①✓

ترم ۷

Subject: .... Date: ۹۲/۶/۲۳ جلسه ی اول : ۲۳/۶/۹۲

درس معماری کامپیوتر ۲ — استاد: سعید صفری

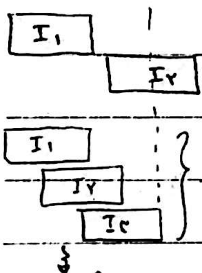
فهرست مطالب:
- ✓ حساب کامپیوتری
- ✓ سیستم نمایش اعداد (حافظه، پردازنده)
- ✓ جمع / تفریق
- ✓ ضرب — Add & Shift
- ✓ تقسیم
- ✓ ممیز شناور

ISA → Instruction set Architector ← اسمبلی پروگرمینگ نمی باشد.
(نرم افزار / ISA / سخت افزار)

- ✓ معماری مجموعه دستورات پردازنده MIPS
- ✓ طراحی پردازنده
- ✓ طراحی مسیر داده Data path
- ✓ طراحی واحد کنترل

single cycle — کارایی ↑ — Multi-cycle — کارایی ↑ — پایپ لاین

- * معیارهای ارزیابی کارایی پردازنده ها
- ✓ طراحی پردازنده به صورت پایپ لاین
  - * مسیر داده پایپ لاین
  - * واحد کنترل پایپ لاین
- * بررسی انواع مخاطره ها و روش های برطرف کردن آن ها — Hazards



→ پایپ لاین (افزایش سرعت پردازش)

overlap → hazard دارد.

Subject: Date:

✓ سلسله مراتب حافظه: همه حافظه‌ها با سرعت پردازنده‌ها رشد نمی‌کنند پس تعیین کننده‌ی سرعت، حافظه‌ها هستند که می‌خواهیم کم بودن سرعت حافظه‌ها را جبران کنیم.

* cache
* Main Memory
* HDD

✓ فرکانس‌های کاری در نسل پردازنده‌ها افزایش پیدا می‌کرد ولی از یک جایی به بعد فرکانس را بالا نبردند چون فرکانس متناسب با توان مصرفی است پس توان بالا می‌رفت و باعث داغ شدن پردازنده می‌شد و همچنین برای وسیله‌ای مثل موبایل باعث زود تمام شدن باتری می‌شد. ✓

✓ پردازنده‌های چند هسته‌ای

✓ واسط ارتباطی ← مبحث I/O - bussing ها و ...

مرجع درس:

✓ D.A. Patterson, J.L. Hennssy, "Computer organization & Design: The Hw/sw Interface". Margan kaufmann Pub. /۲۰۱۵

ارزیابی:

۱۵٪ میان‌ترم 1 شنبه ۲ آذر ← هر کلاس ← تا آخر Multi-cycle

۱۵٪ میان‌ترم ۲ شنبه ۲۳ آذر ← در این امتحان طراحی Data path و واحد کنترل

۳۵٪ پایان‌ترم ← بیشتر از مباحث آخر ترم است که یک پردازنده را به طور کامل می‌خواهد که مثلاً در Hw ها داده شده باشد

۲۰٪ Hw ← کدهای verilog ← مورد دیگر را در پایان‌ترم می‌خواهند ← مثلاً طراحی single و multi در Hw می‌آید

٪۱۰ Final project ← پیاده‌سازی میان‌ترم ۲ } طراحی پایپ‌لاین در امتحان

٪۵ Quiz ← مباحث قبلی

Subject:................ Date:................

جلسه دوم: ۹۲/۶/۲۵

سیستم نمایش اعداد:

پرارزش ترین رقم (MSD) — کم ارزش ترین رقم (LSD)

$$N = (a_{n-1}\, a_{n-2} \cdots a_0 \,.\, a_{-1} \cdots a_{-m})_r$$

$r^{n-1} \quad r^{n-2} \quad r^{0} \quad r^{-1} \quad r^{-m}$ ← وزن جایگاه مورد نظر

سیستم نمایش Positional (سنتی)

با دلیل برطرف کردن افزونگی ← $0 \le a_i \le r-1$

a_i digit (صحیح)

دهدهی: $r=10$: $0 \le a_i \le 9$

دودویی: $r=2$: $0 \le a_i \le 1 \Rightarrow a_i \in \{0,1\}$ → Binary Digit (Bit)

شانزده: $r=16$: $0 \le a_i \le 15$

اشکال مبنای ۱۶ → $(413)_{16}$ { سه رقمی: ۴، ۱، ۳ — دو رقمی: ۴ و ۱۳ }

راه حل ← $a_i \in \{0, 1, 2, \ldots, 9, A, B, \ldots, F\}$

$$|N| = N_r = \sum_{i=-m}^{n-1} a_i r^i$$ ← ارزش عددی

$$N = (42.3)_5 \rightarrow |N| = \underbrace{4\times5^1}_{20} + \underbrace{2\times5^0}_{2} + \underbrace{3\times5^{-1}}_{0.6} = 22.6$$

تبدیل مبناها:

① تبدیل از مبنای r به مبنای ۱۰ ⇐ همان ارزش عددی

② تبدیل از مبنای ۱۰ به مبنای r

$N = 43.125 \Rightarrow N = (?)_2$

43 − (32) = 11 − (8) = 3 − (2) = (1) ⇒ 101011.001

$0.125 \times 2 = (0).25$

$0.25 \times 2 = (0).5$

$0.5 \times 2 = (1).0$

توان های ۲: ۱، ۲، ۴، ۸، ۱۶، ۳۲، ۶۴، ۱۲۸، ۲۵۶، ۵۱۲، ۱۰۲۴

(!!؟) را بطای آز (۱+) ندارد $(n = [n_1 \frac{\log r_1}{\log r_2}] + 1)$

Subject:

Date:

③ تبدیل از مبنای r_1 یا مبنای r_2 (۱۰ ← ۲)

$N = (a_{n-1} \ldots a_0)_r$ — $r^n \Rightarrow$ عدد مختلف قابل نمایش

۰ ⇐ کوچکترین عدد قابل نمایش

بزرگترین عدد قابل نمایش $(r-1)(r-1)\ldots(r-1) = r^n - 1$

729_{10} — $(\underbrace{\quad\quad}_{\text{چند بیت؟}})_2$ — $(512) \leq 729 \leq 1024$ — ۱۰ بیت نیاز است

$(\underbrace{\quad\quad}_{n_1 \text{ رقم}})_{r_1} = (\underbrace{\quad\quad}_{n?})_{r_2}$

$r_1^{n_1} - 1 \leq r_2^{n} - 1 \xrightarrow{\log} n_1 \log r_1 \leq n \log r_2 \Rightarrow n \geq n_1 \frac{\log r_1}{\log r_2}$

$\Rightarrow n = [n_1 \frac{\log r_1}{\log r_2}]$ (!!؟)

جمع:

10100	43
$+00111$	$+38$
11011	81

تفریق:

75	110010
-49	-010101
26	011111

✓ این روش جمع و تفریق ۲ تا سخت افزار برای جمع و تفریق نیاز دارد که مناسب نیست پس به سراغ روشی می رویم که مقادیر منفی را هم بتوانیم نمایش دهیم و با یک سخت افزار کار را تمام کنیم. (مختلف)

$A - B = A + (-B)$

Note book SEVAN

Subject:........................ Date:........................

اعداد علامت دار

← تعداد اعدادی که با n رقم می توان نمایش داد.

$2 \times 2^{n-1}$ ←

① Sign_Manitude

رقم علامت

$N = (a_{n-1}\, a_{n-2} \, \dots \, a_0)_2$ $\qquad a_{n-1} = 0$ ← عدد مثبت

در مبنای ۲ → 2^n $\qquad a_{n-1} = 1$ ← عدد منفی

+۱۴۶ ← ۰۱۴۶ : مبنای ۱۰ $\qquad$ ۰۱۰۱۰ : مبنای ۲

-۱۴۶ ← ۹۱۴۶ $\qquad$ ۱۰۱۰۱

بازهم ۲ تا نمایش برای صفر داریم ←

	S.M.	1's comp.	2's comp.
+۷	۰۱۱۱	۰۱۱۱	۰۱۱۱
+۶	۰۱۱۰	۰۱۱۰	۰۱۱۰
+۵	۰۱۰۱	۰۱۰۱	۰۱۰۱
+۴	۰۱۰۰	۰۱۰۰	۰۱۰۰
+۳	۰۰۱۱	۰۰۱۱	۰۰۱۱
+۲	۰۰۱۰	۰۰۱۰	۰۰۱۰
+۱	۰۰۰۱	۰۰۰۱	۰۰۰۱
+۰	۰۰۰۰	۰۰۰۰	۰۰۰۰
-۰	۱۰۰۰	۱۱۱۱	۰۰۰۰
-۱	۱۰۰۱	۱۱۱۰	۱۱۱۱
-۲	۱۰۱۰	۱۱۰۱	۱۱۱۰
-۳	۱۰۱۱	۱۱۰۰	۱۱۰۱
-۴	۱۱۰۰	۱۰۱۱	۱۱۰۰
-۵	۱۱۰۱	۱۰۱۰	۱۰۱۱
-۶	۱۱۱۰	۱۰۰۱	۱۰۱۰
-۷	۱۱۱۱	۱۰۰۰	۱۰۰۱
-۸	—	—	۱۰۰۰

✓ در همه‌ی روش ها (۳ تا ۱) همیشه در سمت چپ برای اعداد مثبت (۰) داریم و برای اعداد منفی ۱.

مشکل روش ①: ۲ تا صفر ⊕ و ⊖ داریم که برای مقایس اعداد مشکل زاست:

محدوده ی نمایش → $-(2^{n-1}-1) \le \quad \le 2^{n-1}-1$

Subject: Date:

روش محاسباتی: ① اعداد هم علامت:
قدر مطلق ها را باهم جمع کرده:
حاصل هم علامت با اپرند (علامت)

+۲	0010
+۳	0011
+۵	0101

-۲	1010
-۳	1011
-۵	(1)1101

② اعداد غیر هم علامت:
قدر مطلق بزرگتر
قدر مطلق کوچکتر −
← علامت بزرگتر

-۶	(1)110
+۳	0011
-۳	(1)011 ← -۳

✓ باز هم مشکل اول با قبلی ایجاد شد / ۲ سخت افزار مختلف برای پیاده سازی فرآیند با الشا نیاز داریم.

② Diminishe Radix complement
(r-1)'s complement

فرض: N یک عدد مثبت $N = (0\ a_{n-2} \dots a_0)_r$

$-N = (r-1)' comp = r^n - 1 - N$

مثلاً مبنای ۱۰ ← -۴۲۶ +۴۲۶ = ۰۴۲۶ (۴ رقمی) ⇐ ۹۹۹۹ − ۰۴۲۶ ؛ −۴۲۶ = ۹۵۷۳

در روش قبل، بیت علامت جدا بود و در محاسبات شرکت نمی کرد.

در این روش، علامت جزئی از عدد است و مثل بیت های عدد در محاسبات شرکت می کند.

$2^n - 1 -$ = n بیتی با n تا رقم یک ⇐ +۱۴ = 01110 (۵ بیتی)
-۱۴ = ?
$2^5 - 1 - N$ ⇐ 11111 − 01110 = 10001

10001 = یعنی در این روش فقط کافی است اعداد را معکوس کنیم تا با منفی آن عدد برسیم.

Subject:............ Date:............

✓ محدوده‌ی نمایش برای این حالت هم مثل قبلی است.

در روش محاسبات در این روش، اعداد بیت با بیت با هم جمع شده و در صورت ایجاد بیت نقلی از طبقه‌ی آخر، آن را مجددا با نتیجه جمع می‌کنیم.

+۲	۰۰۱۰	-۲	۱۱۰۱
+۳ +	۰۰۱۱ +	-۳	۱۱۰۰ +
+۵	۰۱۰۱	-۵	(۱)۱۰۰۱
			۱
			۱۰۱۰

۱۰۱۰ سه همان ۵- جدول

در این روش محاسبات ۲ بار برشه در حالی که نباید می‌شد.

✓ در این روش اعداد منفی ارزش مکانی خود را از دست می‌دهند پس دیگر "Positional" نیستند.

$101010 0 = ?$ → (⊕ اول می‌کنیم) → ۱۰۱۰۱۱ = +۴۳ → ? = -۴۳

✓ جمع ۲ باره‌ی بیت نقلی هیچگاه رخ نمی‌دهد.

فرض: بیت نقلی در طبقه‌ی i ایجاد شده و بقیه‌ی طبقات آن را فقط انتقال داده و کری ایجاد نمی‌کنند.

$a_{n-1}\ a_{n-2}\ \dots\ a_i\ a\ \dots\ a_1\ a_0$ — با توجه به فرض ما این باید ۰ باشد

$b_{n-1}\ b_{n-2}\ \dots\ b_i\ b\ \dots\ b_1\ b_0$ — تا طبقه‌ی i منتقل نکنند (بعد (i))

$0\ \ S_{n-1}\ S_{n-2}\ \dots\ S_i\ \dots\ S_1\ S_0$

این بیت نقلی در جمع ۲ باره فقط تا طبقه‌ی i ام پیش می‌رود و بقیه‌ی طبقات آن را ادامه نمی‌دهند.

"قابل اثبات است."

③ Radix complement

r's comp.

$$\bar{N} = r^n - N$$

+۴۲۰ → ۰۴۲۰

-۴۲۰ = ?

	۱۰۰۰۰
-	۰۴۲۰
	۹۵۸۰

Subject: Date:

۱۰ | ۱۱۰۰ ✓ تا اولین رقم یک، از سمت راست عینا خودش را
۱۰ | ۱۰۰۱ می نویسیم و سپس ها را معکوس می کنیم.

✓ در روش comp ۲'s، ۲ تا صفر نداریم و از آن جا تا 2^n عدد می توانیم نمایش دهیم یک عدد باقی می ماند

که آن ۰۰۰۰ است که چون رقم سمت چپش ۱ است، آن را با (۸) نسبت می دهیم ⇐ مزیت ها

$-2^{n-1} \leq \quad \leq 2^{n-1}-1$

پس محدوده ی نمایش ما افزایش می شود.

روش محاسبات: اعداد بیت ها بیت با هم جمع شده و در صورت ایجاد بیت نقلی از طبقه ی آخر از آن صرف نظر می کنیم.

+۲	۰۰۱۰	-۲	۱۱۱۰
+۳	+ ۰۰۱۱	-۳ +	+ ۱۱۰۱
+۵	۰۱۰۱	-۵	⊗ ۱۰۱۱

✓ اگر عددی مثل ما را داده شود، نحوه ی نمایش آن به یکی از ۳ حالت بالا را مشخص نکرده ما آن را comp ۲'s فرض می کنیم (چون این عدد در هر کدام از روش ها یک عددی می شود یا بیشترین دارد)

✓ در روش comp ۲'s هم اعداد ارزش مکانی خود را در حالت منفی از دست می دهند.

نکته: برای اعداد منفی مثل ۲ بیت ها ارزش مکانی شان:

ما = ۱۰۱۱۰

ما + ما = ۰

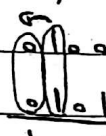
نکته: در روش comp ۲'s، چ عدد مثبت و چ عدد منفی با فرمول زیر قابل نمایش است.

$N = (a_{n-1}\ a_{n-2} \ldots a_0)_2 \Rightarrow N_{10} = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$

Subject: ① بیت نقلی وارد نشد یا طبقه‌ی آخر Date:

ولی از طبقه‌ی آخر خارج نشد

سرریز:



جمع ۲ عدد + ، منفی شد ← منفی

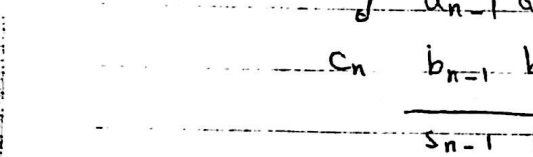
② بیت نقلی نداریم و لی بیت نقلی از طبقه‌ی آخر داریم،

جمع ۲ عدد منفی ، + شد ⇒

از ① و ② : اگر carry ۲ طبقه‌ی آخر یکی نباشد ⇒ overflow رخ می‌دهد.

↓ راه‌های تشخیص overflow

برای ۴ رقم (بیت) ، محدوده‌ی نمایش ۸- تا ۷+ است ولی حاصل ۹- شد پس حاصل غلط است و با آن سرریز یا overflow می‌گوئیم



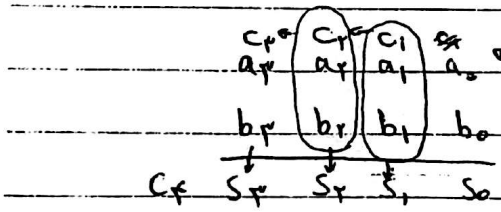
$$OV = \overline{a_{n-1}}\,\overline{b_{n-1}}\,s_{n-1} + a_{n-1}\,b_{n-1}\,\overline{s_{n-1}}$$

$$= c_n \oplus c_{n-1}$$

↳ XOR

چه ده سازی سخت افزاری این ساده تر است.

طرح full adder:

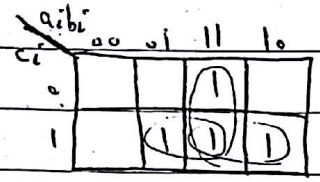


روش تکراری (یعنی ماژول پایه‌ای را می‌سازیم و از آن استفاده می‌کنیم تا حاصل جمع بدست آید.)

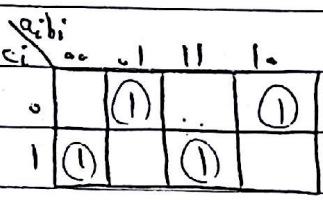
SEVAN Note book جدول صحت در من بعد

Subject:..... Date:.....

a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

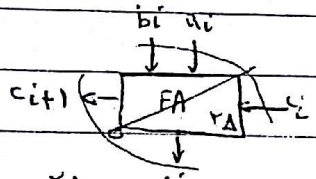


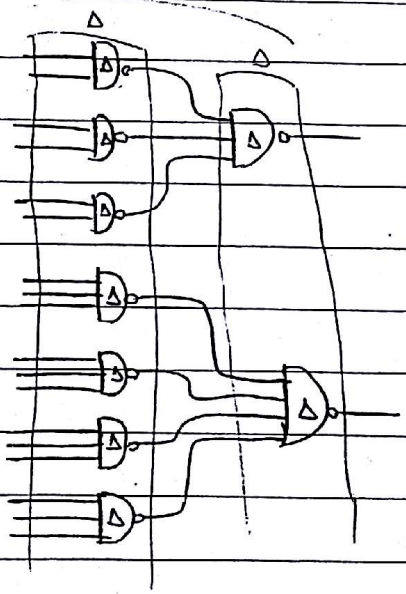
$$c_{i+1} = a_ib_i + a_ic_i + b_ic_i$$



$$s_i = \bar{a}_i\bar{b}_ic_i + \bar{a}_ib_i\bar{c}_i + a_i\bar{b}_i\bar{c}_i + a_ib_ic_i$$

$$= a_i \oplus b_i \oplus c_i$$





جلسه‌ی سوم: ۹۲،۶،۲۵

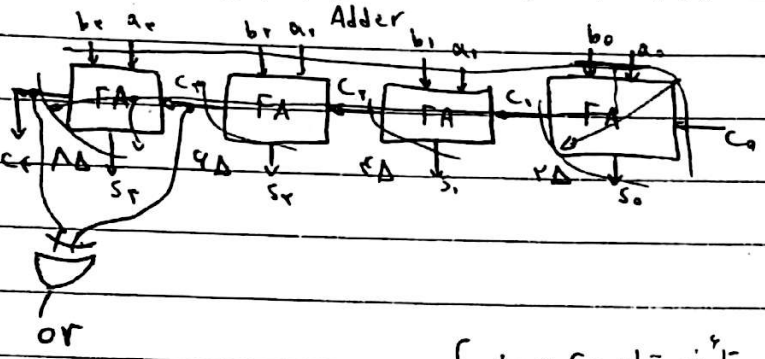
Subject: Date:

$c_4\ c_3\ c_2\ c_1\ c_0$

$a_3\ a_2\ a_1\ a_0$

$b_3\ b_2\ b_1\ b_0$

Carry propagation c_0 $S_3\ S_2\ S_1\ S_0$



این با افزایش بیت ها، بیشتر می شود. (به صورت خطی)

تعداد بیت ها

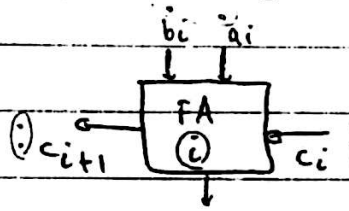
$Delay_{CPA} = n * 2\Delta$

FA { t_c: تأخیر تولید c ; t_s: تأخیر تولید s

میزان تأخیر در حالت کلی $\rightarrow (n-1)t_c + \max(t_c, t_s)$

خال می خواهیم این تأخیر را کم کنیم یا به صورت پیش بینی کنیم c_i ورودی با هر FA چقدر است و سپس همه جمع ها را به صورت پارالل (موازی) انجام دهیم.

a_i b_i	c_{i+1}	
0 0	0	: kill
0 1	c_i	} Propagate
1 0	c_i	
1 1	1	: Generate



توجه: c_i بدون carry، تولید شده توجه

$$c_{i+1} = \underbrace{a_i b_i}_{g_i} + \underbrace{(\bar{a}_i b_i + a_i \bar{b}_i)}_{P_i} c_i$$

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

$$g_i = a_i b_i$$

$$P_i = a_i \oplus b_i$$

$$S_i = a_i \oplus b_i \oplus c_i = P_i \oplus c_i$$

تأخیرها:

$T_{CPA} = n * 2\Delta = 8 * 2\Delta = 16\Delta$ برای ۳۲ ورودی

$T_{CLA} = 8\Delta$ برای ۱۲ ورودی

در این روش با افزایش هر ۴ ورودی، ۲Δ با تأخیر اضافه می‌شود پس تأخیر خیلی کم می‌شود

Subject:..... Date:.....

Carry Look Ahead Adder (CLA)

$i=0$: $c_1 = g_0 + p_0 c_0$

$i=1$: $c_2 = g_1 + p_1 c_1 = g_1 + p_1 g_0 + p_1 p_0 c_0$

این همان مشکل وابستگی با FA قبلی را نیاز دارد

پس برحسب a_0 ، b_0 و c_0 می نویسیم.

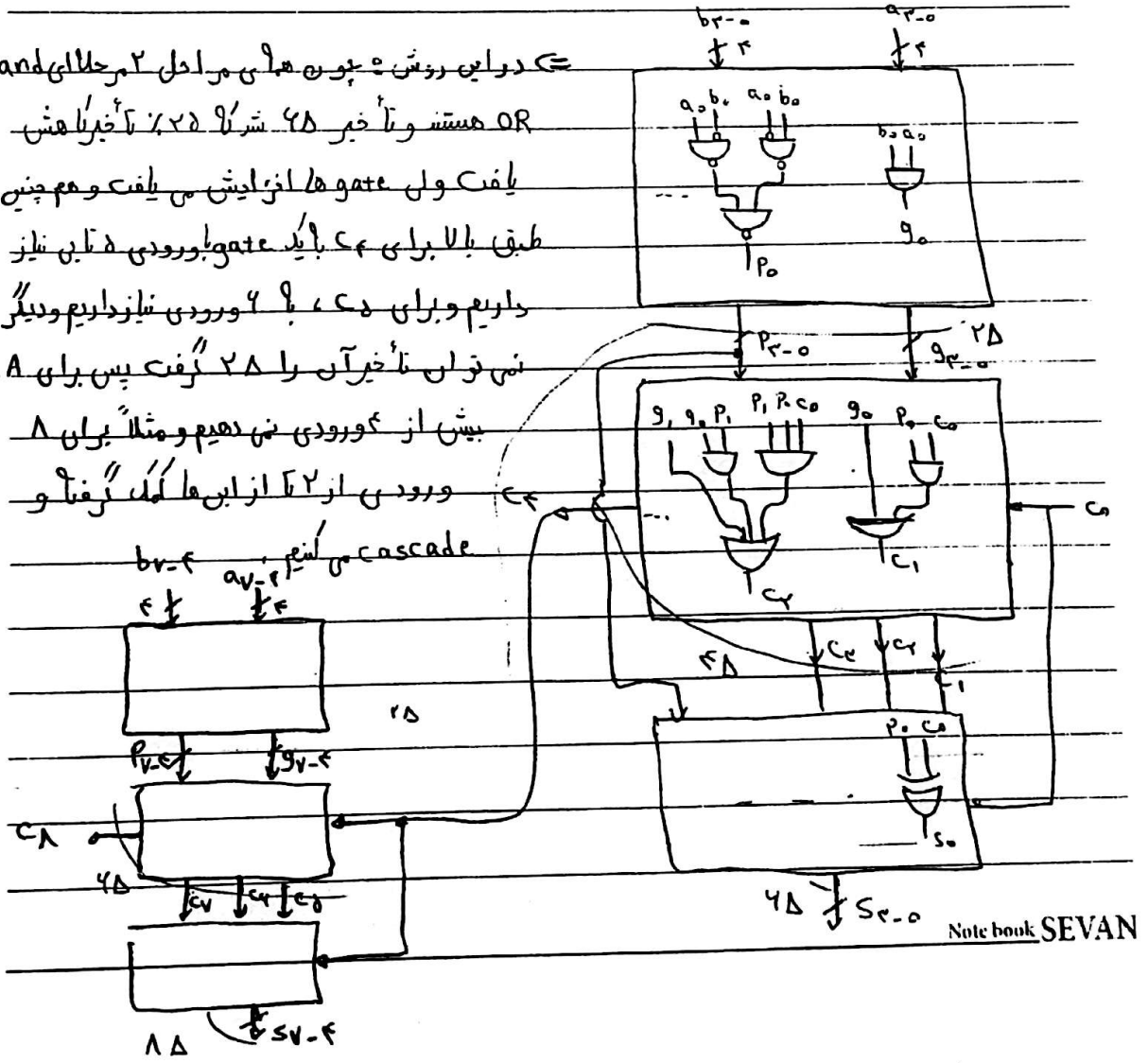
$i=2$: $c_3 = g_2 + p_2 c_2$

$= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$

$i=3$: $c_4 = g_3 + p_3 c_3$

$= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$

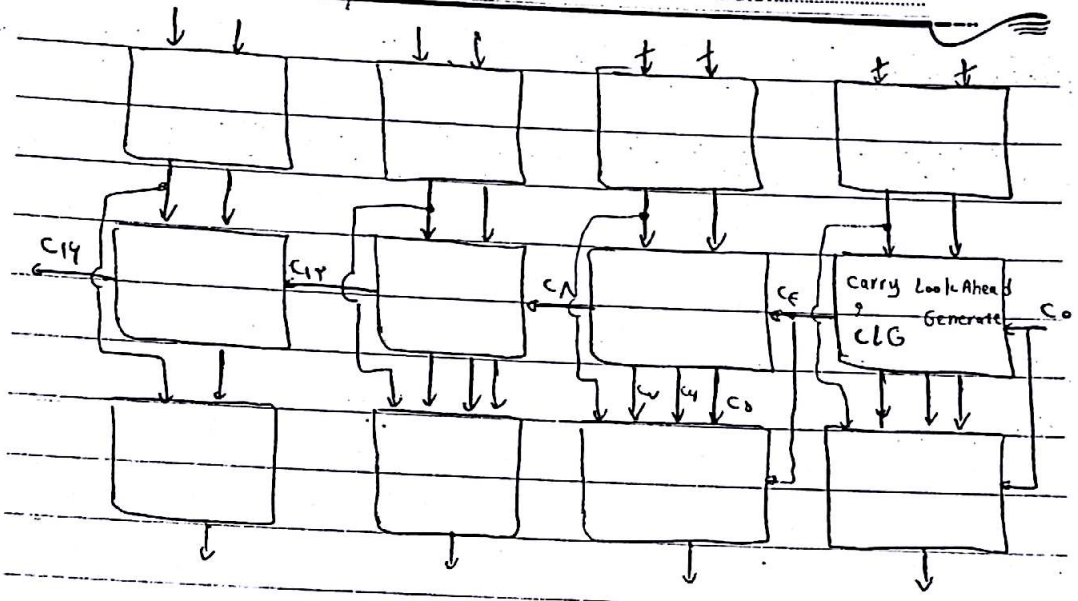
در این روش: چون ۲ مرحله‌ای and و OR هستند و تأخیر کل ۴Δ شد ۲Δ تأخیر داشتیم یافت ولی gate ها افزایش می یافت و هم چنین طبق بالا برای c_4 باید gate با ورودی ۵ تایی نیاز داریم و برای c_5 ، ۶ ورودی نیاز داریم و دیگر نمی توان تأخیر آن را ۲Δ گرفت پس برای CLA بیش از ۴ ورودی نمی دهیم و مثلاً برای ۸ ورودی از ۲ تا از این ها که گرفتیم و cascade می کنیم.



(۱۴)

Subject: (Grouped CLA)

Date:



در شکل با اتصال cascade ها دوبار داگر بتوانیم C_4 و C_8 و C_{12} و C_{16} و ... را تخمین بزنیم با زهم می‌توانیم تأخیر را کم کنیم. (دقیقاً مثل حالت قبل) با یک scale بزرگتر با $C_4 = \underbrace{g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0}_{G_0} + \underbrace{p_3 p_2 p_1 p_0}_{P_0} C_0$ ورودی

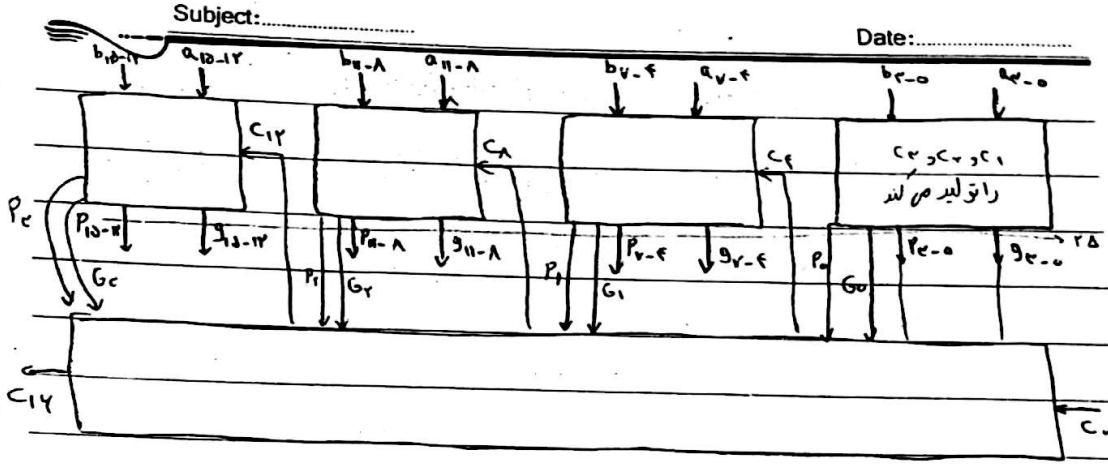
این هم همان Propagate است و یکی از FA [۴] Generate ها بالاخره انجام شده

$$\Rightarrow \{C_8 = G_1 + P_1 C_0 + P_1 P_0 C_0\}$$

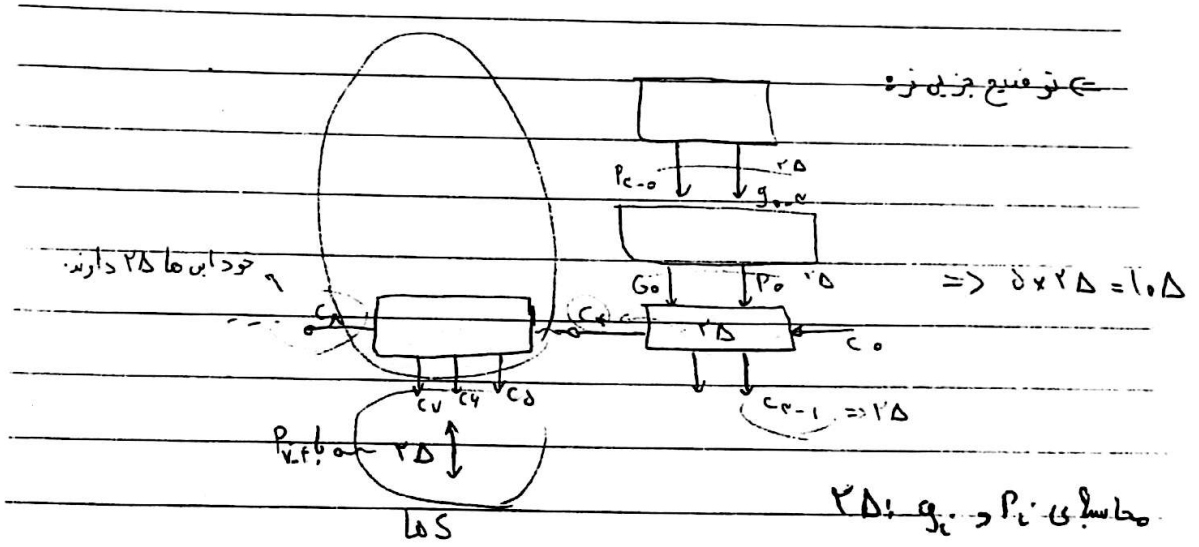
C_{12}

$$\begin{cases} G_0 = g_{c-0} \text{ , } P_{c-0} \\ P_0 = P_{c-0} \end{cases} \qquad C_4 = G_0 + P_0 C_0$$

Subject: Date:



⇐ توضیح جزیره:



$\Rightarrow \Delta \times ۲۵ = ۱۰۵$

محاسبه‌ی P_i و g_i: ۲۵

محاسبه‌ی G_i و P_i: ۲۵

محاسبه‌ی C_4 و C_8 و C_{12} و C_{16}: ۲۵

محاسبه‌ی carry های میانی تمام گروه‌ها (مثل C_1 و C_2 و C_3): ۲۵

محاسبه‌ی لاها: ۲۵

+

تأخیر کلی = ۱۰۵

⇐ بررسی یک نوع (ایده) جدید برای جمع کردن

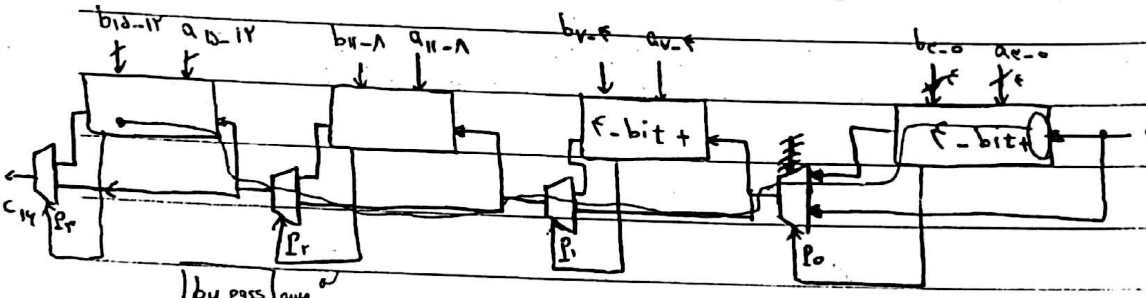
در جلسه بعد

گروه‌های k بیتی

Subject:............................

Date:............................

"Carry skip Adder" را برای ۱۶ بیتی توضیح می‌دهیم.



مسیر by pass
یعنی از 4-bit adder عبور نمی‌کند.

ما همیشه worst case نداریم و مثلاً در بالا اینجور عمل می‌کند که اگر c قرار باشد های Propagate کنند ما با P های بزرگ آن را تشخیص می‌دهیم و با P_i ها block بعدی را انتخاب کرده و c را به آن می‌دهیم و با همین ترتیب باید آن را ادامه دهیم.

worst case بدترین مسیر این است که اولی (4 bit adder) خودش Generate کند و از مسیر by Pass برود و ۲ تای بعدی Propagate کنند (از مسیر by pass) و آخری هم تا آخر بماند که آخری Propagate کند ولی بیرون نریزد.

گروه آخر + n-2 گروه وسطی + گروه اول

تأخیر تولید P_i + $k * T_{FA} + (n-2)\, t_{mux} + (k-1)\, t_{FA}$

(k: k بیت، 4-bit اول — t_{mux}: مالتی پلکسر)

ما های adder ها را k بیتی گرفتیم در حالی که لزومی نداشت و با کم کردن بیت های adder اول و آخری که تقریباً سری عمل می‌کنند را کم کنیم چون تأخیر FA بیشتر از mux است ولی روش هایی وجود دارد که optimization این ها را انجام می‌دهد و تأخیر و تعداد بیت های adder ها را مشخص می‌کند.

پایان binary adder ها

Subject: Date:

جلسه‌ی چهارم ۱، ۷، ۹۲

سیستم نمایش (BCD) Binary coded Decimal

۱۴۲ = ۰۰۰۱ ۰۱۰۰ ۰۰۱۰ BCD

چون در سیستم دهدهی کار می‌کنیم با این اعداد سروکار داریم و با بقیه کاری نداریم.

(همین اعداد می‌تواند در جمع BCD مشکل آفرین باشد)

۰۰۰۰
⋮
۱۰۰۱
۱۰۱۰ ✗
۱۰۱۱ ✗
⋮
۱۱۱۱

۲ + ۳ = ۵

۰۰۰۰ BCD ①
۰۰۱۱ + BCD
۰۱۰۱ = BCD

جمع کننده‌ی Binary (با یک سخت‌افزار جدید)

۷ + ۵ = ۱۲

۰۱۱۱ ②
۰۱۰۱ +
۱۱۰۰ رقم خارج از BCD

۸ + ۹ = ۱۷

۱۰۰۰ ③
۱۰۰۱ +
۱۰۰۰ ① ۰۰۰۱ = رقم مجاز BCD

⇐ ولی حاصل اشتباه است.

k	z_4	z_3	z_2	z_1	Cout	S_3	S_2	S_1	S_0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1
⋮					⋮				
0	1	0	0	1	0	1	0	0	1
0	1	0	1	0	1	0	0	0	0
0	1	0	1	1	1	0	0	0	1
0	1	1	0	0	1	0	0	1	0
0	1	1	0	1	1	0	0	1	1
0	1	1	1	0	1	0	1	0	0
0	1	1	1	1	1	0	1	0	1
1	0	0	0	0	1	0	1	1	0
1	0	0	0	1	1	0	1	1	1
1	0	0	1	0	1	1	0	0	0

حالت ① با ۷ تا

حالت ② با ۷ تا

حالت ③ با ۷ تا

نیاز به تصحیح

ولی ورودی ۳۲ حالت مختلف دارد ما BCD است.

بیشترین ورودی = ۹ + ۹ = ۱۸

$B_c B_r B_1 B_c$ $A_r A_r A_1 A_c$

4-bit binary +

carry $z_4 z_3 z_2 z_1$

C_{in}

Note book SEVAN

Subject: Date:

k	z_8	z_4	z_2	z_1	Cout	S_3	S_2	S_1	S_0
۱۷	۰	۰	۱	۱	۱	۱	۰	۰	۱
	X					X			

تا ۱۹ ← Don't care ها

نکته ۱: در صورت نیاز به تصحیح باید ۶ واحد از حاصل جمع کم شده، بیت carry out را ۱ کنیم. یا ۶ واحد به حاصل اضافه شود.

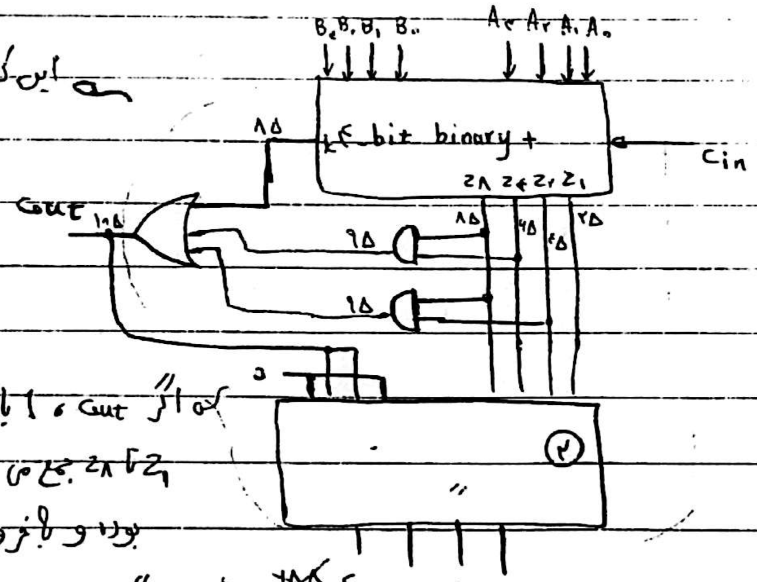
XXXX -
۱۰۱۰ ← مکمل ۲ عدد ۱۰
XXXX +
۰۱۱۰

یعنی هرجا خروجی صحیح است باید ۰ باشد و هرجا خروجی اشتباه است باید ۱ باشد.

$f = k + z_8 z_4 + z_8 z_2$

Cout یا

این کلاً یک بلاک BCD+ است که برای یک رقم با یک رقم (مثلاً ۵+۴) جمع با تاخیر می رود.



که اگر Cout ، ۱ باشد این جا ۶ حاصل می شود و با خروجی z_8 تا z_1 جمع می شود و اگر هم صفر باشد همان z_1 تا z_8 درست بوده و با خروجی می رود.

S_3 S_2 S_1 S_0
که تاخیر در نگاه اول ۱۹۵ ✓

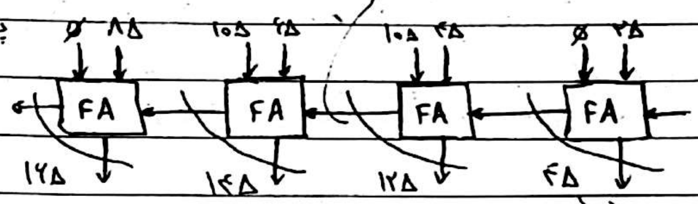
ادامه درس بعد

(۱۸) ۱ حال اگر مقادیر سمت راست جدول زیر را با یک جمع کنیم مکمل ۱۰ ساخته می شود.

در این ۴۵ و ۴۵ و ۱۰۵ برای ورودی ها داریم که ۱۰۵ از همه دیرتر است و خود FA هم ۲۵ تاخیر دارد. پس ۱۲۵ خروجی ها تاخیر داریم. C_{in} با عنوان

Subject: Date:

(۲) بازگردن واحد دوم

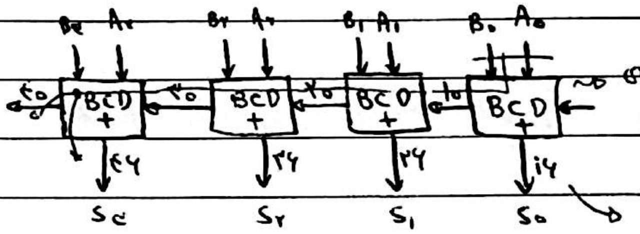


۲۵ خود FA اضافه می کند

که پس با جای ۱۸۵ به ۱۲۸ تاخیر داریم.

این برای جمع ۲ عدد ۴ رقمی است. = C

(۱۵۴۳ + ۷۶۴۸) (مثلاً)



واحدهای جمع کننده تک رقمی BCD

این رابطه را قبلاً داشتیم

$(n-1)t_c + \max(t_c, t_s)$

$3 \times 10 + \max(10, 16) = 46$ ✓

این تأخیرها از قبل به دست آمده (خروجی ۱۴۵ و Cout ، ۱۰۵ تأخیر دارد)

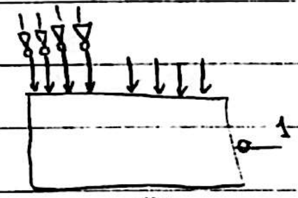
۸ ۴ مکمل ۱۰ عدد ۴ → ۶

مانند باینری که مکمل ۹ را حساب می کنیم و با یک جمع می کنیم.

۹ − ۴ = ۵ → مکمل ۹ عدد ۴

۵ + ۱ = ۶ → مکمل ۱۰ عدد ۴

۸ + ۶ = ~~۱۴~~ ۴



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

مکمل B

قاعده ی خاصی ندارد پس یک نمایش X−3 معرفی می کنیم

ستون اول +۳ بشده است.

BCD	مکمل ۹	X−3 → self complement
0000	1001	0011
0001		0100
0010		0101
0011		0110
0100		0111
0101		1000
0110		1001
0111		1010
1000		1011
1001		1100

✓ برای ساختن BCD با روش X−3 خروجی نیز باید X−3 باشد یعنی مثلاً ۰۰۰۰

چون با هر کدام ۳ واحد اضافه شده $X-3 \xrightarrow{?} 3+3=6 \xrightarrow{6-3} 3$

⇒ یعنی جدول مرحله را باید از ابتدا بنویسیم.

و یک عدد ۱ جمع می کردیم.

و مانند مبنای ۲ مکمل ۹ با یک not ⇐ در این صورت با یک not می توان مکمل ها را ساخت

و تا آخر این را ادامه می دهیم و مانند قبلی کارنومپ و غیره را بدست می آوریم که

در انتها می بینیم با یک gate ، not پیاده سازی می شود.

Subject: Date:

تا ردیف اول آن ... می باشد و همان کارهای قبلی را برای آن انجام دهیم تا در آخر

با یک gate ، not می رسیم

ضرب کننده Multiplier

k y_3 y_2 y_1 y_0 x Multiplicand

m x_3 x_2 x_1 x_0 Multiplier

x_0y_3 x_0y_2 x_0y_1 x_0y_0 ← and ها مانند

x_1y_3 x_1y_2 x_1y_1 x_1y_0

x_2y_3 x_2y_2 x_2y_1 x_2y_0

x_3y_3 x_3y_2 x_3y_1 x_3y_0

+

P_7 P_6 P_5 P_4 P_3 P_2 P_1 P_0 Product

ولی در این صورت ① و ② با هم عمل می کنند

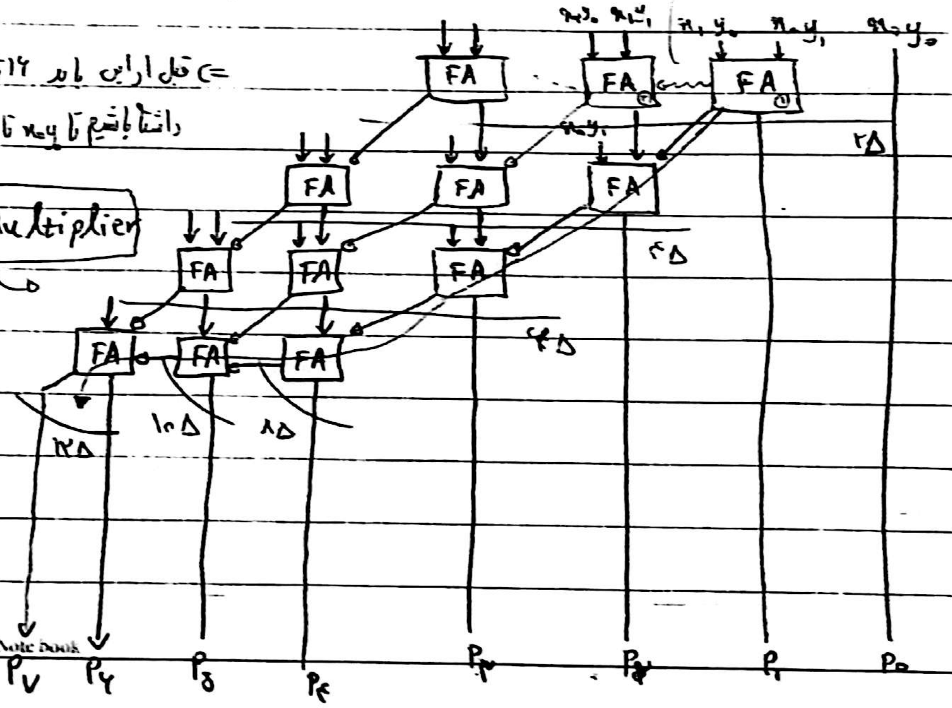
P_7 و carry طبقات آخر

k+m رقم

carry را با FA هم سطح نمی دهیم چون تأخیر را زیاد می کند

=> قبل از این باید ۱۶ تا and گیت داشته باشیم تا x_0y_0 تا x_3y_3 را بسازد

Array Multiplier



SEVAN Note book

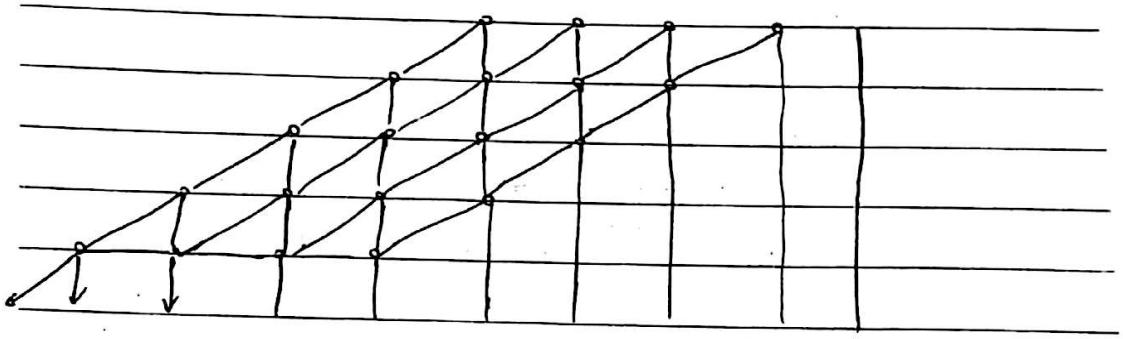
ضرب شیفت با جمع یعنی ضرب ۲

Subject: Date:

تعداد بیت ها

✓ n سطر، هر سطر n-1 عدد FA

✓ cy ها به صورت قطری منتشر می‌شود، بیت در سطر آخر با صورت سطری منتشر می‌شود



✓ این روش سریعترین ضرب کننده است.

✓ گیت AND ← n^2 تا و $n(n-1)$ عدد FA

✓ ← روش دیگر "High Radix Multiplier" ← ضرب در مبنای بالاتر از ۲ انجام می‌شود.

اما با شیفت و جمع این کار را انجام می‌دهند.

Half Adder

✓ در صفحه‌ی قبل، برخی از FA ها در سطر اول و آخر) ۲ ورودی دارند که می‌توان از HA استفاده کرد ولی با علت حفظ شکل از FA کمک می‌گیریم.

✓ ما در این جا از تعداد زیادی FA استفاده کردیم ولی سرعت را بالا بردیم ولی روش دیگر این است که از یک adder استفاده کنیم و مرحله مرحله از آن کمک بگیریم.

$y_3\ y_2\ y_1\ y_0 = y$

$x_3\ x_2\ x_1\ x_0 = x$

جلسه‌ی آینده طراحی add & shift را یاد می‌گیریم.

$x_0 2^0 y$ $\quad P_0 = 0$

$x_1 2^1 y$ $\quad P_1 = P_0 + x_0 2^0 y$ ← adder که کلا داریم از یک

$x_2 2^2 y$ $\quad P_2 = P_1 + x_1 2^1 y$ استفاده می‌کنیم.

$x_3 2^3 y$ $\quad$ ⋮ پس سیستم لذتر می‌شود.

$$P = \sum_{i=0}^{3} x_i 2^i y$$

Subject: Date:

جلسه‌ی پنجم ۹۲/۷/۶:

که روش قبل سرعت بالایی داشت ولی هزینه‌ی زیادی هم داشت ← می‌خواهیم با یک سرعت کنترل هزینه را کنترل کنیم و بیاییم.

$y_3\ y_2\ y_1\ y_0$ *

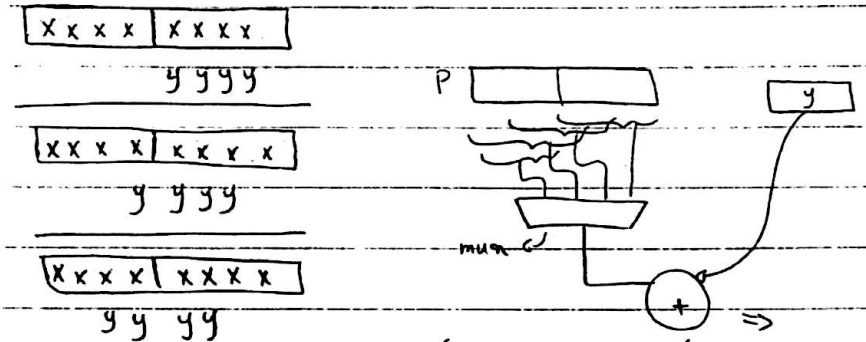
$x_3\ x_2\ x_1\ x_0$

$P = \sum_{i=0}^{3} x_i 2^i y$

$P_0 = 0$ برابر $2n$ بیت

for $i=0$ to 3 شیفت و پاپ با تعداد i

$P_{i+1} = P_i + x_i (2^i y)$



این جمع مشکل دارد چون با یک adder ۴ بیتی و یک mux نیاز داریم و یا با یک adder ۸ بیتی و اضافه کردن صفر نیاز داریم.

روش پیاده‌سازی:

✓ با طی شیفت y به سمت چپ، y را همواره با رتبه‌ی بالای P جمع کرده سپس P را یک بیت به راست شیفت می‌دهیم.

Subject: Date:

P_0 | 0000 | 0000 |

$y_3 y_2 y_1 y_0$

P_1 | 0000 | $y_3 y_2 y_1 y_0$ |

روش ما →

P_0 | 0000 | |

$y_3 y_2 y_1 y_0$

P_1 | $y_3 y_2 y_1 y_0$ | |

$2^{-1} P_1$ | $x y_3 y_2 y_1$ | y_0 |

P_4

روش ما ⇒

$P_0 = 0$

for $i = 0$ to 3 {

$P_i = P_i + x_i y$

$P_{i+1} = P_i 2^{-1}$

}

روش ما ↓

مثال:

```
   1010 *
   1101
   ----
   1010
  0000
 1010
1010
--------
10000010
```

```
      1010 *
      1101
P0    0000   0000
      1010 → x0y
P0    1010      شیفت به راست
P1   (0)10100000
      0000
P1    0101
P2    00101000
      1010
P3    1100
P3    01100100
      1010
P4    0000
P4   (1)00000010
```

از سمت چپ بیت نقلی حاصل جمع را وارد می کنیم.

شیفت به راست

شیفت به راست

در هر ۲

جواب نهایی

تعداد operation = { 3 جمع: تعداد یک های Multiplier ، 4 شیفت: تعداد بیت های Multiplier }

Add & Shift Multiplier

$P_0 = 0$;

for $i = 0$ to 3 {

if ($x_i == 1$)

$P_i = P_i + y$;

$P_{i+1} = P_i 2^{-1}$;

}

Subject: غلیب قلاب ها Date:

مدار دیجیتال متشکل از چه مداری است که در آن تمام FF های موجود در مدار با یک پالس کلاک حساس باشند.

① طراحی مسیر داده Data path Design

② " واحد کنترل controller

همان RTL design Register Transfer Level

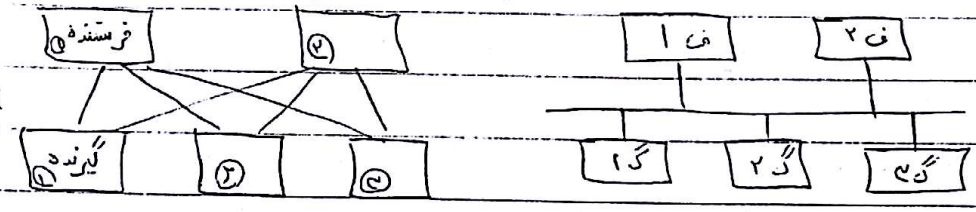
تعریف مسیر داده: مجموعه ای از المان های عملیاتی (+ * -) و واحدهای حافظه (FF و رجیستر و شیفت رجیستر) که توسط "گذرگاه" یا "باس" bus متصل شده اند.

مجموعه ای از سیم ها برای انتقال اطلاعات از نقطه ای به نقطه ای دیگر

Bus:

① باس اختصاصی Dedicated Bus ← در این روش تعداد باس های ما با تعداد ورودی فرستنده و گیرنده برابر است ولی هزینه ی زیادی دارد ولی باهم تداخل ندارند.

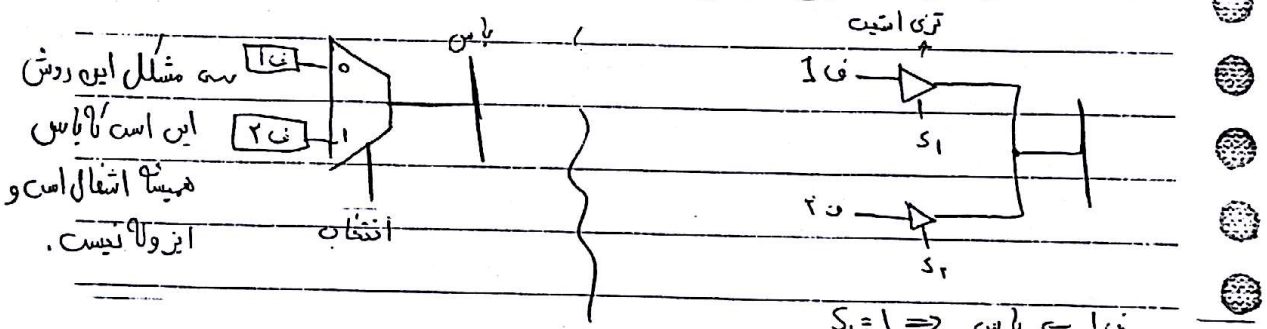
② " مشترک Common ← مزبور فرستنده و گیرنده می باشد ولی هزینه ی



اما این هزینه و سرعت کمتری دارد ولی باید

باس ۹ = ۳×۳

حواسمان باشد در هر لحظه یک فرستنده روی باس اطلاعات بگذارد تا تداخل پیدا نشود.



مثال این روش این است که باس همیشه اشغال است و ایزوله نیست.

ف ۱ ← باس ⇒ $S_1 = 1$

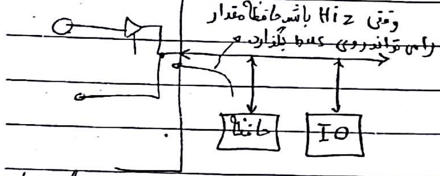
ف ۲ ← باس ⇒ $S_2 = 1$

ایزوله $S_1 = S_2 = 0$ ⇒ باس ← Hi z های امپدانس

SEVAN Note book

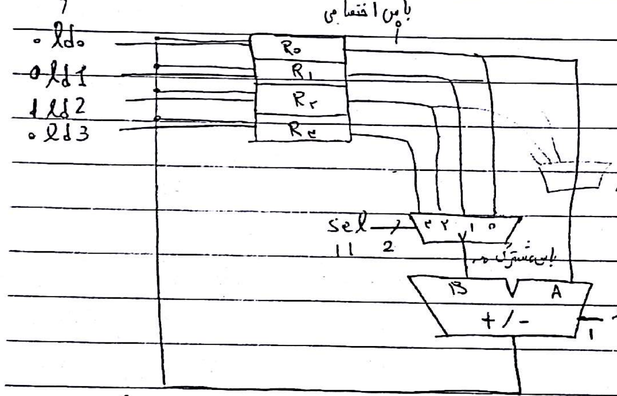
Subject: .... Date: ....

مشکل پیاده‌سازی اول این است که اگر بخواهیم از باس خارجی استفاده کنیم نمی‌توانیم چون همیشه مقدار روی باس قرار دارد. (پس باس خارجی (خصوصاً در رابطه با پردازنده) باید tri state باشد)
پردازنده



ما برای راحتی روش باس مشترک و mux را استفاده می‌کنیم.

سیگنال‌های کنترلی



مسیرداده که clk ها را نشان نداده‌ایم

اگر بخواهیم $R_i \pm R_k$ را داشته باشیم.

$f \begin{cases} + & 0 \\ - & 1 \end{cases}$ flag

کار مسیرداده

$R_j \leftarrow R_0 \pm R_i$

$j = 0, 1, 2, 3$ $\quad i = 0, 1, 2, 3$

به عبارتی مسیرداده: تمام مسیرهای ممکن برای عبور داده در سیستم

مثلاً مسیر: $R_2 \leftarrow R_0 - R_3$ که با مدار روی شکل مشخص شده است.

~~$R_2 \leftarrow R_3 - R_1$~~: این روش در مدار بالا قابل اعمال نیست که این همیشه R_0 است. که این را کنترلر تشخیص می‌دهد.

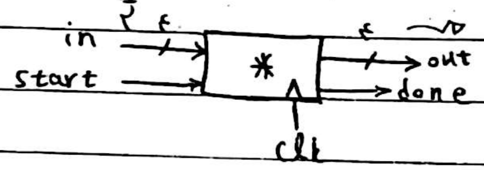
Subject:.................. Date:..................

می خواهیم ضرب کننده ای با ویژگی های زیر طراحی کنیم:

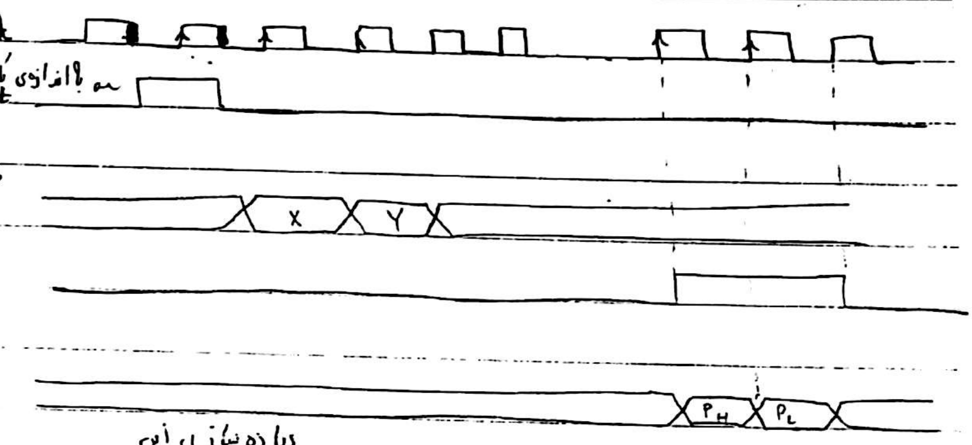
۲ ورودی ۴ بیتی خواهیم داشت

که ۲ ورودی در ۲ clk وارد می شوند.

خروجی هم ۲ در ۲ clk حاضر می شود چون خروجی ۸ بیت است.



timing:



۲n بیتی => $P_0 = 0$ ← پیاده سازی این

for i = 0 to 3 {
  if (x_i == 1)
    $P_i = P_i + y$;
  $P_{i+1} = P_i 2^{-1}$
}

المان های مورد نیاز عبارت اند از: تا بتوانیم X و Y را پای آن بدهیم

① دو رجیستر ۴ بیتی برای ذخیره سازی X و Y با قابلیت لود

② یک رجیستر ۸ بیتی برای ذخیره سازی P با قابلیت clr و لود و شیفت به راست. (شیفت)

که با توجه به کاری که می خواهیم انجام دهیم به راحتی ویژگی و نوع رجیستر ها و Component ها و اتصال آن ها به دست می آید

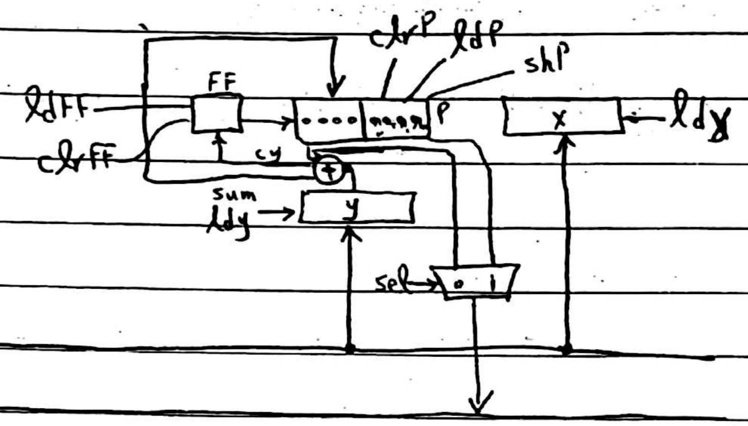
③ یک FF برای ذخیره سازی carry عملیات جمع (که برای انتقال به مرحله بعد (در شیفت)) با لود و clr

④ یک جمع کننده ۴ بیتی که mux ها را هم بخواهیم که در لیست نمی نویسیم

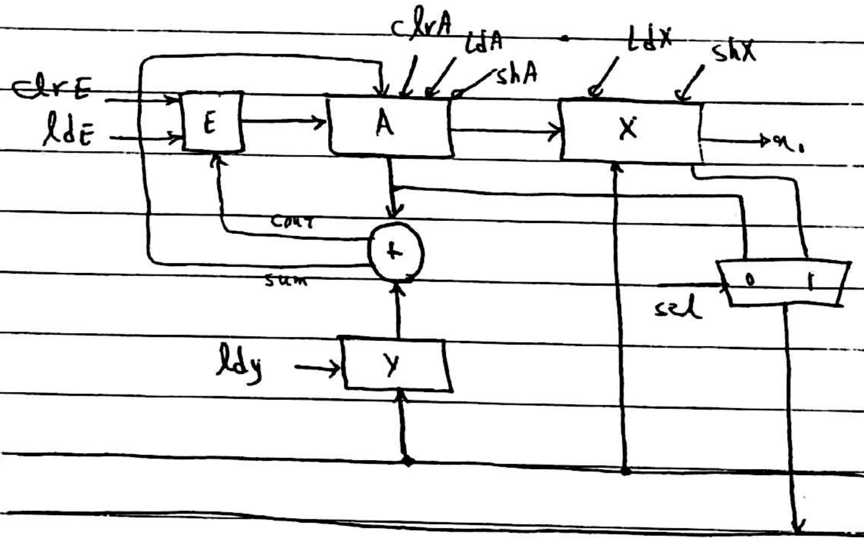
(۲۶) وجود clr در FF به این علت است که در هر مرحله FF را صفر کنیم چون اگر x_i صفر باشد، FF ما 1 را در شیفت به سمت چپ P اضافه می‌کند که اشتباه است. (چون دیگر عمل جمع نداریم)

Subject:........................ Date:........................

دقیقاً RTL های این می‌نویسند. ساده‌سازی Datapath



در Data path بالا می‌توان سمت راست P را با X جایگزین کرد چون مثلاً در مرحله‌ی اول ما فقط به x_0 نیاز داریم و تا آخر با آن کاری نداریم که همین اتفاق هم با شیفت به راست دادن می‌افتد و بیت‌ها از مرحله‌ی اول x_0 بیرون می‌افتد و همین طور برای x_1 و x_2 و x_3 هم تکرار می‌شود و بنابراین از یک رجیستر ۴ بیتی صرفه‌جویی می‌شود.



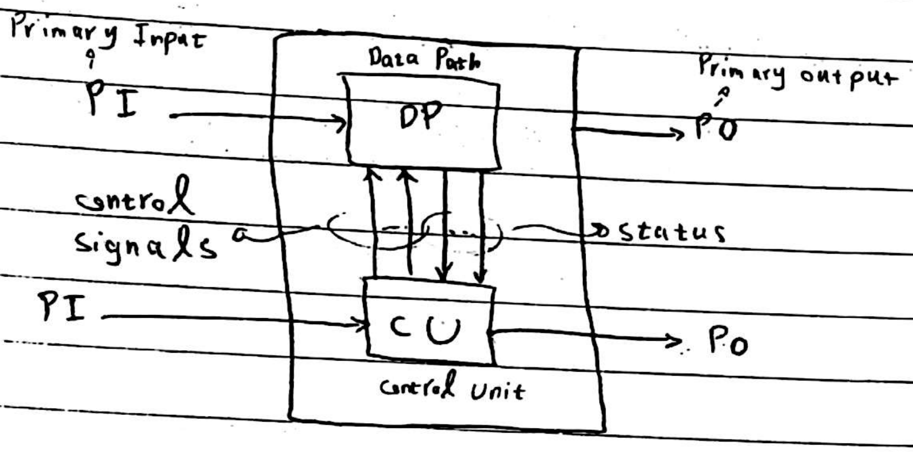
روش بالا بهینه نیست چون مثلاً شیفت و carry را می‌توانستیم در یک کلاک انجام دهیم تا ۳۲ تا،

Note book SEVAN

Enrolment key: CA92 ⇐ اولین HW تا ۲۶ آبان

Subject:

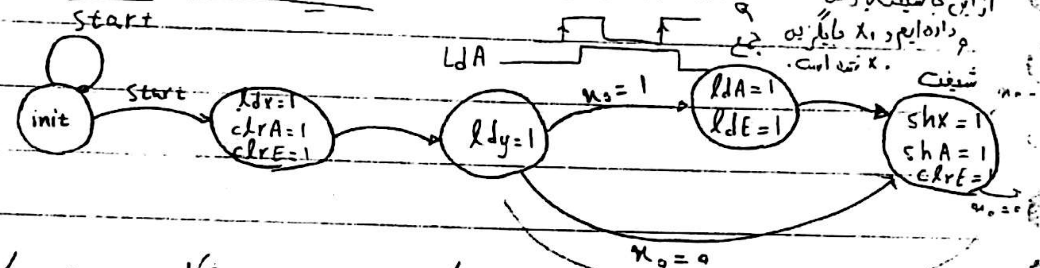
Date:



جلسه ی ششم ۹۲/۷/۸ : (با توجه با مسیر داده ی ص ۲۶ می خواهیم کنترلر را طراحی کنیم.)

Combinational است ⇐ یعنی همیشه جمع در حال انجام است و ما فقط می گوییم جمع را یک جایی ذخیره کن، نتیجه

از این جا شیفت به راست داده ایم و x_1 جایگزین x شده است. جمع

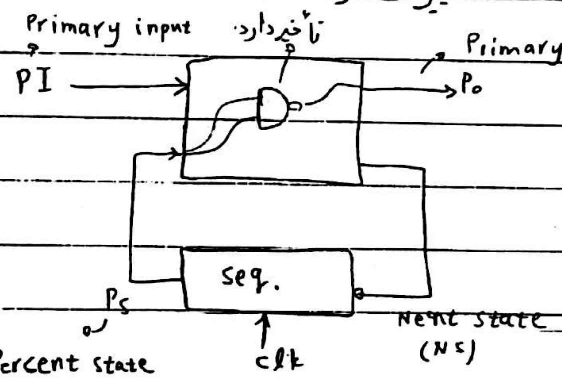


که اگر این قسمت را ۳ بار دیگر با همین صورت تکرار کنیم عمل جمع انجام می شود ـــ مشکل این روش کار نمی کند.

در انتها مدارها ترتیبی (sequential) را از مدل های تی استفاده می کنیم

که مجموعه ای از فلیپ فلاپ ها

⇐ مدارها از نوع Moore است یا تغییرات در state ها است.



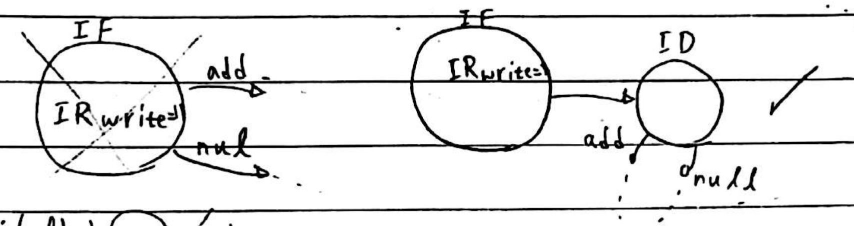
در مدل مور ⇐ $P_o = F(P_s)$

Subject:............................ Date:............................ بسیار مهم

✓ هر نوع عمل ثبت رجیستری داریم (مثل ld یا clr) به علت تأخیرهایی که داریم، ldA در زمان لبه‌ی بالارونده‌ی clk که داریم از state خارج می‌شویم اتفاق می‌افتد که در لبه‌ی بالارونده‌ی ورود به state. ⇐ ما در چک کردن $x_0=1$ یا $x_0=0$ دوم داریم رجیستری را چک می‌کنیم که هنوز وضعیتش معلوم نیست چون shx انجام شده.

✓✓✓ به هیچ عنوان نمی‌توان در عبور از یک state به state بعدی روی رجیستری که در state اول تغییر کرده تصمیم‌گیری کنیم بلکه باید یک clk به آن فرصت دهیم و سپس تصمیم‌گیری کنیم.

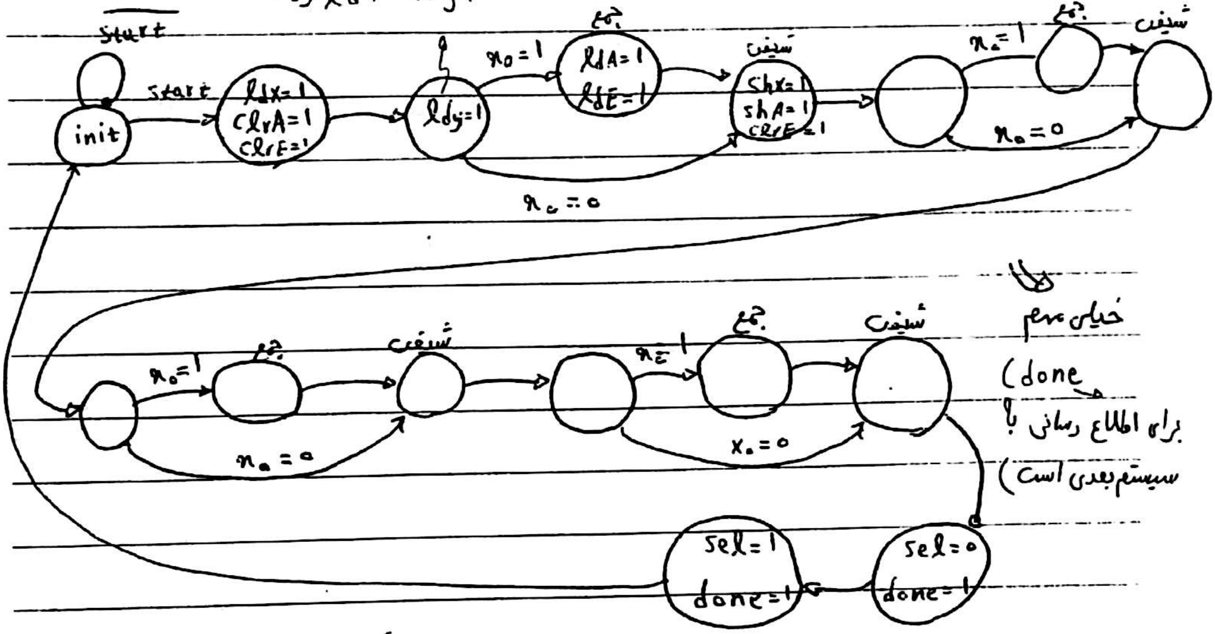
بزرگترین اشتباهی که در طراحی:



کنترل کننده ما که وجود می‌آید در CPU هم همین ⇐ مشکل را داریم.

خالی (clk) ◯ باز باید یک می‌گذاشتیم

اگر این ldx لازم بود؟



خیلی مهم
(done برای اطلاع رسانی با سیستم بعدی است)

که هر دایره به معنی یک clk است.

که گفتیم ماشین بالا

Year: Month: Date:

NOTE BOOK

Seq Part {
always @ posedge clk
ps = ns ;

اگر reset یا clr یا ... داشتیم ← در این جا وارد می شود.

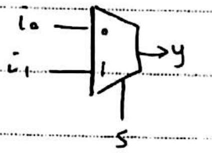
همه‌ی PI ها را باید بنویسیم

Comb. NS {
always @ PI or ps
Switch(Ps)
case 'init: ... ;
case 'S0 : ... ;

تمام ترکیبات ورودی

در این قسمت باید تمام حالات ممکن را رعایت ← کنیم تا ابزار سنتز درست عمل کند

Comb. Po {
always @ ps
begin
ldX = 1'b0 ; ldE = 1'b0;
Switch (Ps)
case 'S0 : {ldx, clrA, clrE} = 3'b111;



if (S == 1'b0)
y = i0 ;
else
y = i1;

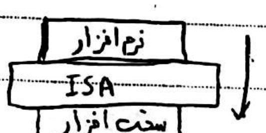
← پس هر if ، باید else داشته باشد.

if (S == 1'b0)
y = i0 ; ← این فرض می کند

اگر else قرار ندهیم مقدار قبلی را می خواهیم نگه داریم و همه‌ی else ها را یک latch قرار می دهد.

← برای این کار می توانیم همه را inactive کنیم و هر وقت خواستیم از هر کدام استفاده کنیم آن را active کنیم.

طراحی پردازنده هم کاملاً به همین صورت است فقط ISA (واسط بین نرم افزار و سخت افزار است) (Instruction set Artichecter) را دارد ← که مثلاً کلاس برنامه و ... را اطلاع می دهد ← بنابراین برای طراحی ابتدا نرم افزار سپس ISA و سپس سخت افزار را طراحی می کنیم.



✓ با یک مثال می توانیم State machine را حل کنیم

Classic ✓ الگوریتم ضرب کننده آرایه ای و Shift & add را مطالعه کردیم

NOTE BOOK Year: Month: Date:

ضرب کننده علامت دار

$$\frac{Y *}{X}$$

حالت ۱: X بدون علامت، Y علامت دار ⇐ همان جمع قبلی است فقط در شیفت به راست ما sign extention انجام دهیم که این معنی که در سمت چپ بیت علامت را وارد می کنیم E.

حالت ۲: X علامت دار

که عمل قبلی قابل قبول نیست.

$X = (x_{n-1}\ x_{n-2}\ \dots\ x_0)_2$

$X_2 = -x_{n-1}2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i$

هم برای منفی و هم برای مثبت صحیح است.

$+12 = 01101$ (-2^4, 2^3, 2^2, 2^1, 2^0)

$-12 = 10011$ ($-16 + 2 + 1 = -12$)

$$\frac{Y *}{x_3\ x_2\ x_1\ x_0}$$

```
      x x x x
    x x x x
  x x x x
- x x x x
```

عمل جمع ما همان قبلی است فقط باید بیت آخر را چک کنیم که اگر صفر بود جمع انجام دهیم اگر ۱ بود باید ⊖ انجام دهیم.

بنابراین در Data path ما ⊕ جمع کننده هم عمل جمع و هم عمل تفریق باید انجام دهد و x_0 هم برای چک نیاز است (در مرحله چهارم)

یک روش دیگر: الگوریتم Booth

$$\frac{X}{011110}$$

۳۲ ۱۶ ۸ ۴ ۲ ۱

$P = ۳۰$ ارزشش

Classic ~~از راست شروع می کنیم که اولین ... $-۲ + ۳۲ = ۳۰$~~ و تفریقش جمع می کنیم

یک که رسیدیم و با ضریب منفی و توان ۲ ضرب می کنیم و جلو می رویم تا به اولین صفر برسیم سپس با علامت + ...

signed digit => یعنی ارقام ها می توانند علامت دار باشد. مثلاً (-۲)(-۲)

الگوریتم Booth برای ۰۱۰۱۰۱۰ مناسب نیست چون تعداد عملیات ها را بالا می برد ولی با آمارگیری که انجام داده اند دیده شده Booth تعداد عملیات کمتری دارد.

مثال: $(+11)\times(-13)$

$+13 = 01101$

$-13 = 10011$

$+11 = 010110$ ← یک صفر اضافه می کنیم و ۲تا۲تا جدا می کنیم.

کد کردن با روش Booth: $1\bar{1}10\bar{1}$

x_i	x_{i-1}	op	
0	0	0	NoP
0	1	1	+
1	0	$\bar{1}$	−
1	1	0	NoP

```
                10011 *
                1 1̄ 1 0 1̄   ← ۲'s comp بالایی
signed extend   00001101
                00000000
                1110011
                001101
                10011
           -----------------
           ⊗ 1011100001
             0100011110  ← برای چک کردن ۲'s comp
```

که ۱۴۳

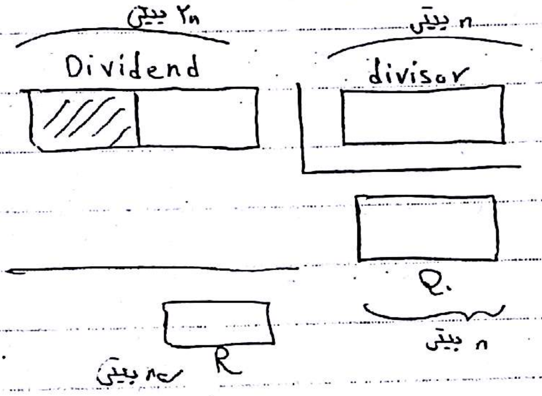
$$\begin{array}{r} 13 \\ \times 11 \\ \hline 143 \end{array}$$

```
P0 = 0 , x(i-1) = 0
for i = 0 to n-1
    if (x(i) x(i-1) = 01)
        P0 = P0 + y
    else if (x(i) x(i-1) = 10)
        P0 = P0 - y ;      جمع با مکمل ۲ عدد y
    shr A x x(i-1)
```

الگوریتم تقسیم که از همه‌ی عملیات‌ها پیچیده‌تر است.

تقسیم اعداد مثبت

که حتماً سمت چپ یک صفر باشد.



n بیتی ۲ بیتی

0100 | 101

سرریز

{سرریز ⇐ divisor ≥ نیمه‌ی رتبه‌ی بالای Dividend}

که اگر نیمه‌ی رتبه بالای dividend از divisor بزرگتر یا مساوی باشد دچار سرریز می‌شویم. ⇐ 0 ≠ divisor

⇐ ما شبیه به با تفریق متوالی تقسیم انجام می‌دهیم.

دو روش عمده برای تقسیم: ① Restoring

② Non-Restoring

$r = q.d + w$ باقیمانده منفی نمی‌تواند

$\frac{10}{\;}$ $\frac{4}{2}$ $10 = 2\times4 + ②$

$d > w$

$w \geq 0$

۴ پارده‌ای

که در روش Restoring ⇐ در مراحل میانی اگر پس از تفریق، باقیمانده منفی شد بلافاصله آن را عوض می‌کنیم یعنی با حاصل بیشتر باقیمانده‌ی پارده‌ای منفی را نمی‌پذیریم.

ولی در روش ② ⇐ اگر به یک باقیمانده‌ی منفی پارده‌ای رسیدیم آن را حذف نمی‌کنیم بلکه باید در مراحل بعد آن را تصحیح کنیم.

Year: Month: Date:

NOTE BOOK

۹۲/۷/۱۳

Restoring: (n Ld)

$w_0 = n$

for $i = 0$ to $n-1$ do

$w_i = 2w_i - d$ ← شیفت به چپ ; divisor

if $(w_i \geq 0)$ ← بیت متناظر q

$w_{i+1} = w_i$; $q_{n-1-i} = 1$;

else

$w_{i+1} = w_i + d$; $q_{n-1-i} = 0$;

مثال: $n=4$

برای شیفت

$w_0 = 0$ ۱۱۰۱

$2w_0 = 1101$

$-d = 11110$

$w_1 = 11011$ ← منفی $q_3 = 0$

$+d = 00010$

$w_1 = 11101$

$2w_1 = 11010$

$-d = 11110$

$w_2 = 11000$ ← $w_2 \geq 0$ $q_2 = 1$

$2w_2 = 10001$

$-d = 11110$

$w_3 = 10011$ ← منفی $q_1 = 0$

$+d = 00010$

$w_3 = 10101$

$2w_3 = 01011$

$-d = 11110$

$w_4 = 01001$ ← باقیمانده $q_0 = 1$

خارج قسمت $0101 = 5$

شیفت به چپ = ضرب کردن دیوزور از نیمای پردازش و اگر حاصل + بود q (بیت متناظر در خارج قسمت) را یک قرار داده و اگر حاصل منفی بود q متناظر را صفر قرار می‌دهیم که اگر حاصل منفی بود آن را با d جمع می‌کنیم ← این را n بار تکرار می‌کنیم.

تعداد بیت‌های تقسیم شونده

Classic

NOTE BOOK Year: Month: Date:

با روش ریاضی برای مبنای ۲ ثابت می‌شود این ۲ روش یکی هستند.

Non-Restoring:

⇐ در این روش تصحیح در همین جا انجام می‌شود:

$w_0 = n$

$w_1 = 2w_0 - d$

for $i = 1$ to $n-1$ do

  if ($w_i \geq 0$)

    $w_{i+1} = 2w_i - d$; $q_{n-1-i} = 1$;

  else

    $w_{i+1} = 2w_i + d$; $q_{n-1-i} = 0$;

if ($w_n < 0$)

  $w_n = w_n + d$; $q_0 = 0$;

else

  $q_0 = 1$

مراحل کمتر ⇐ سریعتر

همان مثال قبل با این روش:

$w_0 = 0\ 0000\ 1011$

$2w_0 = 0\ 0001\ 0110$

$-d = 1\ 1110$

$w_1 = 1\ 1111$ ← منفی ، $q_4 = 0$

$2w_1 = 1\ 1110\ 1100$

$+d = 0\ 0010$

$w_2 = 0\ 0000$ ← صفر ، $q_3 = 1$

$2w_2 = 0\ 0001\ 1000$

$-d = 1\ 1110$

$w_3 = 1\ 1111$ ← منفی ، $q_2 = 0$

$2w_3 = 1\ 1111\ 0000$

$+d = 0\ 0010$

$w_4 = 0\ 0001$ ← مثبت ، $q_1 = 1$

$q_0 = 1$

ما شخص بی‌کسه، در این جا هم برای تقسیم کنندهم می‌توانیم ساختار آرایه‌ای را داشته باشیم.

Classic

Year: Month: Date:

NOTE BOOK

محاسبات ممیز شناور (floating Point):

مثال: یک عدد ۵ بیتی را در نظر گرفته ایم. ۱۰۱۱۰

۱۰۱۱۰۰ +۲۲

۰۱۰۱۱۰ $\frac{1}{2}+\frac{1}{8}+\frac{1}{16}$

۳٫۸۷۵ (بزرگترین عدد در این) ← ۱۰٫۱۱۰ +۲٫۷۵

بسته به اینکه ممیز را کجا در نظر بگیریم ⇐ اعداد ما متفاوت می شوند.

ولی مثلاً ما برای یک application نیاز به عدد ۲۰ داریم ⇐ نحوه ی نمایش

ولی چون ما ممیز را fix در نظر گرفته ایم نمی توانیم آن را نمایش دهیم و محدوده ی نمایش ما کم شده

بنابراین بهتر است بتوانیم این ممیز را جابجا کنیم ← ممیز شناور

$$N = S\ M\ B^{E}$$

که علامت، مانتیس، مبنا (۲)، همواره مثبت

این، ممیز را جابجا می کند و با آن بازی می کند.

مشکل: برای نمایش یک عدد، حالات مختلفی بوجود می آید. مثلاً:

12300×10^{-2}

1230×10^{-1}

123×10^{0}

$12{,}3 \times 10^{1}$

$1{,}23 \times 10^{2}$ ← نماد علمی

$0{,}123 \times 10^{3}$ ← که این را انتخاب می کنیم

همه یک عددند.

$1 \le$ مانتیس < 10 که مبنای ۱۰

* یک عدد بدون علامت *

مبنای ۲ ⇐ نمایش نرمال: $M = 1.xx\ldots x$

$S=0$ ⇐ مثبت

$S=1$ ⇐ منفی

$$N = (-1)^{S}\ M\ 2^{E}$$

توزیع اعداد روی محور اعداد مانند اعداد صحیح یکنواخت نیست.

مثلاً: $1{,}xx * 2^{999}$ ← $-2^{6}+2$

⇐ روی محور فقط قسمت ⊕ را نشان داده ایم در حالیکه برای قسمت منفی هم دقیقاً قرینه ی همین است.

ادامه در صفحه بعد

Classic

NOTE BOOK Year: Month: Date:

$1.00 \times 2^{-2} = 0.0100 = 2^{-2}$

$1.01 \times 2^{-2} = 0.0101 = 2^{-2} + 2^{-4}$

$1.10 \times 2^{-2} = 0.0110 = 2^{-2} + 2^{-3}$

$1.11 \times 2^{-2} = 0.0111 = 2^{-2} + 2^{-3} + 2^{-4}$

} $\Rightarrow 2^{-4}$ تا 2^{-2} فاصله‌ها

$1.00 \times 2^{-1} = 0.100$

$1.01 \times 2^{-1} = 0.101$

$1.10 \times 2^{-1} = 0.110$

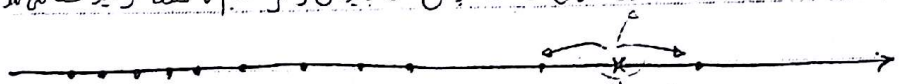
$1.11 \times 2^{-1} = 0.111$

} فاصله‌ها 2^{-3} تا 2^{-2} است

۲ برابر قبلی

نحوه‌ی مختلف گرد کردن: Round to nearest ← even / odd

این عدد ۲ برای ما قابل نمایش نیست پس حتماً باید آن را گرد کنیم تا قطعاً تولید خطا می‌کند.

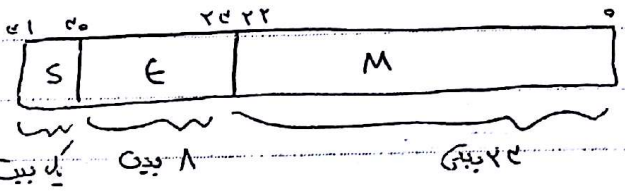


Rounding گرد کردن

پراکندگی اعداد یکنواخت نیست در ممیز شناور ← در اعداد کوچک رزولوشن بیشتر است چون می‌خواهیم آن را تضعیف کنیم ولی در اعداد بزرگ (مثل ۱۰۰۰ و ۱۰۰۱) فواصل زیاد است چون تفاوت زیادی با هم ندارد. (فواصل کمتر)

✓ هر چقدر تعداد بیت‌های مانتیس بیشتر باشد دقت بیشتر است (فواصل کمتر می‌شود در بالا) تعداد بیت‌های توان، محدوده‌ی نمایش را مشخص می‌کند.

* Single precision: استاندارد IEEE برای مشخص کردن بیت‌ها تا پروسورهای مختلف بتوانند با هم کار کنند.



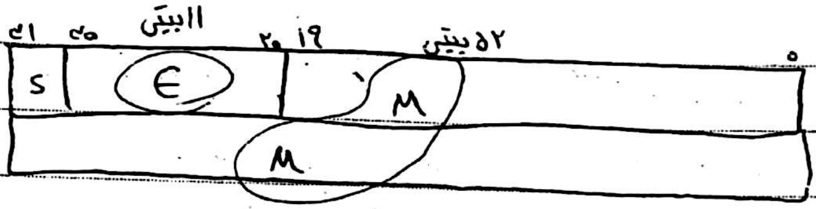
* Double precision: Classic

در کد:

```
float var1;
double var2;
```

Year: Month: Date:

NOTE BOOK



(تعمیم است)

E از این جا به بعد هم مطالب گفته شده درباره ی single precision است (برای Double هم قابل

$1.xx...x * 2^{yy...y}$ = چون 1 همیشه وجود دارد ما در روش single طبق IEEE، 1 را کنار

می‌گذاریم و هنگام محاسبات این 1 را به سمت چپ عدد می‌افزاییم (انگار ۲۴ تا جا برای Mantis داریم)

✓ Implicit on Representation ::

1 نمایش ضمنی

ما از این استفاده می‌کنیم.

Explicit ← اگر یک را نمایش دهیم

نمایش M: بدون علامت

نمایش E: علامت دار — مکمل ۲؟ (تعداد بیت های E)

	2's comp	Biased$(2^{n-1}-1)$
+7	0111	1110
+6	0110	1101
+5	0101	1100
+4	0100	1011
+3	0011	1010
+2	0010	1001
+1	0001	1000
0	0000	0111
-1	1111	0110
-2	1110	0101
-3	1101	0100
-4	1100	0011
-5	1011	0010
-6	1010	0001
-7	1001	0000
-8 (توان خاص)	1000	(+7)

Classic

در این جا (+۷)

$25,4 \times 10^{2}$
$+ \; 14,3 \times 10^{1}$

برای جمع ۲ عدد ممیز شناور باید ابتدا توان ها را مقایسه کنیم (برای اعداد مثبت و یا فقط اعداد منفی مشکلی نداریم چون قدر مطلق هر کدام بزرگتر است آن عدد بزرگتر است) ولی مقایسه ی اعداد مثبت با منفی نتیجه غلط می‌دهد ⇐ ۸ـ را برای تقارن در توان کنار می‌گذاریم (از ۷- تا ۷+)

NOTE BOOK Year: Month: Date:

نمایش E : علامت‌دار Biased ⇐ مزیت: اعداد مثبت و منفی قابل مقایسه‌اند.

مزیت ۲: چون M را 1.xxx...x با این صورت گرفته‌ایم وقتی صفر را در این عدد ضرب می‌کنیم باید عدد صفر شود ولی با علامت وجود ۱ در چپ نمایش عدد صفر ممکن نیست (بنابراین حالتی که همه‌ی Mها را صفر قرار داده و Eها را باید ۸- قرار دهیم تا عدد خیلی کوچک و نزدیک با صفر شود) که

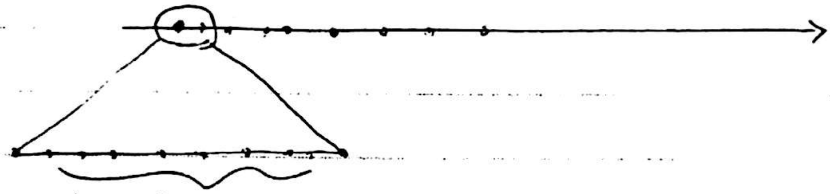
پس نمایش صفر با صورت ← | 0 | 0 - - - 0 | 00 - - - — 0 | صفر درآمد.

(۸ بیت)

حالات خاص: ① ±∞ + k = ±∞ ← همه را با توان ۸- نمایش می‌دهیم ← یعنی بزرگتر از بزرگترین عدد قابل نمایش می‌باشد.

② NaN ۲/۴۴/0 یا Not a Number $(\sqrt{-1,4})$ ← با توان ۸- نمایش می‌دهیم.

③ DeNormal



اعداد غیرنرمال ⇐ را با توان ۸- نمایش می‌دهیم.

✓ با ترکیب مانتیس و بی‌ترال ۸- می‌توان این ۳ حالت خاص را ایجاد کرد.

مثال) ۱۲.۶۲۵- را به صورت Biased نمایش دهید (از چپ)

$-1101.101 * 2^0$; $127 = +130$

$\ominus 1.101101 * 2^{(+3)} \to E$ ← آن قدر صفر در راست اضافه می‌کنیم تا ۲۴ بیتی شود 0 0 0 . . . 0

sign، M

| 1 | 10000010 | 1011010 - - - - 0 |

علامت منفی

C 1 5 A 0 0 0 0 => Hex ۸ رقمی

Classic

که معمولاً این گونه نمایش می‌دهند.

(۳۹)

(کتاب پارسا (سعید صفری)) تمرین‌های هفته‌ی بعد ۲۰ دقیقه‌ی آخر کلاس تا آخر مباحث امروز

Year: Month: Date: NOTE BOOK

✓ عکس پایین متن قبل هم با واقعی امکان پذیر است.

✓ ← مثلا ۲ تا عدد نرمال با توان‌های مختلف داریم ← باید توان‌ها را یکی کنیم ← بهتر است عدد کوچکتر را با عدد بزرگتر برسانیم چون ممکن است ما جا برای نمایش بیت‌ها نداشته باشیم و مجبور شویم یک بیت را دور بریزیم که اگر عدد کوچکتر را با توان بزرگتر برسانیم بیت کم ارزش دور ریخته می‌شود ولی در تبدیل توان بزرگتر با کوچکتر بیت پرارزش دور ریخته می‌شود.

← پس در جمع و تفریق ← بعد از یکسان سازی توان‌ها و انجام عمل جمع ← نرمال سازی انجام می‌دهیم

برای ضرب ← مانتیس‌ها را در هم ضرب و توان‌ها را با هم جمع می‌کنیم ← نرمال سازی در آخر

" تقسیم ← " " " بر " تقسیم و " " از هم کم می‌کنیم ← " " "

$1,432 \times 10^{1}$ ۳ دور ریخته می‌شود

$4,59 \times 10^{2}$ (در تبدیل توان یک با ۲)

✓ با قرار دادن ۳ بیت در سمت راست این نمایش‌های (برای تبدیل توان ۲ با یک) دور ریخته می‌شود

می‌توان انواع محاسبات را انجام داد.

مثلاً وقتی توان‌ها با هم مثلاً ۱۳ تا اختلاف دارند مجبوریم ۱۳ تا بیت از ۲۴ بیت را دور بریزیم (آیا نیاز نیست این ۱۳ تا بیت را در واحد محاسباتی حفظ کنیم یا در واقع ۴۶ بیت در واحد محاسباتی داشته باشیم؟ IEEE می‌گوید با ۳ بیت همه چیز مشخص می‌شود و نیازی به این کار نیست.)

۹۲/۷/۱۵

معماری مجموعه دستورات (ISA) Instruction Set Arch.

| SW |
| ISA |
| HW |

همیشه مراحل طراحی از بالا به پایین است.

اطلاعاتی از سخت افزار بدون جزئیات سخت افزاری (پیاده سازی) در اختیار برنامه نویس قرار می‌دهد تا کد خود را بسازد.

یعنی همیشه یک application را می‌یابیم و برنامه‌های آن را ایجاد می‌کنیم و سپس با ISA سخت افزار را می‌سازیم

Classic