بنام خداوند جان و خرد

معماری کامپیوتر

جلسه دوازدهم: قانون امدال - محاسبات کامپیوتری (واحد ALU)

قانون Amdahl

- در طراحی سیستم‌های کامپیوتری اجزای مختلفی باید درنظر گرفته شوند
- کارایی و بازده بالا در طراحی سیستم‌های کامپیوتری بسیار حائز اهمیت است
- تشخیص اینکه سرمایه‌گذاری روی کدام بخش از سیستم منجر به بازده بالاتری می‌شود: قانون امدال
  - بهبود کارایی سیستم درنتیجه بهبود کارایی اجزای آن حاصل می‌شود
  - بهبود اجزایی از سیستم که محاسبات وعملیات سنگین را برعهده دارند منجر به بهبود بیشتر در کارایی می‌شوند
  - بخش ترتیبی سیستم که قابلیت اجرای موازی ندارد: محدودیت سرعت اجرا
  - پارامتر تسریع (speed up): $\frac{Old\ execution\ time}{New\ execution\ time}$

قانون Amdahl

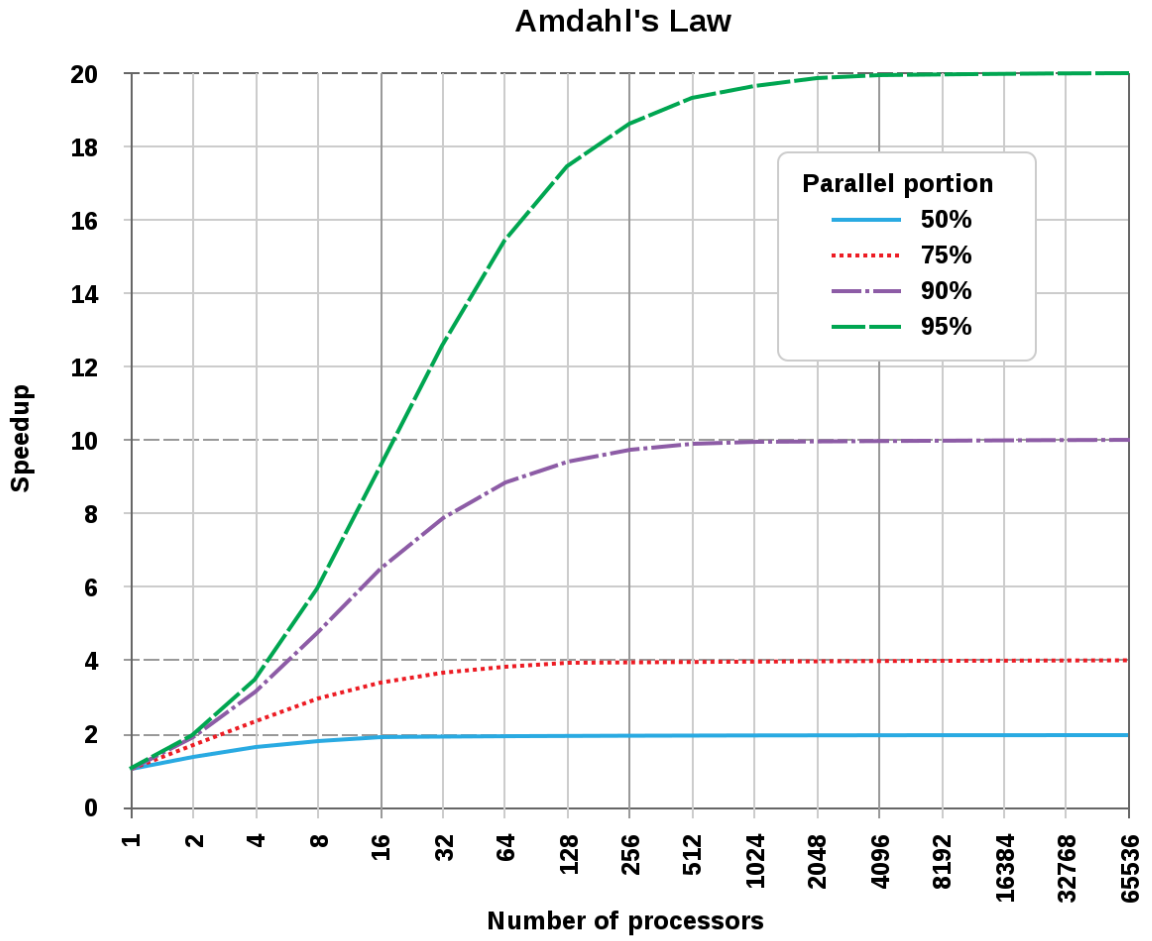
- اگر نسبت دستورات ترتیبی یک برنامه که قابلیت تسریع در آن‌ها نیست به کل دستورات آن f باشد، و بقیه دستورات را بتوان با اجرای موازی p برابر سریع‌تر کرد، میزان افزایش سرعت اجرای برنامه (speed up) حالت دوم به حالت اول برابرست با:

$$\text{Speed-Up} = \frac{1}{f + \frac{1-f}{p}} = \frac{p}{(p-1).f + 1}$$

- درحالت کلی اگر a_i بخش از برنامه را بتوان به‌اندازه p_i موازی کرد:

$$\text{Speed-Up} = \frac{1}{(1-\sum_{i=1}^{n} a_i) + \sum_{i=1}^{n} (\frac{a_i}{p_i})}$$

Amdahl قانون



قانون Amdahl

- مثال: یک برنامه روی کامپیوتر در ۱۰۰ ثانیه اجرا می‌شود، که ۶۰ ثانیه آن مربوط به عملیات ضرب است. اگر بخواهیم اجرای برنامه ۲ برابر سریعتر شود، عملیات ضرب را چقدر سریعتر انجام دهیم؟

Speed up = 2 = 100 / x → x = 50 → mult_old = 60 , mult_new = 50-40=10 → ۶ برابر سریعتر

Amdahl's law: $2 = \frac{p}{(p-1).f+1} = \frac{p}{(p-1).0.4+1}$ → p = 6

قانون Amdahl

- مثال: در مثال قبل اگر بخواهیم تسریع برنامه را ۲.۵ برابر کنیم، به چه میزان موازی‌سازی نیاز خواهیم داشت؟

Speed up = 2.5 = 100 / x → x = 40 → 40 – 40 = 0 impossible ×

Amdahl's law: $2.5 = \frac{p}{(p-1).f+1} = \frac{p}{(p-1).0.4+1}$ → 0 = 1.5 impossible ×

قانون Amdahl

- مثال: یک برنامه در زمان ۸۰ ثانیه بر روی یک رایانه اجرا شده است. ۲۰ درصد زمان برای دستورات ممیز شناور و ۳۰ درصد زمان برای دستورات ضرب اعداد صحیح مصرف شده است. اگر اجرای دستورات ممیز شناور را ۸ برابر و اجرای دستورات ضرب اعداد صحیح را ۶ برابر تسریع کنیم، میزان تسریع برنامه چقدر است؟

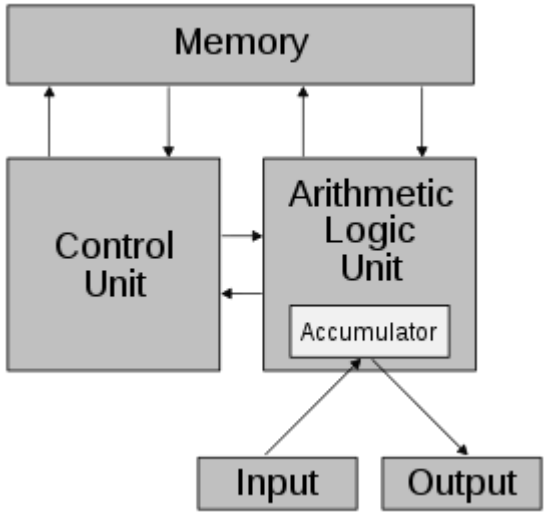
تعمیم قانون امدال:

$$\text{Speed-Up} = \frac{1}{(1-\sum_{i=1}^{n} a_i)+\sum_{i=1}^{n}(\frac{a_i}{p_i})}$$

$$= \frac{1}{0.5+(0.3/6)+(0.2/8)} = \frac{1}{0.5+(0.05)+(0.025)} = 1.74$$

طراحی پردازنده: واحد حساب و منطق

طراحی واحد پردازشگر مرکزی



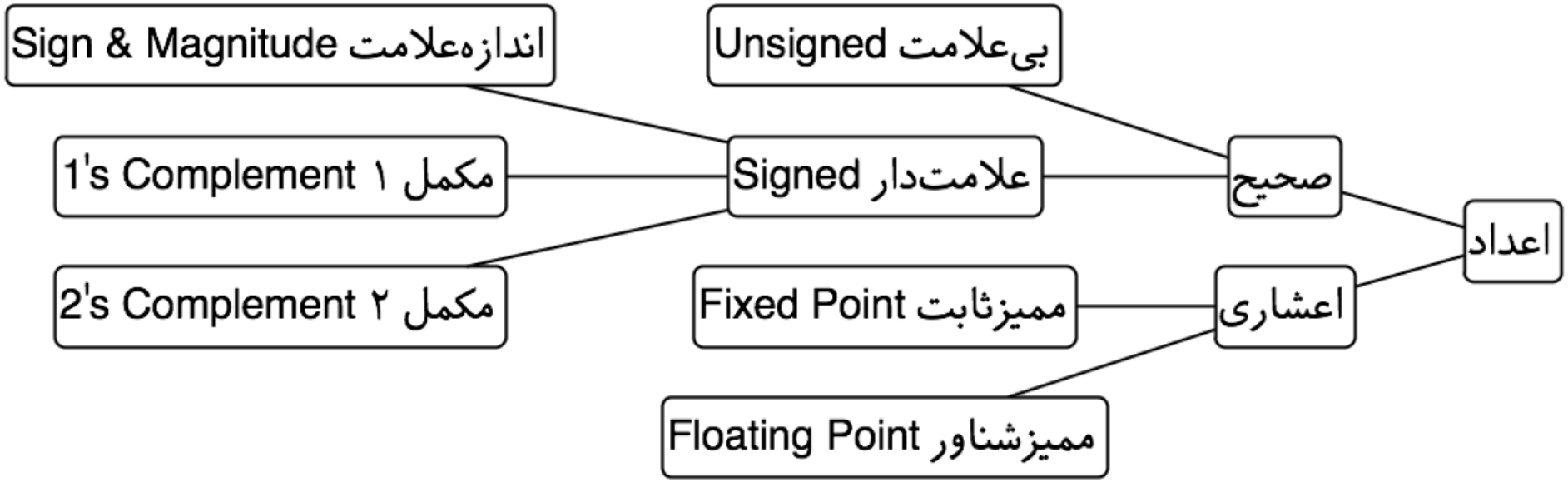
- طبق مدل طراحی معماری کامپیوتر Von Neumann پردازشگر
  - واحد محاسبات و منطق
  - واحد کنترل
- هدف این بخش:
  - آشنایی و بررسی این واحد
  - طراحی این واحد

واحد حساب و منطق (ALU)

- مطالب این فصل:
  - آشنایی با جمع‌کننده‌ها
  - آشنایی با ضرب‌کننده‌ها و تسریع عملیات ضرب
  - آشنایی با الگوریتم و واحد تقسیم‌کننده
  - محاسبات در سیستم اعداد اعشاری ممیز ثابت و ممیز شناور
  - محاسبات اصلی در سیستم نمایش اعداد BCD

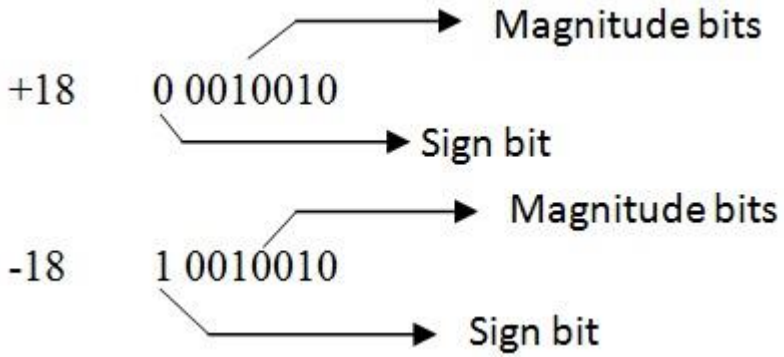
جمع‌کننده (Adder)

- عملیات جمع مهم‌ترین عملیات ریاضی است
- برای طراحی جمع‌کننده لازم است نوع اعداد را بشناسیم:



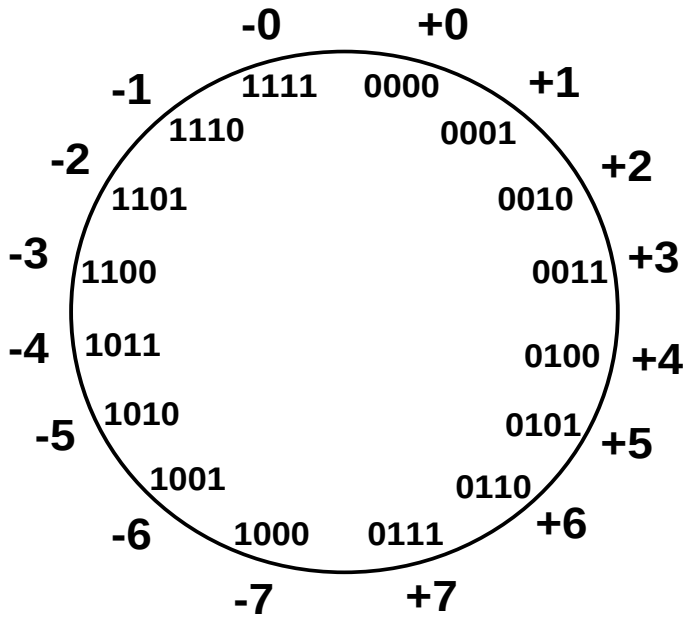
اعداد صحیح علامت‌دار

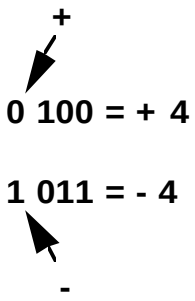
- روش اندازه-علامت (sign-magnitude):
  - اندازه و علامت در قالب دوبخش مجزا ذخیره می‌شوند
  - آخرین بیت (از سمت راست) نشان‌گر علامت است



اعداد صحیح علامت‌دار

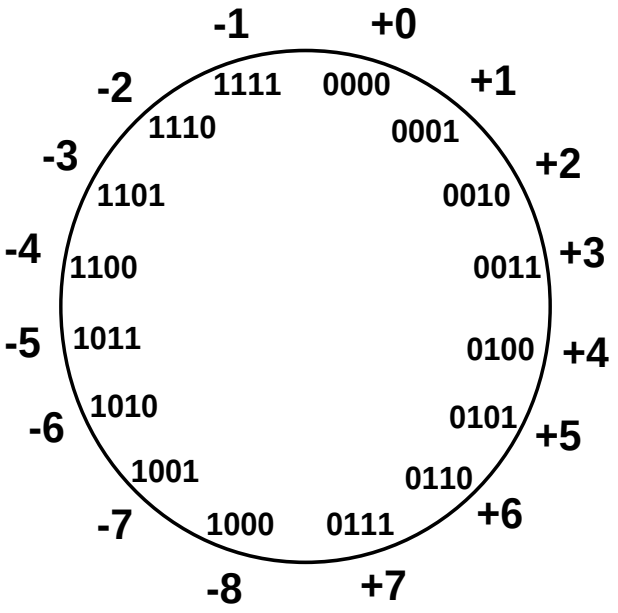
- روش مکمل ۱ (1’s complement):
  - نمایش عدد و علامت باهم در n بیت
  - عدد منفی: مکمل کردن تمام بیت‌ها در نمایش باینری

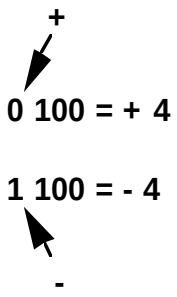




اعداد صحیح علامت‌دار

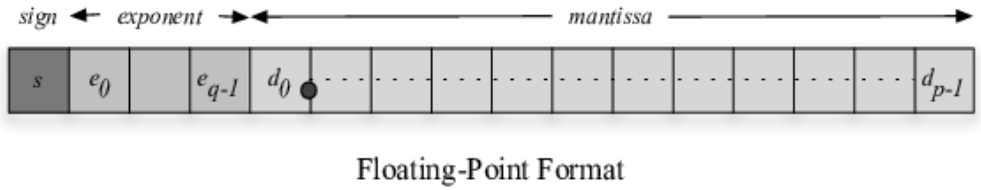
- روش مکمل ۲ (2’s complement):
  - نمایش عدد و علامت باهم در n بیت
  - عدد منفی: مکمل کردن تمام بیت‌ها در نمایش باینری + ۱



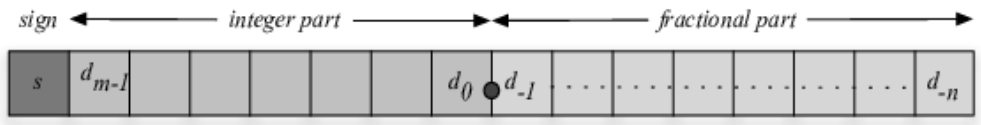


اعداد اعشاری

- اعداد اعشاری ممیز ثابت
  - تعداد بیت‌های ذخیره‌سازی بخش صحیح و اعشاری ثابت است
- اعداد اعشاری ممیز شناور
  - ممیز اعشار از هرجایی برحسب عدد مدنظر می‌تواند قرار گیرد



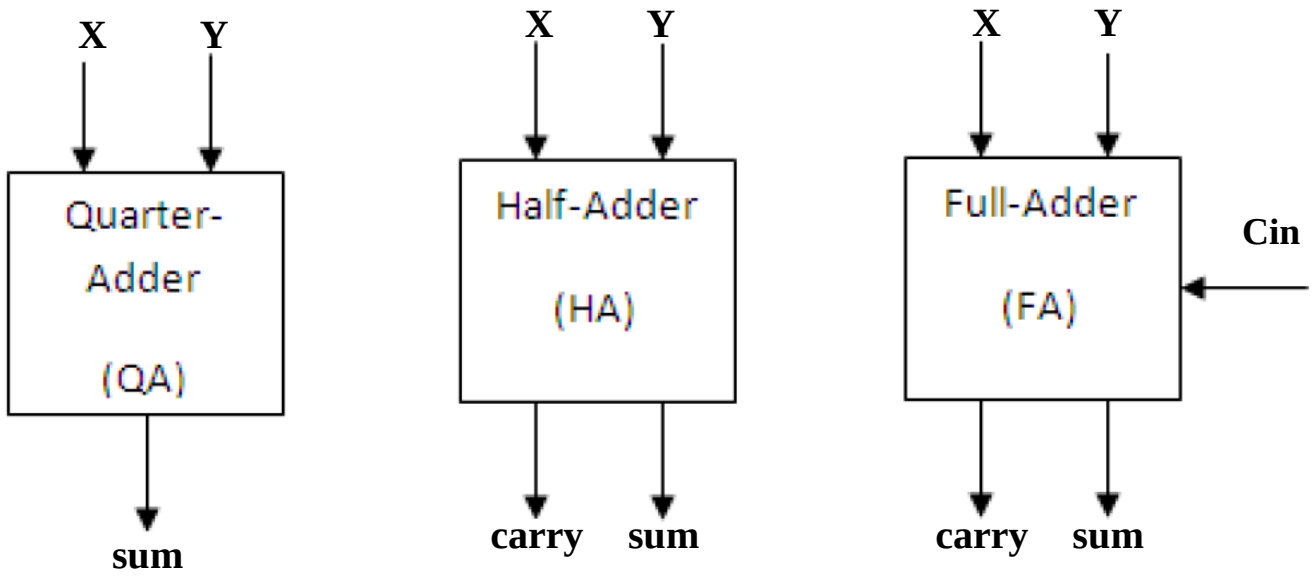
Floating-Point Format



Fixed-Point Format

جمع‌کننده (Adder)

- فرض: اعداد صحیح بدون علامت
- جمع‌کننده‌های پایه:
  - ربع جمع‌کننده (Quarter Adder)
  - نیم‌جمع کننده (Half Adder)
  - تمام جمع‌کننده (Full Adder)



جمع کننده (Adder)

FA

X	Y	C_{in}	C_{out}	S
۰	۰	۰	۰	۰
۰	۰	۱	۰	۱
۰	۱	۰	۰	۱
۰	۱	۱	۱	۰
۱	۰	۰	۰	۱
۱	۰	۱	۱	۰
۱	۱	۰	۱	۰
۱	۱	۱	۱	۱

$$s = sum = x \oplus y \oplus C_{in}$$

$$C_{out} = carry = xy + C_{in}y + C_{in}x$$

HA

X	Y	C	S
۰	۰	۰	۰
۰	۱	۰	۱
۱	۰	۰	۱
۱	۱	۱	۰

$$c = carry = xy$$

$$s = sum = x \oplus y$$

QA

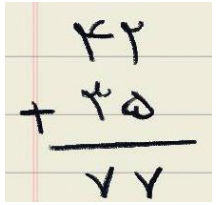
X	Y	S
۰	۰	۰
۰	۱	۱
۱	۰	۱
۱	۱	۰

$$s = sum = x \oplus y$$

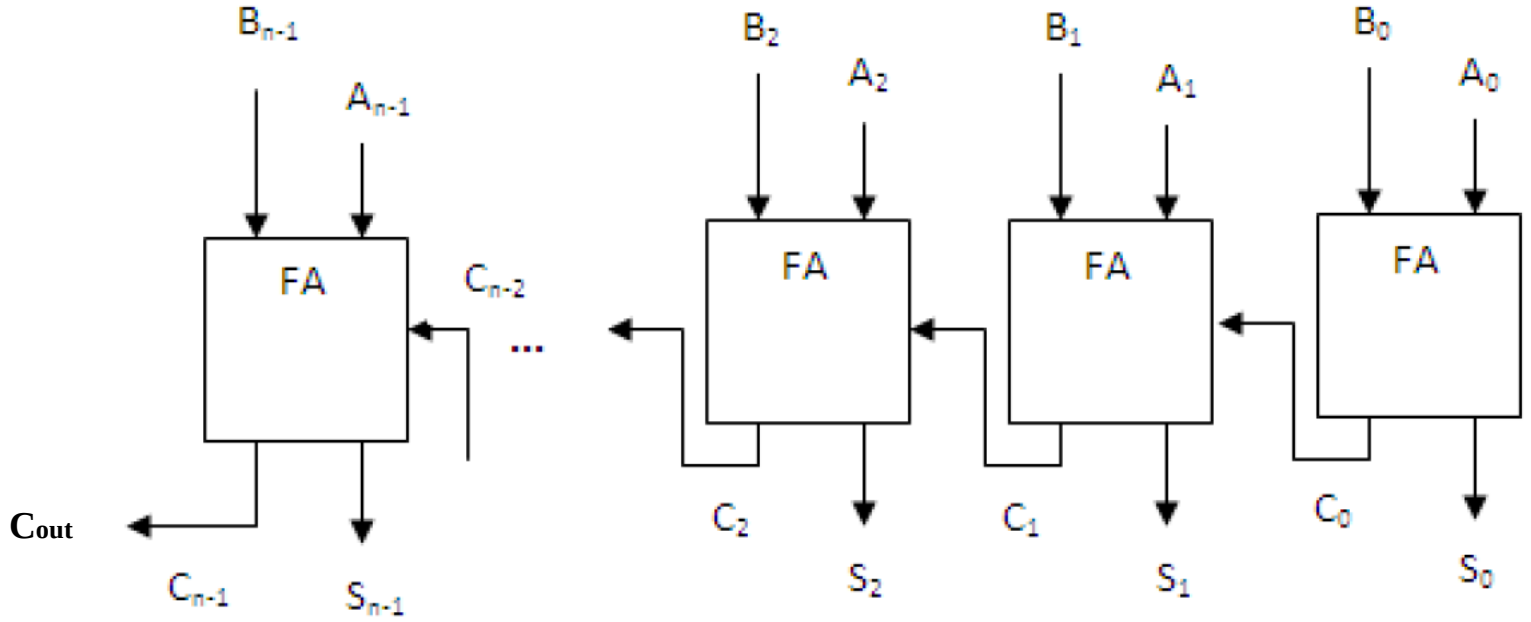
جمع‌کننده (Adder)

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

Ripple Carry Adder



- جمع‌کننده آبشاری
- ایده اصلی: سریال کردن n تمام جمع‌کننده (مشابه عملیات جمع مبنای ده به‌صورت دستی)



معیارهای مقایسه طراحی اجزای ALU

- ارزیابی و بررسی اجزا از نظر کیفیت و هزینه
  - **کیفیت:** تاخیر در محاسبه پاسخ نهایی (Delay)
    - زمان لازم برای دریافت خروجی از لحظه ورود داده
    - برای سادگی، در محاسبه این پارامتر، تاخیر گیت‌ها را ثابت درنظر می‌گیریم (d)
  - **هزینه** (Hardware Cost): تعداد گیت‌های لازم تعداد ترانزیستورهای استفاده شده در طراحی مدار

Ripple Carry Adder

- محاسبه معیارهای تاخیر و هزینه در جمع‌کننده آبشاری

- هزینه:

Cost (ripple carry) = n * Cost (FA) = n * [(C_{xor}) +3*(C_{and}) +C_{or}] = 5n * C_{gate} = **5n**

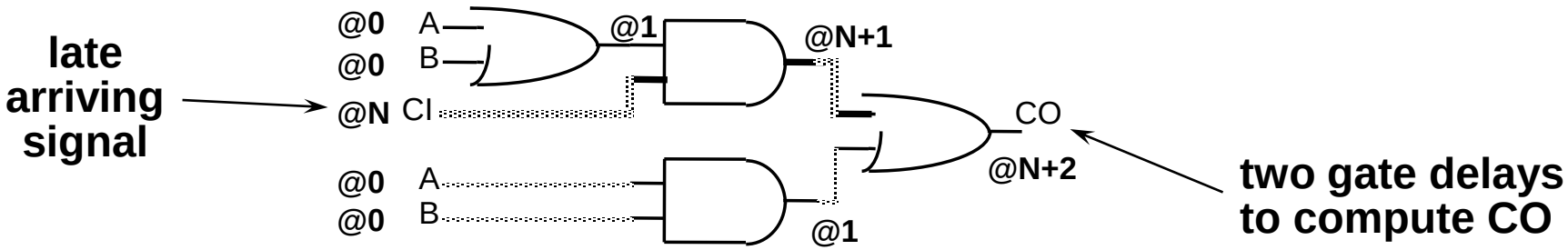
- تاخیر:

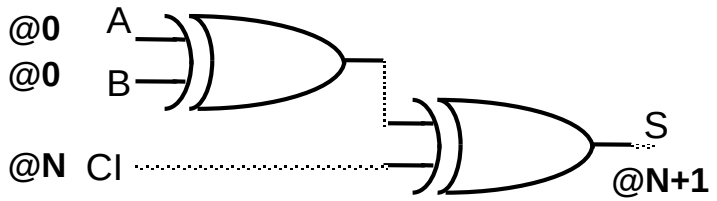
Delay (FA) = Max (delay(sum),delay(carry)) = Max (delay(1 level gate), delay(2 level gate)) = 2d

Delay (RCA) = Delay (sum) = (n-1) 2d + d = (2n-1)d; Delay (carry) = 2nd → Max (d(sum),d(carry)) = **2nd**

Ripple Carry Adder

- محاسبه تاخیر بیت جمع و بیت نقلی





Ripple Carry Adder

- جمع‌کننده آبشاری از نظر طراحی ساده‌ترین است
- تاخیر این نوع جمع‌کننده زیاد و برابر 2nd است
- وابستگی تاخیر به تعداد بیت‌های ورودی
  - رشد خطی تاخیر با افزایش تعداد ورودی‌ها
- از لحاظ هزینه مناسب است ولی از نظر تاخیر کند محسوب می‌شود
  - در نتیجه به سراغ سایر انواع جمع‌کننده می‌رویم