بنام خداوند جان و خرد

معماری کامپیوتر

جلسه سیزدهم: جمع کننده

جمع‌کننده (Adder)

- انواع جمع‌کننده n بیتی:
  - Ripple Carry Adder
  - Carry Look ahead Adder
  - Carry Select Adder
  - Carry Save Adder

Ripple Carry Adder

- سریال کردن n تمام جمع‌کننده (مشابه عملیات جمع مبنای ده به‌صورت دستی)
- تاخیر: 2nd (d تاخیر یک گیت)
- هزینه: 5n
- از لحاظ هزینه مناسب است ولی از نظر تاخیر کند محسوب می‌شود
  - در نتیجه به سراغ سایر انواع جمع‌کننده می‌رویم

Carry Look-ahead Adder

- جمع‌کننده با پیش‌بینی بیت نقلی
  - با هدف کاهش تاخیر ارائه شده است
  - در جمع‌کننده آبشاری، بیت‌های نقلی گلوگاه ایجاد تاخیر بودند:
    - انتقال بیت‌های نقلی از یک FA به FA بعدی زمان‌بر بود
    - لازمه شروع به کار هر FA اتمام کامل کار FA قبلی‌اش بود
  - در جمع‌کننده با پیش‌بینی بیت نقلی (CLA)، هدف تسریع محاسبات بیت نقلی است
    - سریال بودن عملیات RCA را حذف کرده و به‌جای انتشار بیت نقلی، پیش‌بینی انجام می‌شود.

Carry Look-ahead Adder

- در محاسبه بیت‌های نقلی داشتیم:

$C_0=A_0B_0+B_0C_{in}+A_0C_{in} \rightarrow C_0=A_0B_0+C_{in}(A_0+B_0)$

$C_1=A_1B_1+B_1C_0+A_1C_0 \rightarrow C_1=A_1B_1+C_0(A_1+B_1) \rightarrow C_1= A_1B_1+ (A_1+B_1)\ A_0B_0 + (A_1+B_1)\ A_0B_0C_{in}$

- بافرض تعریف:

Generate	$G_i=A_iB_i$
Propagate	$P_i=A_i+B_i$

Carry Look-ahead Adder

- بازنویسی محاسبات بیت‌های نقلی:

Generate	$G_i=A_iB_i$
Propagate	$P_i=A_i+B_i$

$$C_0=G_0+C_{in}\ P_0$$

$$C_1=G_1+G_0\ P_1+P_0\ P_1\ C_{in}$$

$$C_2=G_2+G_1\ P_2+G_0P_1P_2+P_0\ P_1P_2\ C_{in}$$

$$C_i=G_i+C_{i-1}\ P_i$$

$$C_{n-1}=G_{n-1}+G_{n-2}\ P_{n-1}+G_{n-3}P_{n-1}P_{n-2}+\ldots+P_0\ P_1\ldots P_{n-2}P_{n-1}\ C_{in}$$

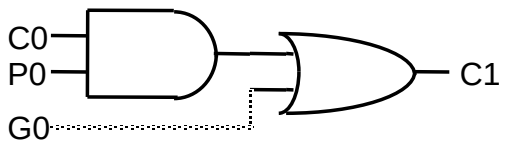
حذف وابستگی بیت‌های نقلی به مرحله قبل

Carry Look-ahead Adder

- امکان محاسبه Pi و Gi های به‌صورت موازی
  - تاخیر محاسبه این دو پارامتر در هر سطح: d (تاخیر یک گیت)
- تاخیر محاسبه بیت نقلی هر مرحله براساس Pi و Gi

$C_i = G_i + C_{i-1} P_i$ ⇒ **دو واحد تاخیر**

- مدار ترکیبی تولید بیت‌های نقلی براساس رابطه بالا: Carry Generator
  - تاخیر این مدار 2d



Carry Look-ahead Adder

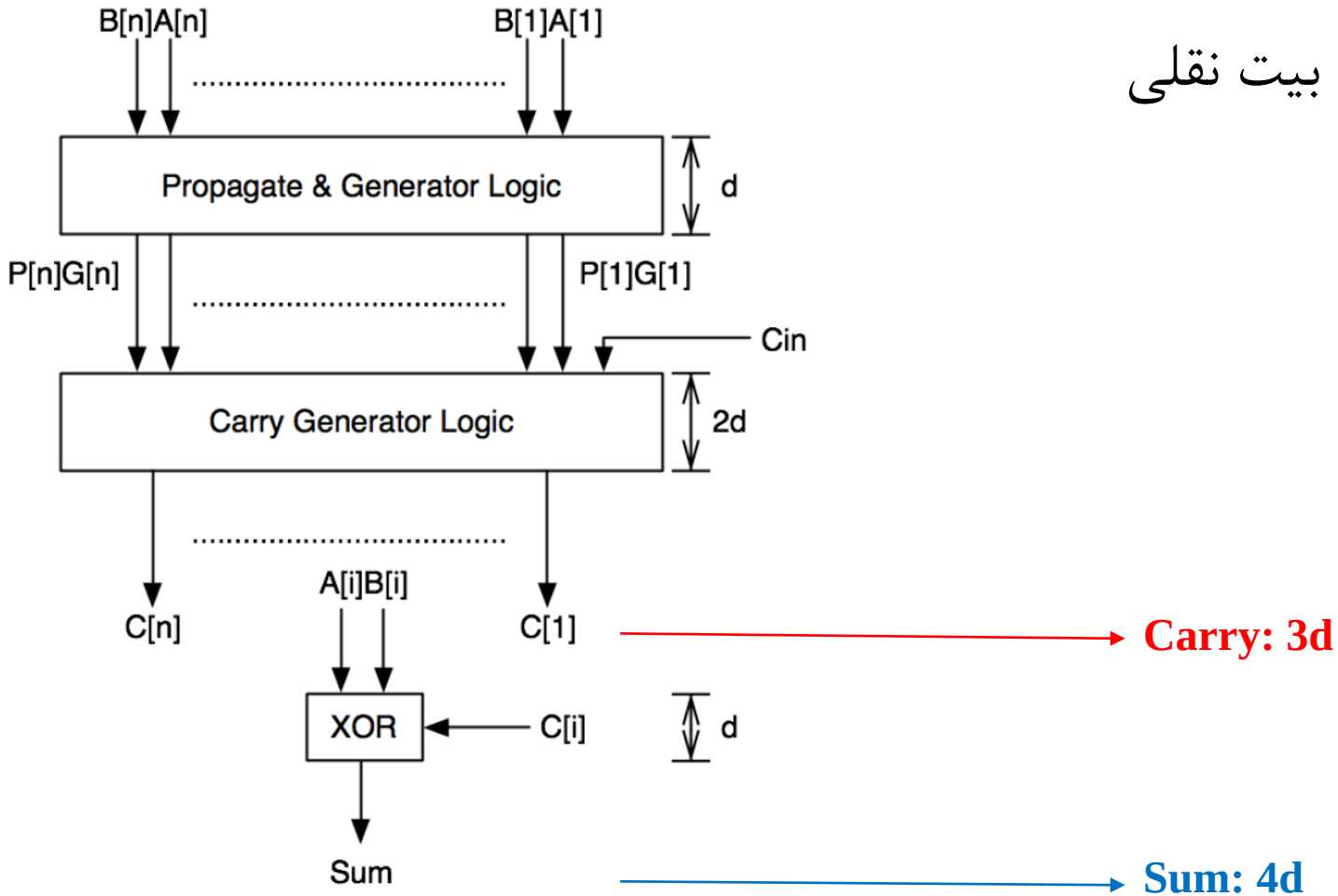
- با دادن بیت‌های نقلی و داده (A,B) به یک تمام جمع‌کننده نتیجه بدست می‌آید
- در این حالت به Cout نیاز نیست پس یک xor برای محاسبه جمع کافی است.
- تاخیر جمع‌کننده:

Delay_carry = 2d + d = 3d

Delay_sum = 3d + d = 4d

→ **Max (sum, carry) = 4d**

Carry Look-ahead Adder



- ساختار جمع‌کننده با پیش‌بینی بیت نقلی

Carry: 3d

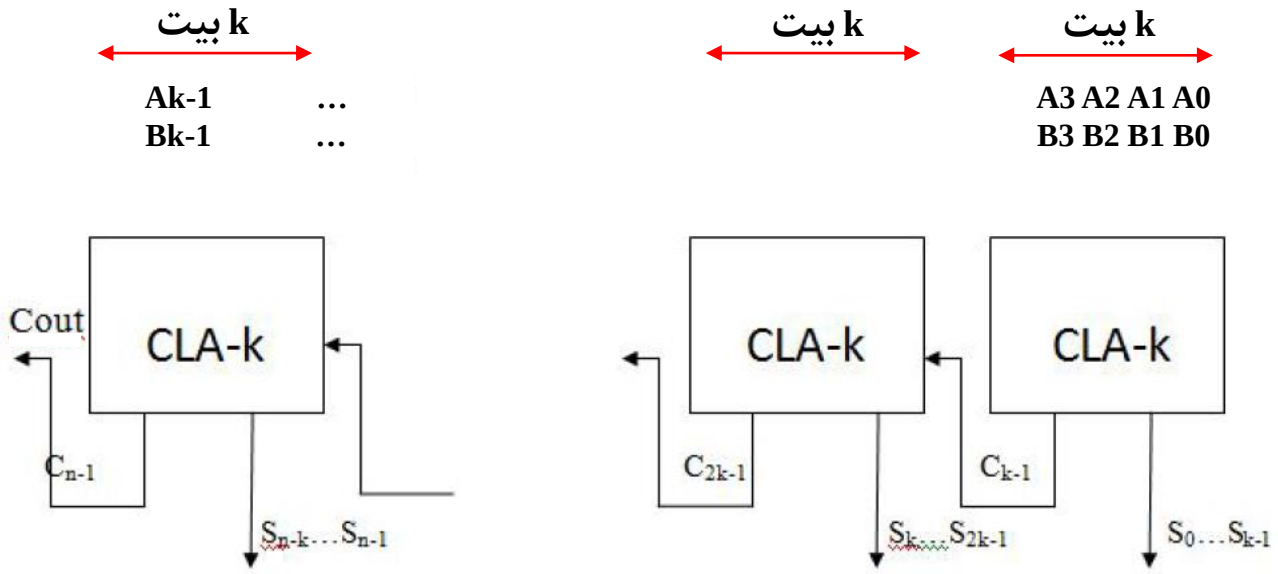
Sum: 4d

Carry Look-ahead Adder

- ارائه CLA تحولی در دنیای جمع‌کننده‌ها بود
  - مستقل کردن تاخیر از تعداد بیت‌های ورودی
  - کاهش مرتبه زمانی جمع‌کننده‌ها از O(n) به O(1)
- در ساختار CLA ارائه شده، گیت‌ها n ورودی درنظر گرفته شدند
  - حداکثر تعداد ورودی گیت‌های ساخته شده برابر ۴
  - پس لازم است طراحی CLAها را از n بیت به ۴ بیت محدود کنیم

Carry Look-ahead Adder

- طراحی CLA براساس ماژول‌های ۴ بیتی
  - اتصال سریال CLA های ۴ بیت



Carry Look-ahead Adder

- تاخیر CLA
  - دیدیم که تاخیر بیت نقلی 3d و بیت جمع 4d می‌باشد پس داریم:

$$Delay\_carry = 3d + 3d + \ldots + 3d = 3\,(n/k)d$$

$$Delay\_sum = 3\,(n/k)d + d$$

  - هرچه k کمتر باشد، تاخیر بیشتر می‌شود
    - k=n بهترین حالت که پیاده‌سازی آن عملی نیست
    - k=1 عملکرد بدتر از جمع‌کننده عادی
    - k حالت معمول برابر ۴

Carry Look-ahead Adder

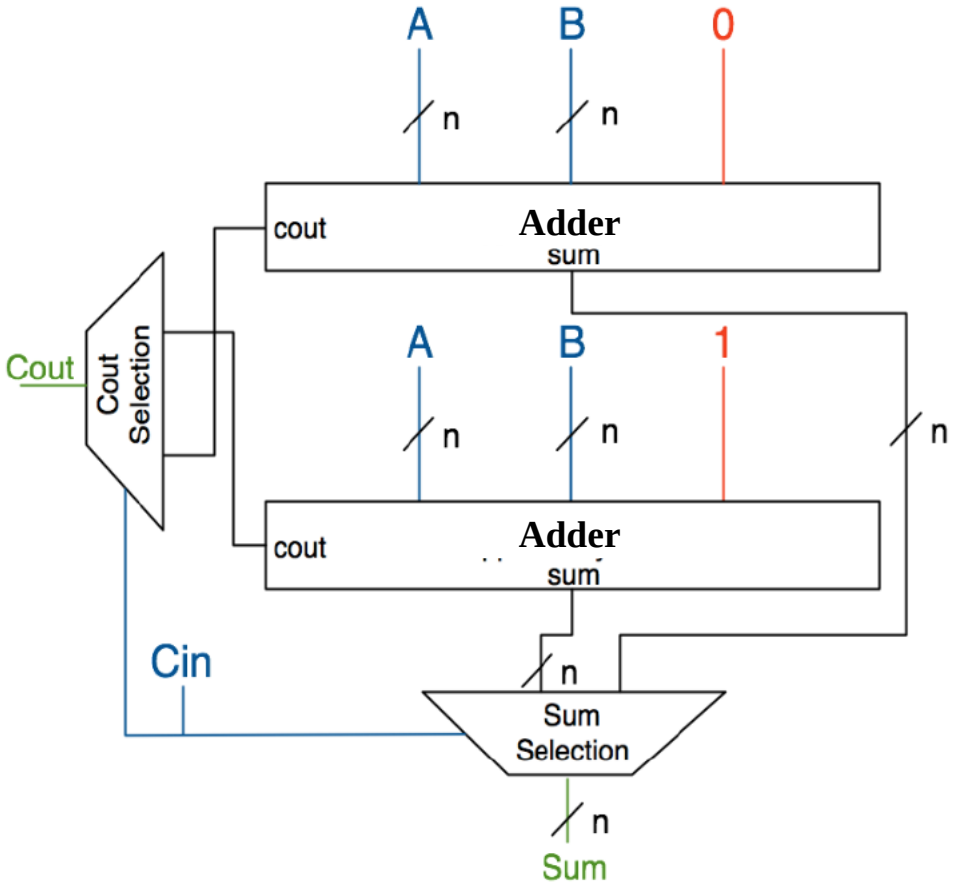
- هزینه CLA
  - محاسبه P و G ها به 2n گیت نیاز دارد
  - محاسبه بیت نقلی به بیش از 5n گیت نیاز دارد
  - هزینه سخت‌افزاری بالا در ازای کارایی و تاخیر بهتر

$$\textbf{Performance} = \frac{1}{Delay * Cost}$$

Carry Select Adder

- جمع‌کننده انتخابی
  - هدف اصلی کاهش تاخیر ناشی از محاسبه و انتقال بیت نقلی است
  - **ایده اصلی:** در محاسبات باینری، رقم نقلی ورودی در مرحله یا «صفر» است و یا «یک»
  - در جمع n بیتی، به ازای هر بیت ورودی یکبار جمع با بیت نقلی «صفر» و یکبار با «یک» انجام شود
  - این دو محاسبه موازی هم قابل اجرا هستند
  - پس از مشخص شدن مقدار بیت نقلی ورودی، نتایج یکی از دو جمع انجام شده انتخاب شده و به خروجی می‌رود
    - پیاده‌سازی انتخاب توسط MUX

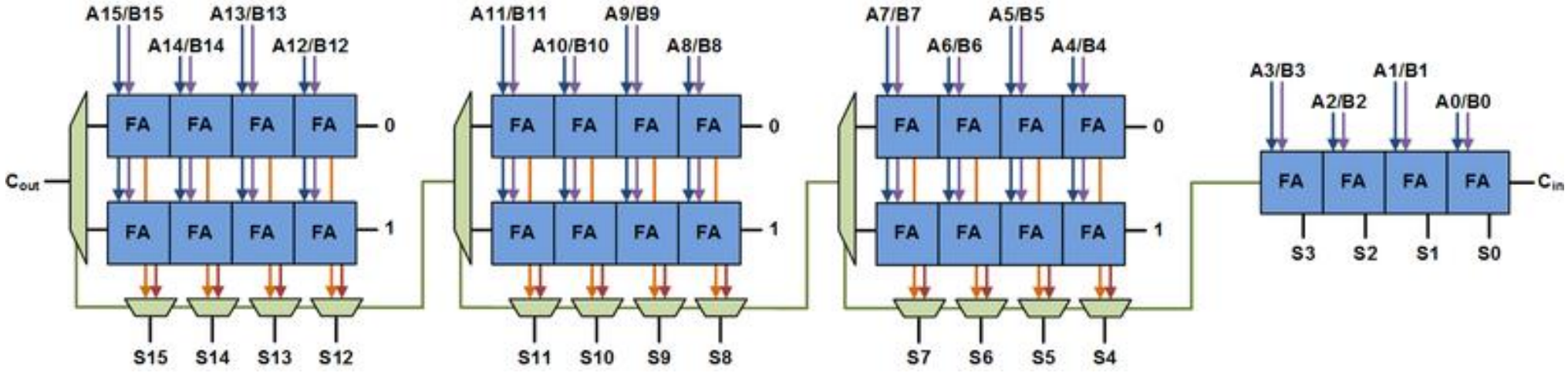
Carry Select Adder



- **ساختار جمع‌کننده انتخابی:**
  - اضافه شدن دو MUX
  - تاخیر بیشتر نسبت به جمع‌کننده ساده
  - هزینه سخت‌افزاری بیشتر
  - باارزش نبودن طراحی فعلی

Carry Select Adder

- استفاده از ایده اصلی و تغییر طراحی
  - تفکیک جمع‌کننده به اجزای کوچکتر و اتصال آن‌ها
    - بطور مثال: اتصال آبشاری بلوک‌های ۴ بیتی CSA



Carry Select Adder

- استفاده از بلوک‌های کوچکتر با اندازه مساوی: Uniform Carry Select Adder
- **تاخیر:**

Delay = k * delay (FA) + ($\frac{n}{k}$-1) * delay (MUX)

Delay_CSA = 4 * 2d + 3 * 3d = 8d + 9d = 17 d

Delay_RCA = 16*2d = 32d

- k تعداد بیت بلوک‌های کوچک مدار است (در شکل قبل: k=4)
- 2d تاخیر FA ها و 3d تاخیر MUXها
- در برخی از طراحی‌ها برای بلوک اول هم MUX اضافه می‌شود

Carry Select Adder

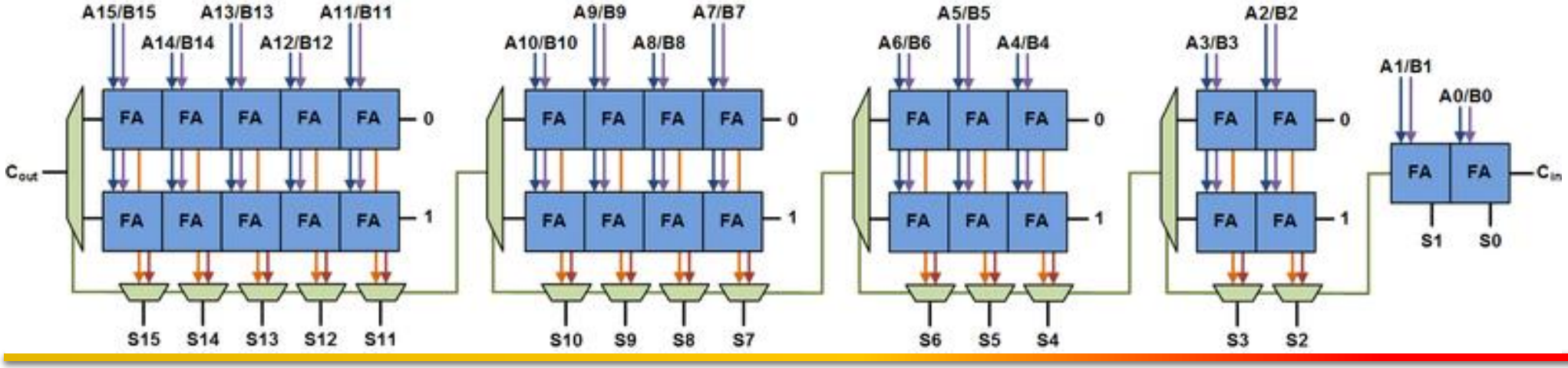
- کارایی جمع‌کننده وابسته است به مقدار k
  - **k=1**: تاخیر از حالت آبشاری بیشتر است
  - انتخاب k مناسب:

Delay(carry select) < Delay (Ripple Carry) →2kd + (n/k)*3d < 2nd

- با حل نامعادله و تعیین k می‌توان به تاخیر بهینه برحسب کاربرد رسید

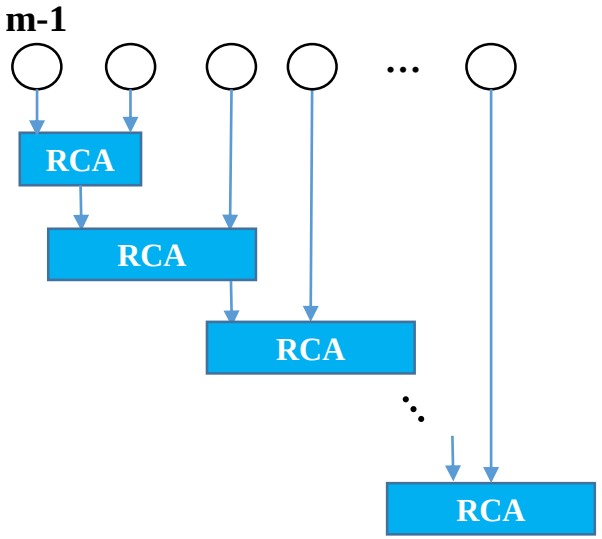
Carry Select Adder

- استفاده از بلوک‌های کوچکتر با اندازه نامساوی: Non-uniform Carry Select Adder
  - کوچک کردن سایز بلوک‌های اول با هدف کاهش زمان انتظار بلوک‌های آخر
  - تاخیر کمتر ولی سخت‌افزار یکسان
  - تاخیر این حالت؟



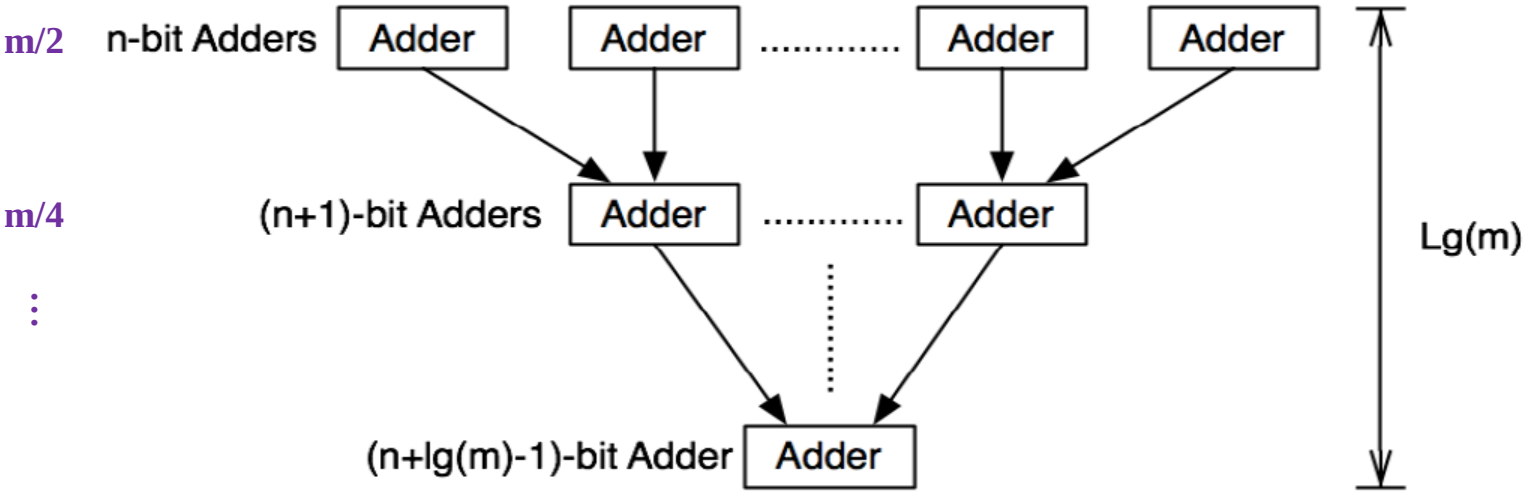
Carry Save Adder

- جمع‌کننده ذخیره بیت نقلی
  - فرض کنید می‌خواهیم m عدد n بیتی را باسرعت بالا جمع کنیم
  - **روش اول:** یک ماتریس m*n تشکیل داده و سطر به سطر با m-1 جمع‌کننده (آبشاری) جمع کنیم
    - تاخیر هر جمع کننده آبشاری n بیتی: 2nd
    - تاخیر این روش بسیار زیاد است : 2n*(m-1)
    - 2n بیت خروجی با احتساب نقلی‌های هر مرحله



Carry Save Adder

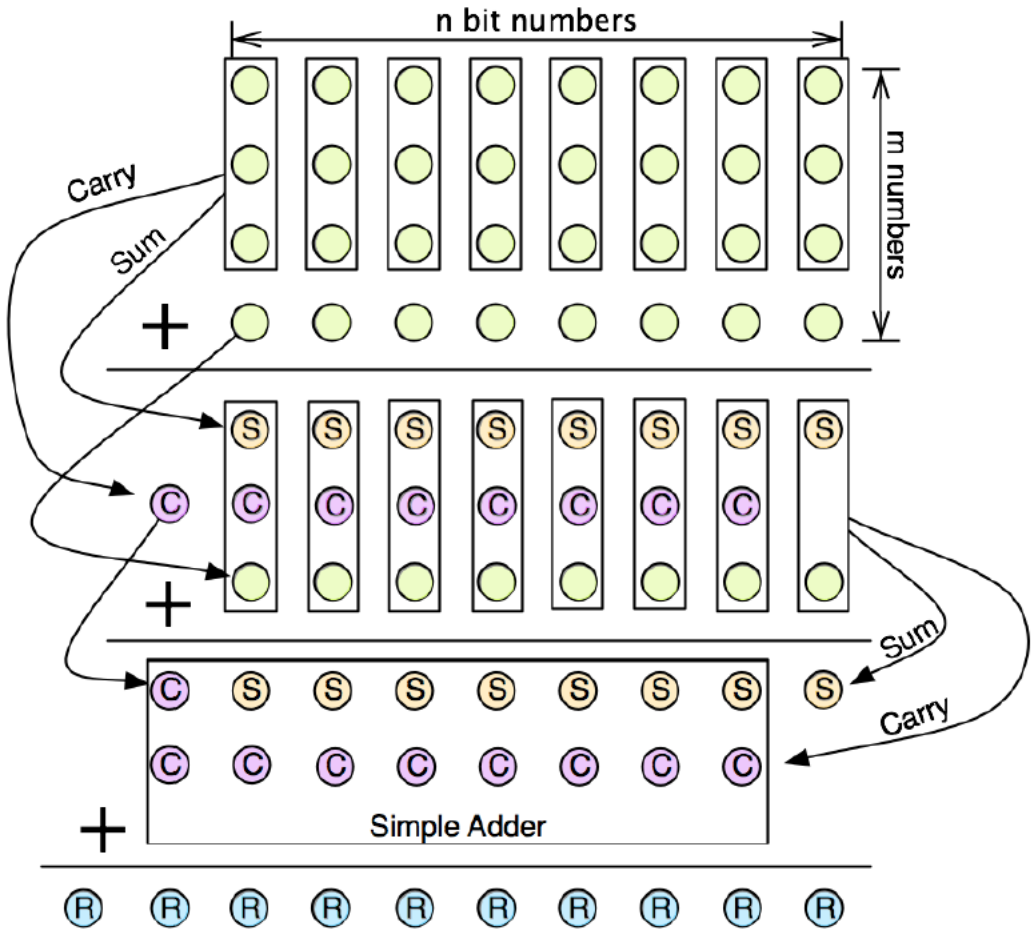
- فرض کنید می‌خواهیم m عدد n بیتی را باسرعت بالا جمع کنیم
- **روش دوم:** استفاده از یک درخت برای جمع با هدف کاهش دادن تاخیر و کاهش طبقات
  - تاخیر = 2nd * ceil(log(m))

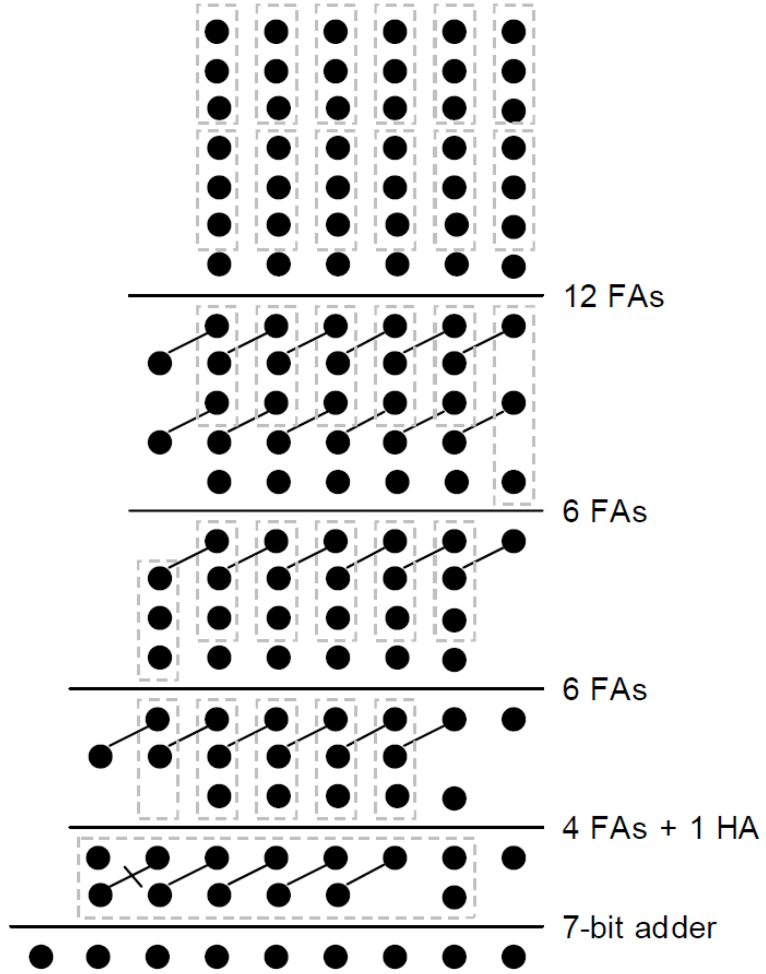


Carry Save Adder

- فرض کنید می‌خواهیم m عدد n بیتی را باسرعت بالا جمع کنیم
  - **روش سوم و بهتر:** جمع‌کننده ذخیره بیت نقلی
    - هر FA یک جمع‌کننده سه تایی: دو ورودی و یک بیت نقلی ورودی
    - با استفاده از FA و به‌طور بازگشتی، دسته‌های سه‌تایی اعداد را به دوتایی تبدیل می‌کنیم (S و Cout)
      - بیت‌های نقلی تولید شده را جدا ذخیره می‌کنیم
      - جمع بیت‌های نقلی با یک سطح شیفت به چپ با حاصل جمع
      - تکرار این روال را تا رسیدن به دسته‌های دوتایی
      - استفاده از جمع‌کننده n+1 بیتی (مانند آبشاری) و پایان عملیات

Carry Save Adder





Carry Save Adder

X:	1 0 0 1 1	
Y:	+ 1 1 0 0 1	
Z:	+ 0 1 0 1 1	
C:	1 1 0 1 1	

X:	1 0 0 1 1	
Y:	+ 1 1 0 0 1	
Z:	+ 0 1 0 1 1	
S:	0 0 0 0 1	

X:	1 0 0 1 1	
Y:	+ 1 1 0 0 1	
Z:	+ 0 1 0 1 1	
S:	0 0 0 0 1	
C:	1 1 0 1 1	
Sum:	1 1 0 1 1 1	

Delay = delay(FA) + delay (5 bit adder)
Cost = 5*cost (FA) + cost (5 bit adder)