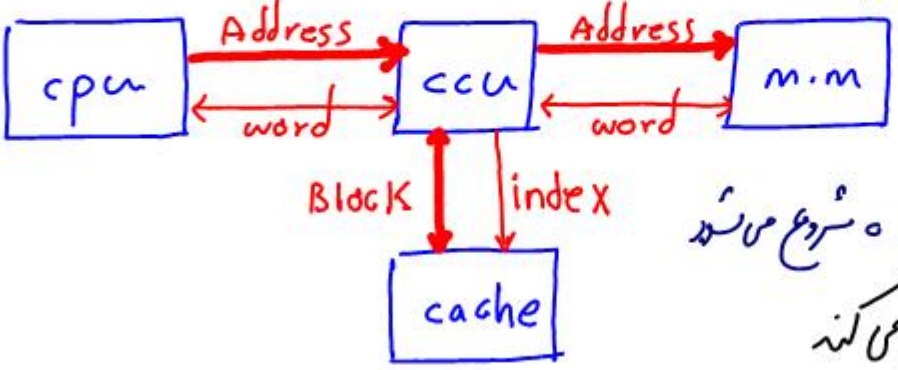
نتیجه‌گیری : دلیل آنکه خانه‌های کش را هم سایز خانه‌های m.m می‌گیرند آن است که علاوه بر LOR زمانی LOR مکانی نیز داشته باشیم ، چون اگر خانه‌های کش را هم سایز خانه‌های حافظه اصلی می‌گرفتند فقط از LOR زمانی استفاده می‌شد در آن صورت .

word : به هر خانه m.m ، word می‌گویند

Block : " " " حافظه کش

Address : به آدرس حافظه اصلی

index : " " " کش که از ۰ شروع می‌شود



توجه) cpu آدرس یک کلمه را تولید می‌کند

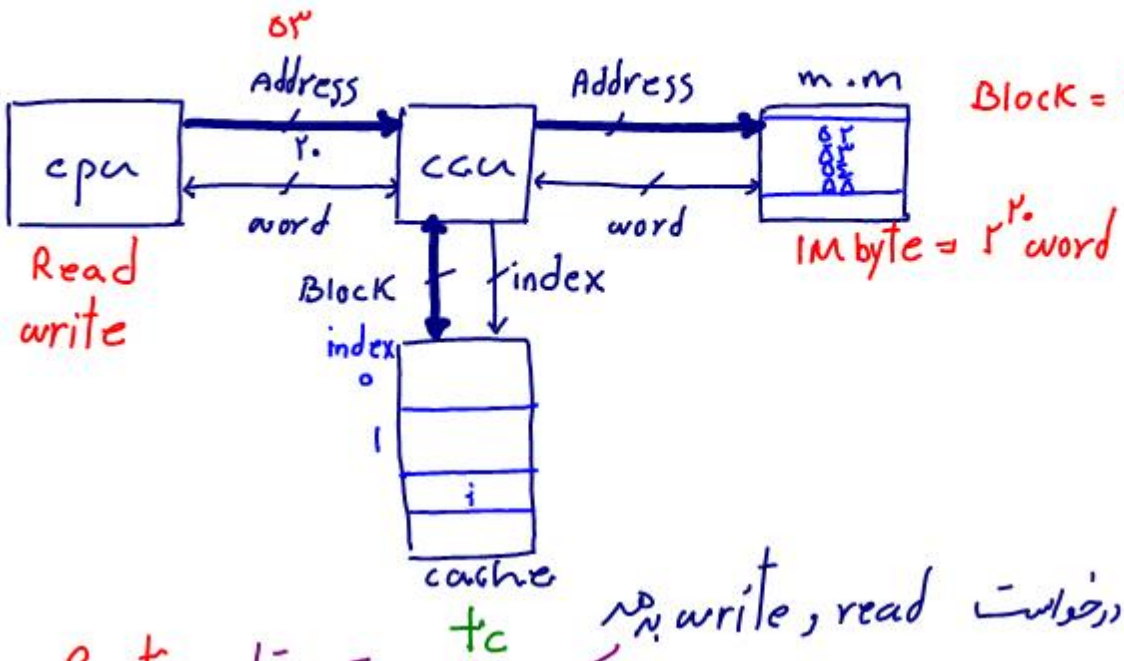
معماری کامپیوتر

رامین رضوی

نکته) کش و ccu از دید cpu، transparent است، یعنی ccu و کش وجود دارند ولی cpu از وجود آنها مطلع نیست

نکته) خیلی مهم است که سرعت ccu یا Y باشد، چون در غیر اینصورت نقض غرض می‌شود، یعنی ما اصلاکش را برای این گذاشتیم که سرعت را ببرد.

نکته) ccu علاوه بر اینکه کارهای کنترلی را انجام میدهد درون اش یکسری رجیستر برای ذخیره اطلاعات دارد



* cpu می‌تواند در درخواست به هر read, write بدهد

* عملکرد سیستم حافظه به ۲ صورت است
  - Defualt : ۱- عملکرد سری سیستم حافظه
  - ۲- " موازی " "

read : فعلا فرض می‌کنیم cpu فقط درخواست read بدهد

- عملکرد سری سیستم حافظه : سری اینطوری است که وقتی cpu یک آدرسی به ccu میدهد، ccu ابتدا کش را می‌خواند تا ببیند که آن داده در کش هست یا نیست (چک کردن tag, validbit و ...) بعد اگر در کش بود که یکسره مورد نظر را به cpu میدهد (این کار زمان نمی‌برد) و اگر در کش نبود میرود

معماری کامپیوتر

رامین رضوی

سراغ حافظه اصلی و یک بلاک از حافظه اصلی می‌خواند و به cache منتقل می‌کند و سپس دو کار انجام میدهد، یکی اینکه کلمه مورد نظر را به cpu میدهد و دیگر آنکه بلاک مورد نظر را در کش می‌نویسد که دارن کلمه به cpu زمان می‌گیرد، همچنین cpu منتظر اینکه cache در کش بنویسد نمی‌ماند

عملکرد سلسله مراتبی

cpu یک آدرس به cache میدهد، حال cache همان موقع که می‌رود سراغ کش همزمان هم سراغ m.m می‌رود حال اگر در کش پیدا شود که داده را به cpu میدهد و تمام ولی اگر در کش نباشد باید قبل عمل می‌شود

hit rate (نرخ برخورد): با h نشان داده میشود و برابر است با: $hit\ rate\ (h) = \frac{\text{تعداد hit ها}}{\text{تعداد درخواست cpu ها}}$ یک احتمال است

miss rate (نرخ نقصان): $m = 1 - h$

متوسط زمان دسترسی به سیستم حافظه (متوسط زمان دسترسی cpu به یک کلمه (آدرس)) (AMAT)

t_c = زمان دسترسی به کش (هرخانه کش یک بلاک است)

t_m = زمان دسترسی به m.m (زمان خواندن هرخانه از m.m که در خانه m.m یک word است)

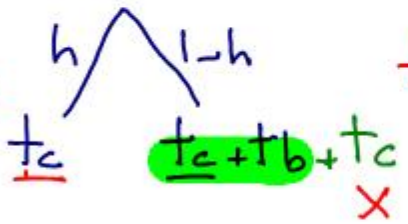
$t_b = Kt_m$; با فرض آنکه بلاک K تایی باشد

* بطور پیش فرض، ما از m.m یک word می‌خواستیم، مگر آنکه در سوال به نحوی اشاره شود که با هر access یک بلاک خوانده می‌شود.

$$t_{avg} = AMAT = ht_c + (1-h)(t_c + t_b) = t_c + (1-h)t_b$$

* ما همیشه زمان کش را داریم، اما اگر در کش نبود زمان بلاک را نیز داریم

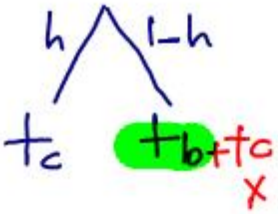
سری:



$t_{avg} = t_c + (1-h)t_b$

عملکرد موازی

موازی:



$t_{avg} = ht_c + (1-h)t_b$

نکته) عملکرد سری و موازی در هنگام miss اختلاف دارند و اختلاف شون نیز به اندازه t_c است

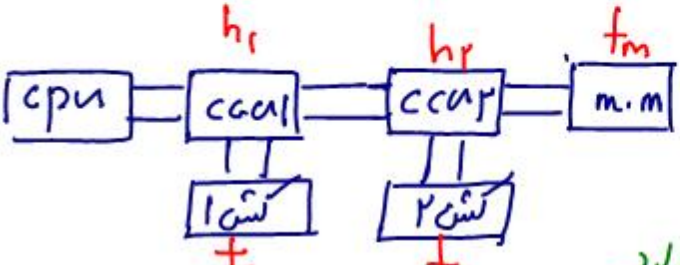
* بنابراین اختلاف سرعت بین سری و موازی قابل توجه نیست ① t_c در مقابل t_b خیلی کوچک است ② hit rate بالاست

- موازی خیلی کم سریعتر است

نکته - ما هروقت به حافظه ها دسترسی انجام می دهیم انرژی بسیار زیادی مصرف می شود (تا وقتی به حافظه ها دست نمی زنیم آنها انرژی ثابت و خیلی کمی را مصرف می کنند که قابل صرف نظر کردن است)

(چون پایه های آدرس حافظه ها خازن های خیلی بزرگی دارند)

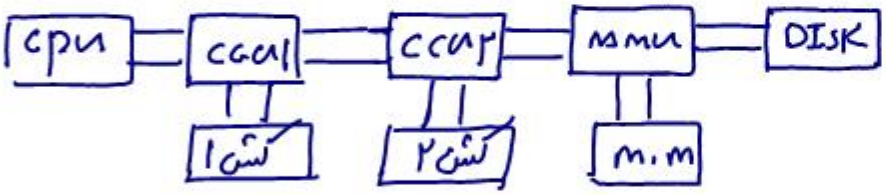
نکته) بین روش سری و موازی از لحاظ سرعت اختلاف ناچیزی وجود دارد ولی از لحاظ انرژی ، انرژی مصرفی عملکرد سری بسیار پایین تر از انرژی مصرفی عملکرد موازی است ، چون در روش موازی ما به ازای تمامی آدرس هایی تولید شده توسط cpu به سراغ m.m می رود .



cache چند لایه : ccu1 از وجود ccu2 و کش ۲ خبر ندارد و فکر میکند سمت راست اش حافظه اصلی و این داستان در کش های بالایه ای بالاتر ادامه دارد

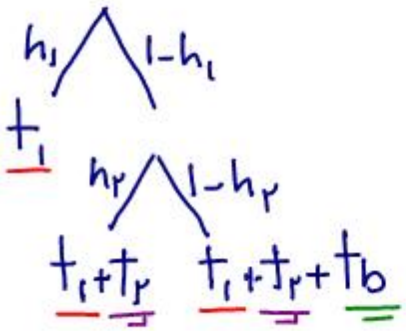
معماری کامپیوتر

رامین رضوی



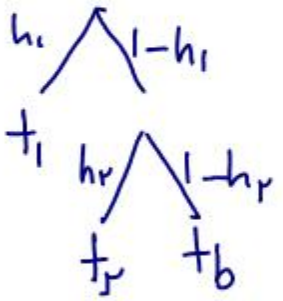
m.m در اینجا نقش کش را برای virtual space که دیسک است بازی میکنه

عملکرد سری برای شکل ۱ :



$$t_{avg} = t_1 + (1-h_1)t_2 + (1-h_1)(1-h_2)t_b$$

عملکرد موازی برای شکل ۱ :



$$t_{avg} = h_1 t_1 + (1-h_1)h_2 t_2 + (1-h_1)(1-h_2)t_b$$

مثال - فرض کنید سیستم حافظه‌ای ۲ لول کش دارد، با توجه به مشخصات داده شده، متوسط زمان دسترسی به سیستم حافظه را بدست آورید ؟

L_1 hit time = 1 cycle

L_1 miss rate = ۵٪

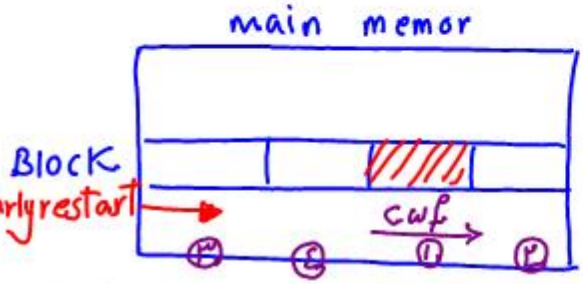
L_2 hit time = ۱۰ cycle

L_2 miss rate = ۲۰٪

L_2 miss penalty = ۵۰ cycle

$$t_{avg} = 1 + 0/05 \times 10 + 0/05 \times 0/2 \times 50 = 2 \text{ cycle}$$

ماوقتی سراغ حافظه اصلی می رویم در واقع گویی جریمه می شویم :



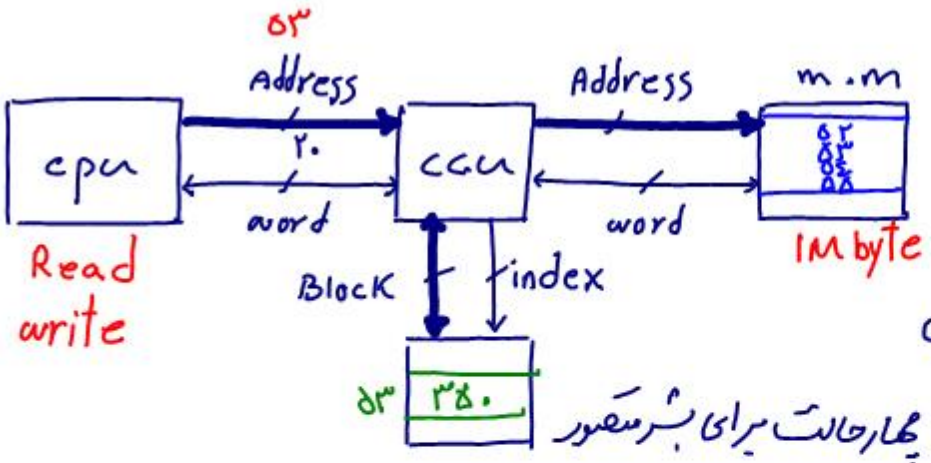
نکته) برای اینکه cpu علاف نشود دو روش برای خواندن word ها از یک بلاک از m.m وجود دارد، یکی early restart و دیگری critical word first

معماری کامپیوتر

رامین رضوی

نکته) در برخی از معماری ها ما دو تا کش داریم instruction cache و Data cache ، روش early restart بیشتر به درد instruction cache ها می‌خورد ، چون دسترسی مان به inst ها معمولاً ترتیبی و روش دوم بیشتر به درد Data cache ها می‌خورد ، چون دسترسی مان به Data ها معمولاً بصورت رندم است .

سیاست‌های نوشتن



وقتی cpu یک آدرسی به ccu میدهد و میگوید برو و در آن آدرس دیتا ذخیره کن و آن آدرس در کش وجود دارد (hit) چهار حالت برای بشر متصور است :

۱- در هیچ کدام از m.m و کش ننویسد X

۲- در m.m بنویسد و در کش ننویسد : X چون طبق LoR زمانی که cpu الان می‌خواهد در یک خانه‌ای از حافظه بنویسد ، احتمال در آینده نیز می‌خواهد دوباره در آن بنویسد یا از آن بخواند

۳- فقط در کش بنویسد : write back (copy back) (پس نویسی)

۴- در m.m و کش بنویسد ، write through (گام نویسی)

- موقعی که miss رخ میدهد ۲ روش وجود دارد

۱- write allocate : بلاک حاوی کلمه مورد نظر را به کش منتقل می‌کنم

۲- write around = no write allocate : بلاک به کش منتقل نمی‌شود

نکته) write back همیشه از روش write allocate استفاده می‌کند ولی write through می‌تواند از ۲ روش nwa و swa استفاده کند که معمولاً از روش nwa استفاده می‌کند

write through: وقتی که cpu می‌خواهد بنویسد و انا خ میدهد می‌توانیم فقط بلاک درون کش را آپدیت کنیم (کاری که روش write back انجام می‌دهد) اما در اینصورت دیتا درون کش و m.m inconsistent خواهد بود، بنابراین در write through داریم:

زمانی که hit خ میدهد (که یک t_c طرف می‌شود برای مقصین این موضوع) ما هم بلاک کش را آپدیت می‌کنیم و هم بصورت موازی کلمه مورد نظر را در m.m می‌نویسیم و چون t_m از t_c بیشتر است، زمان اینکار برابر t_m خواهد بود

حال اگر miss خ بدهد ۲ حالت ممکن است خ دهد، بسته به اینکه از چه سیاستی استفاده کنیم

۱- write allocate: ابتدا بلاک حاوی آن کلمه را از m.m می‌خوانیم، سپس همزمان که بلاک آپدیت شده را در کش می‌نویسیم کلمه مورد نظر را نیز در حافظه می‌نویسیم $t_c+t_b+t_m$

۲- no write allocate: بلاک از حافظه اصلی به کش منتقل نمی‌شود و cpu کلمه مورد نظر را فقط در m.m می‌نویسد t_c+t_m

نکته) پیاده‌سازی این روش بسیار آسان است، اما کارایی خوبی ندارد، چون در این روش با هر درخواست write باید در m.m هم بنویسیم و این روش پهنای باند حافظه را خیلی اشغال می‌کند

نکته) در زمانی‌هایی که با write hit ها پشت سر هم در یک آدرس، مانند زمانی‌هایی که یک counter دارد increment می‌شود این روش خیلی به است

مثال برای نشان دادن کارایی پاسخ این روش.

فرض کنید که از مکانیزم ack استفاده می‌کنیم، اگر ۱۰ درصد دستوراتی که داریم اجرا می‌کنیم عملیات store در حافظه را انجام بدهند و هر write در حافظه ۱۰۰ کلاک طول بکشد. همچنان فرض کنید که CPI بدون cache miss برابر ۱ باشد (Base CPI=1) در اینصورت CPI کلی را محاسبه کنید

به ازای هر دستور بطور متوسط ۱۱ کلاک می‌خواهیم ⇐ effective CPI = 1 + 0/1 x 100 = 11

* یک راه حل برای رفع این مشکل نداشتن write Buffer است. WB همین است که داده‌هایی که قرار است در M.M نوشته بشوند را نگه می‌دارد، در این صف داده‌های جدید در حالی وارد می‌شوند که داده‌های قدیمی در انتظار ارسال به حافظه هستند. اگر CPU می‌خواهد در حافظه بنویسد لزومی ندارد که متوقف بشود تا عملیات write انجام بشود، می‌توانیم داده‌ای بنام WB نگه داریم که همزمان که در کش می‌نویسد در WB نیز بنویسد و بعد از آن CPU می‌تواند به کارش ادامه دهد.

نکته) اگر write buffer پر شود که زمانی ممکن است رخ بدهد که write بصورت burst اتفاق بیفتد و CPU به یک write برسد، در اینصورت CPU باید تا زمانیکه یک خانه WB خالی می‌شود صبر کند (پردازنده باید stall کند) ⇐ برای جلوگیری از این اتفاق به جای اینکه طول یک WB را زیاد کنید WB های بالاتر از L1 ورودی قرار می‌دهند

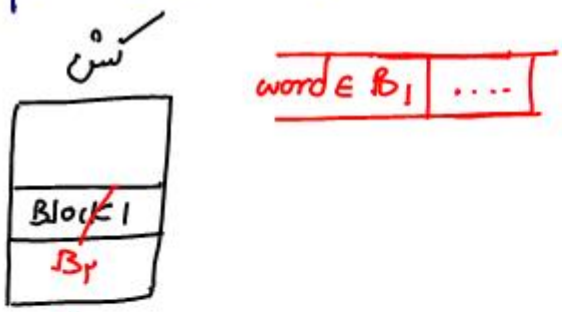
نکته) اگر برنامه (زبان) که حافظه می‌تواند Swrite را در خودش بنویسد کمتر از نرخ تولید write توسط پردازنده باشد، آن گاه هیچ مقداری از بافر نمی‌تواند به ما کمکی بکند

نکته) nwa در جاهایی مناسب است که می‌خواهیم تعداد زیادی آدرس را مقداردهی اولیه کنیم که بعدها ازشون استفاده می‌کنیم. مثلاً وقتی می‌خواهیم یک آرایه بزرگ را initialize کنیم

نکته) در سیاست wt در زمانی که یک core داریم، اگر چند لول کش داشته باشیم وقتی به کش سایزیت می‌رسیم همان موقع باید برویم و بقیه لول‌های حافظه‌مان تا m.m آپدیت کنیم

نکته) این سیاست برای سیستم‌هایی که چند تا core به دیتای دسترسی دارند مناسب است. در سیستم‌های multi core که هر core کش خودش را دارد باید حواسمان به invalidate کردن سایر کش‌ها باشد

نکته) موقعی که یک موجودی مانند DMA یا I/O processor می‌خواهد بصورت مستقیم از حافظه اصلی بخواند باید حواسش به write buffer باشد و یا در حالتی که cpu خواسته کلمه‌ای را بخواند، ccu باید بصورت مداوم کش و wB را بگردد تا مراقب حالتیکه کلمه در کش نیست ولی در wB هست باشد. در واقع باید wB را بخشی از m.m در نظر بگیریم، یعنی هروقت خواستیم از m.m چیزی بخوانیم حتما باید اول wB را چک کنیم



① write word ∈ Block 1 ② read word ∈ B2 : MISS ⇒ B1 (cache) ← B2 (m.m)

③ read w ∈ B1 & MISS ⇒ اگر بریم کلمه‌ای که در m.m هست را به cpu بدهیم غلط است

write back

فرض کنید پردازنده آدرسی داده و می‌خواهد write کند، اگر اینا پیش بیاید پردازنده فقط در کش می‌نویسد (و بیت dirty را یک می‌کند) و اگر miss رخ بدهد، ابتدا بلاک حاوی کلمه از m.m به cache منتقل می‌شود سپس بلاکی که حاوی کلمه آپدیت شده است در کش نوشته می‌شود و در اینجا نیز dirty bit یک می‌شود حال وقتی بخواهیم بلاکی از کش را دور بریزیم ابتدا dirty bit اش را نگاه می‌کنیم، اگر صفر بود با خیال راحت دور می‌ریزیم و اگر ۱ بود ابتدا آن بلاک را در حافظه می‌نویسیم و سپس می‌توان روی آن بلاک کش بلاک دیگری گذاشت

نحوه تعیین db: اینکه cache چطور باید db ها را تعیین کند، مانند قبل اگر داده کش و m.m یکی باشد این بیت ۰ است و اگر یکسان نباشد ۱ است

ما هنگامی که در کش می‌نویسیم ممکن است یک بلاک کش تغییر کند. حال نوشتن در کش به یکی از دو دلیل زیر اتفاق می‌افتد

۱- open نشده بنویسیم ← dirty bit=1

۲- " " بخوانیم ← در حالتی که miss رخ بدهد ← dirty bit=0

توضیح ۲) چون هر وقت این اتفاق می‌افتد miss رخ داده و cache رفته و از حافظه یک کپی برداشته که توی کش بگذارد پس آن چیزی که در کش می‌گذارد با کسی که در حافظه است یکسان است

استفاده از db: ما وقتی از این بیت استفاده می‌کنیم که یک خانه کش را می‌خواهیم replace کنیم

نکته) به dirty bit، modify bit و یا بیت دست خوردگی نیز می‌گویند

معماری کامپیوتر

رامین رضوی

نکته) سیاست write back موجب افزایش کارایی می‌شود مخصوصا موقعی که پردازنده می‌تواند چند write پی‌درپی تولید کند سریعتر از اینکه write ها توسط m.m هندل می‌شوند

نکته) پیاده سازی اش از write through پیچیده‌تر است.

نکته) در write back هم می‌توان از write buffer استفاده کرد

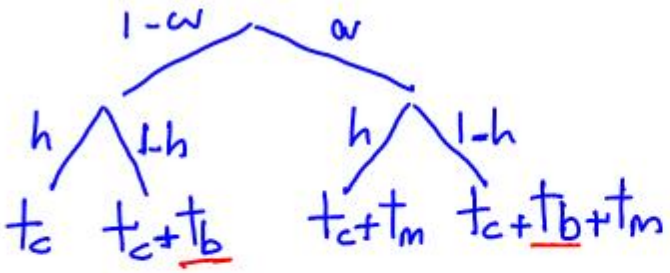
نکته) معمولا در کامپیوترهایی که ما استفاده می‌کنیم L_1 cache از نوع write back است و L_2 cache از نوع write through است.

نکته) در هنگام خواندن برای کاهش مصرف انرژی بصورت سری می‌خوانیم ولی وقتی می‌خواهیم در m.m دنبال بنویسیم بصورت موازی این کار را می‌کنیم

سوال- t_{avg} را در سیاست wt وقتی از wa استفاده می‌کند محاسبه کنید

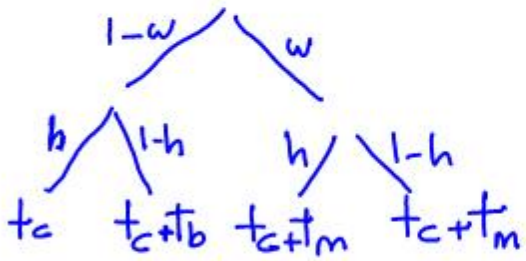
$w=$ درصد نوشتن‌ها $\Rightarrow$ چقدر از درخواست‌هایی که cpu میدهد نوشتن و چقدر خواندن است

عملکرد سری



$$t_{avg} = t_c + (1-h)t_b + wt_m$$

سوال) سوال ۲ را در حالت wa حل کنید



$$t_{avg} = t_c + (1-w)(1-h)t_b + w t_m$$

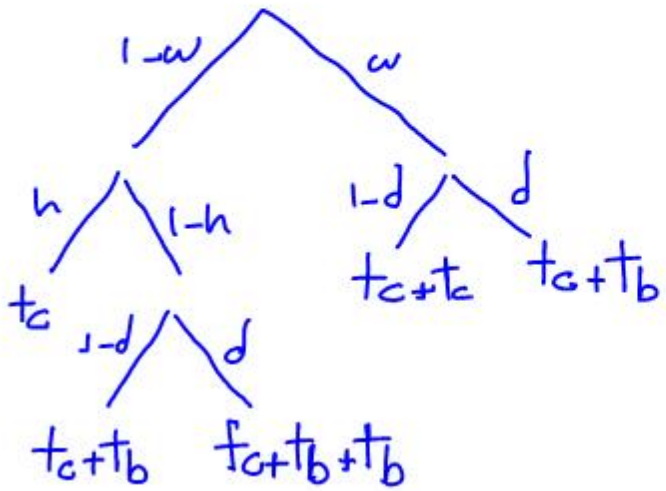
سوال: t_{avg} را برسیاست w_b بدست آورید؟

احتمال آن که بلاکی که درکش است از زمان حضورش درکش تاکنون تغییری داشته یا نه $= w_b$

$w_b \leq w \rightarrow$ درصد نوشتن ها

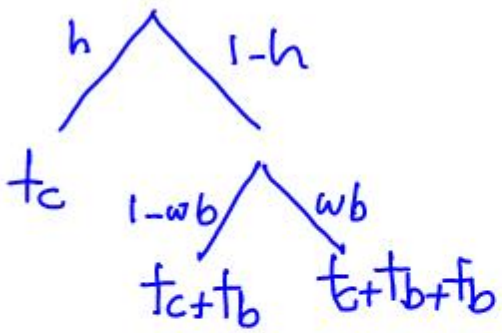
t_c : hit
t_{c+t_b} : miss
} خواندن

t_c : hit
t_{c+t_b} : miss
} نوشتن



$w_b \leftarrow$ درکش می‌نویسم

$w_t \leftarrow$ هم کدش می‌نویسیم و هم در m.m



معماری کامپیوتر

رامین رضوی

کش به ۳ روش ساخته می‌شود

۱- Direct map

۲- Fully associative ـ (انجمنی) associative

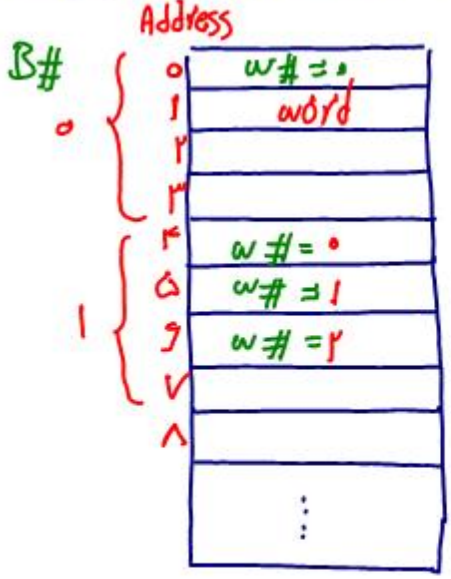
۳- set associative

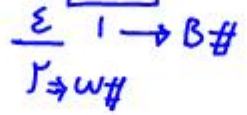
مقدمات

cache size : تعداد بلاک‌های کش

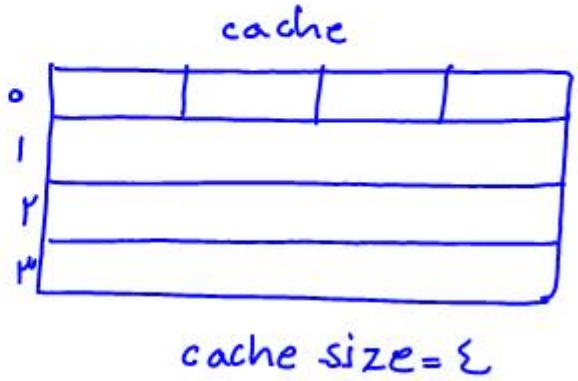
Block size : هر بلاک کش از چند word تشکیل شده

- اگرچه بلاک برای حافظه‌ست و برای کش است ولی خانه‌های حافظه را (چند word) به بلاک‌ها دسته‌بندی می‌کند
- Blockهای حافظه شماره‌گذاری شدن از ۰ است، همان طور که indexهای درون کش هم شماره‌گذاری شدن از ۰ است. که به این شماره‌ها B# می‌گوییم
- wordهای داخل یک بلاک شماره‌گذاری می‌شوند که به آنها word# (w#) می‌گوییم، w# نیز از ۰ شروع می‌شوند





مثال Block size = ۴ و



* آدرسی که cpu تولید می‌کند و miss می‌شود، cache باید بفهمد که آن آدرس در کجا main است و در کدام بلاک است و آن بلاک شامل چه کلمه‌هایی است.

Decimal

۵۳ | ۸ ⇒ Block size
۴۸ | ۶ ⇒ B#
۵ ⇓
w#

مثال: Block size = ۸

- cache می‌آید تقسیم کند، چون عمل تقسیم بسیار کند است و باعث کند شدن کش می‌شود. راه‌حل: در هر مبنایی تقسیم کردن بر توان‌هایش مبنا بسیار راحت است و فقط با جدا کردن یکسری رقم (سمت راست) از اعدادمان امکان پذیر است

⇓

با توجه به اینکه آدرس‌های cpu باینری است و مبنایش ۲ است حتما باید Block size توانی از ۲ باشد.

۵۳ = ۱۱۰۱۰۱ (۵ ۴ ۳ ۲ ۱ ۰)
باقیمانده ← w# | ← حاصل قسمت B#

کلمات این بلاک = ۴۸، ۴۹، ۵۰، ۵۱، ۵۲، ۵۳، ۵۴، ۵۵

آدرس: [B# | w#] ← n بیت

Block size = 2^n

نکته) Block size را توانی از دو می‌گیرند تا واحد کنترل کش برای به دست آوردن Block number و word number نیازی به تقسیم کننده نداشته باشد

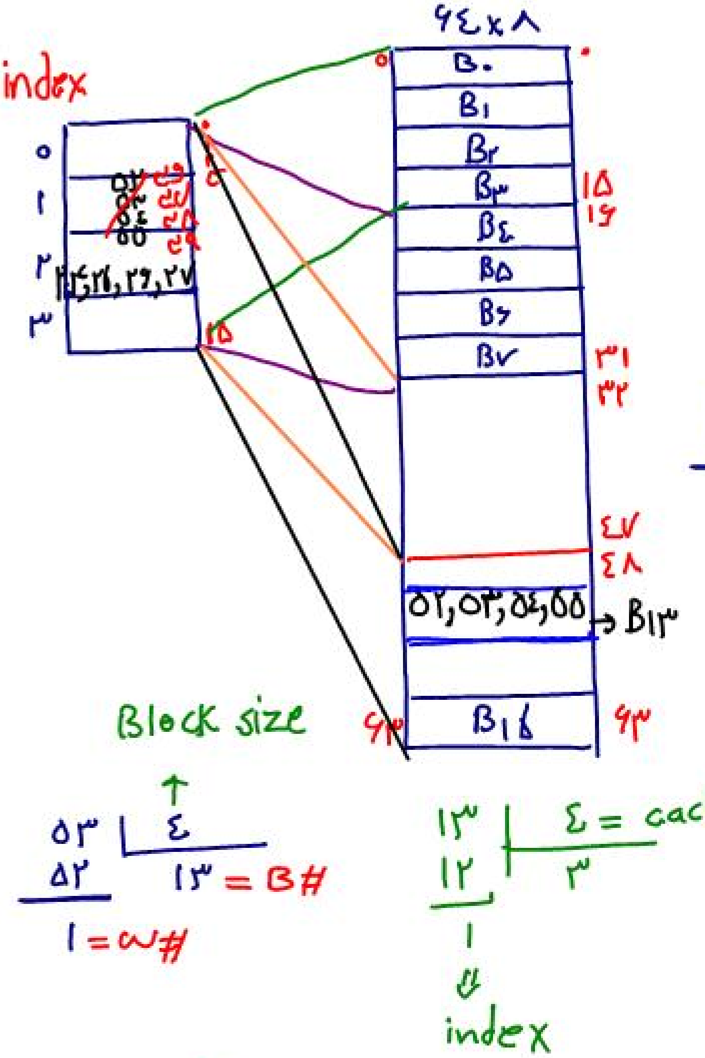
نکته) چرا cache نیازمند B# و w# دارد؟ چون باید با توجه به آدرسی که در اختیار دارد بتواند بفهمد که باید چه کسانی را prefetch کند

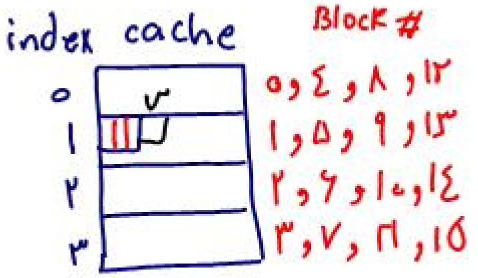
Direct map

در این روش جای هر بلاک در کش مشخص است

یعنی فرمول نگاشتی برای این کار وجود دارد که طبق این فرمول برای هر بلاک واقع در حافظه یک index در کش تولید می‌شود (آدرس یک بلاک در کش تولید می‌شود)

Block size = 4





$53 \div 4$: $13 = B\#$, $1 = w\#$

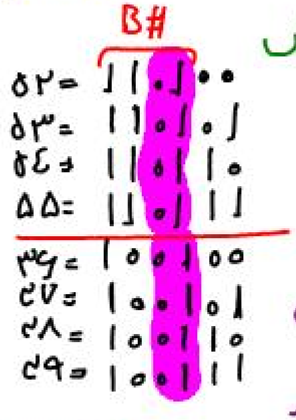
$27 \div 4$: $6 = B\#$, $3 = w\#$

$37 \div 4$: $9 = B\#$, $1 = w\#$

cpu = 53, 55, 27, 37 (m, h, m, m)

$53 \div 4$: $13 = B\#$, $1 = w\#$

$13 \div 4$ (4 = cache size): 3، باقیمانده 1 ← index



* آدرس‌های هم بلاک B# یکی است

* بلاک‌های هم ایندکس، ایندکس‌های یکسانی دارند

53 = | 1 1 | 0 1 | 0 1 | (B#، w#، index)

آدرس: | tag | index | w# |

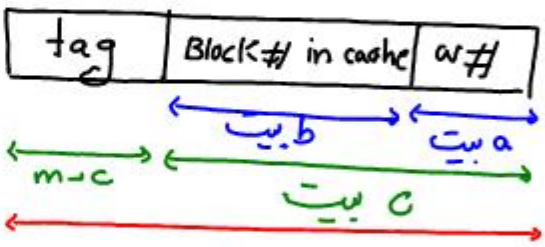
index = آدرس بلاک در حافظه اصلی (B#) mod تعداد بلاک‌های کش (cache size) = آدرس بلاک در کش

" = کلمه " " mod کلمات کش " " = " کلمه در کش " "

Address | Block size ⇒ باید توانی از ۲ باشد
B# in m.m | cache size = # of Block in cache
tag
w#
offset
word offset
displacment
کدام کلمه آن بلاک
index = Block # in cache

حتما توانی از ۲ است

تعداد کلمات حافظه اصلی = 2^m
حجم کش = 2^c word
Block size = 2^a



⇒ cache size = تعداد بلاک های کش = $\frac{2^c}{2^a} = 2^{c-a}$ ⇒ $b = c-a$
2^b

* آیا می توان مستقیم tag را بدست آورد ؟

$tag = \log \frac{\text{حجم حافظه}}{\text{حجم کش}} = \log \frac{2^m}{2^c} = \log 2^{m-c} = m-c$

نکته) وقتی حافظه ۴ برابر کش است ، در هر index احتمال حضور ۴ تا وجود دارد که بلاک آنها تمایز قائل شویم ⇐ $\log_2$ این تعداد ، بیت نیاز داریم

سوال - سایز کش در مثال قبل را بدست آورید

Data+tag+valid
$16 \times 8 + 4 \times 2 + 4 \times 1 =$

نکته) اگر گفتند سایز کش n است در سوالی منظور سایز Data اش است

معماری کامپیوتر

رامین رضوی

نکته - به سیاست write through، هم نویسی هم می‌گویند

نکته - به بلاک‌های کش line هم می‌گویند (cache line)

نکته - داخل حافظه با یک مقایسه کننده داریم، و می‌دانیم مقایسه کننده هرچه بزرگتر باشد کندتر می‌شود، بنابراین به‌جای آنکه از B# استفاده کنیم، قسمت کم ارزش B# را به‌عنوان Lable برای هر بلاک استفاده کنیم، چون بلاک‌ها هم index، قسمت کم ارزش B# تشان با هم مساوی است (همان index)، از tag برای lable زدن استفاده می‌کنیم. بنابراین تعداد بیت‌های مقایسه کننده به اندازه تعداد بیت‌های tag است.

آدرس | B# | w#

مثال - فرض کنید حافظه اصلی 1M x 16 bit است، هر کلمه ۱۶ بیت است، کش دو گیگ کلمه دارد هر بلاک ۶۴ بایت است، فرمت آدرس در نگاشت مستقیم را به‌دست آورید؟

1 word = 16 bit = 2 Byte

1 Block = 64 Byte = 32 word

حجم حافظه اصلی = $1M \times 16bit = 2^{25}$ word

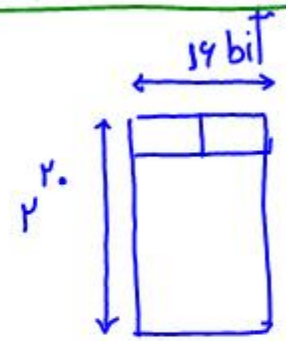
حجم کش = $2K word = 2^{11}$ word

تعداد بلاک‌های کش = $\frac{\text{حجم کش}}{\text{اندازه بلاک}} = \frac{2^{11} word}{2^{5} word} = 2^{6}$

$tag = \log \frac{2^{20} word}{2^{11} word} = \log 2^{9} = 9$

Lable

tail 20



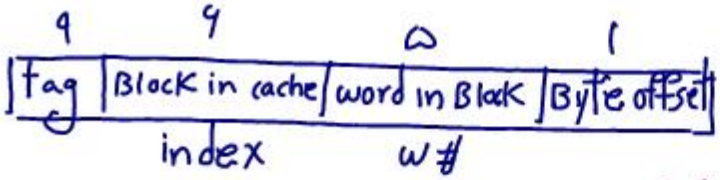
معماری کامپیوتر

رامین رضوی

مثال - فرض کنید در مثال بالا پردازنده به بایت آدرس بدهد ← پردازنده بایت آدرس پذیر است

- اگر پردازنده بایت آدرس پذیر است یعنی واحد خواندن و نوشتن cpu بایت است

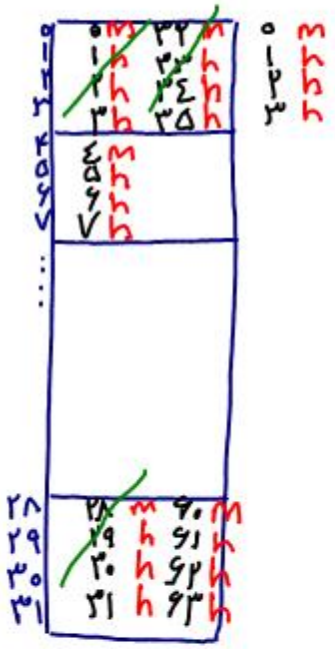
- اگر واحد آدرس پذیر شما با اندازه word شما یکسان نباشد، مثل همین مثال، آن‌گاه فیلد دیگری بنام Byte offset (Byte in word) به سمت راست فرمت آدرس مان اضافه می‌شود.



* حواستان باشد ! بیت اضافه را به تگ اضافه نکنید

$1M \times 16\,bit = 1M \times 2\,Byte = 2^{21}\,Byte$

مثال- فرض کنید یک کش شامل ۸ بلاک ۴ کلمه‌ای است، پردازنده‌ای از آدرس ۰ تا ۶۳ را ۱۰ بار متوالیا تولید می‌کند، hit rate را محاسبه کنید؟

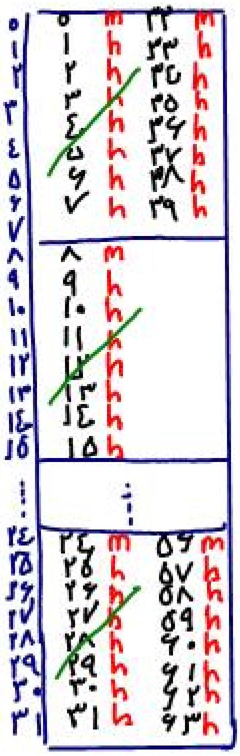


cpu = M ۰ و ۱ و ۲ و ۳ و ۴ و ۵ و ۶ و ۷ و ۸ و .... و ۶۳

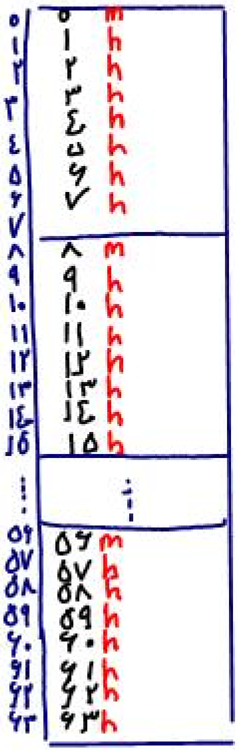
$hit\ rate = \frac{3}{4} = 0,75$ $miss\ rate = 1 - \frac{3}{4} = \frac{1}{4}$

$hit\ rate = \frac{\quad}{10 \times 64}$

با فرض ۴ بلاک ۸ کلمه‌ای



با فرض ۸ بلاک ۸ کلمه‌ای



$$\text{miss rate} = \frac{8}{10 \times 64} = \frac{1}{80}$$

$$\text{hit rate} = 1 - \frac{1}{80} = \frac{79}{80}$$

$$\text{hit rate} = \frac{7}{8} = \%87.5$$

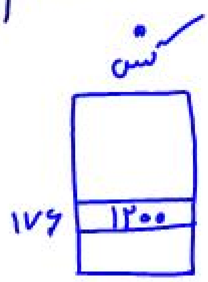
سوال - فرض کنید حافظه اصلی 4K × 8bit (word) است، کش دارای ۶۴ بلاک است و هر بلاک ۱۶ بایت است، آدرس ۱۲۰۰ در کدام بلاک کش قرار دارد؟

روش ۱-

$$64 \times 16\,\text{word} = 2^{10}\,\text{word} = \text{حجم کش}$$

(۶۴: تعداد بلاک کش، ۱۶: اندازه هر بلاک)

۱۲۰۰ در کدام سطر کش است $= 1200 \bmod 1024 = 176$



$$176 \div 16 = 11,\ \text{remainder } 0$$

$11 = \text{index} = B_{11}$ in cache

روش ۲-

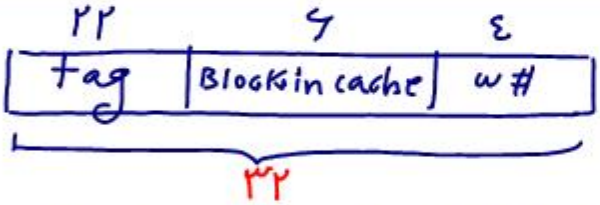
۱- اول ببینیم ۱۲۰۰ در کدام بلاک m.m است

۲- آن بلاک m.m در کدام بلاک کش می‌افتد

1- $1200 \div 16 = 75 = B\#$، باقیمانده $0 = w\#$

$75 \div 64 = 1$، باقیمانده $11 = \text{index}$

روش ۳- فرمت آدرس را بدست می آوریم



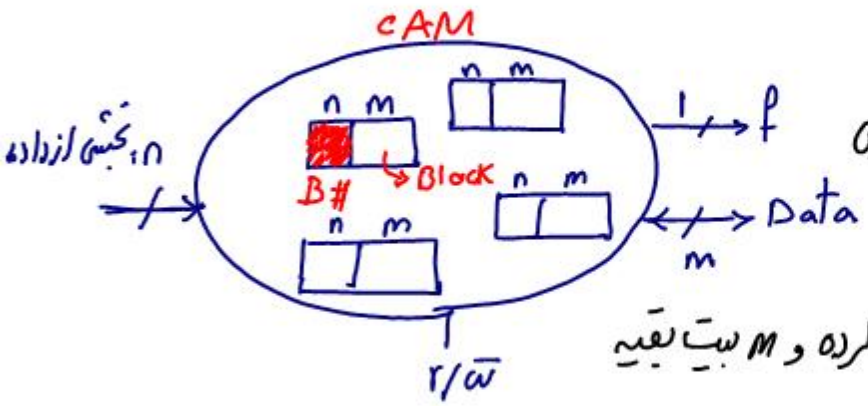
حجم حافظه = ۶۴ word = $2^{2} \times 2^{4} = 2^{6}$ word

۱۰ ۹ ۸ ۷ ۶ ۵ ۴ ۳ ۲ ۱ ۰
۱۲۵۰ = ۰۰۰۱ ۰ ۰۱۰۱۱ ۰۰۰۰
index

Associative

کش associative از یک sRAM خاص بنام CAM (content addressable memory) استفاده می کند، CAM حافظه ای است که آدرس ندارد (مانند مغز انسان) و هروقت که ما می خواهیم داده ای را از درون اش پیدا کنیم باید بخشی از داده را بهش بدهیم، حالا اگر آن داده درون CAM باشد، CAM برما بقیه داده را میدهد، اگر هم نباشد می گوید نیست

مثلاً اگر داده های ما n+m بیتی باشند ما (CCU) n بیت را بعنوان ورودی به CAM می دهیم و CAM بصورت موازی (مانند مغز) n بیت ورودی را با همه داده های درون اش مقایسه می کند و اگر آن داده درکش باشد بقیه اش را به ما میدهد، پس به تعداد تمام عناصر حافظه CAM مقایسه کننده لازم است تو) چون درکش هر خانه یک بلاک است، تعداد مقایسه کننده ها به تعداد بلاک ها است و مقایسه کننده ها هرکدام tag بیتی هستند



۱- اگر f=0 باشد یعنی داده در CAM نبوده و مقداری روی پایه دیتا معتبر نیست

۲- اگر f=1 باشد، یعنی داده را در CAM پیدا کرده و m بیت بقیه داده را روی پایه Data قرار داده

معماری کامپیوتر

رامین رضوی

توجه) ما در معماری کامپیوتر حالتی که داده‌ای که به ورودی CAM می‌دهیم با چند تا match بشود را نداریم

نکته - CAM از جنس SRAM است ولی از SRAM هایی که بصورت عادی استفاده می‌شود خیلی گران‌تر است. چون به تعداد خانه‌هایش مقایسه‌گر دارد.

نکته - از کش Associative در کامپیوترهای ما استفاده نمی‌شود چون نه تنها گران است بلکه چون مقایسه‌گرهای زیادی دارد سخت افزار حجیمی دارد و همین طور باعث می‌شود توان مصرفی سیستم بسیار بالا رود و ...

نکته - CAM از SRAM های معمولی بسیار کندتر است حدود ۲ تا ۵ برابر کندتر است

- CAM وقتی می‌خواهد کسی از کش اخراج کند، LRU را اخراج می‌کند، در واقع مکانیزم CAM replacement LRU (least recently used) است.

توجه) این را نگفتیم اخیرا گفته استفاده نشده ← به اشتباه به همه اشاره می‌کند

LRU: یعنی کسی (بلاکی) که از آخرین استفاده‌اش بیشترین زمان گذشته باشد

نکته) با توجه به اینکه پردازنده در رفتارش LOR زمانی وجود دارد بهتر است برای replace کردن بدون کش عنصر LRU را replace کنیم.

نکته) در حافظه‌های عادی ما وقتی می‌خواهیم از کش بنویسیم هم دیتا و هم آدرس را می‌دهیم کش، اما مار CAM آدرس نداریم و بنابراین نمی‌توانیم به CAM بگیم در کجا بنویسد و بنابراین موقعی که CAM می‌خواهد یک بلاک را در خودش بنویسد و جا ندارد، آن را بطور پیش فرض روی عنصر LRU می‌نویسد

چگونه عنصر LRU را پیدا می‌کنه CAM

- بیایم و یک ساعت در CAM قرار بدهیم و به بلاک ها بر اساس زمان دسترسی بهشون label بزنیم و این ساعت باید در حد ns دقت داشته باشه زیرا access ها در cpu به بلاک با سرعت بالایی انجام میشود

به ۲ دلیل این کار را نمی‌کنیم:
① نیازمند پیاده سازی یک ساعت بسیار دقیق دارد این روش
② این زمان دقیق باید برای هر خانه نگهداری شود که سربار حجیم زیادی را به هر خانه CAM اعمال می‌کند

راه حل

ما برای اینکه بفهمیم عنصر LRU چه کسی است نیازی نداریم که زمان دقیق دسترسی ها را بدانیم، بلکه فقط کافی است ترتیب دسترسی ها را بدانیم به این ترتیب سربار به شدت کاهش می‌یابد و در این حالت ما برای هر بلاک کافی است فقط به تعداد $\log$ تعداد بلاک ها کش بیت نگه داریم

الگوریتم نوشتن در CAM: به اینصورت است که دور عنصر LRU می‌زنید که عنصر LRU دارای بزرگترین برچسب است (time stamp) در اینصورت عنصر جدید برچسب ۰ به خودش می‌گیرد و برچسب سایر بلاک یکی زیاد می‌شود

مثال - فرض کنید ۸ بلاک در کش دارید:

سه بیت برای Time stamp کافی است

معماری کامپیوتر

رامین رضوی

الگوریتم خواندن در CAM : در هنگام خواندن هر عنصری ممکن است خواهد خوانده شود، حال برچسب عنصری که خوانده می‌شود را ۰ می‌کنیم و برچسب بلاک هایی که ترتیبشان از بلاک خوانده شده کمتر است را ۱ واحد زیاد می‌کنیم

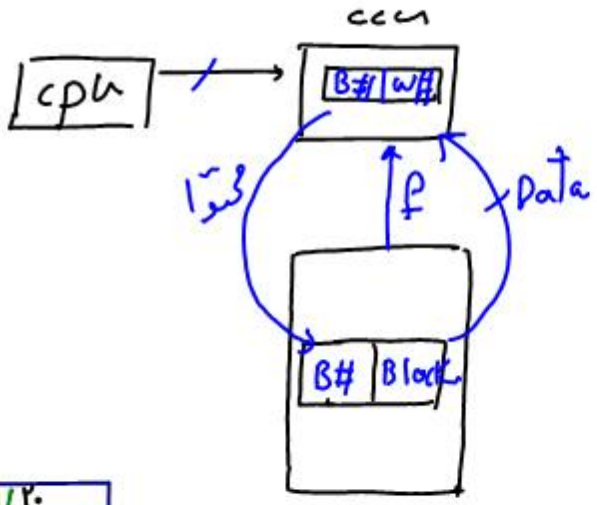
cpu عنصر ۴ را می‌خواند

۷ ، ۶ ، ۵ ، ۴ ، ۳ ، ۲ ، ۱ ، ۰

۱ ۲ ۳ ۴

* ما قبلاً گفتیم که برای آنکه بدانیم هر بلاکی که در CPU است، کدام بلاک حافظه اصلی است (بدانیم صاحب اش چه کسی است) نیاز هست که B# هر بلاک را همچون lable در کنار بلاک مورد نظر در CPU بنویسیم

مشتقی ما در Direct می‌توانستیم صرفنظر کنیم و کل B# را بنویسیم و از قسمت index صرف نظر کنیم ولی CAM، index ندارد که بتوانیم از آن صرف نظر کنیم، بنابراین در اینجا مجبوریم کل B# را همچون tag در نظر بگیریم

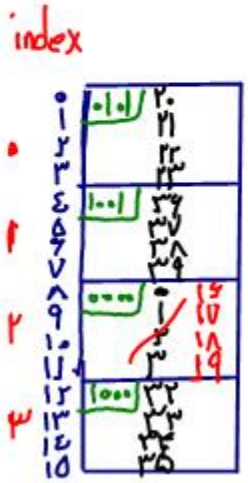


- فرض کنید کشی شامل ۴ بلاک ۴ کلمه‌ای داریم، حافظه اصلی نیز ۶۴ کلمه ۸ بیتی دارد ← حافظه ۱۶ بلاک دارد

cpu آدرس های ۰ تا ۶۳ را تولید می‌کند در ۶ بیت

cpu = 21, 38, 1, 36, 23, 33, 17

m m m h h m



21	۴
20	۵
1 = w#	

۴ ۲
tag | w#
B#

حجم کش = ۱۶×۸ + ۴×۴ + ۴×۱
Data tag valid

سیاست های جایگزینی (Replacement policy)

آن سیاستی است که ما استفاده می‌کنیم برای evict کردن یک line (بلاک)

victimize = replace = cast out = evict *

* وقتی کش جا ندارد و ما دنبال یک بلاک متعلق به یک آدرس می‌گردیم که در کش نیست و کش پر است می‌گوییم capacity miss اتفاق افتاده

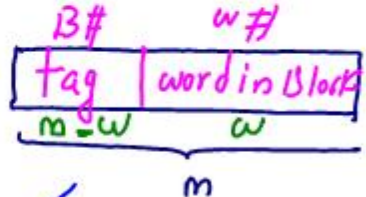
انواع سیاست های جایگزینی

: FIFO
: LRU
: LFU least frequently used : کسی که اخیرا کمتر استفاده شده
: Random ← از روش‌های psedu random استفاده می‌شود ← مواردی که رندم واقعی تولید کند نداریم

کارایی شون زیاد خوب نیست
: MRU most recently used
: MFU " frequently "
: NRU not recently "
: NMRU not most " "
: optimal : اونی را بیرون که در آینده دیرتر استفاده می‌شود

فرمت نگاشت آدرس در کش انجمنی

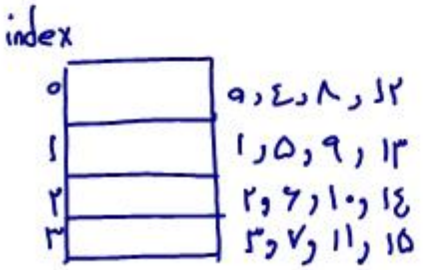
تعداد کلمات حافظه اصلی = 2^m
تعداد بلوک‌های m.m = 2^w " " " بلاک



بیت‌های tag مستقیم = $\log \frac{\text{اندازه m.m}}{\text{اندازه بلاک}} = m-w$

نکته: در این فرمت هیچ اثری از حجم کش نیست

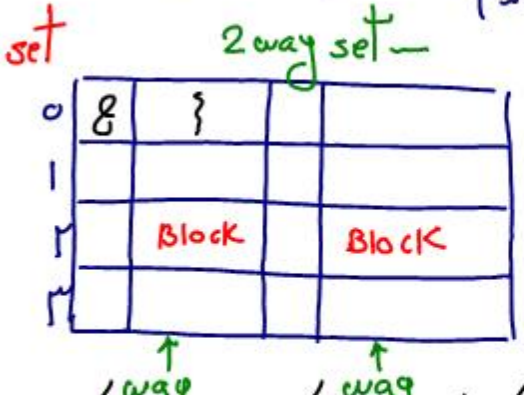
- Direct map کش خیلی خوبی است ولی در یک شرایط خاص ممکن است به عمل کند، زمانی به عمل می‌کند که cpu دو آدرس متوالی که تولید می‌کند، آدرس هایی باشند که باهم، هم index اند، که معنی اش این است که دو تا خانه حافظه را دست زدیم که به اندازه سایز کش ازهم دورند



نکته) فاصله دو خانه هم index به اندازه طول کش یا مضربی از کش است که معمولا با توجه به LOR مکانی که در حافظه پردازنده وجود دارد خیلی بعید است که cpu، با دو خانه با چنین فاصله‌ای کار داشته باشد، زیرا طول کش ها واقعی شلا MB ٤ است در کامپیوترهای multi task این اتفاق اصلا بعید نیست زیرا در این سیستم ها task ٤ در حافظه پخش هستند و هر task در یک جایی قرار دارد و اتفاقا ممکن است فاصله این task ٤ به اندازه مضاربی از سایز کش باشد بنابراین وقتی context switch می‌کنیم داده یک task ممکن است داده تسک دیگری را از کش اخراج کند

set associative

بیایم و برای رفع مشکل بالا بیایم و از هر index چند بلاک بذاریم. بنابراین در اینجا یک خانه کش شامل چند بلاک است.



۰ و ٤ و ۸ و ۱۲
۱ و ۵ و ۹ و ۱۳

فرض کنید cpu آدرس یک word را تولید می‌کند که آن word در بلاکی است که آن بلاک الان قرار دارد باشد در کش باشد جایش در set ۰ کش است، حال mmu کل set ۰ را می‌خواند (چون هر set یک خانه SRAM است) و حال mmu، تگ آدرس آن که cpu تولید کرده را بصورت موازی با tag بلاک های درون set ۰ مقایسه می‌کند و اگر hit بود word مورد نظر را به کش

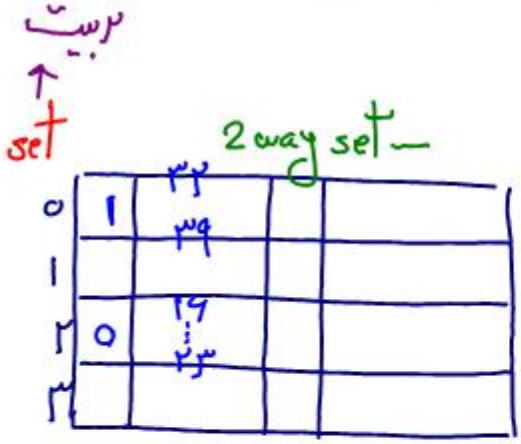
معماری کامپیوتر

رامین رضوی

می دهد و اگر miss بود بلاک شامل word مورد نظر از حافظه به cache آورده می شود و بعد با توجه به یک سیاست جایگزینی جای یکی از بلاک ها آن ست می نشیند و آن ست هم توسط cache در کش نوشته می شود

* هرخانه کش یک set associative یک set است.

مثال: فرض کنید سایز هر بلاک ۸ باشد و همین طور کش ۸ تا بلاک دارد و کش ما 2-way است. حال فرض کنید شروع آدرس ۲۳ تولید شده است.



۰,۴,۸,۱۲
۱,۵,۹,۱۳
۲

۲۳ | ۸ = B-size
۱۶ | ۲ = B#
۷ = w#

۲ | ۴ = cache size
۰ | ۰
۲ = index

تعداد set های کش

s# in cache

۳۷ | ۸
۳۲ | ۴ = b#
۵

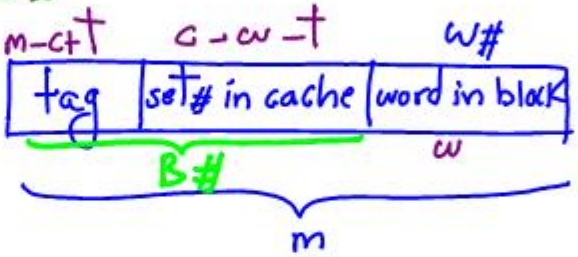
۴ | ۴
۴ | ۱ = tag
۰ = index

(تعداد set های کش) mod (شماره بلاک در حافظه) = جای بلاک i از حافظه در چه set از کش است

تعداد سطر کش = i mod جای کلمه i از حافظه در چه سطری از کش

شماره set کش = $\left\lfloor \frac{K}{m} \right\rfloor$ ← فرض کن که گفته در سطح K کش و فرض کن که هر بلاک m کلمه دارد

فرمت آدرس



تعداد کلمات حافظه اصلی = 2^m
" " " کش = 2^c
" " " بلاک = 2^w
K-way = 2^t

کش چند بلاک دارد = $\frac{2^c}{2^w} = 2^{c-w}$
" " " ست = $\frac{2^{c-w}}{2^t} = 2^{c-w-t}$

معماری کامپیوتر

رامین رضوی

* محاسبه tag بصورت مستقیم

$$tag = \log \frac{\text{حجم حافظه}}{\text{حجم کش}} + \log K = 2^t$$

سوال- آیا مکانیزم replacement در اینجا LRU است؟ خیر - چون ما که LRU کل کش را پیدا نکردیم و فقط LRU را بین افراد یک ست پیدا می‌کنیم

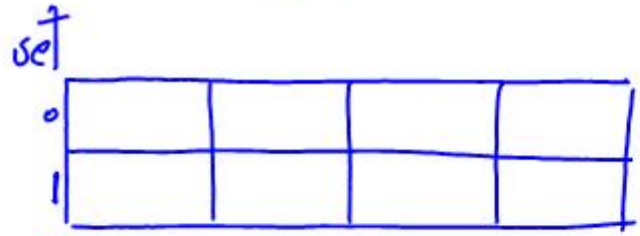
← بنابراین hit rate ش از Direct بهتر است ولی از associative بدتر است زیرا داده کش تمام انجمنی، LRU کل کش را اخراج می‌کردیم.

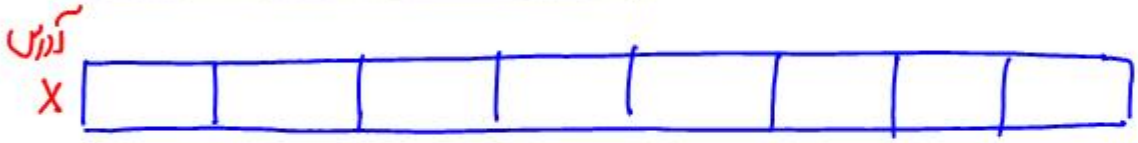
نکته: در یک کش 4way set associative ما اگر تا ۴ تا تسک موازی داشته باشیم مطمئن هستیم که به مشکل برنمی‌خوریم. حتی اگر ۵ تا تسک موازی هم داشته باشیم خیلی خیلی باید بدشانس باشیم که هم این ۵ تا دوبه دو فاصله‌شون اندازه مضربی از کش باشه.

← معمولا بیشتر از 4way نمی‌گیرند ← به دلیل بالا

← هرچقدر 4way را بیشتر کنیم گمان گران‌تر می‌شود چون تعداد تقایسه کننده‌ها موازی بیشتر می‌شود

نکته: با فرض ثابت بودن حجم کش





⇒ Associative

set associative حالت خاصی از DM است: پس Direct map: 1 way = 1

Fully Associative ← است ← کل کش = تعداد بلاک‌های کش } K

معماری کامپیوتر

رامین رضوی

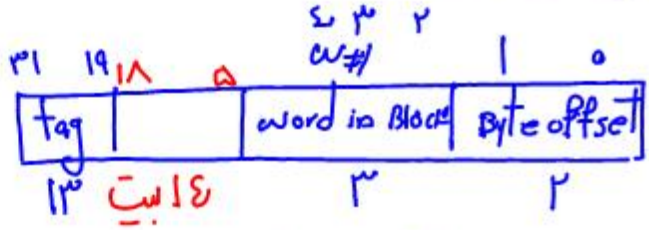
مثال - فرض کنید یک کش داریم ساتز 512K word است (در قسمت دیتا) و در آدرس دهی کش آدرس فیزیکی بصورت زیر تفسیر می شود

bit 0-1 : byte offset

bit 2-4 : word "

bit 19-31 : tag

ساختار این کش به چه صورت است؟



این ذیت یا می توان Direct باشد یا set Associ- . پس در واقع سوال این است که این Direct است یا set

هر بلاک = 8 word

آدرس بایت را بدهد ⇒ 1 word = 4 Byte آدرس بایت cpu پذیرا است

سایز کش = 512K word = $2^9 \times 2^{10}$ word = 2^{19} word

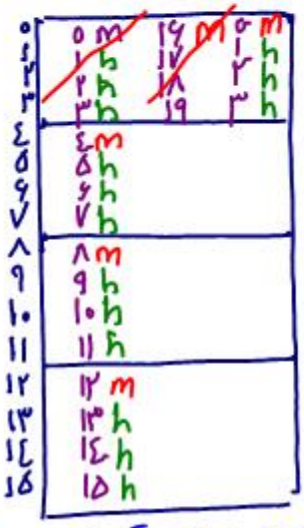
تعداد بلاک های کش = $\frac{2^{19}\ word}{2^3\ word} = 2^{16}$ ⟶ اگر فیلد مدنظر قرار بود B# باشد باید 16 بیت می بود پس S# است

تعداد ست های کش = $\frac{2^{16}}{2^{12}} = 2^{14}$ ⟶ 4 way set Associative

سوال - فرض کنید کش شامل 4 بلاک 4 کلمه ای است و پردازنده از آدرس 0 تا 16 را 10 بار متوالیا تولید می کند، hit rate را محاسبه کنید اگر

الف) از روش Direct استفاده کنیم

ب) " " Fully ass- " و با سیاست جایگزینی LRU استفاده کنیم



تعداد miss = ۵ × ۱۰ = ۵۰

hit rate = ۱ − ۵۰/۱۷۵

تعداد loss = ۵ + ۹ × ۲ = ۲۳

hit rate = ۱ − ۲۳/۱۷۵

مثال: فرض کنید یک کش انجمنی داریم، بلاک های کش ۸ کلمه‌ای هستند، در ابتدا خالی اند، اگر سیاست نوشتنان write allocate, no write allocate باشد، hit rate را محاسبه کنید

write allocate

write	m[۱۰۰]	m
"	m[۲۰۰]	m
"	m[۱۰۴]	m
read	m[۱۰۲]	h
write	۱۰۰	h
read	۱۰۵	h
write	۱۰۷	h
read	۲۰۲	h
write	۲۰۰	h

۱۰۰ ⌊۸ = ۱۲ ، ۹۶ / ۴

۲۰۰ ⌊۸ = ۲۵ ، ۲۰۰ / ۰

hit rate = ۶/۹

no write allocate

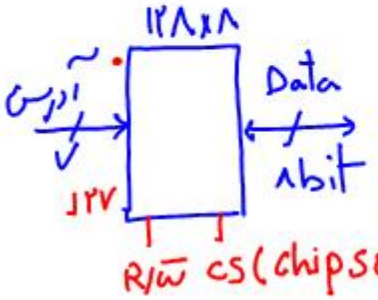
m
m
m
m
h
m
h
m
h
↓

hit rate = ۳/۹

بزرگ بزرگ سازی حافظه ها :

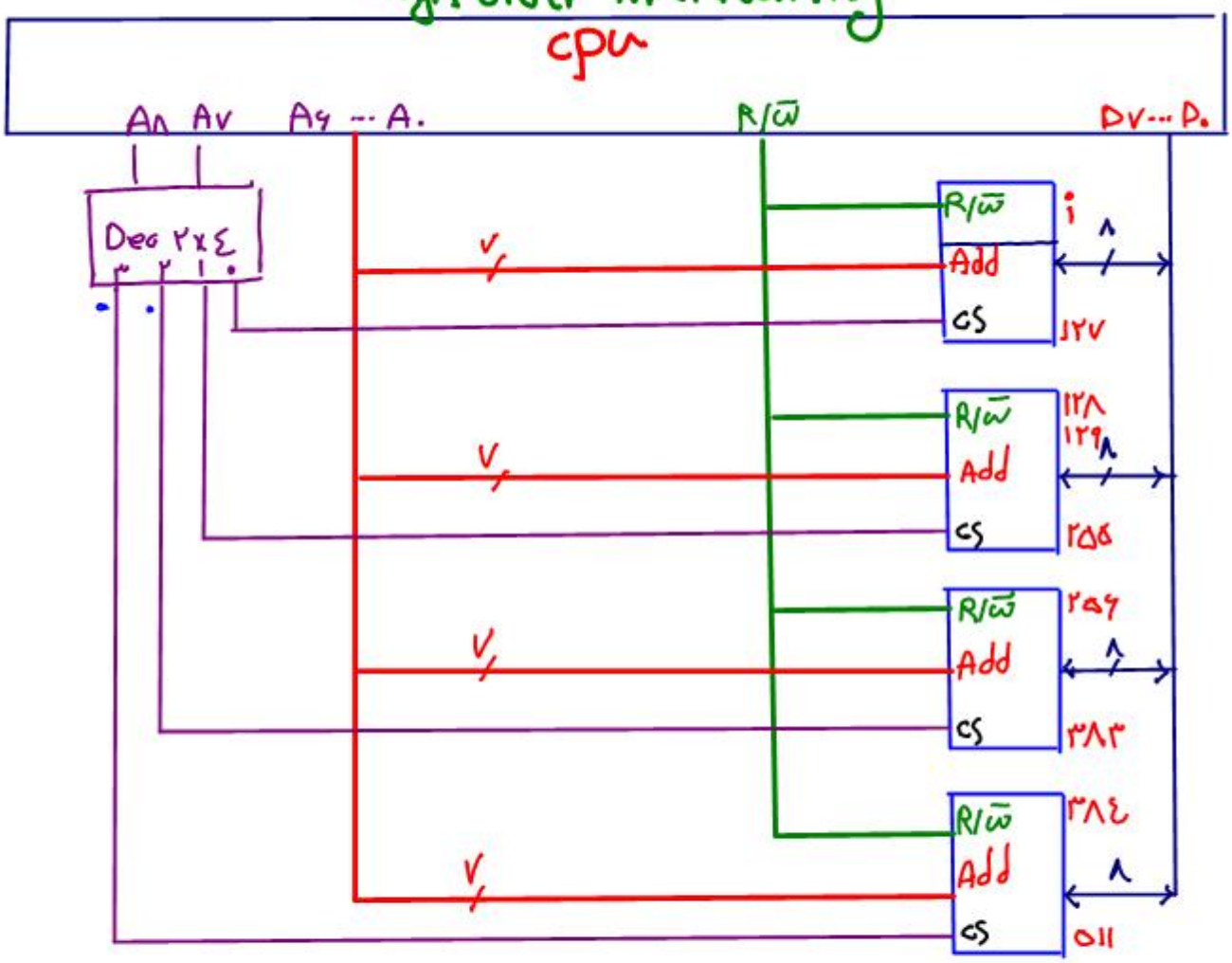
می خواهیم با استفاده از حافظه های کوچکتر حافظه بزرگتر بسازیم

مثال ـ با استفاده از ترانه های RAM ۱۲۸x۸ یک حافظه ۵۱۲x۸ بسازید



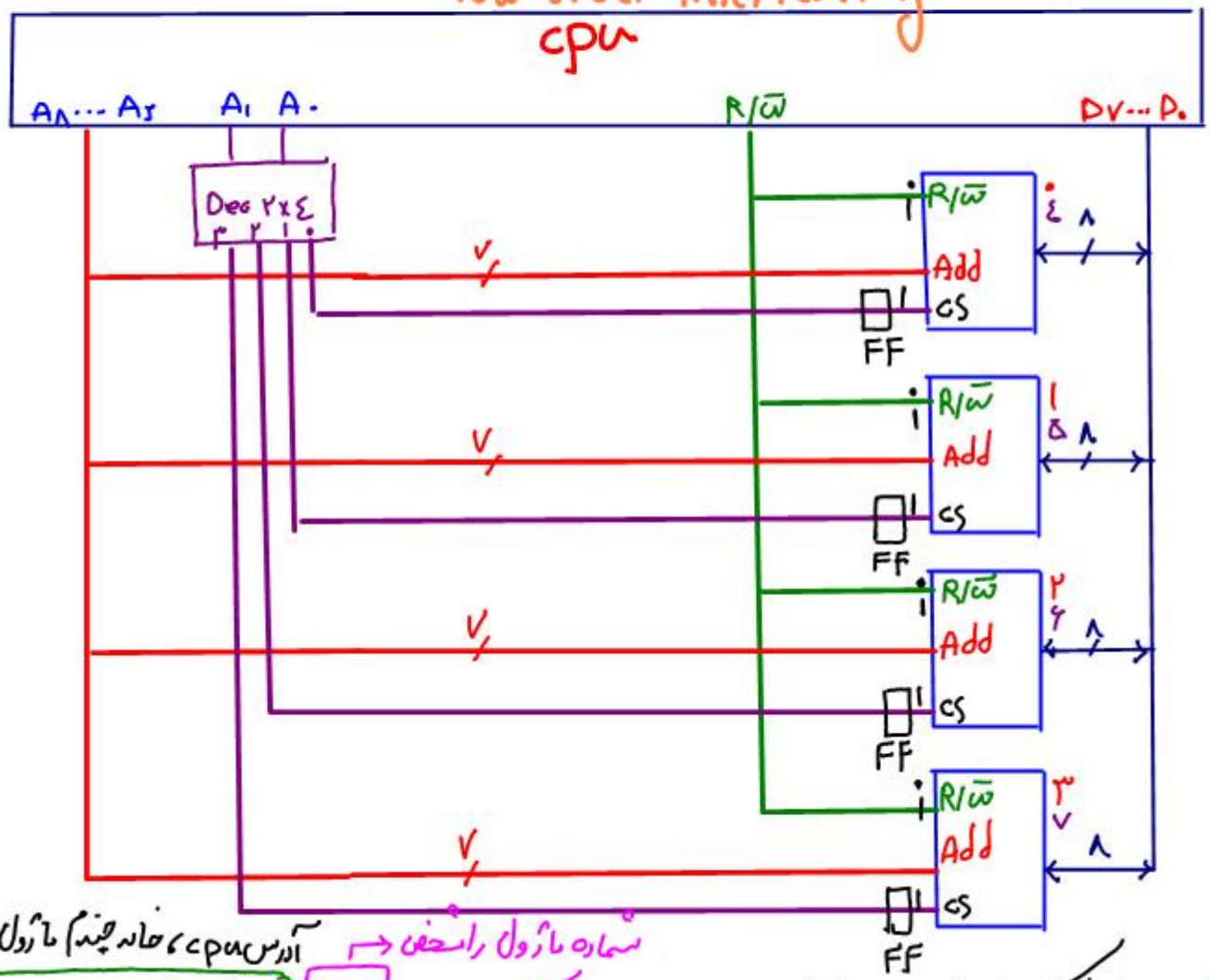
Data high impedanc : ۰= CE OR CS
خواندن : ۱ = R/W̄ ۱=
نوشتن : ۰ = R/W̄

high order interleaving



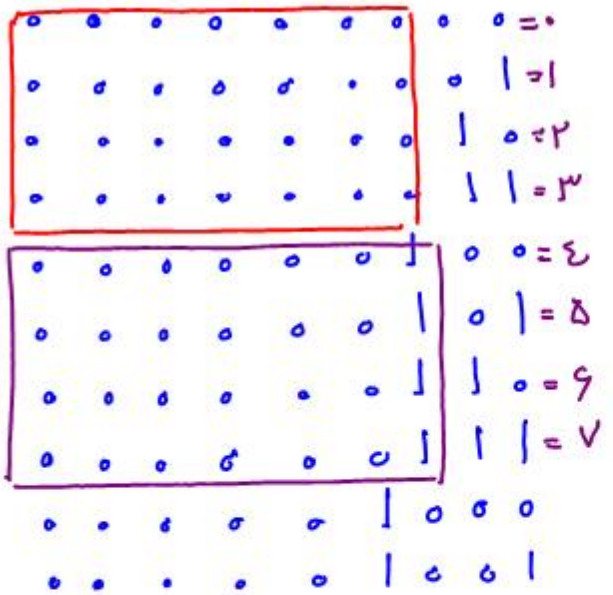
نکته) روش high order point of failure ندارد . در روش low چون همه برنامه ها در همه ترانه ها دیتاهاشون پخش شده ، اگر یک ترانه خراب شود ، کل سیستم از کار می افتد

low order interleaving



آدرس cpu، خانه چندم ماژول مورد نظر برود ← شماره ماژول را مشخص می‌کند

A۸ A۷ A۶ A۵ A٤ A۳ A۲ A۱ A۰



سوال - فرض کنید تاخیر هر ماژول ۱۰۰ns باشد و تاخیر دیکدر ۵ns باشد، خواندن ٤ کلمه ۰ و ۱ و ۲ و ۳ در دو روش high و low چقدر زمان می‌برد؟

در روش high order آخرین ۴۰۵ns و در روش low آخرین ۱۲۰ns شد

توجه) اگر K تا ماجول داشته باشیم و از روش low order استفاده کنیم، با هر access می‌توانیم K تا کلمه متوالی را بخوانیم (که این کلمات آدرس های متوالی هستند). در روش low order آدرس های متوالی در خانه های هم شماره ماجول های مختلف قرار دارد که ما نیز با هر access می‌توانیم خانه های هم شماره ماجول های مختلف را بخوانیم.

	۰ (4K)	۱ (4K+1)	۲ (4K+2)	۳ (4K+3)
۰	۰	۱	۲	۳
۱	۴	۵	۶	۷
۲	۸	۹	۱۰	۱۱
۳	۱۲	۱۳	۱۴	۱۵

* باقیمانده یک عدد باینری به 2^K می‌شود K بیت سمت راست آن عدد.

مثال: ۱۰۱ = ۱۱۰۰۱۰۱ (بیت های ۶ ۵ ۴ ۳ ۲ ۱ ۰)

توجه) ما مدل های دیگر برگ برگ سازی نیز داریم، ما الان low و high را گفتیم ولی مثلاً در شکل بالا هر ۲ خط دلخواهی از آدرس را می‌توان به دیگری داد.

فرض کنید A_1 و A_2 را به دیگری بدهیم

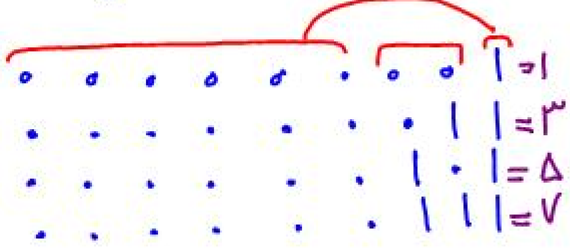
A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
۰	۰	۰	۰	۰	۰	۰	۰	۰	= ۰
۰	۰	۰	۰	۰	۰	۰	۰	۱	= ۱
۰	۰	۰	۰	۰	۰	۰	۱	۰	= ۲
۰	۰	۰	۰	۰	۰	۰	۱	۱	= ۳
۰	۰	۰	۰	۰	۰	۱	۰	۰	= ۴
۰	۰	۰	۰	۰	۰	۱	۰	۱	= ۵
۰	۰	۰	۰	۰	۰	۱	۱	۰	= ۶
۰	۰	۰	۰	۰	۰	۱	۱	۱	= ۷
۰	۰	۰	۰	۰	۱	۰	۰	۰	= ۸
۰	۰	۰	۰	۰	۱	۰	۰	۱	= ۹

	۰	۱	۲	۳
۰	۰	۲	۴	۶
۱	۱	۳	۵	۷
۲	۸	۱۰	۱۲	۱۴
۳	۹	۱۱	۱۳	۱۵

* این مدل برگ برگ سازی برای خواندن آدرس های زوج یا فرد متوالی خوب است.

نکته مهم: یکسری از آدرس هایی را که می‌خواهید سریع بخوانید آنها را بصورت باینری بنویسید، سپس فرق شون (تفاوت شون) را بگیرید و به Dec بدهید. اینکه باید به دنبال چند بیت اختلاف باشید

به تعداد ماجول های حافظه بستگی دارد و برابر ۵۰۱ تعداد ماجول های حافظه به دنبال اختلاف می گردیم



سوال – فرض کنید که چهار ماجول حافظه داریم و می خواهیم ۰ و ۵ را سریع بخوانیم، چطور برگ برگ سازی کنیم

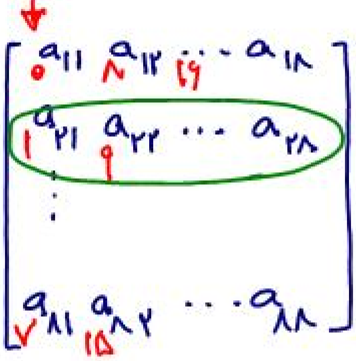
فرض کنید آدرس که از cpu در می آید ۲۰ بیتی است.

$A_2 \quad A_0$

0 = 0 .... 00 0 0 0

5 = 0 .... 00 1 0 1

سوال – فرض کنید ماتریس ۸x۸ که هر عنصرش یک کلمه است، بصورت ستونی از آدرس ۰ در حافظه ذخیره شده است، حافظه از ۸ ماجول ۸K کلمه ای تشکیل شده است، چگونه برگ برگ سازی کنیم تا سرعت دسترسی به درایه های سطر دوم ماتریس افزایش یابد؟



$A_5 A_4 A_3$

1 = 00 000001

9 = 00 — 001001

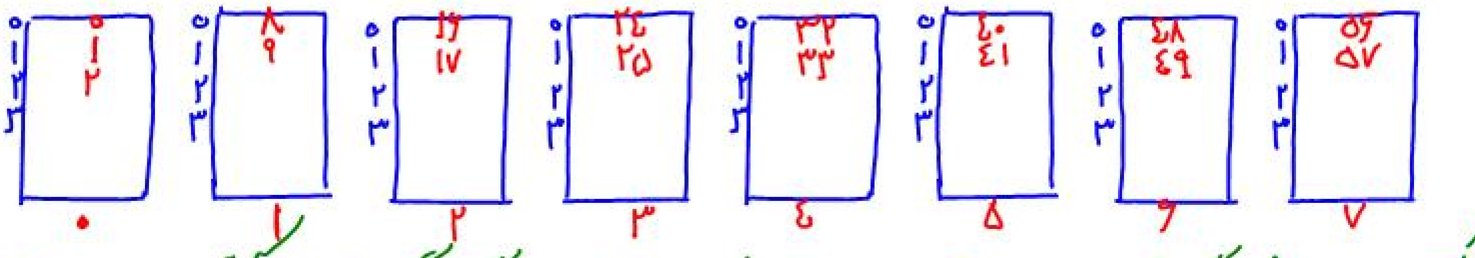
17 = 00 — 010001

25 = 00 — 011001

33 = 00 — 100001

* نحوه برگ برگ سازی این سوال به درد سریع خواندن سطر های ماتریس می خورد

حجم حافظه $= 8 \times 8 \times 2^{10}$ word $= 2^{16}$



نکته) در این مثال اگر بخواهیم ستون ها را سریع بخوانیم، low order اگر برگ برگ سازی کنیم یعنی A_2, A_1, A_0 را به Dec میدهیم، زیرا low برای دسترسی به خانه های متوالی خوب بود

معماری کامپیوتر

رامین رضوی

زبان‌های HDL (Hardware description language) ، زبان‌هایی هستند برای توصیف سخت‌افزار.

برخی از زبان‌های HDL ، RTL ، ASM و MP است. ASM و MP نسخه‌هایی تکمیل یافته RTL هستند. با هرکدام از این زبان‌ها که سخت‌افزار را توصیف کنید بعد می‌توانید واحد کنترل ماشین را با سه روش زیر بسازید، که این سه روش در ۲ دسته کلی قرار می‌گیرند.

روش‌های طراحی واحد کنترل
- hardwired
  - SR-FF (JK-FF)
  - one hot (Direct)
  - ...
  - → gate level
- μ-programmed. (RR = Rom_Register) → component level

اگر واحد کنترل از نوع RR نباشد می‌گویند واحد کنترل اش Hard wired است. علت این نام‌گذاری این است که در روش‌های Hard wired تغییر نوع کارکرد سخت‌افزار ممکن نیست، درصورتیکه اگر واحد کنترل ماشین Micro programmed باشد، می‌توانیم کارکرد ماشین‌مان را upgrade کنیم.

- الگوریتم ضرب ۲ عدد A و B با استفاده از جمع‌ها سوالی را بنویسید:

$\bar{F}_1 S$: $R_1 \leftarrow A, R_2 \leftarrow B$ و $R_3 \leftarrow 0$ و $F_1 \leftarrow 1$ و $E \leftarrow 0$

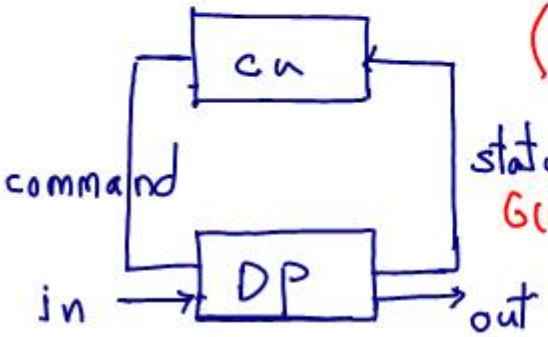
$F_1 G(R_1, R_2)$: $R_2 \leftarrow R_1$, $R_1 \leftarrow R_2$

$F_2 OR(R_1)$: $R_1 \leftarrow R_1 - 1$; $R_3 \leftarrow R_3 + R_2$

$\bar{F}_2 F_1$: $F_2 \leftarrow 1$

$F_2 \overline{OR(R_1)}$: $E \leftarrow 1$ و $F_1 \leftarrow 0$ و $F_2 \leftarrow 0$ و $R_4 \leftarrow R_3$

- هرجا که ما محاسباتی انجام میدهیم آن مربوط به DP است.
- DP بخشی است که مسئول انجام عملیات پایه مثل swap، مثل مقایسه، مثل جمع، transfer



کنترل ولکد کنترل ۲ مرحله دارد : (ولکد کنترل از ۲ بخش تشکیل می‌شود)

۱) پیاده سازی تنگ‌ها الت (ساخت sequencer)

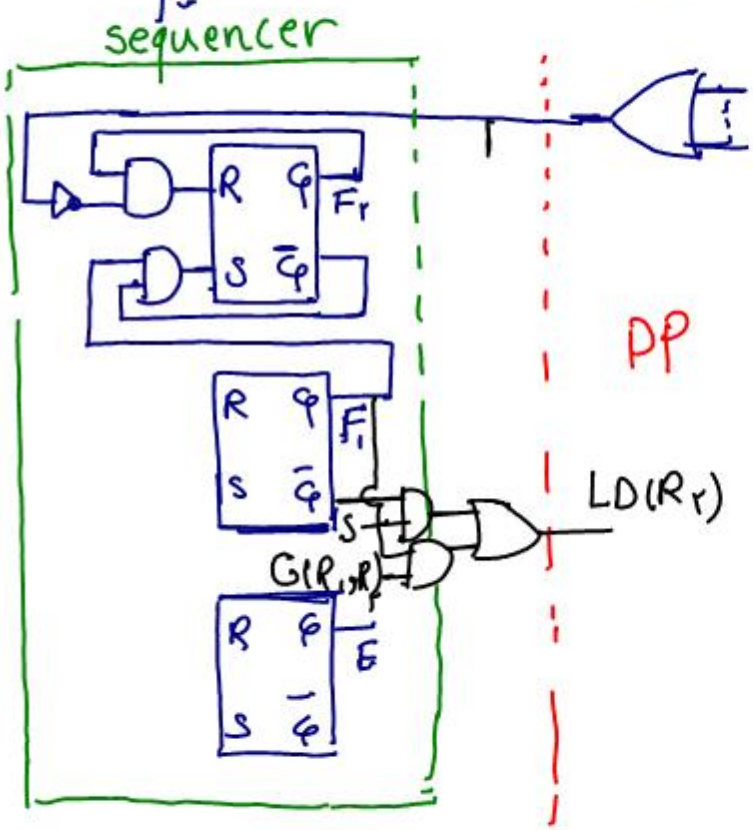
۲) ایجاد سیگنال‌های command

چیزهایی تک بیتی اند و برای مشخص کردن ترتیب کارهستند Flag هستند . مثلاً در RTL

قبل ستا فلگ F_1 , F_2 و E داشتیم .

برای پیاده سازی ولکد کنترل سه روش وجود دارد :

۱) پیاده سازی با FF و RS : در این روش به تعداد فلگ‌ها FF می‌گذاریم



* ولکد کنترل در واقع یک FSM است

بنابراین هر ولکد کنترل نشدنی یک FSM است

بنابراین هر مدار کنترلی که ما بخواهیم طراحی کنیم

می‌توانیم FSM اش را بدست آوردیم و سپس آن

را از روی FSM اش بدست بیاوریم

برای اینکه روش دوم ساخت واحد کنترل یعنی روش one hot یا Direct را معرفی کنیم، ابتدا باید نیاز داشته باشیم از زبانهای توصیف سخت افزار را بنام ASM معرفی کنیم.

ASM (Algorithmic state machine)

به علت Text base بودن RTL امکان اشتباه در آن زیاد بود، بنابراین ASM مطرح شد.

زبان ASM در واقع همان زبان RTL است و فقط نسخه گرافیکی آن است، حتی در این حد که می توانیم بگوییم RTL گرافیکی.

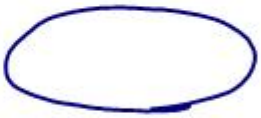
- در فلوچارت Box ها بصورت سری و پشت سرهم اجرا می شوند و ما در فلوچارت مولتی پلکسر نداریم، اما در ASM chart جعبه Box ها می توانند بصورت موازی اجرا شوند، بنابراین ASM را مثل فلوچارت نخوانید چون که دچار اشتباه می شوید.

نکته: ASM فقط گرافیکی نیست و ASM هم گرافیک دارد و هم text . text و دیدن شکل های گرافیکی می آید

- چارت ها و نمودارهای ASM از انتقال سه جعبه زیر بهم ساخته می شود.

جعبه حالت
(state Box)

جعبه تصمیم گیری
(Decision Box)



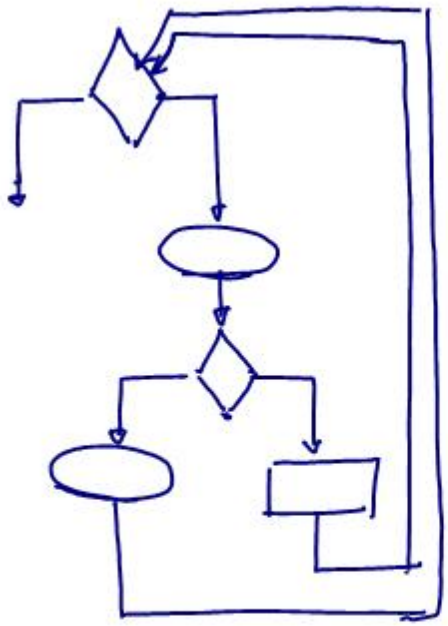
جعبه مشروط
(conditional Box)

معماری کامپیوتر

رامین رضوی

- زبان Asm یکی قوانین (یکی syntax) برای گرامیک اش و یکی syntax نیز برای Text اش دارد.

۱) قوانین مربوط به گرامیک: پشت جعبه مشروط فقط می تواند جعبه تصمیم یا لنه (پشت بعضی فقط لوزی بیاد) حلقه ها هم جهت دار ما حتما شامل state box باشند



۲) قوانین مربوط به Text: داخل جعبه حالت و جعبه مشروط باید MI بیاید و داخل جعبه تصمیم گیری باید condition بیاید

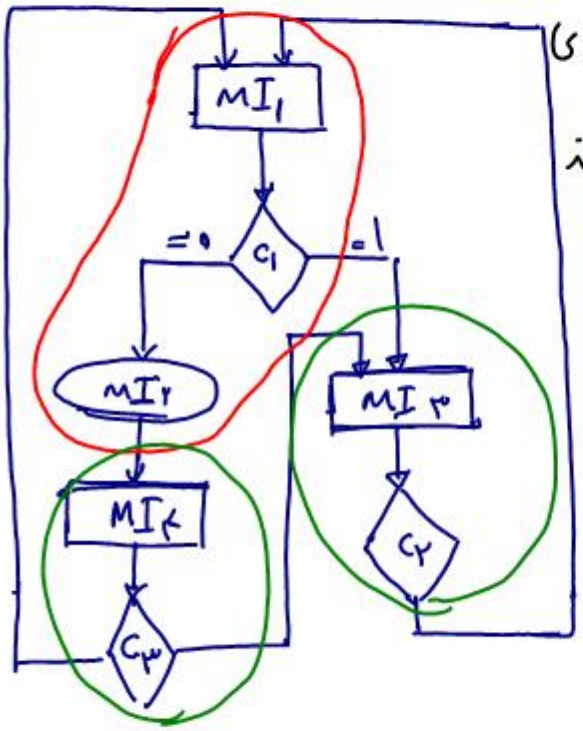
Ar : Asm-Block

نکته) به ازای هر state box یک Asm block داریم

- اگر کسی می خواهد یک چارت Asm را بخواند اولین کاری که باید انجام بدهد این است که باید Asm بلاک های آن چارت Asm را پیدا کند و ما تا Asm block ها یک چارت Asm را پیدا نکنیم قادر نیستیم آن Asm چارت را بفهمیم

روش پیدا کردن Asm block ها مربوط به هر جعبه حالت (state box):

از جعبه حالت شروع می کنیم و در جهت فلش ها می ریم جلو، اگر به جعبه تصمیم گیری رسیدیم باید هر ۲ شاخه را برویم جلو و این جلو رفتن را تا رسیدن به اول یک جعبه حالت دیگر جلو می بریم (ادامه می دهیم)



Asm block ها این را مشخص می‌کند که کدام قسمت‌های RTL با هم موازی و کدام قسمت‌ها که با هم سری هستند

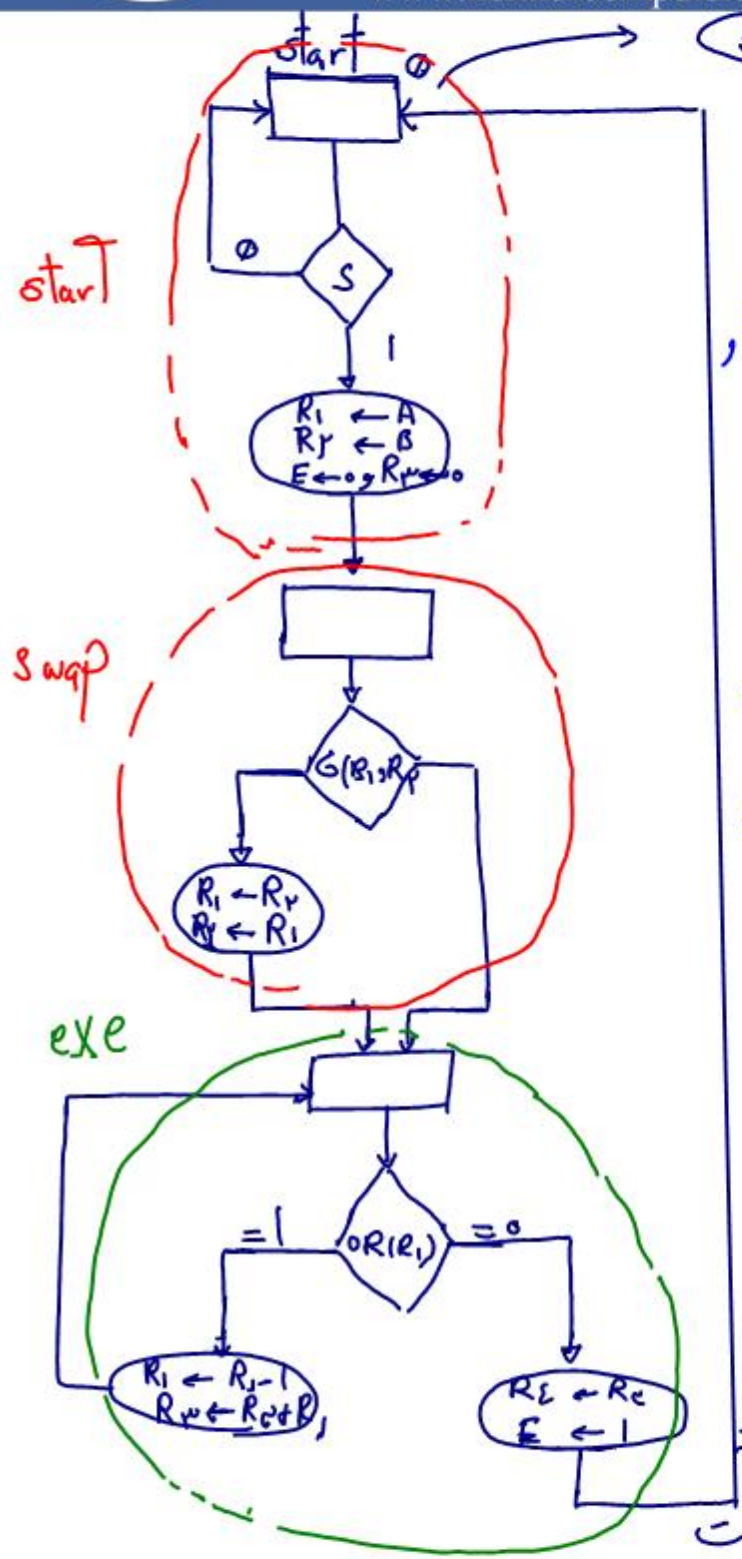
* در چارت‌های ASM تمام محتوای داخلی ASM block ها با هم موازی اند و قسمت‌های سری بین ASM block ها است، یعنی ASM block ها بصورت سری و به ترتیب اجرا می‌شوند، اما محتوای داخلی هر ASM block بصورت موازی و در 1 کلاک انجام می‌شود

$C_1 = 0$: MI_1 و MI_2

$C_1 = 1$: MI_1

نکته) شما هر وقت وارد یک ASM-block بشوید MI مربوط به همه حالت آن ASM-block اجرا می‌شود.

مثال) چارت ASM ضرب دو عدد A و B را بکشید و مدار کنترل سیستم را به روش one hot بسازید



* برای ساخت سیستم مان از روی چارت ASM باید DP و واحد کنترل را بسازیم

- در چارت ASM، DP را از روی Text و واحد کنترل را از روی شکل گرافیکی می سازیم.

- ساخت DP از روی چارت ASM عینا شبیه ساخت DP از روی RTL است، پس ما فقط بخش واحد کنترل را طراحی می کنیم. حال می خواهیم روش دوم طراحی واحد کنترل یعنی one hot را بگوییم

ساخت واحد کنترل شامل ۲ قسمت بود:

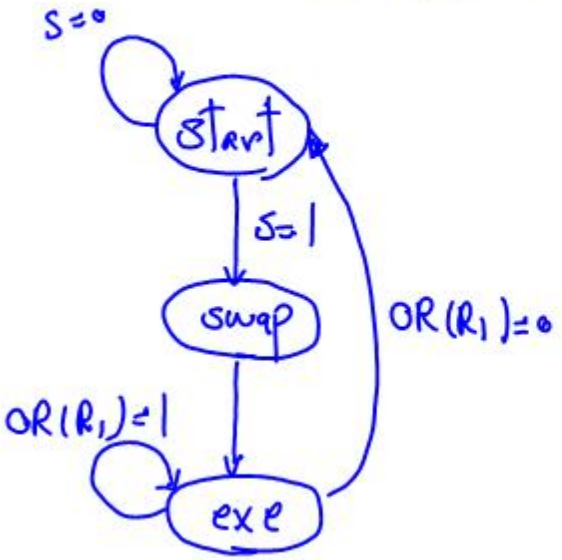
۱) ساخت sequencer

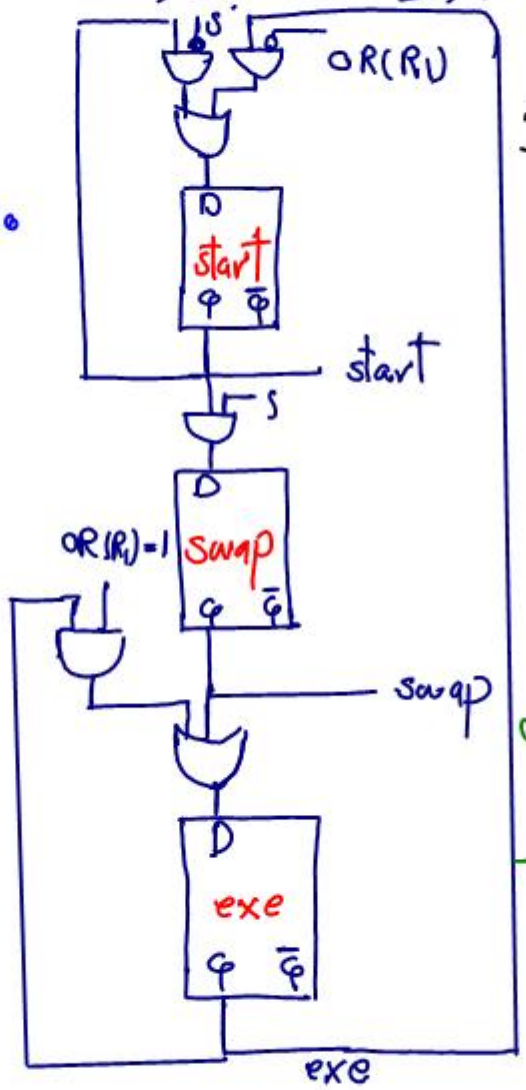
۲) ساخت سیگنال های command

ساخت سیگنال های command شبیه نوشتن RTL است، حال می خواهیم روش جدیدی را برای ساخت sequencer بگوییم.

۲) روش one hot یا Direct (مستقیم) = روش طبیب فلاپ های D

روشی است برای ساخت واحد sequencer . برای این کار ابتدا می آیند و از چارت ASM یک نمودار حالت استخراج می کنند (state diag = FSM استخراج می کنند) ، این کار را به این صورت انجام میدهند که هر ASM block را تبدیل به یک حالت می کنند ، بنابراین به تعداد ASM block state خواهیم داشت ، بعد از رسم FSM به ازای هر state یک D-FF می گذاریم ما در هر لحظه از زمان که در هر state ای هستیم باید FF مربوط به آن حالت مقدار 1 داشته باشد و سایر FF ها مقدار 0 داشته باشند





* برای ساخت واحد کنترل باید از خودمان بپرسیم که :
کی به start می رویم ؟
کی به swap ؟
کی به exe ؟

توجه ۱) روش Direct حجم سخت افزارش خوب نیست ولی در مدارات دیجیتال قسمت پیچیده و کند ما Dp است و واحد کنترل بخش پیچیده و کند ما نیست، برای همین کسی به دنبال کمینه کردن واحد کنترل نیست

توجه ۲) شما می توانید برنامه تان را با RTL بنویسید ولی واحد کنترل تون را با روش Direct بسازید همین طور ممکن است ASM داشته باشید ولی واحد کنترل تون را با روش FF ها یا R-S بسازید

نکته) ASM به راحتی قابل ترجمه به RTL :

ما دو نوع چارت ASM داریم :

انواع ASM
- moor : مورد علاقه طراح سوال نیست — ASMای که جعبه شرطی ندارد
- mealy :

نکته) ASM هایی که جعبه مشروط دارند را نمی توانیم واحد کنترل شون را بصورت micro prog بسازیم، بنابراین کسانی که ASM می نویسند ولی می خواهند واحد کنترل شون micro pro در باشد باید ASM شان مور باشد

نکته) Asm moore از Asm mealy تعداد کلاک بیشتری می خواهد. چون دیگر خیلی از کارهایی را که قبلا می توانستیم بصورت موازی در یک کلاک انجام بدهیم را حالا دیگر نمی توانیم انجام دهیم و بنابراین تعداد state box ها بیشتر و در نتیجه تعداد Asm block ها بیشتر و در نتیجه تعداد کلاک ها بیشتری خواهیم داشت

نکته ششم: طراح علاقه دارد که به state Box، بگوید جعبه عملیاتی یا جعبه انتقال
همچنین به چارت Asm، چارت عملیاتی می گوید

همچنین به Asm مورعلاقه دارد و در آنجا ما جعبه شرطی نداریم، و طراح در Asm مور
به جعبه تقسیم جعبه شرطی می گوید.

نکته) در واحد کنترل به روش one hot:

۱) تعداد FF = تعداد state box = تعداد ASM block = تعداد state در FSM

۲) در گیت های AND = ۲ برابر تعداد جعبه های تصمیم (لوزی ها) * تعداد AND ها همواره زوج است.

۳) " " OR = به تعداد join ها.

- برای اینکه روش سوم ساخت واحد کنترل یعنی روش RR (Rom-Register) یا micro programmed را معرفی کنیم، ابتدا یکی دیگر از زبانهای توصیف سخت افزار بنام micro programing را معرفی می کنیم.

زبان MP: شبیه زبان RTL است اما برخلاف RTL جای خطوط در آن مهم است یعنی اگر خطوط برنامه را جابجا کنیم، خروجی برنامه تغییر می کند، پس از این نظر شبیه C و جاوا و ... است. همانطور که در RTL ساختار هر خط مشخص است، در MP نیز ساختار هر خط مشخص است، و همانطور که هر خط RTL در یک پالس ساعت اجرا می شود هر خط

MP هم در یک کلاک اجرا می‌شود.

MI, if c then L ⇒ lable

این را پاس‌های قبلی تعیین می‌کند

condition

بعنوان مثال یک کد MP بصورت زیر است:

L_1: MI_1, if c_1 then L_1
MI_2, if c_2 then L_2
MI_3, if c_3 then L_1
L_3: MI_4, if c_4 then L_1

۱) اگر جایی نیاز به پرش نداشتیم و می‌خواستیم، خط بعد اجرا شود، کافی است شرط را ۰ بگذاریم

$MI_0 \leftrightarrow MI$

۲) اگر خواستیم حتما بپرد: MI, goto $L_2 \equiv MI$, 1 L_2:

۳) MI می‌تواند تهی باشد مثلا ۱۰ if $\bar{s}$ then L_3:

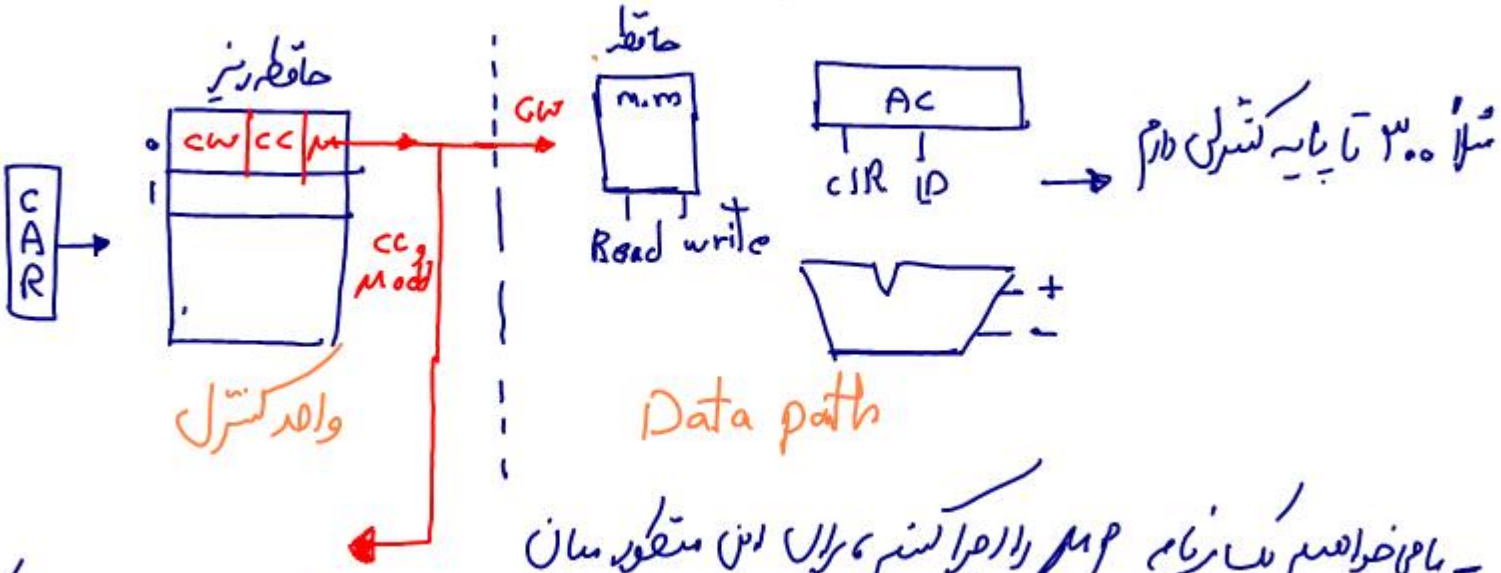
مثال) برای ضرب ۲ عدد A و B در هم به زبان MP کد بنویسید:

L_1: if $\bar{s}$ then L_1
$R_1 \leftarrow A, R_2 \leftarrow B, R_3 \leftarrow 0, E \leftarrow 0$
if $\overline{G(R_1, R_3)}$ then L_2
$R_1 \leftarrow R_2, R_2 \leftarrow R_1$
L_2: if $\overline{oR(R_1)}$ then L_3
$R_1 \leftarrow R_1 - 1, R_3 \leftarrow R_2 + R_3$, goto L_2 = if 1 then L_2
L_3: $R_4 \leftarrow R_3, E \leftarrow 1$, goto L_1

نکته) MP به دو دلیل از ASM و RTL کندتر است. دلیل اول وجود MI ها می باشد
دلیل دوم این است که کلاک های MP پریود بزرگتری دارد.

نکته) MI و condition ها را باید DP تولید کند و بقیه مربوط به CW می شود

پیاده سازی واحد کنترل - روش RR (Rom-Register) یا همان micro programmed



مثلا ۳۰۰ تا پایه کنترلی دارم

- ما می خواهیم یک برنامه MP را اجرا کنیم، برای این منظور ما این
هر کدام از خط های MP را به صورت که باینری درمی آورند (دیزاسمبل می کنند) حال بعد از اینکه
کل کدمان را دیزاسمبل کردیم می آییم و حاصل را در داخل Rom می ریزیم، که باینری خط صفرم
MP در خانه ۰ ریز حافظه و همین ترتیب قرار می گیرد
پس وقتی به Rom آدرس ۵ بدهیم که باینری مربوط به خط ۵ بیرون می آید و قسمتی از آن
نام کلمه کنترلی برای اجرا MI تعیین به سمت DP و قسمتی از آن نام که شرط و پرش
که مشخص می کند خط بعد MP که باید اجرا شود چه هست به سمت خود واحد کنترل برمی گردد

هر خط MP بصورت زیر به کد باینری تکمیلی تبدیل می‌شود

MI, if c then L

CW/MI	CC/MOP	μ address

آدرس بعدی

condition control

CW = تمام بیت‌های command که قرار است به DP داده شود تا یک MI اجرا شود را کنار هم می‌چینیم. (DP در هر کلاک یک MI اجرا می‌کند)

در واقع لغت کنترلی کد شده MI ها است.

حال اگر بعنوان مثال در DP ۳۰۰ تا پایه کنترلی داشته باشیم طول CW، ۳۰۰ بیت است حال ما با ۳۰۰ بیت CW، 2^{300} تا لغت کنترلی مختلف می‌توانیم داشته باشیم، حال مگر ما چند تا MI داریم؟ مثلاً ۱۰۰ تا، پس خیلی از این 2^{300} تا احتمالی هستند و نیازی نیست بخشون این لغت‌های کنترلی اضافی به چه دلیل می‌توانند به در بخورند (اضافی باشند).

۱) یکی از اینها غیرمجاز اند و اینها باعث می‌شوند به سخت افزار آسیب برسد.

۲) خیلی از این ها MI های بی ربط هستند که کاربردی برای ما ندارد

۳) تعداد بسیاری نیز لغت‌های کنترلی مختلف یک MI را نشان می‌دهد.

* ما با ۳۰۰ بیت می‌توانیم 2^{300} تا MI را کد کنیم، در صورتیکه ما فقط می‌خواهیم ۱۰۰ تا MI را کد کنیم، ما ۱۰۰ تا MI را با ۷ بیت می‌توانیم کد کنیم اما ما ۳۰۰ بیت استفاده کردیم، این یعنی ۲۹۳ بیت اضافه در هر خط حافظه نیز

ما بدنبال کاهش سایز Rom هستیم، چون اگر سایز کوچک شود قیمت تمام شده، سرعت و توان مصرفی بالا می رود.

روش های کاهش سایز Rom:

1) تغییر syntax، mp

2) استفاده از np

3) روش MI های عمودی

نکته مهم: معمولاً ۱و۲ را با هم استفاده می کنند

روش ۱: تغییر syntax، mp:

الگوی mp که ما استفاده می کنیم بصورت L then، if و MI ولی اگر به برنامه های mp نگاه کنیم متوجه می شویم که خط های بسیار بسیار کمی بصورت L then، if و MI هستند و اکثرا یا MI تنها هستند و یا L then، if تنها. برای رفع این مشکل میگیم هر خط mp را یا بصورت MI نداریم و یا بصورت L then، if. در اینصورت در خانه های ریزحافظه دو مدل نوبت خواهیم داشت و طول نوبت بزرگتر است که طول ریزحافظه را مشخص می کند

حافظه ریز

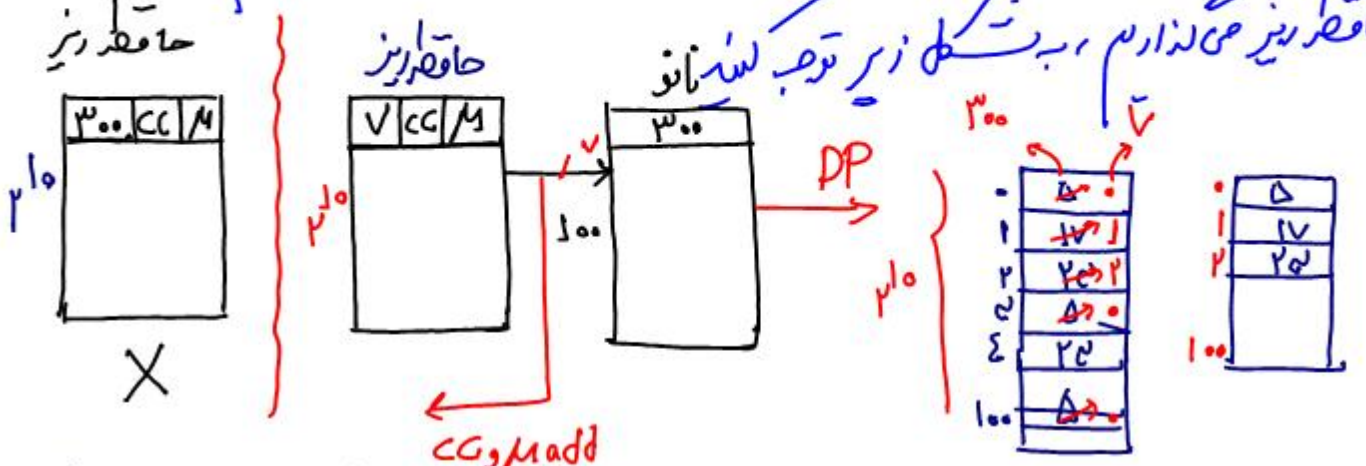
mop	cw
mop	madd

* چون cw خیلی بزرگ است معمولاً این روش خیلی به تنهایی چاره ساز نیست و برای همین این را معمولاً با np ترکیب می کنند

عرض رجیستر حافظه = mop طول + max{ cw و maddress }

روش ۲) استفاده از نانو: این قسمت به کوچک کردن cw کار دارد فقط. این روش به قسمت MI کار دارد و به if ، then ، L کاری ندارد.

مانمی خواهیم مثلاً ۱۰۰ تا MI را که می توانیم با ۷ بیت کدگذاری کنیم با ۳۰۰ بیت کدگذاری کنیم، از این همین مامی آییم و لغات کنترلی متمایز را در یک حافظه دیگر بنام نانو می گذاریم و سپس آدرس آنها را در حافظه ریز می گذاریم، به شکل زیر توجه کنید:



حجم حافظه ریز در ابتدا = $2^{10} \times (300 + c + 10)$

بعد از استفاده از نانو = $2^{10} \times (7 + c + 10)$

صرفه جویی = $2^{10} \times (300 + c + 10) - 2^{10} \times (7 + c + 10)$

$2^{10} \times 293 \Rightarrow 293K$

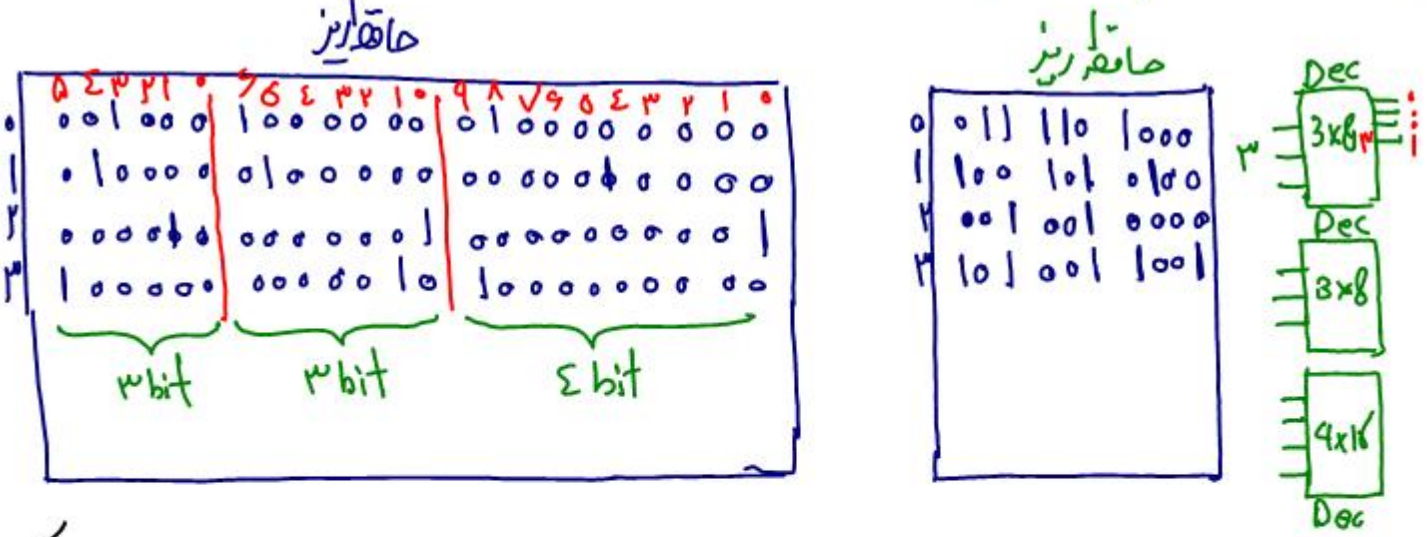
* ترکیب روش ۱ و ۲ با هم.

اگر در حافظه ریز از نمایش ۲ فرسته استفاده کنیم

طول رجیستر دستور بدون نانو = طول{ maddr و طول cw } + max{ طول mop }

با نانو = طول{ madd و nano add } + max{ طول mop }

روش ۳) ریز دستورات عمودی : این روش از یک ویژگی جالب لغت های کنترلی استفاده می کند ویژگی این است که در لغت های کنترلی ● خیلی زیاد است . حال باید لغت های کنترلی را به زیر گروه هایی خرد کنیم که این زیر گروه ها داراں فقط ۱ بیت ۱ باشند



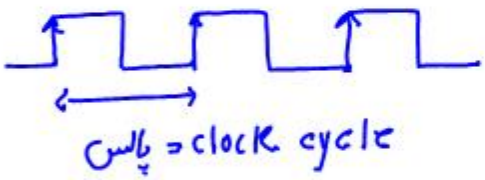
* ما سه روش گفتیم برای فشرده سازی MP : روش دوم و سوم را می شود با هم استفاده کرد اما روش اول را می توانیم هم در کنار روش دوم و هم در کنار روش سوم با هم داشته باشیم اگر کسی از np یا Dec استفاده می کند معمولا سایز Rom برایش مهم است ، اگر سایز Rom مهم است ، پس به احتمال زیاد ، از گزینه ۲ فرمتینگ استفاده می کند .

معماری کامپیوتر

رامین رضوی

کارایی پردازنده ها

مروری بر کلاک :



ولتاژ تا فید است ← پریود کلاک = T = مدت زمان یک کلاک = clock cycle time

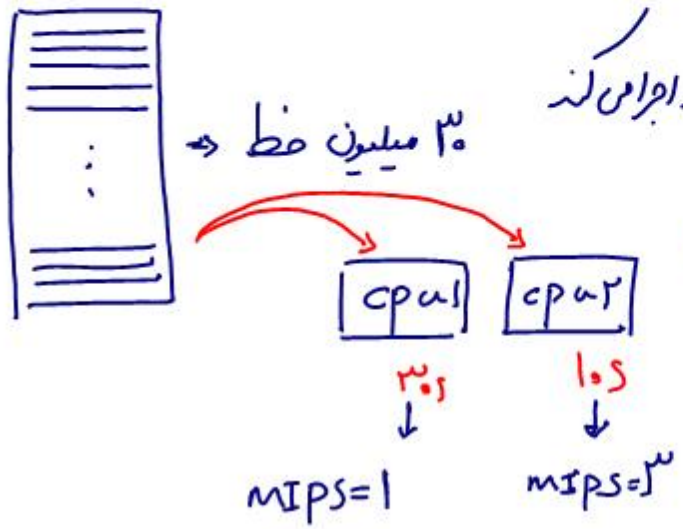
فرکانس = clock rate = $f = \frac{1}{T}$ = هرتز

$\frac{1}{s} = Hz$ ، $\frac{1}{H} = s$ ⇐ ثانیه و هرتز عکس هم اند

فرکانس یک سیستم ۱۰۰۰ هرتز است.

MIPS (million instruction per sec)

متوسط تعداد میلیون دستوری که پردازنده در یک ثانیه اجرا می کند



نکته ۱ - bench mark = برنامه محک = افزار سنج

نکته ۲ - MIPS لزوما معیار خوبی برای سنجش کارایی پردازنده ها نیست زیرا ممکن است ماشینی دارای MIPS کمتر باشد ولی کارایی اش بهتر باشد

مثال - پردازنده ای دارای چهار کلاس دستور العمل است ، یک برنامه محک روی این پردازنده اجرا شده است که مشخصات این برنامه بصورت زیر است ، اگر نرخ کلاک پردازنده ۲۰۰ MHZ باشد MIPS این پردازنده را بدست آورید؟

کلاس دستورات cpu	درصد دستورات برنامه از هر کلاس (گروه)	متوسط تعداد کلاک لازم برای اجرا هر دستور
A = جمع و تفریق اعداد صحیح	٪۳۰	٤
B = ضرب و تقسیم	٪۲۰	٧
C = دستورات کنترلی	٪۱۰	١٤
D = محاسبات اعشاری	٪٤۰	١٥

۱۰۰۰ دستور محاسبات اعشاری داریم که برای اجرا شدن هرکدام اش بطور متوسط به ۱۰ کلاک نیاز داریم

برنامه
۱۰۰۰ خط

متوسط تعداد کلاک به ازا هر دستور = cpI = avg clock per instruction

$$cpI = \frac{30\times 4 + 20\times 7 + 10\times 14 + 40\times 10}{100} = 8$$

۱ دستور برای اجرا شدن بطور متوسط ۸ کلاک نیاز دارد

↳ در ۱ کلاک $\frac{1}{8}$ یک دستور اجرا می‌شود

$Ipc = avg\ inst\_per\ clock = \frac{1}{8}$

متوسط زمان اجرا یک دستور = تعداد کلاکی که یک دستور برای اجرا نیاز دارد × متوسط زمان یک کلاک = cpI × T

متوسط زمان اجرا برنامه = n × cpI × T

$$\text{متوسط زمان اجرا یک دستور} = 8 \times \frac{1}{200MHz} = \frac{8}{200}\times 10^{-6}\, s = \frac{8}{200}\,\mu s$$

دستور	زمان اجرا
۱	$\frac{8}{200}\,\mu s$
?	۱۶

$$\Rightarrow ? = \frac{1\times 16\times 200}{8\,\mu s} = 25\times 10^{6} \Rightarrow MIPS = 25$$

۲۵ میلیون دستور بطور متوسط در ۱ ثانیه در این ماشین اجرا می‌شود

معماری کامپیوتر

رامین رضوی

$$MIPS = \frac{\text{تعداد دستورات برنامه}}{\text{(زمان اجرا برنامه)}\mu sec} = \frac{n}{(n \times cpI \times T)\mu sec} = \frac{1}{\text{زمان اجرا یک دستور (}\mu sec\text{)}} = \frac{f_{(MHz)}}{cpI}$$

نکته) واحد مدت زمان یک کلاک ثانیه است ولی چون واحد ثانیه در دنیا کامپیوتر خیلی بزرگ است، ما در دنیا کامپیوتر از مشتقات ثانیه استفاده می‌کنیم

- در دنیا کامپیوتر واحد هرتز که برای فرکانس داریم خیلی کوچک است، به همین علت از مشتقات هرتز استفاده می‌کنند

$ds = 10^{-1}s$	$KHz = 10^{3}Hz$
$cs = 10^{-2}s$	$MHz = 10^{6}Hz$
$ms = 10^{-3}s$	$GHz = 10^{9}Hz$
$\mu s = 10^{-6}s$	$THz = 10^{12}Hz$
$ns = 10^{-9}s$	
$ps = 10^{-12}s$	
$fs = 10^{-15}s$	
$as = 10^{-18}s$	

توجه) واحدهای K, M, G, T وقتی بابیت و مشتقات آن استفاده می‌شود بصورت توان است.

$K bit = 2^{10} bit$

$M bit = 2^{20} bit$

$G \text{ " } = 2^{30} \text{ " }$

$T \text{ " } = 2^{40} \text{ " }$

سوال - برنامه P داراى ۲۲۳۱۳ دستور است، این برنامه را روی دو ماشین زیر اجرا کرده ایم

الف - ماشین A که داراى cpI برابر ۲٫۴ و cycle time برابر ۲ns است

ب - " B " " " " ۳ و فرکانس برابر ۲۰۰MHz است

کدام ماشین سریعتر است و چند برابر سریعتر است؟

$$\text{سرعت} = speed\ up = \frac{\text{سرعت ماشین A}}{\text{B " "}} = \frac{\text{متوسط زمان اجرا برنامه روی B}}{\text{A " " " " "}} = \frac{n_B \times cpI_B \times T_B}{n_A \times cpI_A \times T_A}$$

$$\frac{\text{سرعت A}}{\text{B "}} = \frac{3 \times 1 \times 2.4}{200 \times 10^6 \times 2 \times 10^{-9}} = \frac{10}{6} = 1.67$$

سرعت B، ۱٫۶۷ برابر A است

معماری کامپیوتر

رامین رضوی

مثال- در یک برنامه ۵۰ درصد زمان اجرا مخصوص محاسبات اعداد اعشاری است و ۳۰٪ از زمان اجرا مربوط به دستورات جمع و تفریق صحیح است، برای بهبود این برنامه دو پیشنهاد وجود دارد

۱- دستورات معیز شناور را دو برابر سریع کنیم

۲- " جمع و تفریق را شش " " "

$0,5t + 0,3t + 0,2t$

↓ ↓

جمع و تفریق صحیح — محاسبات اعشاری

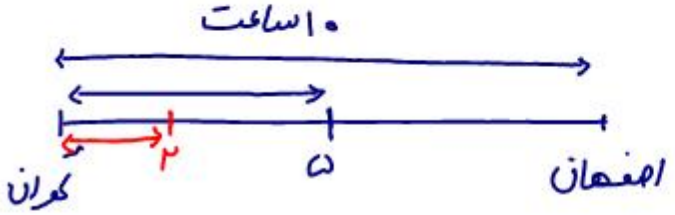
کدام پیشنهاد بهتر است؟

- این دو پیشنهاد به یک اندازه روی زمان اجرا تاثیر می‌گذارد

۱- $\frac{0,5t}{2} + 0,5T = 0,75t$

۲- $0,7t + \frac{0,3t}{6} = 0,75t$

مثال- اگر ۲۰ درصد زمان اجرا یک برنامه را ۵ برابر سریع کنیم کل برنامه چقدر سریع می‌شود؟



۱- بیش از ۳ برابر

۲- " " ۲,۵ "

۳- " " ۲ "

۴- کمتر " " "

$speed\ up = \frac{\text{سرعت بعد از بهبود}}{\text{" قبل "}} = \frac{\text{زمان قبل بهبود}}{\text{" بعد از "}} = \frac{1}{0,84} = \frac{1}{\frac{0,8t}{5} + 0,2t} = \frac{1}{0,84} = \frac{100}{84} \simeq 1,19 = 1,2$

fraction

- اگر f درصد زمان اجرا یک برنامه را p برابر سریع کنیم، کل برنامه چقدر سریع‌تر می‌شود؟

$0 < f < 1$

$speed\ up = \frac{\text{سرعت بعد از بهبود}}{\text{" قبل "}} = \frac{\text{زمان قبل بهبود}}{\text{" بعد از "}} = \frac{1}{(1-f)t + \frac{ft}{p}} = \frac{1}{1-f+\frac{f}{p}}$

Flops (floating point operation per second)

Flops مانند mips معیاری است برای اندازه گیری کارایی پردازنده ها ، Flops برابر است با:

متوسط تعداد دستورات ممیز شناور که در 1 ثانیه اجرا می شود

- flops بیشتر برای مقایسه پردازنده هایی استفاده می شود که در آنها عملیات ممیز شناور عملیات غالب و اصلی است

$$\text{فرمول Flops} = \frac{\text{تعداد عملیات ممیز شناور}}{\text{زمان اجرای کل برنامه}}$$

تست - در یک کامپیوتر زمان اجرای یک برنامه 100s است ، 50 درصد زمان اجرا صرف عملیات ممیز شناور می شود و تعداد دستورات ممیز شناور 10 میلیون دستور العمل است . Flops برنامه چقدر است

	تعداد دستور	زمان اجرا
	10 میلیون	100s ~~50~~
	?	1

$\Rightarrow ? \Rightarrow 0.2 \times 10^6$ گزینه 2 X

1- Flops 0.05 میلیون

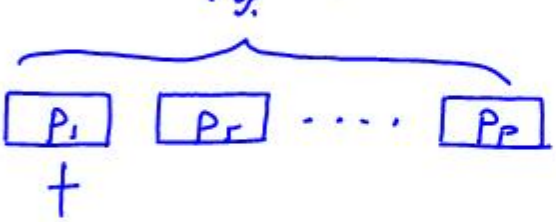
2- 0.2 " "

3- 0.1 " "

4- 0.5 " "

فصل موازات و پایپ‌لاین

سوال ـ فرض کنید برنامه ای روی یک پردازنده اجرا شده و زمان اجرایش + شده است، حال همین برنامه را روی p پردازنده مشابه (همگون) تقسیم کرده‌ایم، در حالت ایده آل انتظار داریم که زمان اجرایمان $\frac{+}{P}$ بشود، اما این حالت ایده آل هیچ وقت پیش نمی‌آید. یک علت این امر این است که معمولا قسمتی از دستورات یک برنامه بهم وابسته است و نمی‌توان آنها را موازی با هم اجرا شون کرد و همچنین برخی از دستورات نمی‌توانند از هم جدا بشوند و باید روی یکپا پردازنده اجرا بشود، و هر چند چنین دستوراتی در برنامه‌ها را معمولا با f که $0 < f < 1$ نشان می‌دهند و به f مولفه ترتیبی نیز می‌گویند. حال با وجود این f زمان اجرا روی p پردازنده را محاسبه کنید



ایده ۱: f درصد را به هم به یک پردازنده و $1-f$ درصد باقیمانده را بین $p-1$ پردازنده پخش کنم

قانون آمدال

$$\text{زمان اجرا} = \max\left(ft, \frac{(1-f)t}{p-1}\right)$$

ایده ۲: ابتدا f درصد وابسته را به یک پردازنده بدهیم و صبر کنیم تا اجرایش تمام شود و سپس که همه p پردازنده خالی شد $1-f$ درصد باقیمانده را روی همه پردازنده‌ها پخش کنیم

$$\text{زمان اجرا}: ft + \frac{(1-f)t}{p}$$

پس فرض مان را ایده ۲ می‌گذاریم.

* سرعت p پردازنده را نسبت به سرعت یک پردازنده محاسبه کنید

$$\text{speed up} = \frac{\text{سرعت p پردازنده}}{\text{سرعت 1 پردازنده}} = \frac{\text{زمان اجرا برنامه روی 1 پردازنده}}{\text{زمان اجرا برنامه روی p پردازنده}} = \frac{\cancel{+}}{\cancel{+}(f + \frac{1-f}{p})} = \frac{1}{f + \frac{1-f}{p}}$$

$$\text{Max speed up} = \frac{1}{f} \xrightarrow{\text{مثلا } f=0.1} \text{Max speed up} = 10 \Rightarrow \text{اضافه کردن cpu}$$

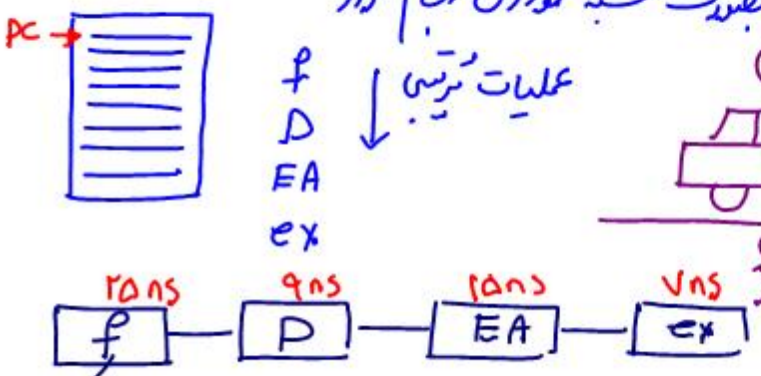
$p \to \infty$

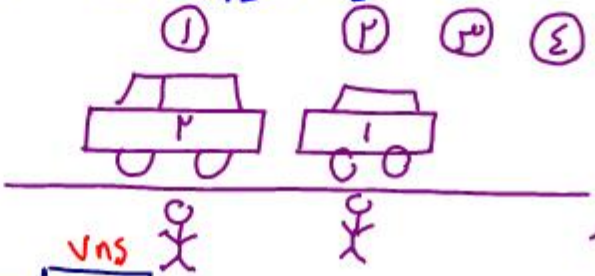
تانک خالی مناسب است

$$\text{efficiency (کارایی)} = \frac{\text{speed up}}{\text{number of processor}}$$

پایپ‌لاین

روشی است که باکمک آن می‌توان عملیات ترتیبی را بصورت شبه موازی انجام داد





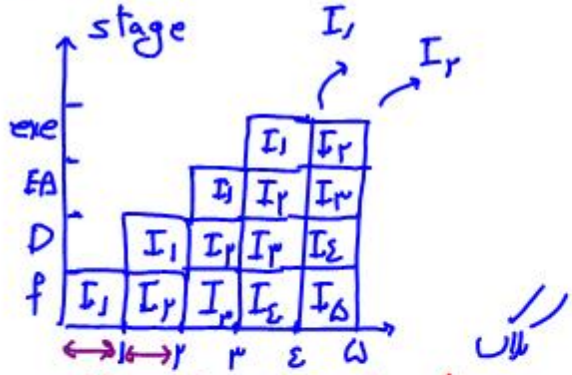
۱ـ چرخ ۴ را ندارند
۲ـ صندلی ۴ را ندارند
۳ـ شیشه ۴ تا را ندارند
۴ـ فرمان دست ندارند

سوال ـ اگر سیکل دستور چهار مرحله داشته باشد و هر مرحله (هر stage) + کلاک طول بکشد و n دستور داشته باشیم، speed up را حساب کنید؟

$$\text{speed up} = \frac{S_{pip}\ (\text{سرعت پایپ})}{S_{no\ pip}\ (\text{سرعت بدون پایپ})} = \frac{\text{زمان no pip}}{\text{زمان pip}} = \frac{n \times 4t}{4t + (n-1)t} = \frac{4n}{n+3}$$

$$\text{max speed up} = 4$$

$n \to \infty$ (تعداد دستورات)



در کلاک ۴ چند instruction هایی در پایپ ما هستند

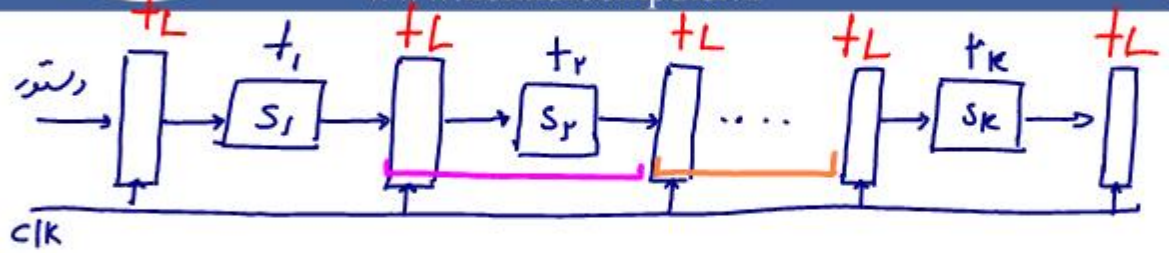
multi clock cycle diagram نمایش

بررسی کلی پایپ‌لاین K تایی

از آنجایی که تاخیر تبدیل‌ها لزوما مساوی نیست، می‌آیند و بین تبدیل‌ها ثبات می‌گذارند و به ثبات‌ها کلاک می‌زنند، بنابراین مشکل وتداخلی پیش نمی‌آید

معماری کامپیوتر

رامین رضوی



$T_{clk} = \max(t_1, t_2, \dots, t_K) + t_L$

زمان اجرا n دستور روی پایپ لاین

زمان اجرا n دستور = $KT + (n-1)T = (K-1+n)T$

دستور اول باید هر پایپ را طی کند و به انتها پایپ برسد (چون در این صورت پایپ پر می شود) از اون به بعد هر دستور با یک کلاک می افتد بیرون (اجرا می شود)

زمان اجرا n دستور بدون pip ← پس در این حالت ما ثبات نمی خواهیم

زمان اجرا no pip = $n \times (t_1 + t_2 + \dots + t_K)$

$speed\ up = \frac{n \times (t_1 + \dots + t_K)}{KT + (n-1)T}$

محاسبه speed up در حالت نهایی :

زمان اجرا n دستور بدون پایپ با یکسری فرضیات

فرضیات : ۱- تاخیر لبه ها یکسان باشند

۲- " ثبات های میان stage ها را ناچیز در نظر می گیریم

$T_{clk} = \max(t_1, t_2, \dots, t_K) + t_L \xrightarrow{t_1 = t_2 = \dots = t_K} T = t_1 = t_2 = \dots = t_K$

زمان اجرا n دستور در no pip = $n \times KT$

محاسبه speed up در ایدئال

$speed\ up = \frac{S_{pip}}{S_{no\ pip}} = \frac{t_{no\ pip}}{t_{pip}} = \frac{nKT}{(K-1+n)T} = \frac{nK}{K-1+n}$

Max speed up = K ← تعداد stage ⇒ خط لوله ای که K تا stage دارد می تواند سرعت را حداکثر K برابر کند

$n \to \infty$

throughput = تعداد دستورات انجام شده در واحد زمان = $\frac{\text{تعداد دستورات}}{\text{زمان اجرا}} = \frac{n}{KT + (n-1)T}$

توان عملیاتی یا گذردهی

فرکانس ۱۰۰ هرتز = در ۱ ثانیه ۱۰۰ تا لبه می‌بینی

$$\max_{n\to\infty} \text{throughput} = \frac{1}{T} = f$$

cpI : avg clock per instruction : متوسط تعداد کلاک به ازای هر دستور = تعداد کلاک‌ها / تعداد دستورات

$$= \frac{K+n-1}{n}$$

$$\max_{n\to\infty} cpI = 1$$ ⇐ به ازای هر دستور بطور متوسط ۱ کلاک می‌خواهیم

f دستور → f کلاک → ۱ ثانیه

هر کلاک ۱ دستور

مثال ـ فرض کنید یک پایپ‌لاین شش بخش داریم و تاخیر بخش‌ها به ترتیب ۲۰، ۱۰، ۱۵، ۱۵، ۱۰، ۱۵ ns است؛ اگر تاخیر لچ ۵ns باشد، max سرعت این پایپ به حالت غیر پایپ را بدست آورید؟

$$\text{speed up} = \frac{\text{سرعت pip}}{\text{no pip}} = \frac{\text{زمان no pip}}{\text{pip}} = \frac{n\times(20+10+\cdots)}{6\times25+(n-1)25} = \frac{85n}{6\times25+(n-1)25}$$

$$\max_{n\to\infty} \text{speed up} = \frac{85}{25} = 3.4$$

مثال ـ ۱۰۰ دستور روی یک پایپ ۸ بخشی اجرا شده است؛ زمان اجرا به چه میزان است اگر:

الف) دستورات بدون مشکل و پشت سرهم وارد پایپ‌لاین شوند

ب) " بصورت ۵ دسته مجزا ۲۰ دستوری وارد پایپ شوند

ج) " " " " از ۱ تا ۲۰، ۲۱ تا ۵۰، ۵۱ تا ۶۰، ۶۱ تا ۷۰، ۷۱ تا ۱۰۰ وارد پایپ شوند

د) دستورات بصورت ۶ دسته مجزا ۱ تا ۲۰، ۲۱ تا ۵۰، ۵۱ تا ۶۰، ۶۱ تا ۷۰، ۷۱ تا ۸۰ و ۸۱ تا ۱۰۰ ـــ شوند

الف) $8T + 99T = 107T$

ب) $5\times(8T+19T) = 135T$

ج) $(8T+19T)+(8T+29T)+(8T+9T)+(8T+9T)+(8T+29T) = 135T$

* فقط تعداد دسته‌ها مهم است و ربطی دسته‌ها اهمیتی ندارد

نکته در پایپ K بخشی اگر یک دسته به دستورات اضافه شود ما K-1 کلاک به زمان قبلی مان اضافه می‌شود (K-1 کلاک جریمه می‌شویم)

نکته) به ازای هر بار پر کردن پایپ ما باید ۱-K کلاک جریمه بدهیم ← هر بار پر کردن پایپ‌لاین به ۱-K کلاک زمان نیاز دارد

⇒ $135T + (8-1)T = 142T$

n دستور به صورت m دسته مجزا وارد یک پایپ K بندی شده است، زمان اجرا را حساب کنید

دستور اول هر دسته تمام پایپ را می‌روند و با m تا دستور اول دسته داریم، سایر دستورات مهم نیست که در کدام دسته هستند و هرکجا که باشند هرکدام با ۱ کلاک اجرا می‌شوند

(تفکر ۱) $mxKT + (n-m)T$

(تفکر ۲) همیشه می‌توانیم فرض کنیم که همه چی گل و بلبل است ← یعنی یک دسته داریم و همه دستورات در این دسته هستند و هیچ مخاطره‌ای هم رخ نمی‌دهد و پس زمان رایدست آوردیم و در انتها جریمه‌ها را محاسبه کنیم و به زمانی که بدست آورده‌ایم اضافه کنیم

زمان اجرا = $\underbrace{KT + (n-1)T}_{\text{فرض کردیم ۱ دسته داریم}} + \underbrace{(m-1)(K-1)T}_{\text{جریمه‌ها}}$

(تفکر ۳) هر دستور به هرحال ۱ کلاک را که می‌خواهد ← بنابراین n دستور n کلاک می‌خواهد و چون m دسته داریم، هر دسته ۱-K کلاک ما را جریمه می‌کند

زمان اجرا = $nT + m(K-1)T$

حل قسمت د مثال قبل با این سه تفکر

تفکر ۱: $6x8T + 94T = 142T$

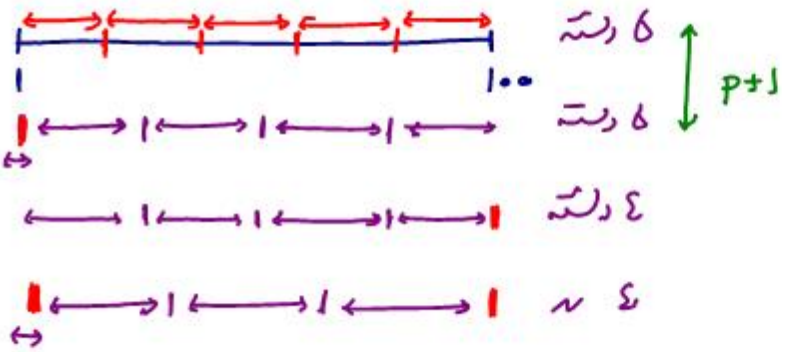
تفکر ۲: $8T + 99T + 5x7xT = 142T$

تفکر ۳: $100T + 6x7xT = 142T$

معماری کامپیوتر

رامین رضوی

نکته) اگر p پرش داشته باشیم، این p پرش یا p+1 دسته ایجاد می‌کند (Defualt) یا p دسته p+1 دسته در صورتی ایجاد می‌شود که دستور آخر برنامه پرش نباشد و گر آخرین دستور برنامه پرش باشد p دسته ایجاد می‌شود



تست) در یک برنامه n دستور وجود دارد که ۳۰ درصد آنها پرش است، وقتی دستورات پرش وارد پایپلاین می‌شوند ماشین اجازه ورود سایر دستورات به پایپلاین را نمی‌دهد تا پایپلاین خالی شود، حال اگر پایپلاین ۸ تایی باشد حداکثر تسریع را محاسبه کنید

تعداد دستورات پرش = ۰/۳n

- دستور آخر برنامه پرش نباشد (Defualt) ← ۰/۳n+۱ دسته = دستور آخر برنامه پرش است ← ۰/۳n دسته

روش ۱: $speed\ up = \frac{t_{no\ pip}}{t_{pip}} = \frac{n \times 8T}{(0/3n+1) \times 8T + (n-(0/3n+1))T} = \frac{8n}{2/4n+0/7n+7}$

روش ۲: $speed\ up = n = \frac{n \times 8T}{8T + (n-1)T + 0/3n \times 7T} = \frac{8n}{8+n-1+2/1n}$

روش ۳: $n = n = \frac{n \times 8T}{nT + (0/3n+1) 7T} = \frac{8n}{n+2/1n+7}$

$$\max speed\ up = \frac{8}{3/1} = 2/58 \quad (n \to \infty)$$

مخاطرات پایپلاین در خط لوله دستور العمل

لـ آن چیزهایی (موانعی) که باعث می‌شود همه‌چی عالی پیش نرود

انواع مخاطرات

۱- مخاطرات ساختاری (structural Hazard)
۲- " " داده‌ای (Data " "
۳- " " کنترلی (control " "

مخاطره ساختاری

اگر دو دستور از پایپلاین بخواهند بصورت همزمان از ۱ منبع سخت افزاری استفاده کنند

مثال- در یک پردازنده شکل پایپلاینش بصورت زیر است، اجرا ۱۰۰ دستور load و store در این ماشین چند کلاک می‌خواهد اگر:

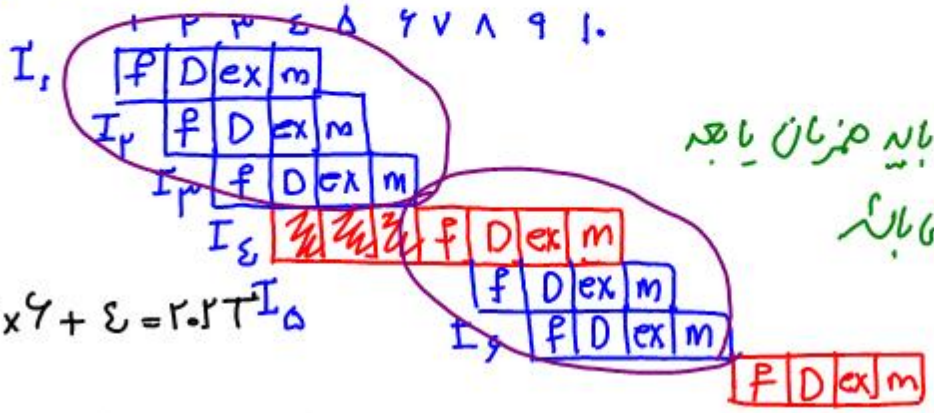
الف) حافظه دستور و دیتا مجزا باشد
ب) " " " مشترک باشد

توجه) در این ماشین دو بند هست که سراغ حافظه می‌رود یکی fetch است که f سراغ دستور می‌رود و mem سراغ داده می‌رود

۱- در حافظه بنویسیم نتیجه دستور را
۲- از حافظه بخوانیم: مثلا وقتی که دستور load است

الف) همه‌چی گل و بلبل خواهد بود و در پایپ وقفه‌ای رخ نمی‌دهد

زمان اجرا = $4T + 99T = 103T$



زمان اجرا = $33 \times 6 + 4 = 202T$

ب)
* fetch یک دستور باید همزمان با چه از Decode دستور قبلی باشد

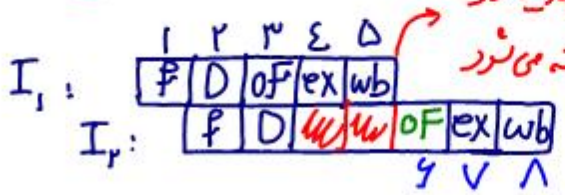
مخاطره داده‌ای یعنی وابستگی بین دستورات، مدل‌هایی مختلف از مخاطره داده‌ای وجود دارد که ۱ مدل آن به اینصورت است که ممکن است دستوری محتوای یک ثبات را نیاز داشته باشد که محتوا آن ثبات قرار است به روز شود، اما هنوز به روز نشده و مقدار قبلی اش (قدیمی اش) در آن است و اگر دستور ما برود و این مقدار را بخواند، مقدار قدیمی را خوانده بنابراین باید صبر کند تا مقدار ثبات آپدیت بشود و بعد آن را بخواند، پس مقداری علاف می‌شود

مثال- با فرض آن که پایپ پردازنده‌ای بصورت زیر باشد، اجرا ۲ دستور زیر در این ماشین چند کلاک طول می‌کشد؟

I_1: $R_1 \leftarrow R_1 + R_2$ write

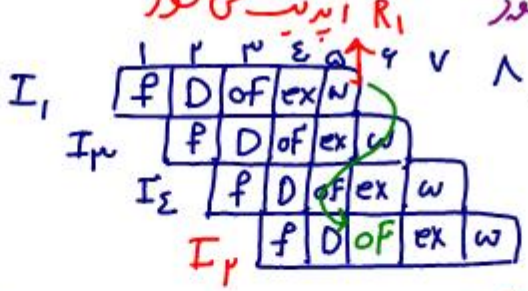
I_2: $R_4 \leftarrow R_1 + R_5$ Read ⇒

* مشکلی که من اینجا دارم در مورد آن صحبت میکنم مشکل (Raw)



* این دو دستور الزاما به هم وابسته نمی‌بودند باید در ۶ کلاک انجام می‌شد ولی حالا در ۸ کلاک انجام می‌شود

I_1: $R_1 \leftarrow R_1 + R_2$
I_2: $R_4 \leftarrow R_1 * R_5$
I_3: $R_3 \leftarrow R_3 - R_2$
I_4: $R_9 \leftarrow R_7 / R_8$

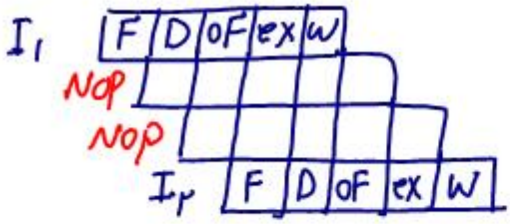


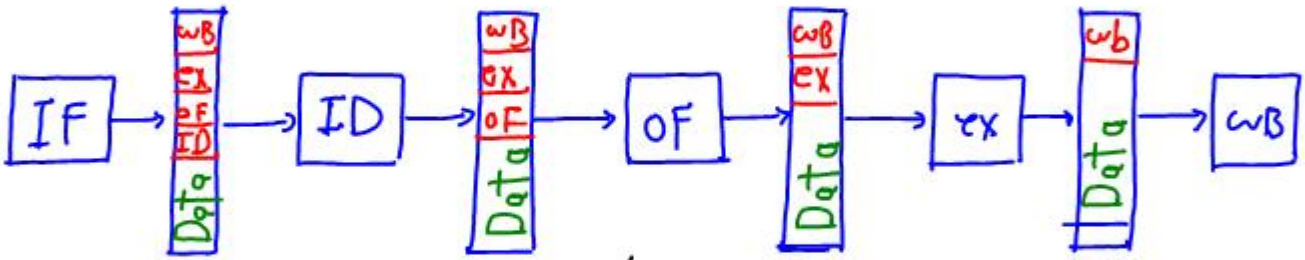
* با جابجا کردن دستورات مشکل را حل کردیم، در این حالت ۴ دستور در ۸ کلاک انجام شد ← به این کار می‌گویند جابجایی دستورات (delayed load) داریم لود را به تاخیر می‌اندازیم

برای رفع مخاطره داده‌ای روش‌هایی وجود دارد از جمله

۱- روش جابجایی دستورات، این روش نرم افزاری است و برخی از کامپایلرها سعی می‌کنند با جابجا کردن دستورات، دستوراتی را که بهم وابسته هستند را از هم دور کنند و بین آنها دستورات

دیگری قرار دهند بطوریکه منطق برنامه تغییری نکند ، اما اگر کامپایلر نتواند دستوراتی مانند دستور ۳و۴ در مثال بالا پیدا کند که بین دستورات ۱و۲ قرار بدهند ، بین این ۲ دستور ، به تعداد لازم دستور nop وارد می‌کند ، nop دستوری است که همه مراحل پایپ‌لاین را طی می‌کند ولی عملی در لثر این دستور انجام نمی‌شود (رخ نمی‌دهد) . با دستور nop تعداد کلاک کم نمی‌شود ولی حجم سخت افزار کاهش می‌یابد





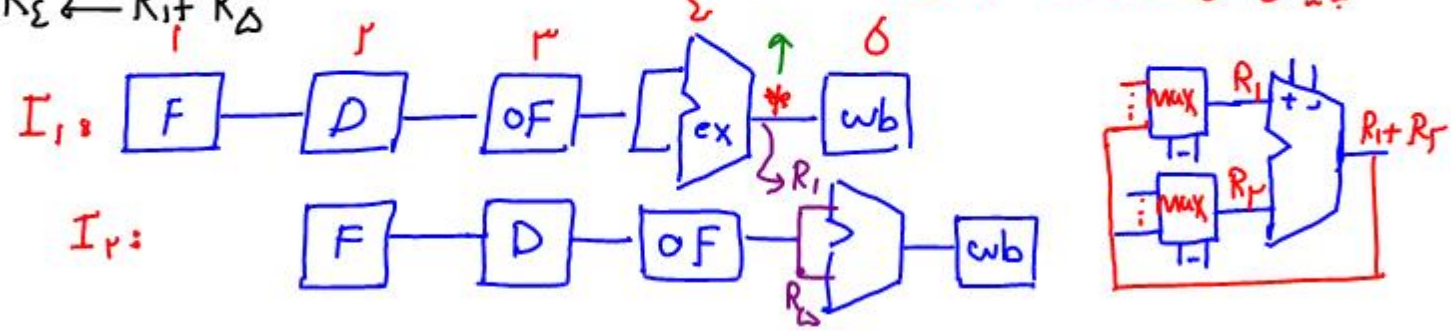
۲- روش سخت افزاری همین کار (همین عملکرد) out of order execution نام دارد که به اینصورت است که به جای اینکه کامپایلر قبل از اجرا دستورات را جابجا کند و این مشکل را حل کند ، در حین اجرا cpu این کار را انجام میدهد (سخت افزاری که اضافه کرده‌ایم)

۳- روش by passing or operand forwarding or internal forwarding

$I_1: R_1 \leftarrow R_1 + R_2$
$I_2: R_5 \leftarrow R_1 + R_5$

مقدار R_1 اینجا هست

پیش فرستادن داخلی عملوند



این روش سخت افزاری است و مسیرهایی داخل پایپ ماین اضافه می شود که به وسیله آنها می توانیم داده ها را درون پایپ برای دستورات بعدی که نیاز دارند forward کنیم. مثلاً در اینجا نتیجه اجرا یک دستور که در خروجی ALU وجود دارد همزمان که در ثبات مقصد نوشته می شود به ورودی ALU نیز وارد می شود تا دستورات بعدی اگر این مقدار را نیاز داشته باشند بتوانند از آن استفاده کنند

نکته مهم: همیشه نمی توانیم با استفاده از forwarding مشکل را حل کنیم و گاهی اوقات مجبور هستیم Nop اضافه کنیم

نکته: ما سه مدل مخاطره داده ای بصورت RAW، WAR، WAW داریم که آنچه که ما بررسی کردیم RAW است. WAR و WAW در پایپ لاین هایی که ما بررسی می کنیم رخ نمی دهد و ایجاد مشکل نمی کند و اینها معمولاً در Super Scalar ها مشکل ایجاد می کنند. البته باید با این مفاهیم آشنا باشید

f	D	EA	ex
f	D	EA	ex
f	D	EA	ex

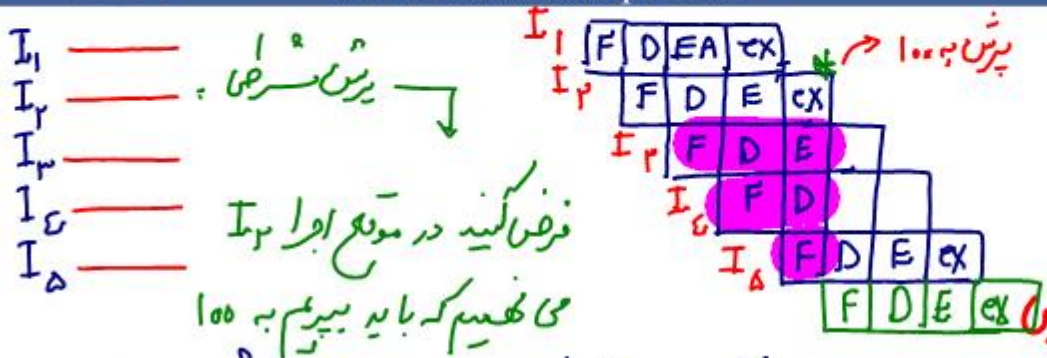
۳- مخاطره کنترلی

در اثر وجود دستورات انشعاب شرطی به وجود می آید

فرض کنید ما دستورات را پشت سرهم وارد پایپ می کنیم، حال فرض کنید یک دستور انشعاب شرطی وارد خط می شود، این دستور ۲ حالت دارد، یا می پرد (taken می شود) یا نمی پرد (not taken می شود)، اما ما در بدو exe متوجه می شویم که آیا می پریم یا نمی پریم اگر نپرد که مشکلی نیست ولی اگر بپرد و ما دستورات بعد از این دستور را به ترتیب وارد پایپ کرده باشیم، دستورات بعد از دستور انشعاب که وارد پایپ شده اند همگی اشتباه هستند و باید اینها را از پایپ برنیم دور و همچنین حالت سیستم را به قبل از ورود دستور انشعاب برگردانیم، که اگر پایپ بزرگ باشد هزینه این کار زیاد است.

معماری کامپیوتر

رامین رضوی



روش های رفع مخاطره کنترلی

۱ـ به محض اینکه تشخیص دادیم دستور انشعاب شرطی وارد پایپلاین شده، جلو ورود سایر دستورات به پایپلاین را بگیریم تا نتیجه انشعاب مشخص شود و بعد از اینکه نتیجه انشعاب مشخص شد از جایگاه پرش به ما می‌گوید دستورات را وارد پایپلاین کنیم

ـ در این روش اگر بخواهیم حتی یک دستور اضافی وارد پایپ نشود باید یک مقایسه کننده ساده به واحد های fetch اضافه کنیم که op دستوراتی که وارد می‌شوند را با op دستورات پرش شرطی مقایسه کند و در صورت برابری اجازه ورود دستور بعدی به پایپ را ندهند

۲ـ جابجایی دستورات (Delayed branch) : نرم افزاری است این روش.

در این روش کامپایلر به جای آن دستوراتی را که حتما باید اجرا شوند (چه پرش بپرد و چه نپرد) را بعد از دستور پرش وارد پایپ می‌کند، البته نباید منطق برنامه بهم بریزد حال مانند قبل اگر چنین دستوراتی پیدا نشد به تعداد لازم Nop وارد پایپ می‌کنیم

در هر سیکل که cx قرار دارد، به اندازه شماره آن بیت یکی کمتر

۳ـ پیش گویی انشعاب (Branch prediction) : پیش بینی کنیم که پرش شرطی که وارد پایپ می‌شود می پرد یا نمی‌پرد، برای پیش گویی ۲ روش وجود دارد

۱ـ پیش گویی استاتیک ۲ـ پیش گویی دینامیک

① استاتیک یعنی یک حدس ثابت. مثلاً پیش گویی می‌شود که دستورات انشعاب رو به عقب مانند for را همیشه taken بگیریم، حال اگر هنگام اجرا taken نشد که خوش

for(i=1 to 10000){

}

به‌حال‌مان ولی اگر not taken شد جریمه‌اش را می‌دهیم

a: ——

⋮

——

Isz B=-10000

BUN a

۲- پیش‌گویی دینامیک: دینامیک یعنی پیش‌گویی بر اساس سابقه، در واقع سابقه دستورات پرش در جداولی نگهداری می‌شود و بر اساس آن پیشگویی انجام می‌شود

مثال - زمان اجرا ۱۰۰ دستور در یک پایپ‌لاین 4-stage-3-issue چند است؟

به این ماشین، superscalar درجه ۳ نیز می‌گویند

F	D	EA	EX
F	D	EA	EX
F	D	EA	EX

تفکر ۱: فرض کنید سه پایپ بصورت موازی دارند فعالیت می‌کنند در اینصورت آن پایپی که ۳۴ دستور را می‌خواهد اجرا کند زمان کل را مشخص می‌کند

$4T + 33T = 37T$

۱	۲	۳	۴	۵
F	D	EA	EX	
F	D	EA	EX	
F	D	EA	EX	
	F	D	EA	EX
	F	D	EA	EX
	F	D	EA	EX
	⋮			
دسته ۳ تایی ۳۴ام		F	D	EA	EX
		F	D	EA	EX
		F	D	EA	EX
			F	D	EA	EX

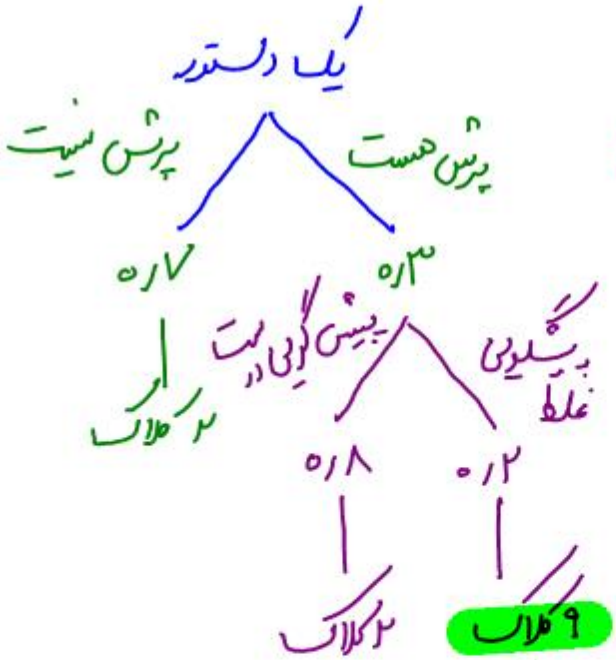
تفکر ۲ -

$4T + 32T + T = 37T$

معماری کامپیوتر

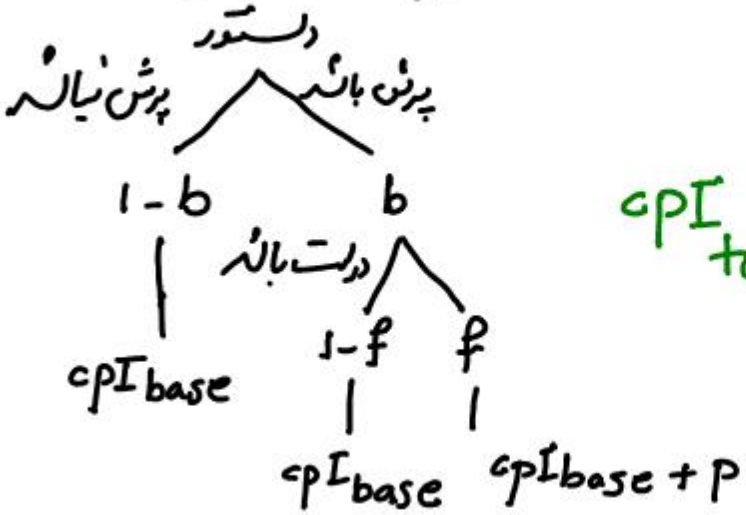
رامین رضوی

سوال - در یک پردازنده پایپ ۸ بندی است، اگر جریمه ان در کار نیامدن هر دستور بطور متوسط ۲ کلاک طول می‌کشد، در این برنامه ۳۰ درصد دستورات پرش هستند، که ۸۰ درصد آنها درست پیش گویی می‌شوند، پیش گویی غلط ۷ کلاک جریمه دارد، بطور متوسط هر دستور چند کلاک طول می‌کشد؟



$$CPI_{total} = 0/7 \times 2 + 0/3 \times 0/8 \times 2 + 0/3 \times 0/2 \times 9$$

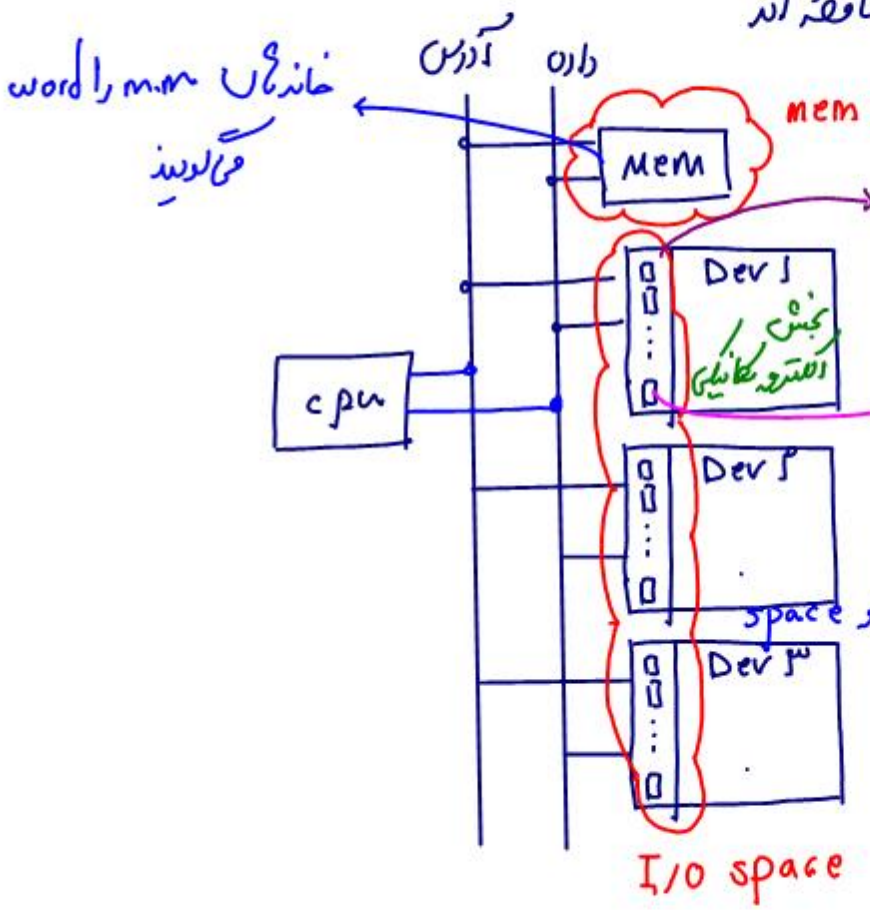
* احتمال اینکه دستور پرش باشد (branch) b =
* " " پیشگویی غلط باشد (fault) f =
* CPI_{base} = متوسط تعداد کلاک به ازای هر دستور بدون در نظر گرفتن جریمه
* تعداد کلاکی که در صورت پیشگویی غلط جریمه می‌شویم: (penalty) p



$$CPI_{total} = CPI_{base} + bfp$$

I/O

- هم وسایل جانبی از دید cpu بخشی از حافظه اند



تقسیم شده

1- mem space

2- I/O space

اگر همه پورت‌ها را کنار هم بذاریم می‌شود فضای آدرس دهی I/O و از I/O space برای نوشتن ارتباط cpu با وسایل جانبی استفاده می‌شود

port‌ها به ۳ دسته تقسیم می‌شوند

1- command : از دید cpu فقط نوشتنی است

2- Data : این پورت دوطرفه است هم خواندنی و هم نوشتنی است

3- status : برای cpu فقط خواندنی است

فرض کنید Dev1 پرینتر و آدرس پورت کاش بصورت زیر است.

status : ۱۳۰ H

data : ۲۸۸ H

command: DDD H

command

۱: کاراکتری که بهت میدم و چاپ کن
۲: کاغذ را بیرون بده
۳: خاموش شو
۴: برو به standby
۵: از standby خارج شو

status

۰: جوهرم تمام شده
۱: کاغذ درون گیر کرده
۲: من خاموشم
۳: کاغذم تموم شده
۴: مشغول چاپ ام
۵: من standby
۶: من آماده چاپ ام

- می خوایم یک برنامه بنویسیم که اون رو با داشتن یک pointer بیشتر فرخوانی اش کنیم و پرینتر آن رشته را چاپ کند. فرض کنید pointer به ابتدا رشته در ثبات R۲ قرار دارد و همین طور در ادامه رشته ای که می خواهیم چاپ کنیم ۰ قرار گرفته

معماری کامپیوتر

رامین رضوی

- cpu یک میلیون دستور در ثانیه اجرا کند
- پرینتر هر $\frac{1}{100}$ ثانیه یک کاراکتر چاپ کند ← در هر ثانیه ۱۰۰ کاراکتر چاپ می‌کند
- cpu در $\frac{1}{100}$ ثانیه ۱۰۰۰۰ دستور انجام میدهد

```
1- LD R1,[13.H]
2- cmp R1,6
3- jNE error
4- L2: LD R3,[R2];
5- cmp R3,0;
6- jE end
7- L1: LD R1,[13.H];
8- cmp R1,4;
9- jE L1
10- cmp R1,6
11- jNE error
12- sTR 1,R1
13- sTR R1,[DDD H]
14- sTR R3,[288 H]
15- INC R2
16- jmp L2
17- end: Ret
```

busy waiting

busy waiting بهتر داشتم و پیوسته

پورت status یک وسیله جانبی را می‌خواستم

در اصطلاح به این کار pool کردن یا سرکشی کردن می‌گویند

- روش interrupt :

در این روش یک وقفه یا به روی پردازنده به نام INT برای وقفه وجود دارد.

معماری کامپیوتر

رامین رضوی

$\bar{I}C_1: MI_1$

$\bar{I}C_2: MI_2$

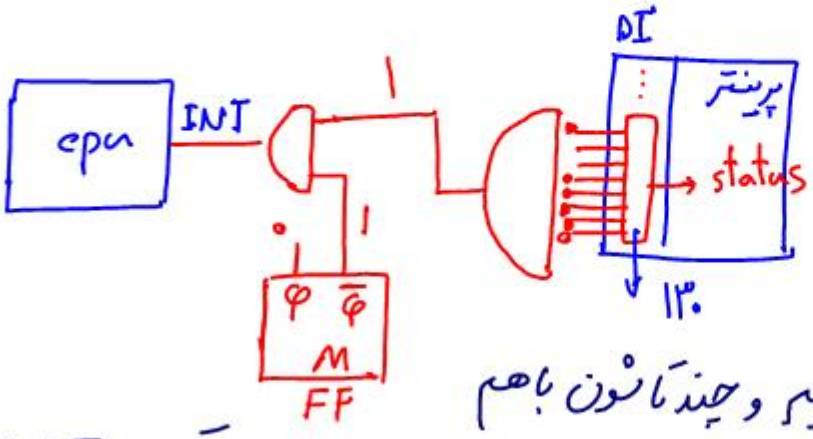
$\bar{I}C_3: MI_3$, if INT=1 then I←1;

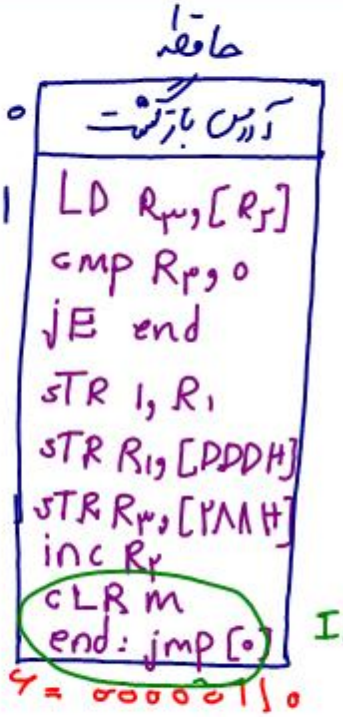
⋮

IT_0: AR←0, DR←PC, cL←0, M←1;

⋮

IT_3: PC←1, I←0, cL←1, M[AR]←DR





اگر status پرینتر این بود وقفه

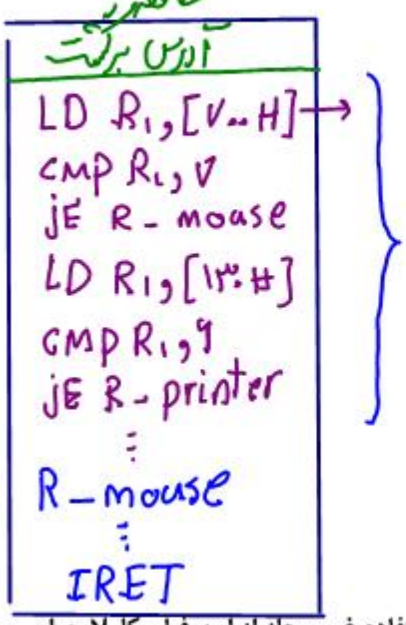
من آماده ام

* اگر چند وسیله جانبی داشته باشیم و چند تاشون باهم

وقفه بدهند، این سوال مطرح میشه که اول به کدوم سرویس بدهیم

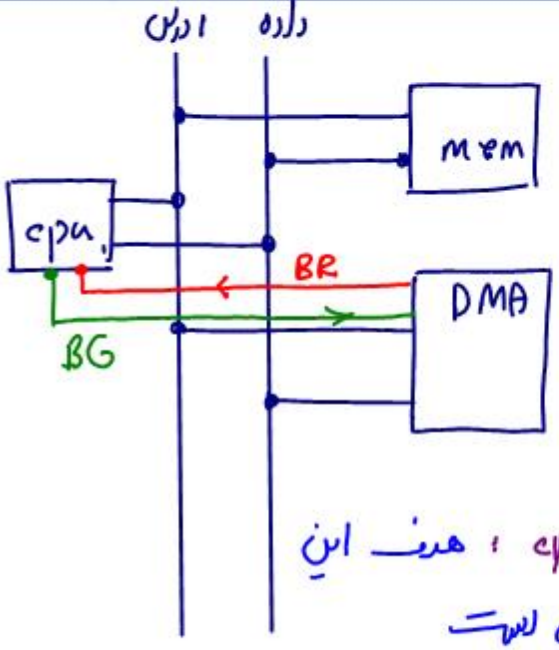
مدیریت وقفه بصورت نرم افزاری

" " " سخت افزاری



DMA :

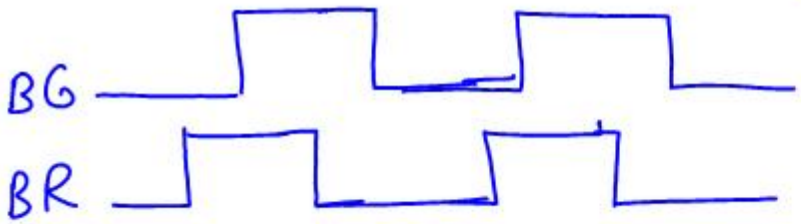
DMA دو تا مد کاری دارد که هرکدام با دیگری تناقض است و اینکه DMA در چه مدی عمل کند بستگی به command ای دارد که cpu به DMA میدهد



۱) cycle stealing یا cycle steal : هدف این مد جلوگیری از اتلاف وقت cpu است

۲) مد Burst : اصلا به دنبال جلوگیری از اتلاف وقت cpu نیست ولی در این مد وقت cpu تلف هم می شود. در این مد DMA دنبال این است که در نقل و انتقال داده به یک وسیله جانبی هیچ فاصله ای بین داده ها را نپذیرد و داده ها را پشت سر هم بفرستد، چون برخی از وسایل جانبی نیاز دارند که دیتا ها بصورت Burst بهشون برسه

مدهای DMA



سوال) اگر در مد Burst ، cpu معطل می شه، خب چرا داشته و ولی دیگه این کار را انجام بدهد، خب خودش انجام بده ① DMA کجره

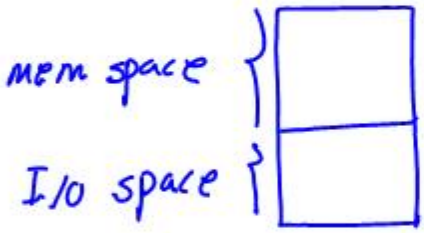
② اگر cpu انجام بده بین داده ها فاصله میفته

* فرض کنید cpu بخواهد ۲۰۰۰ لغت را از خانه ۵۰۰ حافظه بخواند و به وسیله جانبی منتقل کند

```
1. STR 0009 R1
2. L1: LD R2, [R1]
3. STR R2, [216H]  ⟵ فرض کنید آدرس پورت دیتا وسیله جانبی 216H
4. INC R1
5. cmp R1, 2500
6. JNE L1
```

* نحوه دسترسی به فضا mem space , I/O space

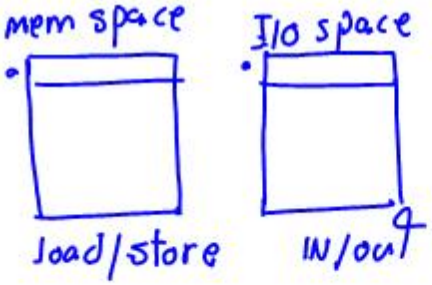
۱) memory mapped IO



از دستورات یکسانی برای دسترسی به mem space

و I/O space استفاده می‌شود و چون از دستورات یکسانی استفاده می‌شود آدرس‌های دو space باید با هم متفاوت باشد

۲) I/O mapped IO

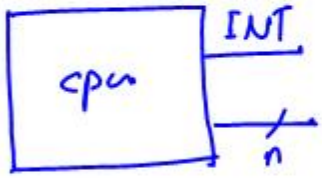


مدیریت وقفه بصورت سخت افزاری

در مدیریت وقفه بصورت سخت افزاری هر وقت که پایه وقفه cpu یک می‌شود، cpu آدرسی که قرار است به آن سپرد که روتین وقفه را اجرا کند را روی یک پایه دیتی آدرس دریافت می‌کند (pc ← address)

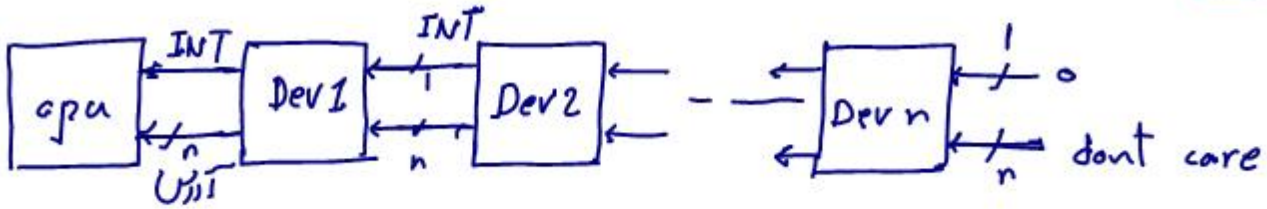
معماری کامپیوتر

رامین رضوی



* هر وسیله جانبی وقتی وقفه میدهد باید یک آدرس هم به cpu بدهد که cpu بپردازد به آن آدرس، پس در این حالت هر وسیله جانبی باید آدرس رقمی وقفه خود در حافظه را داشته باشد

daisy chain



۱	۱۵۰۰
۲	۳۰۰۰
⋮	⋮

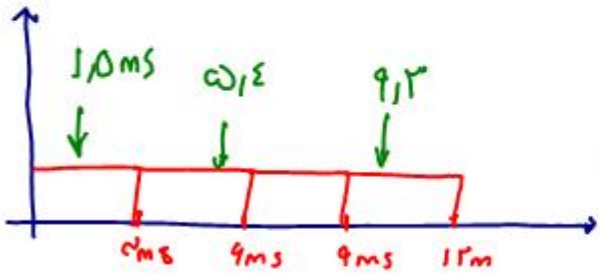
همگام سازی:

single strobe : در سرعت هر دو وسیله جانبی تغییر هست ولی کمی

همیشه از دیگری کندتر است

Hand shak : در هر لحظه حال Device کند می تواند عوض شود

single strobe:



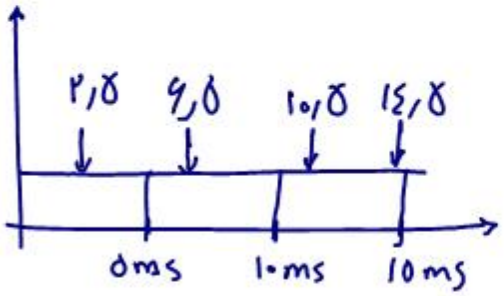
سناریو ۱: فرستنده هر ۳ms یکبار اطلاعات بفرستد

و گیرنده هر ۳,۹ ms " " را بخواند

سناریو دوم : فرستنده کندتر از گیرنده

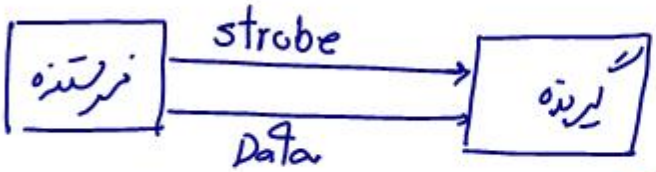
سرعت فرستنده : 5ms

" گیرنده : 4 ms



سناریو ۱ : فرستنده تندتر

" ۲ : فرستنده کندتر



پس strobe را همیشه می‌دهیم به دست طرف کندتر

تا بتواند با استفاده از آن به طرف سریعتر علامت دهد

الگوریتم : فرستنده ابتدا داده را روی خط قرار میدهد بعد با strobe علامت میدهد

و میگه بگیر گیرنده نیز هرگاه علامت را دریافت کرد داده را از خط میخواند

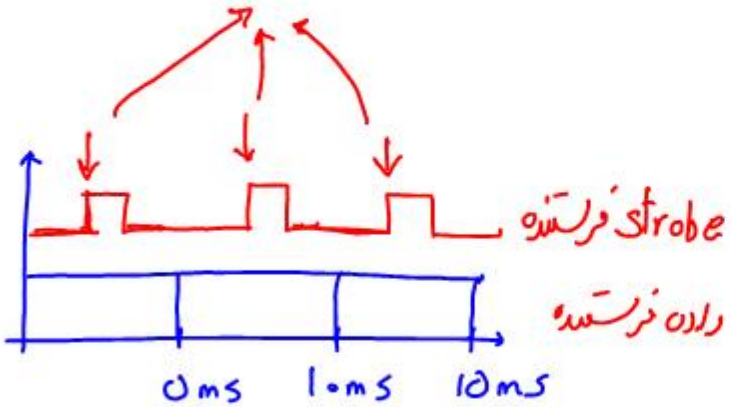
در این نقاط گیرنده می‌خواند

فرض کنید لبه مثبت strobe = بگیر

فرستنده کندتر

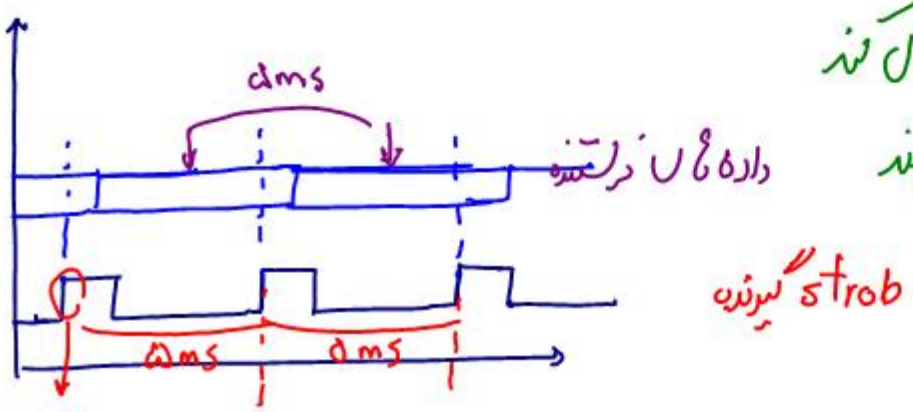
strobe دست فرستنده است و ارتباط

فرستنده آغاز کننده

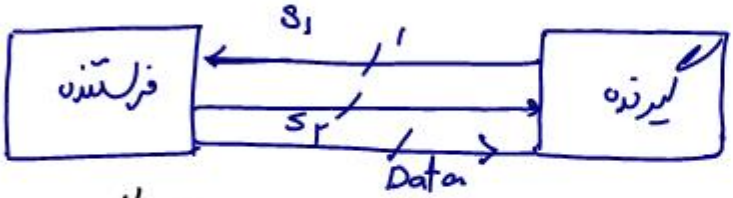


فرستنده تندتر : strobe دست گیرنده است می‌تواند ارتباط گیرنده آغاز کننده است

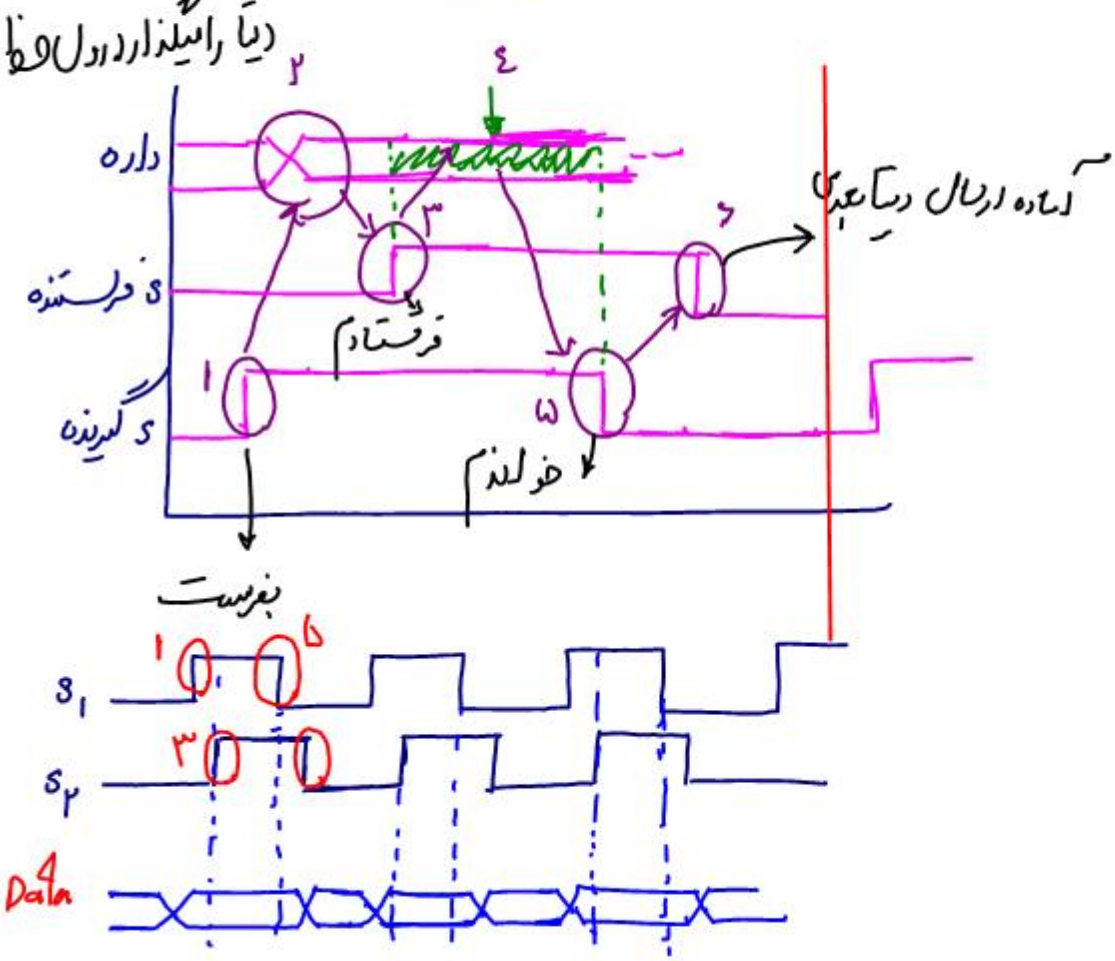
فرستنده هر 5ms داده ارسال کند
و گیرنده نه 5ms یکبار بخواند



Handshake



روش کار:



بهترین مدل ضرب آرایه‌ای از لحاظ تاخیر:

وقتی است که ما برای جمع partial product ها از CSA استفاده کنیم و در جمع پیروش CSA هر FA کری‌اش را می‌دهد به FA جنوب غربی‌اش.

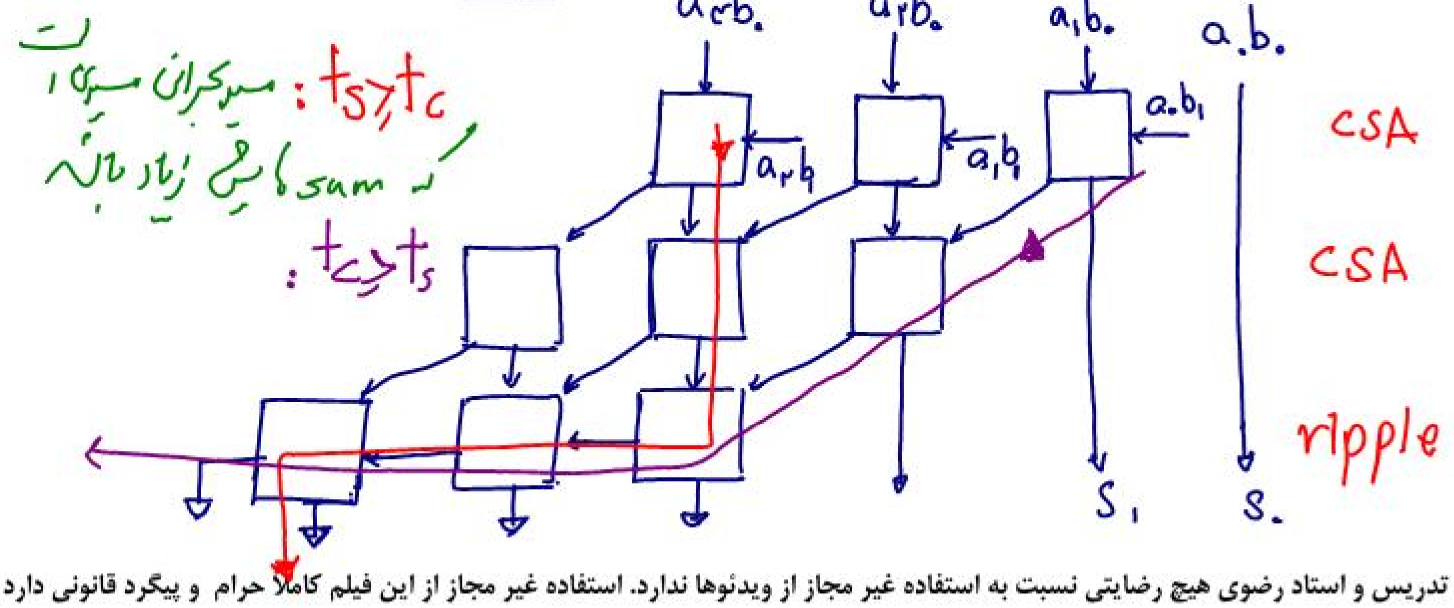
برای ضرب $m \times n$، n ردیف میذاریم و در هر ردیف $m-1$، FA می‌گذاریم، بنابراین در کل $n(m-1)$ تا FA داریم. FA های سطر اول می‌توانند HA نیز باشند. تعداد گیت های AND نیز $m \times n$ است.

حالا مثلاً اگر بخواهیم $A = a_3a_2a_1a_0$ را در $B = b_2b_1b_0$ ضرب کنیم، شکل مدار زیر حاصل می‌شود

$a_3a_2a_1a_0 = m$ $\quad m > n$

$\times \quad b_2b_1b_0 = n$

۳ ردیف داریم که در هر ردیف ۳ تا FA می‌خواهیم



معماری کامپیوتر

رامین رضوی

ما فقط این FA ها ردیف آخرکری داریم، و تعدادشون هم یکی کمتر از تعداد FA ها است.

$t_{AND} + n\,t_{sum} + (m-2)t_c$: $t_s \geq t_c$

$t_{AND} + (n-1+m-1)t_c = t_{AND} + (n+m-2)t_c$: $t_c \geq t_s$

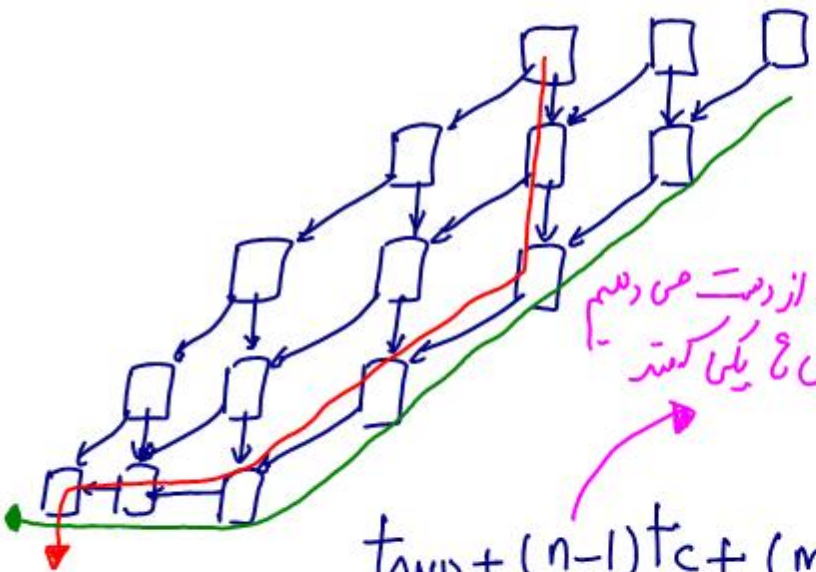
$\Rightarrow m > n$

یکی از تعداد سطرها کمتر داریم

به تعداد FA ها ردیف آخرکری داریم

مثال برای $m \leq n$:

ضرب ۴x۵ : ۵ ردیف داریم، در هر ردیف ۳ تا FA داریم:



$t_s \geq t_c$

$t_c \geq t_s$

این تعداد آخرکری را که به خاطر عمودی پایین آمدن از دست می دهیم در نهایت جبران اش می کنیم، بنابراین تعداد کری ها یکی کمتر از تعداد سطرها خواهد بود

$t_{AND} + (n-1)t_c + (m-1)t_s$: $t_s \geq t_c$

$t_{AND} + (n-1+m-1)t_c$: $t_c \geq t_s$

$\Rightarrow m \leq n$

عمقی که می آییم پایین همیشه یکی کمتر از تعداد FA ها همان است، و در نهایت هر یک sum داریم.

روش های تقسیم:

$A : \varphi \mid \frac{B}{\varphi}$ if $A \geq B \longleftrightarrow V_{21}$

روش ۱) مقایسه ای: A را با B مقایسه کن، اگر خورد سمت راست φ یک $\frac{\quad}{A}$ بذار (اگر در بار اول خورد بلوک ریزالت) و A را منهای B کن و پس $A:\varphi$ را شیفت به چپ بده و اگر هم نخورد، سمت راست φ صفر بذار، پس A,φ را شیفت به چپ بده و دوباره همین کار را تکرار کن.

در روش مقایسه A را با B مقایسه می‌کنیم (این بدان معناست که ماشفت اعدادی میان مقایسه داریم و می‌دانیم که مقایسه کننده حجیم است) بعد اگر لازم بود تفریق می‌کنیم و اگر لازم نبود اصلا تفریق نمی‌کنیم.

n تا شیفت
n تا مقایسه
به تعداد یک های خارج قسمت (φ) تفریق داریم: متوسط $\frac{n}{2}$ تفریق

روش ۲) روش جبرانی (restoring)

می‌گوید بیا و بدون اینکه مقایسه کنی A را منهای B کن، حالا نتیجه را چک کن، اگر نتیجه + بود، معنی اش این است که خوب کاری کردیم که A را منهای B کردیم، یعنی A واقعا از B بیشتر بوده، و بنابراین برو و تو خارج قسمت ۱ بذار و پس A,φ را شیفت به چپ بده و برو مرحله بعد، ولی اگر بدبختانه نتیجه - است، یعنی اشتباه کردیم که A را منهای B کردیم، حالا باید بیایم

و کار خودمان را جبرانی کنیم یعنی: $(A-B)+B$ و در خارج قسمت ۰ بذاریم و پس $A:\varphi$ را شیفت چپ بده و برو به مرحله بعد.

* ممکن است ما در ۱ مرحله هم جمع داشته باشیم و هم تفریق

n تا شیفت

n تا تفریق: هر مرحله تفریق را داشتیم، حالا اگر اشتباه کرده بودیم جمع هم داشتیم

تعداد عمل جمع: متوسط $\frac{n}{2}$: به تعداد ۰ ها ی خارج قسمت،

روش ۳) روش غیرجبرانی (non-restoring):

جبرانی: $A-B$ ← شیفت + : خارج قسمت ۱

$A-B$ ← " " − : " " ۰

بعد شیفت: $(A-B)+B$

به چپ می‌آوردم و می‌رفتم مرحله بعد: $A \leftarrow 2A$، حالا در مرحله بعد دوباره $A \leftarrow 2A-B$

غیرجبرانی وقتی $A-B$ مثبت است یا جبرانی وقتی می‌کند و فرق این دو تا این است که $A-B$ منفی است. اما وقتیکه حاصل $A-B$ منفی بود و اشتباه کرده باشیم، غیرجبرانی میگه جبرانی نکن و خارج قسمت را ۰ بگذار و شیفت به چپ بده و برو مرحله بعد، اما به جای اینکه در مرحله بعد تفریق کنی جمع کن.

$A \leftarrow 2(A-B)$ ← $2(A-B)+B$ ← $A \leftarrow 2A-2B+B$ & $A \leftarrow 2A-B$

n تا شیفت

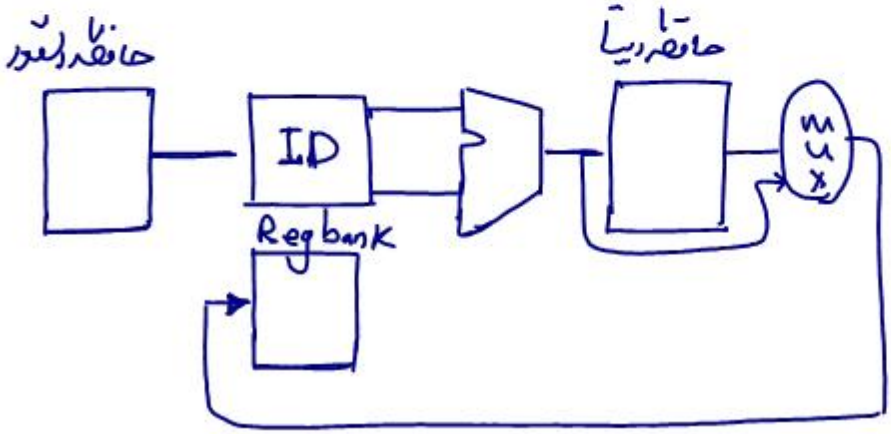
تعداد تقریبی mux : متوسط $\frac{n}{2}$: به تعداد 1 های خارج قسمت

" جمع mux : " " " " " " : " " " " " "

نکته) در غیر جبرانی ما در هر مرحله یا تقریب داریم یا جمع

نکته) در غیر جبرانی شاید نیاز باشد ما در مرحله آخر یک جمع اضافی داشته باشیم

ماشین های MIPS :



هم رمزگشایی انجام می‌شود و هم رجیسترهای مورد نیاز خوانده می‌شود

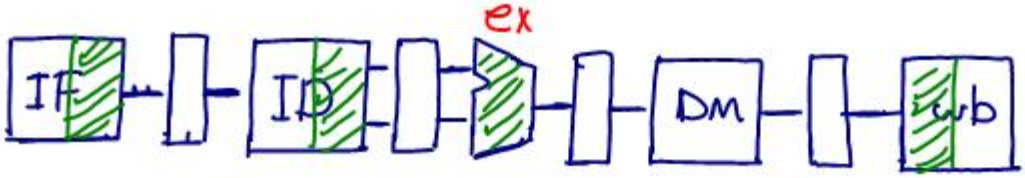
دستورات load و store با حافظه دیتا کار دارند

انواع دستور :

دستورات R-type : دستوراتی که بر روی کار با رجیسترها استفاده می‌شوند.

add $s0, $t0, $t1:
$t1 ← Destination
$s0 ← رجیستر سورس اول
$t0 ← رجیستر سورس دوم

$t1 : s0 + t0
نتیجه از طریق فرعی Alu می رود تا دوتا دوتا Reg file نوشته شود



* در MIPS نوشتن در نیمه اول کلاک انجام می شود و خواندن در نیمه دوم کلاک.

* در دستورات R-type ، IF, ID و ex و wb استفاده می شود و mem استفاده نمی شود

دستور load : قرار است Data را از قسمت Data memory بردارد و بیاد توی Reg file بنویسه. همه واحدها در دستور load استفاده می شوند (Alu در اینجا برای تولید آدرس خانه ای از Data mem که می خواهیم دیتایش را برداریم و به Reg file منتقل کنیم استفاده می شود)

دستور store : این استور مقدار یک ثبات را در یک خانه حافظه ذخیره می کند.

دستور store از واحد wb استفاده نمی کند

دستور Branch on equal : در اینجا فقط واحد IF و ID و ex استفاده می شود

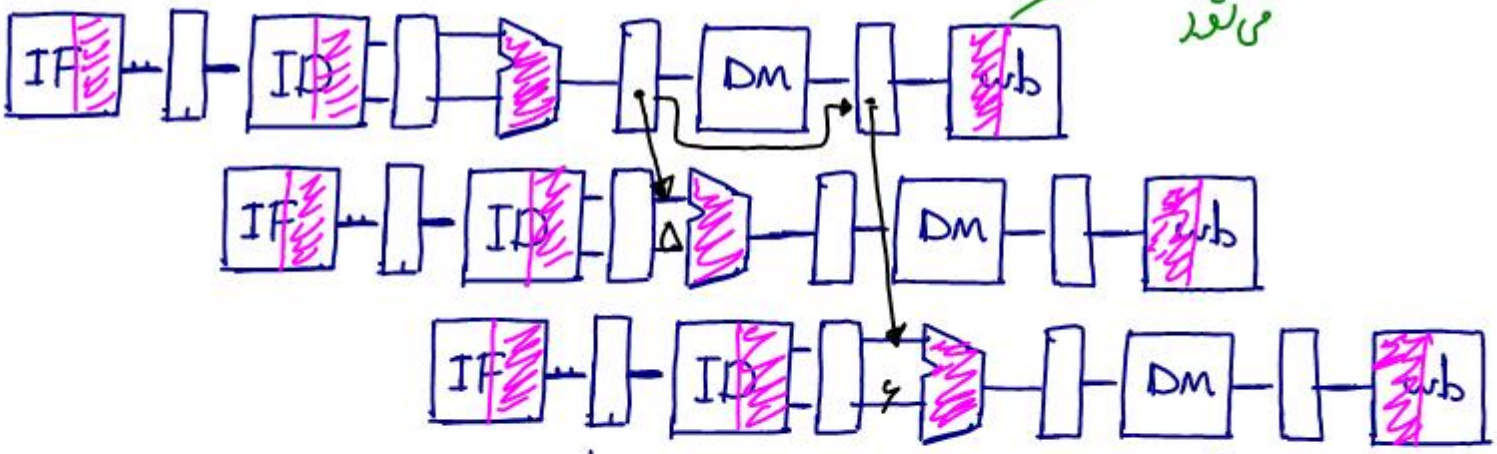
* یک دستور می تواند دستوری را بیاورد درحالیکه ۲ دستور بعد از آن ، به مقدار آن نیاز داشته باشند که با forwarding شاید بتوان مشکل هر ۲ دستور بعد را حل کرد.

```
sub  $2, $1, $3
and  $12, $2, $5
or   $13, $6, $2
```

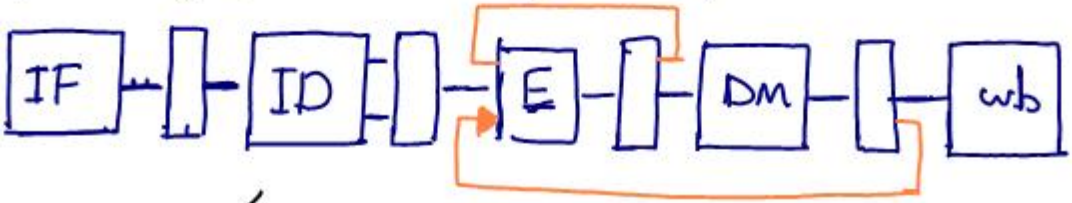
این دو به مقدار ثبات 2 نیاز دارند
که با فرولد دینگ می توانیم مشکل

هر دو را حل کنیم

S2 اینجا نوشته می شود



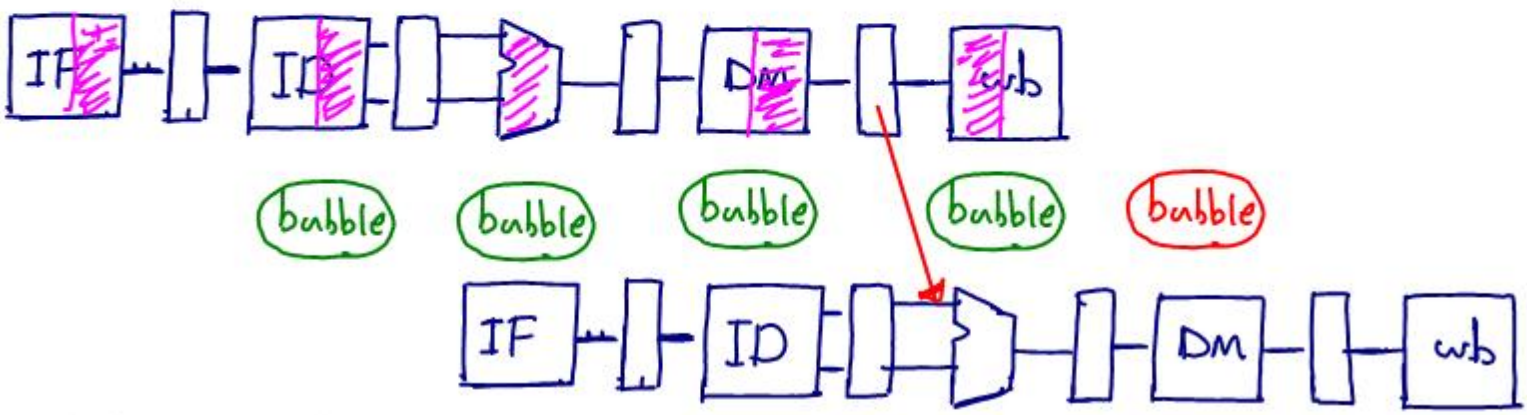
* forwarding فقط از خروجی Alu نیست و از جاهای دیگر نیز می تواند باشد



* با استفاده از forwarding همیشه می توانیم از stall پرهیز کنیم.

```
lw  $ s0, 20($t1)
sab $ t2, $s0, $t3
```

از آدرس ۲۰ بعلاوه محتویات t1 مقداری خوانده می شود
و در s0 نوشته می شود



سوال) اگر کد زیر را به دستورات اسمبلی تبدیل کنیم و با پایپ لاینهای ماشین mips اجرا کنیم چند کلاک طول می‌کشد؟

A = B + E
C = B + F

B در خانه ۰، E در خانه ۴، F در خانه ۸، A در خانه ۱۲ و C در خانه ۱۶ ذخیره شود

```
lw   $t1, 0($t0)      ← ۵ کلاک می‌خواهیم تا pip پر شود
stall →
lw   $t2, 4($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
stall →
lw   $t4, 8($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

13 cycle

```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
lw   $t4, 8($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

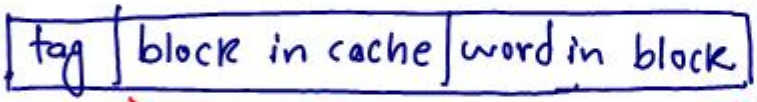
11 cycle

$t1$ می رود در B
$t2$ " " E
A بود در خانه 12
$t4$ " " F

سوال) اگر سایز بلاکمان را در نگاشت مستقیم زیاد کنیم چه اتفاقی می‌افتد؟ (در حالتی که حجم کشمان ثابت است)

- larger block shoud reduce miss rate
  - Due to special locality

ما هر دفعه داریم دیتای بیشتری را می‌آوریم در کش، بنابراین علت locality مکانی hit rate مون بالا می‌رود، اما چون گفته شده حجم کشمان ثابت است بنابراین کشمان چاق‌تر ولی کوتاه‌تر می‌شود



این ۲ فیلد بود که کش را آدرس دهی می‌کرد ← ثابت است

با افزایش block in cache، word in block کاهش می‌یابد، بنابراین احتمال برخورد ۲ بلاک با هم افزایش می‌یابد، بنابراین hit rate کم می‌شود.

مثلاً وقتی block in cache سه بیت باشد احتمال برخورد ۲ بلاک با هم $\frac{1}{8}$ ولی وقتی block in cache دو بیت شود احتمال برخورد ۲ بلاک $\frac{1}{4}$ می‌شود.

بنابراین لزوماً با زیاد کردن سایز بلاک ما به جواب بهتری نمی‌رسیم

- But in a fixed size cache

  larger blocks → fewer of them ⇒

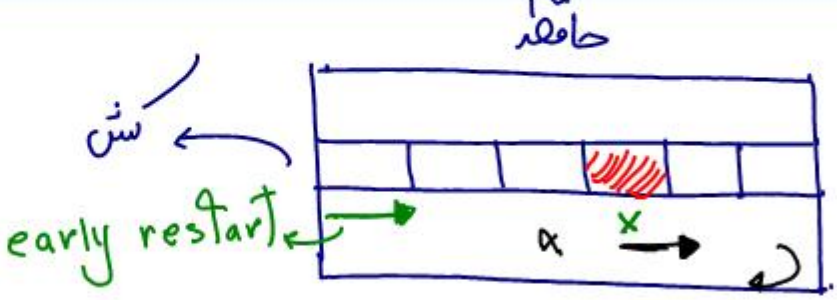
  1) more competition ⇒ increase miss rate
  2) larger block ⇒ pollution (آلودگی)

ما با اضافه کردن سایز بلاک دیتای بیشتری را می‌آوردیم توی کش، اما لزوماً به همشون نیاز احتیاج پیدا نکنیم و در نتیجه احتمالاً قسمتی از آن را بیخودی آوردیم اش

* کشمان را پر کردیم ولی ازش استفاده نکردیم

معماری کامپیوتر

رامین رضوی



early restart : خواندن از حافظه بایت به بایت انجام می‌شود، پس بایت اول حافظه خوانده می‌شود و در بایت اول بلاک مربوطه در کش قرار می‌گیرد، پس بایت دوم خوانده شده به بایت دوم بلاک مربوطه منتقل می‌شود، این روند ادامه دارد تا به پایتی برسیم که CPU آن را می‌خواهد، در اینصورت کش به CPU اطلاع می‌دهد، که دیتای که می‌خواهی آماده شده و می‌توانی آن را برداری و به کار ادامه دهی، حال کش خودش بقیه بایت‌ها را از حافظه برمی‌دارد و بقیه بلاکش در کش را پر می‌کند

critical word first: word ای که بحرانی هست را اول از همه می‌آوریم در کش

نکته: روش early restart بیشتر به درد Instruction cache ها می‌خورد، چون دسترسی ما در مورد Instruction ها معمولاً ترتیبی است - و روش دوم بیشتر به درد Data cache ها می‌خورد زیرا دسترسی ما به Data ها معمولاً یک حالت رندم دارد.

سوال) فرض کنید ۴ تا ماژول (8K word) حافظه به صورت low و high بانک بانک سازی شده‌اند، سرعت low چند برابر high است اگر

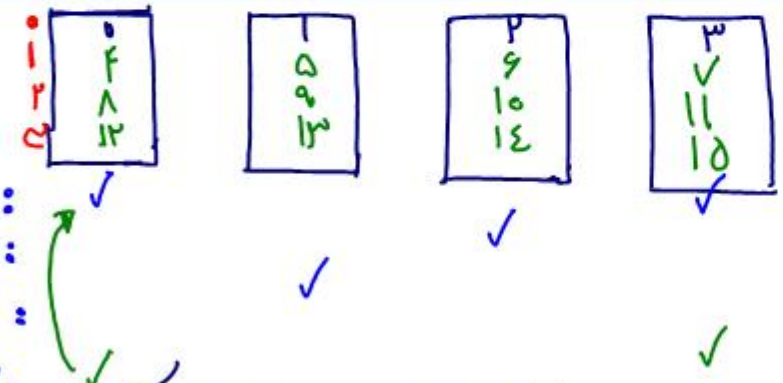
الف) بخواهیم با stride یک بخوانیم (با فاصله آدرسی ۱)

ب) " " " " " (" " ۲)

ج) " " " " " (" " ۳)

معماری کامپیوتر

رامین رضوی



اولین درخواست :
دومین " :
سومین " :
چهارمین " :

الف) می‌خواهیم با فاصله آدرس ۱ بخوانیم. یعنی پردازنده آدرس‌های متوالی را تولید می‌کند

در low با هر access چهار آدرس متوالی را می‌توانیم داشته باشیم

ب) * در high همیشه با هر درخواست یک کلمه خوانده می‌شود

cpu = ۰, ۲, ۴, ۶, ۸, ۱۰, ۱۲, ۱۴, ...

در low با هر درخواست دو کلمه را می‌توانیم به cpu بدهیم. سرعت low دو برابر high است.

ج)

cpu = ۰, ۳, ۶, ۹, ۱۲, ۱۵, ۱۸, ۲۱, ...

اولین درخواست = ۰ و ۳
دومین " = ۶
سومین " = ۹
چهارمین " = تناوب

→ $\frac{\text{تعداد کلمات خوانده شده}}{\text{تعداد درخواست}} = \frac{۴}{۳}$

نکته: اگر m تراشه داشته باشیم و بخواهیم با فاصله آدرسی K بخوانیم آن‌گاه اگر $m > k$ باشد، سرعت low، $\frac{m}{k}$ برابر سرعت high و اگر $k \geq m$ باشد آن‌گاه سرعت low با high فرقی ندارد.

معماری کامپیوتر

امین رضوی

سوال) فرض کنید حافظه از ٤ تا ماژول تشکیل شده و می‌خواهیم متناوبا با فاصله آدرسی ۲ و ۳ بخوانیم تسریع low به high چقدر است؟

cpu = ۰، ۲، ۵، ۷، ۱۰، ۱۲، ۱۵، ۱۷، ۲۰، ۲۲، ۲۵، ۲۷، ۳۰، ۳۲، ۴۵، ۴۷، ٤٠

۰	۱	۲	۳
٤	۵	۶	۷
۸	۹	۱۰	۱۱
۱۲	۱۳	۱٤	۱۸
۱۶	۱۷	۱۸	۱۹
۲۰	۲۱	۲۲	۲۳
۲٤	۲۵	۲۶	۲۷

درخواست اول: ✓ ✓ ✓ ✓

" دوم: ✓

" سوم: ✓

" چهارم: ✓ ✓

" پنجم: ✓

" ششم: ✓

درخواست اول = ۰، ۲

" دوم = ۵، ۷

" سوم = ۱۰

" چهارم = ۱۲، ۱۵

" پنجم = ۱۷

" ششم = ۲۰، ۲۲ x تناوب

جواب = ۵/۵

سوال) عدد BCD ، A را در نظر بگیرید برای محاسبه $5 \times A$ حداقل سخت افزار مورد نیاز چقدر است

$0101 \times A =$

A می تواند ۰ تا ۹ باشد، بنابراین حاصل نهایی می تواند ۴۵ باشد که ۲ رقم BCD برای نمایش حاصل کافی خواهد بود.

$0\,a_3\,a_2\,a_1\,0\,a_0\,0\,a_0$

$101 \times a_3a_2a_1a_0$	=	$0\,a_3\,a_2\,a_1$	$0\,a_0\,0\,a_0$
0000	=	0000	0000
0001	=	0000	0101
0010	=	0001	0000
0011	=	0001	0101
0100	=	0010	0000
0101	=	0010	0101
0110	=	0011	0000
0111	=	0011	0101
1000	=	0100	0000
1001	=	0100	0101

سوال) حداقل سخت افزار مورد نیاز برای $50.A$ چیست؟

$5 \times 10 \times A = 10 \times 5A \rightarrow 0\,a_3\,a_2\,a_1\,0\,a_0\,0\,a_0\,0000$

سوال) اگر A دو رقمی باشد حداقل سخت افزار لازم برای $50.A$ چقدر است؟

$50(A_2A_1)_{10} = 50(A_1 + 10A_2) = 10 \times 5A_1 + 10^2 \times 5A_2$

FA, HA, HA, C, C

$\quad\quad 0\ a_3\ a_2\ a_1\ 0\ a_0\ 0\ a_0\ 0000 = 10 \times 5A_1$

$+\ 0\ a_3\ a_2\ a_1\ 0\ a_0\ 0\ a_0\ 0000\ 0000 = 10^2 \times 5A_2$

$0\ a_3\ a_2\ a_1\ C\ \downarrow\ \downarrow\ \downarrow\ 0\ a_0\ 0\ a_0\ 0000$

۲ تا HA و یک FA.

سوال) اعداد ۳ بیتی A و B و C و یک جمع کننده ۴ بیتی داریم مطلوبست محاسبه؟

$16\times(A+B \text{ div } 2+20)+2C+1$

$20 = 10100$ (بیت‌های ۴ ۳ ۲ ۱ ۰)

$2C+1 = c_2c_1c_0 1$ *

$B \text{ div } 2 = b_2b_1$

$16A = a_2a_1a_0\,0000$

$16(B \text{ div } 2) = b_2b_1\,0000$ *

$16\times 20 = 10100\,0000$ *

$101011 \rightarrow 32+11=43$

$43 \div 2 = 21$، باقیمانده 1

① $21 = 10101$ ← خارج قسمت

جمع کننده ۴ بیتی

$$
\begin{array}{r}
1\,0\,1\,b_2\,b_1\;|\;c_2\,c_1\,c_0\,1\\
+\;0\,0\,a_2\,a_1\,a_0\;|\;0\,0\,0\,0\\
\hline
1\;\downarrow\downarrow\downarrow\downarrow\;c_2\,c_1\,c_0\,1
\end{array}
$$

سوال) محاسبه $3A+132$ (A یک عدد دودویی بدون علامت ۷ بیتی است)

روش ۱) $3A=2A+A$

روش ۲)

$$
\begin{array}{r}
a_6a_5a_4a_3a_2a_1a_0\\
\times\qquad\qquad 11\\
\hline
1\;a_6\;a_5\;a_4\;a_3\;a_2\;a_1\;a_0\\
a_6\;a_5\;a_4\;a_3\;a_2\;a_1\;a_0\\
1\;0\;0\;0\;0\;1\;0\;0\\
\hline
\end{array}
$$

یک جمع کننده ۸ بیتی

$132 = 10000100$ (بیت‌های ۷ ۶ ۵ ۴ ۳ ۲ ۱ ۰)

به جای ۱۳۲ می‌ذاشتم ۱۲۹.
با ۱ جمع کننده ۸ بیتی می‌شد

سوال) $125 \times A - 100(A \bmod 4) + 2$ (A ، BCD است)

$a_3 a_2 a_1 a_0$

	$000a_3$		$0a_1a_0a_1$	$0a_0 0a_0$
125×0 =	0000	0000	0000	0000
125×1 =	0000	0001	0010	0101
125×2 = 250:	0000	0010	0101	0000
125×3 = 375:	0000	0011	0111	0101
125×4 = 500:	0000	0101	0000	0000
125×5 = 625:	0000	0110	0010	0101
125×6 = 750:	0000	0111	0101	0000
125×7 = 875:	0000	1000	0111	0101
125×8 = 1000:	0001	0000	0000	0000
125×9 = 1125:	0001	0001	0010	0101
+	0000	0000	0000	0010 = 2
−	0000	$00a_1a_0$	0000	0000

$a_1 a_0$

				A		$0a_2 0a_2$
0	=	0000	−	0000	=	0000
1	=	0001	−	0001	=	0000
2	=	0010	−	0010	=	0000
3	=	0011	−	0011	=	0000
5	=	0101	−	0100	=	0101
6	=	0110	−	0101	=	0101
7	=	0111	−	0110	=	0101
8	=	1000	−	0111	=	0101
0	=	0000	−	1000	=	0000
1	=	0001	−	1001	=	0000

اعداد (عدد) های یک رقمی A و B مفروض اند

$505A + 2 \times (B \bmod 2) + 200$