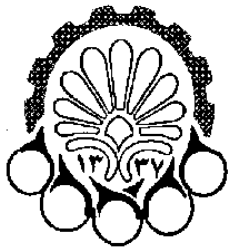


دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر و فناوری اطلاعات

جزوه درس

معماری کامپیوتر

Computer Organization & Design

نسخه ۵.۱

دکتر زرندی

فهرست

فصل اول
مروری بر مدار منطقی
و حافظه‌های رایانه

در این بخش در ابتدا یادآوری مختصری از درس مدار منطقی می‌شود سپس با زبان verilog آشنا می‌شویم که برای استفاده از آن برنامه ModelSim پیشنهاد می‌شود که در ضمیمه ۱ به طور جامعی مورد بررسی قرار گرفته است.

سپس با حافظه‌های رایانه آشنا می‌شویم. در ابتدا حافظه‌ها را بر اساس سرعت و هزینه در سلسله مراتب حافظه مورد بررسی قرار می‌دهیم و سپس می‌کوشیم تا با کمترین هزینه بیشترین سرعت را داشته باشیم. همانطور که می‌دانید حافظه اصلی ارزان اما سرعت آن کم است پس برای اینکه سرعت را بالا ببریم مقداری حافظه پرسرعت را میان حافظه ا صلی و پرداز شگر قرار می‌دهیم(cashe) و می‌کو شیم با شیوه‌های مختلف این ارتباط را سریعتر کنیم

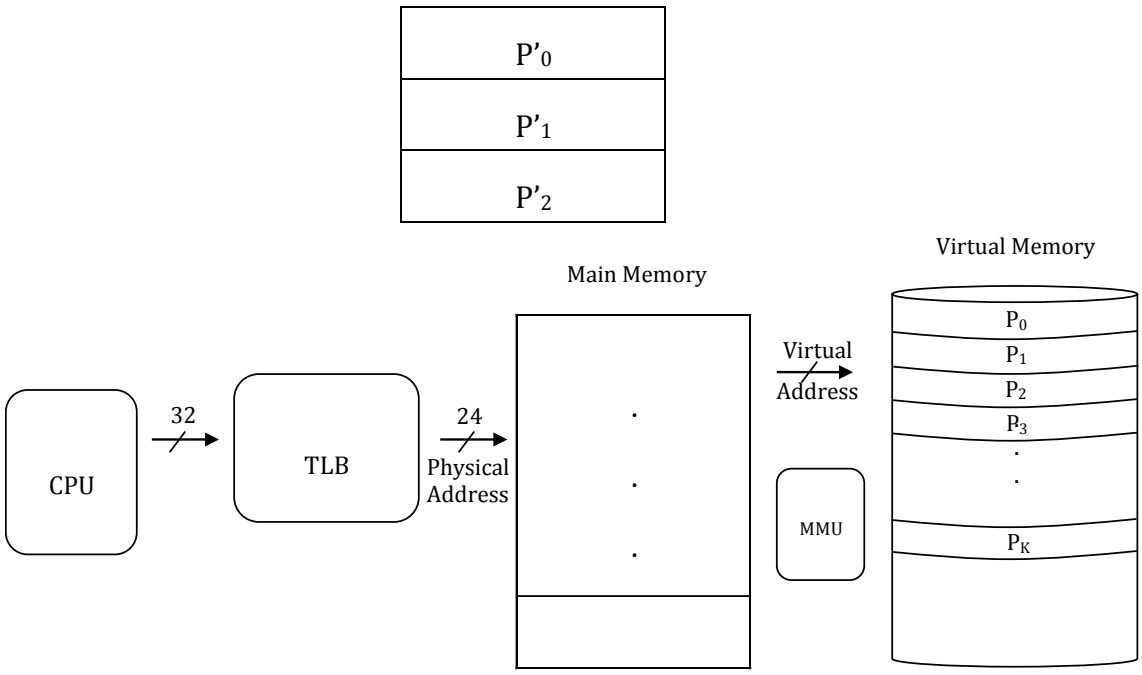
سپس فاکتورهای کارایی یک سیستم مورد بررسی قرار می‌گیرد.

TLB

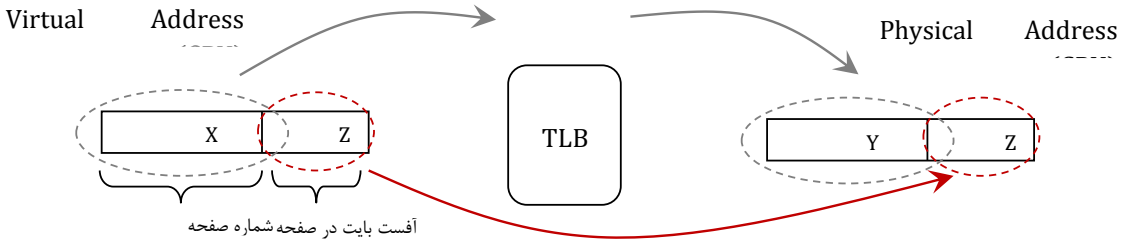
وقتی CPU ۳۲ بیت آدرس میدهد و حافظه ۱۶MB و باس آدرس ۲۴ بیتی است استفاده از ۲۴ بیت کم ارزش کارایی بالایی به ما نمیدهد. به همین دلیل هارد دیسک مطرح و از آنجایی که هارد دیسک خیلی کند و لخت است از آن به صورت یک Virtual Memory در کنار Main Memory استفاده شد. این امر به طور کلی دو مزیت دارد:

- استفاده از کل ۳۲ بیت
- سرعت نسبتاً بالا

به این منظور هارد دیسک و Main Memory به واحد هایی به نام page تقسیم شدند که از آن برای آدرس دهی استفاده شد. به علاوه اندازه هر page وابسته به عوامل مختلفی از جمله سرعت دیسک ،. .. میباشد.



از آنجا که آدرسی که از طرف CPU فرستاده میشود ۳۲ بیتی و Main Memory ۲۴ بیتی است از TLB به عنوان مترجم ا ستفاده مي شود. به این صورت که آدرسVirtual از سمت CPU را به آدرس فیزیکی در Main Memory ترجمه میکند (شماره page اصلی را گرفته و شماره page در Main Memory را میدهد).



یک TLB(Translation Lookaside Buffer) در واقع یک حافظه میانجی در CPU اســـت که قســـمتهایی از جدول نگاشت pageها را برای ترجمه آدرس مجازی به فیزیکی در خود نگاه میدارد.

این جدول میتواند به عنوان مثال به شکل زیر باشد:

P	P’
0	2
1	0
2	X
3	X
.	.
.	.
.	.
232-1	1

اما در این حالت بسیاری از خانهها معتبر نیستند. برای حل این مشکل و بهینه سازی، جدول را برعکس کرده و به صورت زیر تغییر دادند تا فضای کمتر و بهینه تری را اشغال کند.

P	P’
1	0
99	1
0	2
2	3
.	.
.	.
.	.
6	K

یک TLB نوعا یک حافظه CAMا ست و بر ا ساس محتوا ج ستجو کرده و این گونه عمل میکند که آدرس واقعی دریافت شـــده را با تک تک pageها مقایســـه میکند و اگر برابر بودند یک hit رخ میدهد و آدرس فیزیکی را بر میگرداند و در غیر این صورت (داده در Main Memory نباشد) miss رخ داده است.

TLB در مواردی نیز به دلیل توان مصـــرفی بالای حافظههای CAMبا مکانیزم k-way set associative مورد استفاده قرار میگیرد.

در حالتی که miss رخ میدهد MMU که معمولاً در خود TLB جا دارد، داده مورد نظر را با اســتفاده از الگوریتم LRU(Least Recently Used) در Main Memory جایگزین میکند.

به عبارتی میتوان گفت که TLB تقریباً شــبیه یک Fully Associative Cache عمل میکند که خانه‌های آن به tagهای موجود در cache شباهت دارند.

و اما مســئله ای دیگر محل cache در کنار TLB اســت. اگر cache قبل از TLB قرار بگیرد به این معنی خواهد بود که داده‌های هارد دیســک را در بر دارد که در این حالت کارایی بســیار بالا میرود ولی بزرگتر شــدن آدرس‌ها باعث ایجاد هزینه بیشتر نیز میشود.

و اگر بعد از TLB باشـــد به این معنی خواهد بود که داده‌های مجود در Main Memory را در خود نگاه میدارد که این بار کارایی و هزینه حالت قبل را نخواهد داشت.

به علاوه با توجه به اینکه ترکیب متوالی cache و TLB وقت گیر است این دو را به صورت موازی در کنار هم به کار میگیرند. با این حال در مواردی نیز cache قبل و اکثراً بعد از TLB قرار میگیرد.

برآورد کارایی[1]

علم حیل، همان علمی است که راه های شناخت تدابیر و شیوه های عملی کردن مفاهیم ریاضی در صنعت را مشخص می سازد و نشان می دهد که چگونه می توان مفاهیم عقلی ریاضی را در اجسام طبیعی آشکار نمود. .. یکی از علوم حیل(=مهندسی)، علمی است که پیرامون ساختن ابزار و وسایل برای صنایع عملی مورد استفاده قرار می گیرد.

[1] Performance Evaluation

تفاوت كارايي و بازدهي عملياتي

ما در این مجال، تفاوتی بین کارایی(Performance) و بازدهی عملیاتی(Throughput) قایل نیستیم. زیرا در صفحات آینده خود را به ریزپردازنده و کارایی تعریف شده برای آن محدود می کنیم. لیکن می توان گفت بازدهی عملیاتی(Throughput) زیر شاخه ای از کارایی(Performance) است.

- کارایی(Performance): اصطلاحی کلی است. می توان آن را ایده آل کردن فرآیند در سیستم‌ها دانست. موضوع این ایده آل سازی می تواند اجزا و مشخصات گوناگون سیستم باشد: گاهی زمان پاسخ سیستم به یک درخواست(Execution Time)، گاهی Clock، گاهی سرعت، گاهی توان مصرفی (Power) و. .. .

تعریف غیر دقیق بازدهی عملیاتی طبق کتاب پترسون به شرح زیر است:

- بازدهی عملیاتی (Throughput): مجموع کاری که یک سیستم در یک زمان مشخص می تواند انجام دهد.

به عنوان مثال مدیر یک شبکه علاقه مند است تا بازدهی عملیاتی سرور بالا باشد تا درخواست های بیشتری را در طول روز به سرانجام برساند.

كنفرانس هاي جهاني

کنفرانس هایی هم در دنیا داریم که تنها پیرامون کارایی سیستم های کامپیوتری و دست یابی به کارایی بالا (High Performance) است. از آن جمله است :

1. HPCA (High Performance Computer Architecture)
2. ISCA
3. MICRO

هر وقت به دنبال اطلاعات به روز هستید، به کتابچه های این‌ها مراجعه کنید.

تبيين اهميت موضوع

همان طور که در سخن فارابی اشاره شده بود، هنر مهندسی پیاده سازی مدل های ذهنی در جهان بیرونی است. مهندسی کامپیوتر هم این گونه است. حال فرض کنید برای مدلی که از یک کامپیوتر در ذهنمان داریم، ۲ نمونه

ساخته شده است. سوال اساسی این جاست "کدام یک از این ۲ رایانه، مطلوب تر است؟". مطلوب بودن یک کامپیوتر کاملا بستگی به کاربر و کاربری مورد انتظار او از کامپیوتر دارد. لذا نکته مهم این است که کدام یک برای "کارِ" کاربر، مناسب تر است. ما از شاخص کارایی(Performance) برای برآورد این "مناسب بودن" و "مقایسه" دو دستگاه در این زمینه، استفاده می کنیم.

کاربر	کار مورد انتظار از رایانه	مفهوم کارا بودن رایانه برای کاربر	نگرانی کاربر پیرامون قیمت رایانه
سازمان هواشناسی	پردازش حجم زیادی از داده‌ها	قدرت پردازشی رایانه	نگران نیست
سازمان هوافضا	کنترل باله های موشک	سریع، دقیق و بی درنگ عمل کردن	نگران نیست
گرافیست	کارهای گرافیکی سنگین	از پس تولید جلوه های بصری و تصاویر، در زمان کوتاهی برآید.	کمی نگران
کاربر خانگی	بازی، مرور اینترنت و برخی نیازهای روزمره	از هر نظر نسبتا قابل قبول عمل کردن(سرعت، قدرت محاسباتی و. ..)	کاملا نگران

جدول ۱ - مثالي از تفاوت مفهوم كارايي در زمينه هاي مختلف

اما نکته ای که در نظر تمامی کاربران اهمیت دارد، این است که رایانه نسبتاً "سریع" عمل کند. لذاست که می توان گفت که کارایی با هر تعریفی که بیان شود در رابطه زیر صدق می کند:

$$\text{Performance} \sim \frac{1}{\text{Execution time}}$$

که تعریف زمان اجرا(Execution time) که گاهی از آن با Response Time یاد می شود، طبق تعریف کتاب پترسون به قرار زیر است:

- زمان اجرا (Execution Time): مجموع زمانی که یک رایانه نیاز دارد تا یک وظیفه کاری(Task) را به سرانجام برساند. شامل: زمان اجرا شدن روی CPU، زمان دسترسی به حافظه، زمان صرف شده توسط سیستم عامل برای پاسخ گویی به درخواست های برنامه و. .. .

ما فعلا از رابطه بالا برای بررسی کارایی استفاده می کنیم. در انتهای این بخش، عامل هزینه(cost) را هم برای تحلیلی واقع گرایانه تر در نظر خواهیم گرفت.

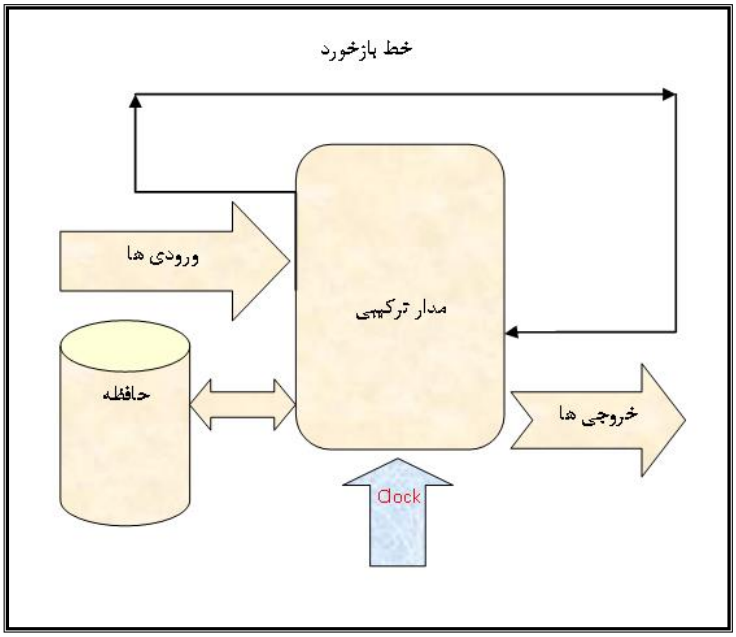
مثال) برنامه X روی کامپیوتر A و B به ترتیب ۱۰ و ۱۵ ثانیه طول می کشد. مشخصات دیگر رایانه‌ها را یکسان در نظر بگیرید. مشخص کنید که کامپیوتر A چند برابر B کاراتر است؟

پاسخ:

$$\frac{\text{Performance A}}{\text{Performance B}} = \frac{\text{Execution Time B}}{\text{Execution Time A}} = \frac{15}{10} = 1.5$$

وابستگي زمان اجرا به عوامل

می دانیم تقریبا تمامی کامپیوتر های متداول امروزی، بر اساس مدل زیر عمل می کنند:



لذا می توان گفت که هر عملیات طی چندین کلاک انجام می شود. اگر فرکانس کاری ریزپردازنده برابر f باشد آن گاه هر کلاک، زمانی برابر $\frac{1}{f}$ نیاز دارد. لذا داریم :

$$\text{زمان اجرای برنامه(Execution Time} = \frac{\text{تعدادکلاک های مصرفی برنامه}}{f}$$

می دانیم که هر برنامه از چندین دستور تشکیل شده است. اگر تعداد دستورات برنامه n باشد داریم:

$$\text{زمان اجرای برنامه(Execution Time} = \sum_{i=1}^{n} \frac{\text{تعدادکلاک های مصرفی دستورiام}}{f}$$

CPI

CPI (Clock per Instruction) یک پارامتر پرکاربرد در برر سی کارایی (Performance) یک رایانه ا ست. بنا به تعریف برای مجموعه دستور العمل X با k دستور داریم:

- تعداد کلاک به ازای دستور در مجموعه دستورالعمل های X :

$$CPI_X = \frac{\text{تعداد کلاک های مصرفی برای همه } X \text{ دستور مجموعه}}{K}$$

این پارامتر را برای ۴ مجموعه از دستور العمل‌ها محاسبه می کنند:

۱. یک تک دستور از مجموعه دستورات ریزپردازنده(Instruction set):

$$CPI_i = \text{تعداد کلاک لازم برای اجرای دستور i ام}$$

در این حالت، به مفهوم اندیس CPI دقت شود.

۲. مجموعه تمامی دستورات ریزپردازنده (Instruction Set). در این صورت :

$$CPI = \frac{\sum_{i=1}^{\text{تعداد دستورات } instruction\ set} \text{تعداد کلاک مصرفی برای دستور } i \text{ ام}}{\text{تعداد دستورات } instruction\ set} = \frac{\sum_{i=1}^{\text{تعداد دستورات } instruction\ set} CPI_i}{\text{تعداد دستورات } instruction\ set}$$

۳. مجموعه دستورات یک برنامه خاص :

$$CPI = \frac{\sum_{i=1}^{\text{تعداد دستورات برنامه}} \text{تعداد کلاک مصرفی برای دستور } i \text{ ام}}{\text{تعداد دستورات برنامه}}$$

۴. کلاس j ام از مجموعه دستورات ریزپردازنده(Instruction set) :

$$= \frac{\sum_{i=1}^{instruction\ set \text{ تعداد دستورات کلاس } j \text{ ام}} \text{تعداد کلاک مصرفی برای دستور } i \text{ ام این کلاس}}{instruction\ set \text{ تعداد دستورات کلاس } j \text{ ام}} CPI_j$$

در این حالت، به مفهوم اندیس CPI دقت شود.

اکنون که با CPI آشنا شدیم می توانیم با رابطه زیر نیز آشنا شویم:

$$\text{زمان اجرای برنامه(Execution Time)} = \sum_{k=1}^{t} \frac{C_k . CPI_k}{f}$$

t تعداد کلاس های دستورات ریزپردازنده است.

C_k تعداد دستورات برنامه که متعلق به کلاس k ام دستورات ریزپردازنده هستند، است.

CPI_k نرخ کلاک به ازای دستور برای مجموعه دستورات کلاس k ام دستورات ریزپردازنده است.

نکته: در این مبحث، تمام دستوراتی از مجموعه دستورالعمل‌ها که تعداد کلاک یکسانی مصرف می کنند در یک کلاس از Instruction Set قرار می گیرند.

نکته: همان طور که گفتیم، زمان اجرا را فقط ریزپردازنده تعیین نمی کند. عواملی مانند حافظه سیستم، سیستم عامل و. .. نیز در زمان اجرا موثر اند. مثلا دستورات باید از حافظه واکشی (Fetch) شوند. لذاست که تعداد کلاک مصرفی برنامه به آن عوامل نیز مربوط می شود. برای همین می توان گفت که فرمول های بالا با کمی تسامح بیان شده اند.

IPC

IPC (Instruction per Clock) پارامتر مشابهی است که گاه به گاه مورد استفاده قرار می گیرد. تعریف آن به قرار زیر است:

- نرخ تعداد دستور انجام شده به ازای یک کلاک در مجموعه دستورالعمل های X :

$$IPC_X = \frac{K}{\text{تعداد کلاک های مصرفی برای همه } K \text{ دستور مجموعه } X}$$

مشـابه آن چه در مورد CPI گفتیم، مجموعه X می تواند ۴ حالت گوناگون داشـته باشـد که در بخش قبل بحث شد. رابطه زیر ۲ مفهوم بیان شده اخیر را به هم پیوند می زند:

$$CPI \times PCI = 1$$

نکته: CPI و IPC بدون واحد هستند.

سه پارامتر اساسي

برای تعیین کارایی یک سیستم به زمان اجرای برنامه روی آن روی آوردیم. سوالی که مطرح است این است که چه پارامترهایی در تعیین این زمان موثر است. مطابق روابط صفحات قبل داریم :

$$\text{زمان اجرای برنامه (Execution Time)} = \frac{\text{تعداد دستورات} \times CPI}{f}$$

لذاست که می توان گفت کارایی یک سیستم کامپیوتری به ۳ عامل زیر مرتبط است:

۱. CPI دستورات برنامه که ساختمان و نحوه طراحی ریزپردازنده تعیین کننده آن است.

۲. تعداد دستورات برنامه که Instruction Set و الگوریتم کامپایل کردن برنامه تعیین کننده آن است.

۳. فرکانس کاری ریزپردازنده.

اگر در تحلیل خود از کارایی سـیسـتم‌ها، هر کدام از این ۳ عامل را در نظر نگیریم، ره به جایی نخواهیم برد. لذا نباید فریب بهینه بودن تنها یکی از موارد بالا را خورد. ممکن اسـت از خود بپرسـید که "پس چرا در بازار تنها روی فرکانس کاری ریزپردازنده تبلیغ می شود؟" پا سخ این جا ست که شرکت های سازنده معمولا سازگاری (compatibility) یک نسل از ریزپردازنده‌ها را رعایت می کنند و از سویی برنامه های کاربردی و سیستم های عامل مطابق با اشـتراکات موجود بین Instruction set ریزپردازنده های یک نسـل نوشـته می شـوند، لذا عملا پارامترهای "CPI" و "تعداد دستورات برنامه" برای حجم زیادی از برنامه هایی که کاربر خانگی یا اداری اجرا می کند، به ازای ریزپردازنده های مختلف یکسـان اسـت. لیکن یک تحلیل جامع باید این ۲ پارامتر را نیز در نظر بگیرد.

MIPS

MIPS (Million Instruction per Second) مقیاسی دیگر برای مقایسه کارایی ۲ سیستم است.

$$\text{MIPS} = \frac{1}{10^6} \times \frac{\text{تعداد دستورهای اجرا شده}}{\text{زمان طی شده به ثانیه}}$$

واحد این پارامتر را هم معمولا MIPS می نامند. اگر د ستورات اجرا شده روی ۲ سی ستم دقیقاً یکی با شد، آن سیستمی که MIPS بیشتری دارد، کاراتر است. لیکن اگر دستورات متفاوت باشد هیچ قضاوتی نمی توان کرد.

رابطه زیر را برای MIPS داریم:

$$\text{MIPS} = \frac{\text{تعداد دستورات اجرا شده}}{Execution\ Time \times 10^6} = \frac{f}{\text{CPI} \times 10^6}$$

تمرین ۵: MIPS پردازنده خود را حساب کنید.

نکته: همان گونه که گفتیم اگر دستورالعمل های اجرا شده یکسان نباشند می توان مثالی زد که ریزپردازنده ای که دارای MIPS بالاتری باشـــد، دارای زمان اجرای بدتری باشـــد. لذا در حالت کلی رابطه MIPS و کارایی لزوماً مستقیم نیست.

مثال اول کارایی :

برنامه ای در کامپیوتر A در ۱۰ ثانیه اجرا می شـــود. نرخ کلاک در A برابر ۴۰۰ مگاهرتز ا ست. در کامپیوتر B در ۶ ثانیه اجرا می شود. نرخ کلاک B را بد ست آورید. در ضمن می دانیم که برای هر دســتور، کامپیوتر B به ۱. ۲ برابر تعداد کلاک لازم روی A به کلاک نیاز دارد.

$$1.2 \times CPI_A = CPI_B$$

تعداد کلاک لازم $1.2 \times Instruction\ Count \times CPI_A = B$

$$6 = \frac{(InstructionCount.CPI_B)}{f_B} = \frac{(InstructionCount.CPI_A)\times 1.2}{f_B} = \frac{f_A \times 10 \times 1.2}{f_B}$$

$$f_B = 800 \times 10^6\ Htz$$

مثال دوم کارایی :

یک برنامه بر روی ۲ رایانه با اجرا شده است. از آن جایی که کامپایلرها متفاوت بوده است، ۲ کد مختلف به دست آمده است. اگر کلاس های دستورات A , B ,C دارای CPI زیر باشند و تعداد دستورات از هر کلاس در ۲ کد مختلف به قرار زیر باشد مطلوب است رایانه کاراتر را بیابید. هم چنین رایانه ای که دارای MIPS بالاتری اســـت را بیابید. آیا این ۲ یکســـان هستند؟ فرکانس کاری هر دو رایانه را ۱ مگاهرتز فرض کنید.

	تعداد دستور کلاس A	تعداد دستور کلاس B	تعداد دستور کلاس C
کد برنامه روی ماشین ۱	۵	۱	۱
کد برنامه روی ماشین ۲	۱۰	۱	۱

کلاس دستور	CPI
A	1
B	2
C	3

$$\text{MIPS1}=\frac{\text{f}}{CPI\ OF\ PROGRAM \times 10^6}=\frac{1}{\frac{5\times1+1\times2+1\times3}{1+1+5}}=0.7$$

$$\text{MIPS2}=\frac{\text{f}}{CPI\ OF\ PROGRAM \times 10^6}=\frac{1}{\frac{10\times1+1\times2+1\times3}{10+1+1}}=0.8$$

$$\frac{1}{\text{Execution time}}\sim \text{Performance}$$

$$\text{Performance 1}\sim\frac{\text{f}}{\text{Instruction Count}\times\text{CPI}}=\frac{\text{f}}{7\times\frac{5\times1+1\times2+1\times3}{1+1+5}}=10^5$$

$$\text{Performance 2}\sim\frac{\text{f}}{\text{Instruction Count}\times\text{CPI}}=\frac{\text{f}}{12\times\frac{10\times1+1\times2+1\times3}{10+1+1}}=0.07\times10^5$$

لذا کامپیوتر ۱ کاراتر است و کامپیوتر ۲ دارای MIPS بیشتری است. لذا در این مثال دیدیم که MIPS با کارایی لزوماً نسبت مستقیم ندارد.

دو سبک طراحی

در ابتدای اختراع رایانه، رایانه‌ها دارای مجموعه دستورالعمل های اندک و ساده ای بودند. با رشد فناوری رایانه و همه گیر شـــدن آن، طراحان به ســـمت رایانه هایی با مجموعه دســـتورالعمل های پیچیده رفتند. لیکن پس از چندین سال متوجه شدند که این پیچیدگی تبعاتی چند به همراه دارد:

- پروسه طراحی را پیچیده و هزینه بر می کند.
- زمان ارایه محصول به بازار (Time To Market) را افزایش می دهد. امری که سبب عقب افتادن شرکت موتورولا از شرکت اینتل در بازار ریزپردازنده های رایانه های شخصی در قرن گذشته میلادی شد.
- عیب یابی مدارات و تضمین صحت عملکرد آن را برای کاربرد های حیاتی(Critical) مشکل می سازد.

لذا طراحان دوباره به سمت مجموعه دستورالعمل های اندک و ساده بازگشتند. این بازگشت ابتدا توسط محافل دانشگاهی در دهه اواخر دهه ۸۰ و ابتدای دهه ۹۰ میلادی رخ داد.

CISC (Complex Instruction Set Computer) و RISC (Reduced Instruction Set Computer)

نام ۲ دسته از رایانه هاست که دارای ایده های معماری متفاوتی هستند. اگرچه هردو دسته از الگوریتم فون نیومن بهره می برند لیکن در مشخصات زیر با هم متفاوت اند :

نوع مشخصه	مشخصه RISC	مشخصه CISC	ملاحظه
تعداد و تنوع دستورات در مجموعه دستور العمل	کم	زیاد	
تعداد کلاک مصرفی برای دستورات مختلف	ثابت و کم	متغیر و زیاد	برای همین در RISC ها می توان پهنای کلاک را کاهش داد.
شیوه های آدرس دهی	محدود و کم	زیاد و متنوع	معمولا انواع آدرس دهی های غیر مستقیم را در RISCها نداریم.
تعداد بایت مصرفی هر دستور	ثابت و کم	متغیر و گاهی زیاد	
تعداد ثبّات های عمومی	کم	زیاد	مثلا برای دستور جذر در یک رایانه CISC به چندین و چند ثبّات میانی نیاز است.
تعداد عملوند های دستورالعمل	ثابت و کم	زیاد و متغیر	

تعداد دستورات لازم برای یک برنامه مشخص	بیشتر از CISC	کمتر از RISC	

البته گاهی هر دوی این دیدگاه‌ها را می توان در یک ریزپردازنده جست و جو کرد. مثلا هسته پنتیوم توسط دیدگاه RISC ساخته شده است لیکن سعی شده است که کل ریزپردازنده از بیرون همانند CISC به نظر آید.

نکته: در عبارات مربوط به محاسبه CPI، در رایانه های RISC کران عبارت Σ افزون تر است ولی پهنای کلاک کمتری دارد.

نکته: در عبارات مربوط به محا سبه CPI، در رایانه های CISC کران عبارت Σ کوچک تر ا ست ولی پهنای کلاک بیشتری دارد.

قانون ۸۰-۲۰

این قانون شهودی و حدودی روی بسیاری از پدیده‌ها حکم فرما ست. مثلا شما با ۲۰ در صد از تلاش آرمانی ممکن ا ست تا ۸۰ در صد نتیجه نهایی را بگیرید. یا ۲۰ در صد مجموعه د ستورالعمل های یک کامپیوتر، نیاز ۸۰ در صد کاربران را برآورده می کند.

قانون آمدال

آقای آمدال _که سرپرستی پروژه IBM mainframes را چندی به عهده داشت اولین بار این قانون را ارایه کرد. این قانون به ما کمک می کند تا دریابیم که روی کدام بخش از یک سیستم باید سرمایه گذاری بیشتری بکنیم تا بازده بیشتری بدست آوریم.

- قانون آمدال: اگر نسبت د ستورات ترتیبی یک برنامه به کل د ستورات آن f با شد(یعنی نتوان در اجرای آن بخش تسریع کرد) و الباقی برنامه را بتوان_به هر طریقی_ p برابر سریع تر اجرا کرد، میزان افزایش سرعت برنامه(Speed-Up)، یعنی نسبت زمان اجرا در حالت دوم به حالت اول، از رابطه زیر به دست می آید:

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

نکته: حالت تعمیم یافته قانون آمدال؛ اگر a_i نسبت از برنامه را بتوان به میزان p_i برابر سریع تر اجرا کرد :

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

تمرین ۵: قانون آمدال را اثبات کنید.

مدت زمان اجرای برنامه قبل از افزایش ســـرعت را x می نامم. مدت زمان اجرای برنامه بعد از افزایش سرعت را y می نامم.

این گونه نسبت افزایش سرعت برابر مقدار زیر است:

$$speedUp = \frac{x}{y}$$

ازسویی دیگر طبق فرض مساله داریم:

$$x = x.f + x.(1-f)$$

که جمله اول عبارت بالا بخشی است که سرعت آن تغییر نخواهد کرد و قسمت دوم عبارت بالا بخشی است که تغییر خواهد کرد.

پس از تغییر سرعت، بخش دوم در $\frac{1}{p}$ زمان قبل اجرا می شود لذا برای y داریم:

$$Y = x.f + (\frac{x.(1-f)}{p})$$

لذا برای نسبت $\frac{x}{y}$ داریم :

$$speedUp = \frac{x}{y} = \frac{x.f + x.(1-f)}{x.f + (\frac{x.(1-f)}{p})} = \frac{f+1-f}{f+\frac{1-f}{p}} = \frac{1}{f+\frac{1-f}{p}}$$

مثال اول آمدال:

یک برنامه روی یک کامپیوتر در ۱۰۰ ثانیه اجرا می شـــود که ۶۰ ثانیه آن مربوط به دستورالعمل های ضرب برنامه است. دستورالعمل های ضرب چقدر سریع تر شوند تا اجرای برنامه ۲,۵ برابر سریعتر گردد؟

$$Speed\text{-}Up = 2.5 = \frac{1}{f + \frac{1-f}{p}} = \frac{1}{.4 + \frac{.6}{p}} \rightarrow 0=1 \text{ (تناقض)}$$

پس چنین کاری امکان ندارد. راه حل دیگر این بود که اگر برنامه بخواهد 2.5 برابر ســـریعتر شود یعنی باید در $\frac{1}{2.5} \times 100 = 40$ ثانیه اجرا شود. لیکن می دانیم که دستورات غیر از ضرب ۴۰ ثانیه زمان می خواهند. این یعنی باید سرعت اجرای بخش ضرب بی نهایت باشد تا هیچ زمانی نبرد. اما این موضوع امکان ندارد.

مثال دوم آمدال :

یک برنامه در زمان ۸۰ ثانیه بر روی یک رایانه اجرا شـده اسـت. ۲۰ درصـد 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؟

مطابق تعمیم قانون آمدال داریم:

$$Speed\text{-}Up = \frac{1}{(1-.3-.2) + \frac{.3}{6} + \frac{.2}{8}} = 1.74$$

مثال سوم آمدال :

تابع ریشـــه دوم اعشـــاری در یک برنامه گرافیکی به طور معمول به کار می رود. فرض کنید زمان اجرای این تابع، ۲۰ درصـــد زمان از زمان اجرای برنامه گرافیکی را مصـــرف کند. ۲ راه کار برای بهبود برنامه موجود است. پیشنهاد اول: تابع ریشه دوم را ۱۰ برابر سریع تر کنیم.

پیشنهاد دوم: همه دستورات ممیز شناور را ۲ برابر تسریع کنیم.

این عملیات اعشاری ۵۰ درصد زمان کار گرافیکی را به خود مشغول می کند. کدام راه کار بهتر است؟

پیشنهاد اول: $Speed\text{-}Up = \frac{1}{(1-.2)+\frac{.2}{10}} = 1.22$

پیشنهاد دوم: $Speed\text{-}Up = \frac{1}{(1-.5)+\frac{.5}{2}} = 1.33$

لذا راه کار دوم بهتر است.

بستر آزمایشBenchmark

گفتیم که کارایی یک سیستم را معمولا با اجرای یک برنامه روی آن اندازه می گیریم. ولی نگفتیم چه برنامه ای. مسلماً برای عادلانه بودن قضاوت ما باید برنامه های آزمایشگر برای همه سیستم های هم ردیف، یک سان باشد. برنامه های بســـتر آزمایش این وظیفه را به عهده دارند. برای ســـرورها برنامه های بســـتر آزمایش ویژه ای داریم. برای کامپیوتر های نهفته(مانند موبایل ها) برنامه های آزمایش ویژه خودشـــان راداریم_مانند MIBENCH_ و بالاخره برای پردازنده های رایانه های شـــخصـــی PC نیز برنامه های بســـتر آزمایش ویژه ای وجود دارد_مانند SPEC CPU 2000.

البته این برنامه‌ها به ۲ دسته زیر تقسیم می شوند:

۱. CINT که عملیات اعداد صحیح را انجام می دهد.

۲. CFP که عملیات اعداد اعشاری انجام می دهد.

فصل دوم

ALU

همانطور که می‌دانید طراحی یک CPU، شـــامل بخش‌های متفاوتی اســـت. یکی از این بخش‌ها واحد منطق و محاسبات است. در این بخش قصد داریم که این واحد را مورد بررسی قرار دهیم و در نهایت آن را طراحی کنیم.

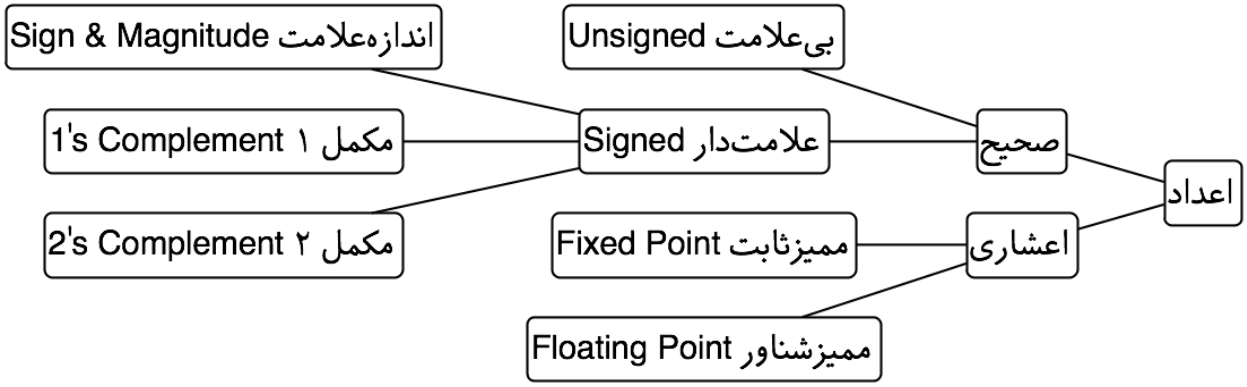
در ابتدا با جمع کننده‌ها آ شنا می شویم، سپس می‌خواهیم عملیات ضرب را در پرداز شگر طراحی کنیم. در این مرحله ابتدا با ضـرب کننده ترتیبی آشـنا می‌شـویم، که برای سـرعت بخشـیدن به آن از الگوریتم بوث اسـتفاده می‌شود. سپس با نحوه‌ی اجرای عملیات تقسیم آشنا خواهیم شد.

در پیاده سازی اعداد اعشاری ابتدا از شیوه ممیز ثابت استفاده می‌کنیم، اما می‌بینیم استفاده‌ی ما بسیار محدود می شود. سپس از ایده‌ای همانند نمایش علمی اعداد ا ستفاده می‌کنیم. هر چند خواهیم دید چون این اتفاق در فضای محدودی رخ می‌دهد دقت کار ما پایین خواهد بود.

در پایان با نمایش کاربردی BCD آشنا می‌شویم و عملیات پایه (جمع، تفریق، ضرب و تقسیم) آن را مورد بررسی قرار می‌دهیم.

جمع کننده‌ها:

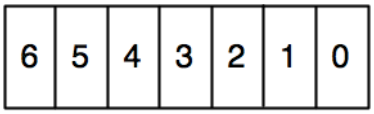
۲ عدد n بیتی داریم، ابتدا باید بدانیم این اعداد چه ساختاری دارند.



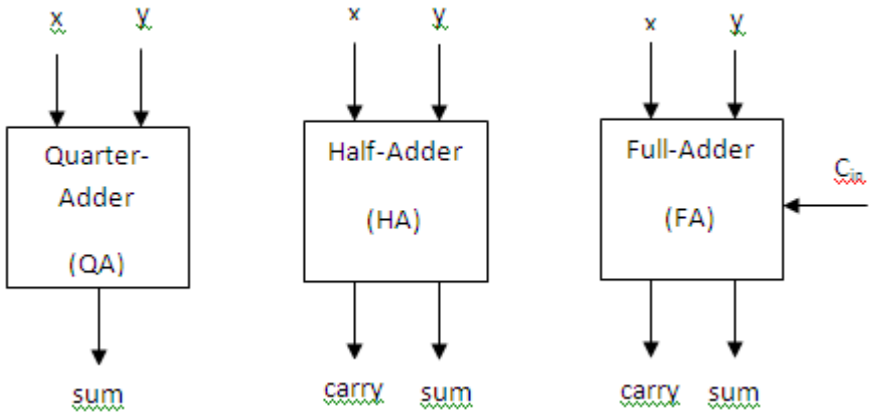
در روش اندازه علامت، علامت و عدد به صورت جدا از هم نگهداری می‌شوند. آخرین بیت (از سمت راست) نشان دهنده‌ی علامت است. در مکمل ۲ علامت و عدد در یک n بیت با هم نشان داده می‌شوند. در هر حال با وجود اعداد علامت دار دامنه‌ی اعداد قابل نمایش نصف می‌شود. چون هم اعداد مثبت داریم و هم اعداد منفی. در هر دو حالت علامت می‌تواند از طریق بیت آخر تشخیص داده شود. البته در روش مکمل ۲، برای پیدا کردن اندازه باید مکمل دوی عدد منفی را حساب کرد.

به طور کلی پیاده سازی اعمال حسابی نیازمند دانستن نوع عدد است و هر کدام نیز ملاحظات خاصی دارند.

برای سادگی فرض می‌کنیم اعداد صحیح و مثبت هستند (بدون علامت) بعدها تغییرات لازم برای هر نوع عدد را جداگانه بررسی خواهیم کرد.



در طراحی مدار معمولا اندیس گذاری از صفر و از سمت راست آغاز می‌شود.



:Quarter Adder

$s = sum = x \oplus y$

X	Y	S
٠	٠	٠
٠	١	١
١	٠	١
١	١	٠

$c = carry = xy$

$s = sum = x \oplus y$

:Half-Adder

X	Y	C	S
٠	٠	٠	٠
٠	١	٠	١
١	٠	٠	١
١	١	١	٠

:Full-Adder

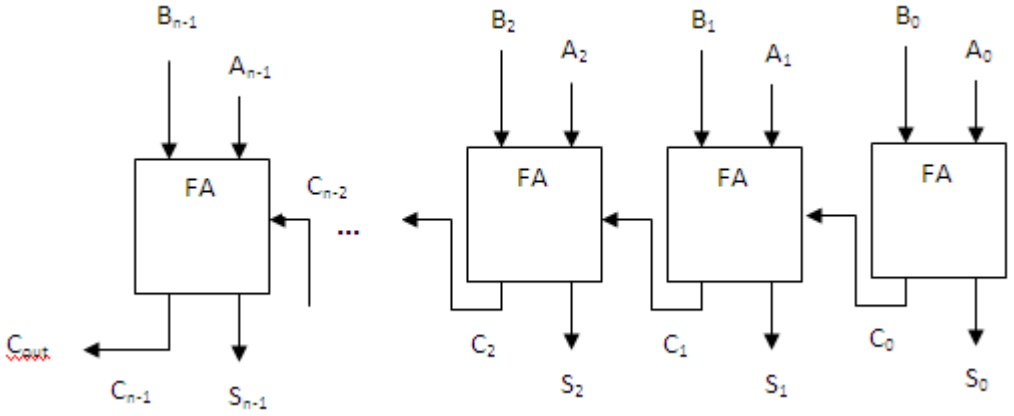
X	Y	C_{in}	C_{out}	S
۰	۰	۰	۰	۰
۰	۰	۱	۰	۱
۰	۱	۰	۰	۱
۰	۱	۱	۱	۰
۱	۰	۰	۰	۱
۱	۰	۱	۱	۰
۱	۱	۰	۱	۰
۱	۱	۱	۱	۱

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

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

جمع کننده‌ی آبشاری (Ripple-Adder)

برای ساختن جمع کننده‌ی n بیتی یکی از روش‌های ساده استفاده از چند FA پشت سر هم است که در واقع همان تکنیک مورد استفاده‌ی انسان در محاسبات مبنای ۱۰ است.



در زمان ساخت ALU فاکتورهای مختلفی ظاهر می‌شوند که نقش اساسی در انتخاب نوع طراحی ما خواهند داشت. از این رو لازم است که هر کدام از اجزای مورد نیاز از نظر کیفیت و هزینه مورد سنجش قرار بگیرند.

در اینجا منظور از کیفیت، تاخیر در محاسبه پاسخ نهایی، و منظور از هزینه، تعداد گیت‌های لازم برای طراحی آن مدار است.

- HW cost: هزینه‌ی سخت افزاری- معمولا با تعداد ترانزیستورهای استفاده شده یا تعداد gate سنجیده می‌شود.
- Delay: به معنای زمان لازم برای دریافت خروجی از لحظه‌ی ورود داده است. برای ساده شدن محاسبات لازم برای بدست آوردن میزان تاخیر در مدار، میزان تاخیر هر گیت را یک مقدار ثابت فرض می‌کنیم.

در جمع کننده‌ی آبشاری داریم:

$$Cost(RippleCarry) = n * \text{Cost}(FA) = n[1(C_{xor}) + 3(C_{and}) + 1(C_{or})] = 5 * n(C_{gate}) = 5n$$

اما در مورد Delay، ابتدا لازم است ببینیم یک FA خود چه مقدار تاخیر دارد.

اگر فرض کنیم مقدار تاخیر ثابت گیت‌ها d است، خواهیم داشت

$$\begin{matrix}\text{Sum تاخیر} = \text{d} \\ \text{Cout تاخیر} = 2\text{d}\end{matrix} \Rightarrow Full\ Adder \text{ تاخیر } = 2\text{d}$$

در نتیجه برای جمع کننده‌ی آبشاری داریم:

$$Delay \begin{cases} Delay_{sum} = (n-1)2d + d = (2n-1)d \\ Delay_{Cout} = 2nd \end{cases}$$

جمع کننده‌ی آبشاری از ساده ترین نوع جمع کننده هاست. اما تأخیر آن 2 nd است و می‌توان بهتر از این طراحی انجام داد. از آنجایی که میزان تأخیر آن وابسته به تعداد بیت‌های ورودی است، با افزایش تعداد بیت‌ها شاهد رشد خطی زمان لازم برای محاسبه خروجی هستیم. به همین دلیل زمانی که می‌خواهیم هزینه‌ی کمتری بپردازیم ولی زمان مهم نیست از آن ا ستفاده می‌کنیم. اما اگر زمان مهم با شد از جمع کننده‌های دیگر ا ستفاده خواهیم کرد.

جمع کننده با پیش‌بینی بیت نقلی (Carry Look-ahead Adder (CLA)):

در محاسبه‌ی بیت‌های نقلی (Carry) داریم:

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

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

حال G و P را به صورت زیر تعریف می‌کنیم:

Generate $G_i=A_iB_i$

Propagate $P_i=A_i+B_i$

پس خواهیم داشت:

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

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

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

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

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

پس C_{n-1} یک SOP است که فقط G_iها و P_iها در آن ظاهر شده‌اند.

می‌توان G_iها و P_iها را پس ازتأخیر زمانی d به دست آورد. زیرا همزمان و به صورت موازی بیت‌های متناظر را and و or می‌کنیم. در مرحله‌ی بعد G_iها و P_iها را به یک مدار منطقی ترکیبی می‌دهیم و پس از 2d تأخیر، carryها را خواهیم داشت.

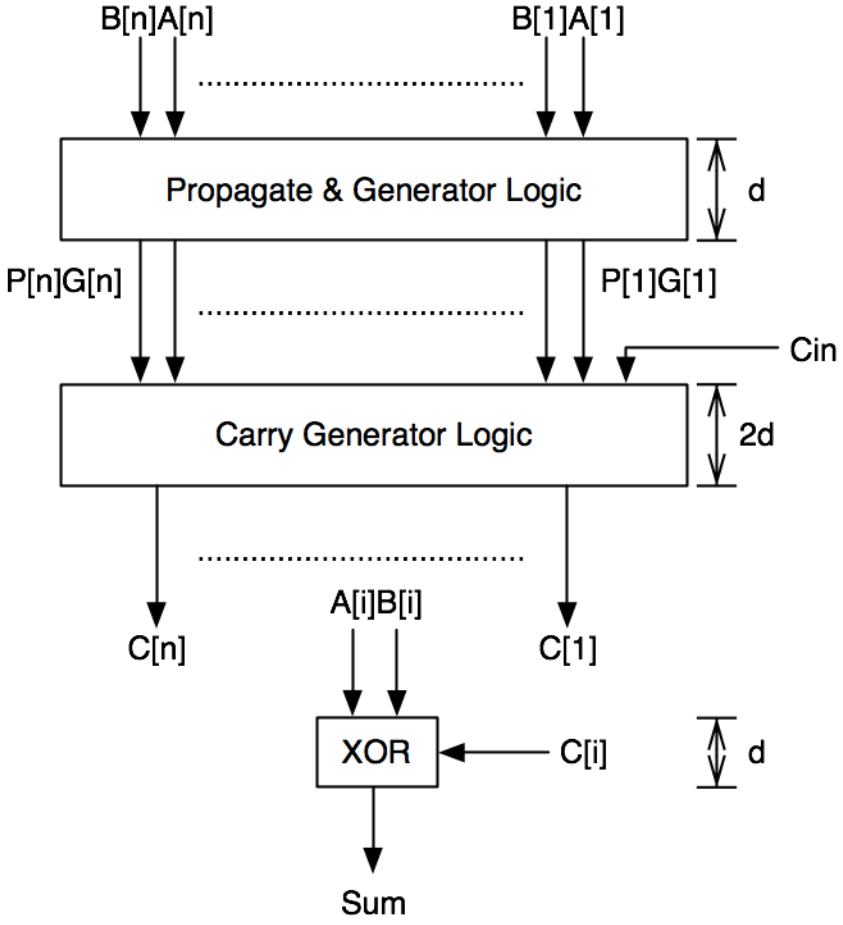
به این مدار ترکیبی که بیت‌ها را می‌گیرد و carry را به ما تحویل می‌دهد، Carry Generator و مدار جمع کننده‌ی حاصل از آن را Carry Look-ahead Adder می‌گویند.

با داشتن carryها و بیت‌های A و B، می‌توان با یک تمام جمع کننده (Full Adder) حاصل نهایی را حساب کرد. اما اینجا به C_{out} هم نیازی نداریم پس می‌توانیم از یک xor استفاده کنیم. این جمع کننده نیز پس از تأخیر زمانی d حاصل را محاسبه می‌کند (فقط یک گیت xor)، بنابراین در نهایت داریم:

$Delay_{sum}=4d$

$Delay_{carry}=3d$

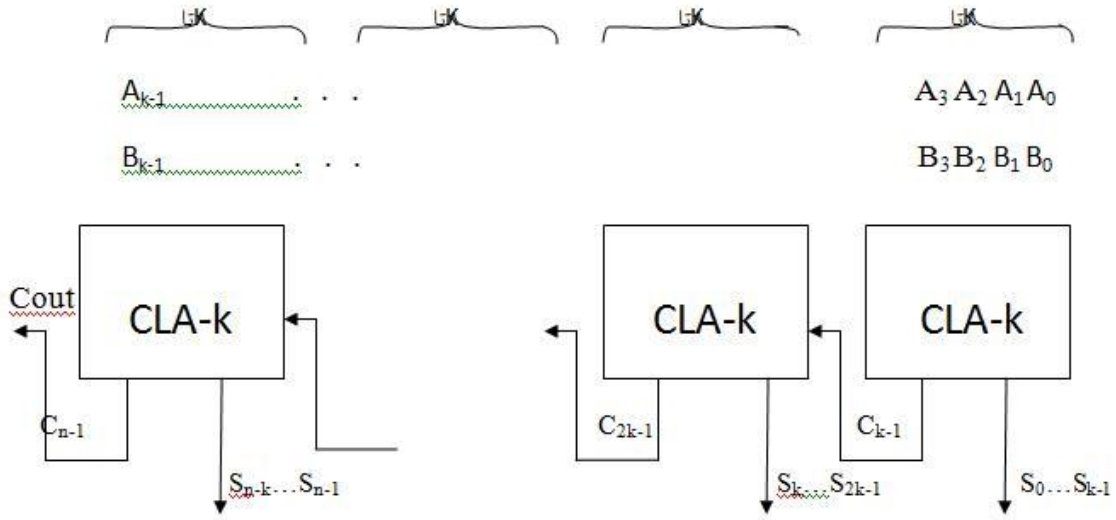
در زیر یک مدار جمع کننده با پیش بینی بیت نقلی را مشاهده می‌کنید.



طراحی Carry Look-ahead Adder تحول بـ سیار مهمی در جمع کننده‌ها به وجود آورد زیرا این اولین بار بود که تأخیر در یک جمع کننده به تعداد بیت‌های ورودی وابسته نبود. این ابداع باعث شد که مرتبه‌ی زمانی جمع کننده‌ها از $O(n)$ به $O(1)$ کاهش یابد.

در Carry Look-ahead Adder (CLA) که طراحی کردیم هر and، حداکثر n ورودی خواهد داشـت. در حالی که چنین and هایی در عمل وجود ندارند. با بررسـی ماکزیمم تعداد ورودی‌های گیت and در بازار می‌بینیم که حداکثر and ها ۴بیتی اند، پس می‌توان Carry Look-ahead Adder(CLA)های ۴ بیتی تولید کرد.

برای جمع اعداد n بیتی در عمل تعدادی Carry Look-ahead Adder (CLA)۴ بیتی را کنار هم می‌گذارند و مداری شبیه Ripple Carry Adder طراحی می‌کنند که البته سریع تر از Ripple Carry Adder ساده خواهد بود.



اگر فرض کنیم که این CLAهای k بیتی را بتوانیم طوری بسازیم که مثل قبل بیت نقلی را با 3d تأخیر و حاصل جمع را با 4d تأخیر به ما بدهد، خواهیم داشت:

$\text{Delay}_{\text{carry}} = 3d + 3d + \cdots + 3d = 3(\frac{n}{k})d$

$\text{Delay}_{\text{sum}} = (\frac{n}{k} - 1)3d + 4d = (3\frac{n}{k} + 1)\, d$

هرچه k کمتر با شد تأخیر بی‌شتر می شود. اگر k=n مانند همان n-bit-CLA عمل خواهد کرد و تأخیرها همان 3d و 4d خواهد شد. اگر k=1 مانند تمام جمع کننده‌ی عادی عمل می‌کند(و حتی بدتر از آن زیرا مدار در این حالت پیچیده تر شده و تاخیر افزایش می‌یابد). در حالت معمول k را ۴ قرار می‌دهند.

در این روش محاسبه‌ی G_iها و P_iها 2n گیت نیاز دارد و با محاسبه‌ی C_iها تعداد گیت‌ها از 5n بیشتر می‌شود و هزینه‌ی سخت افزاری هم بالا می‌رود اما در عوض کارایی تا حد خوبی افزایش و تأخیر کاهش می‌یابد.

در این مرحله از طراحی کارایی را به شکل زیر تعریف می‌کنیم:

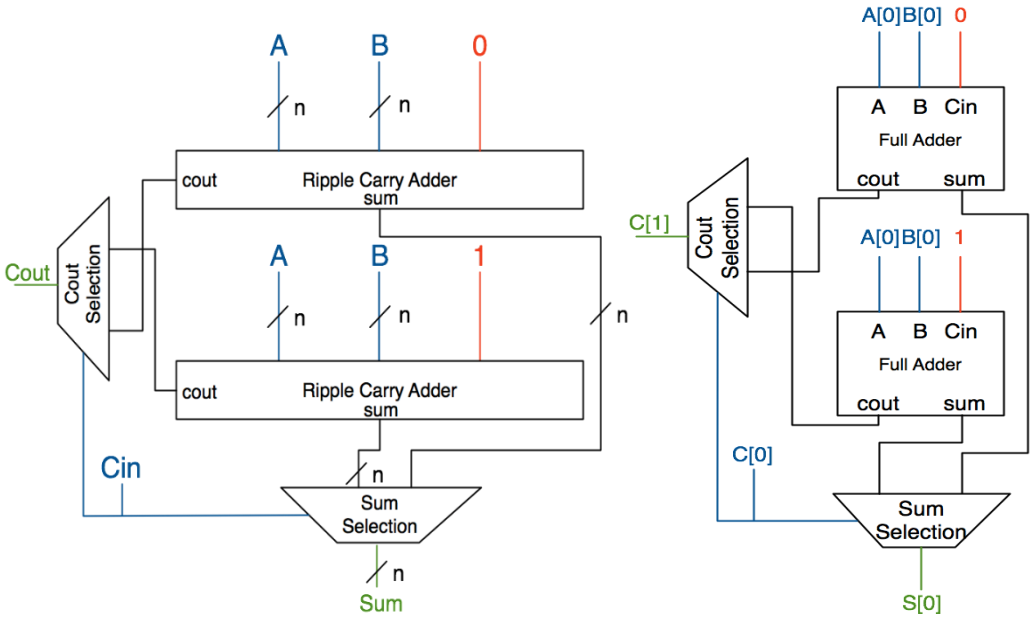
$\text{performance} \sim \frac{1}{Delay * Cost}$

مطابق با این تعریف، افزایش کارایی متناسب است با کاهش تاخیر و هزینه ساخت مدار.

جمع کننده‌ی انتخابی (Carry Select Adder):

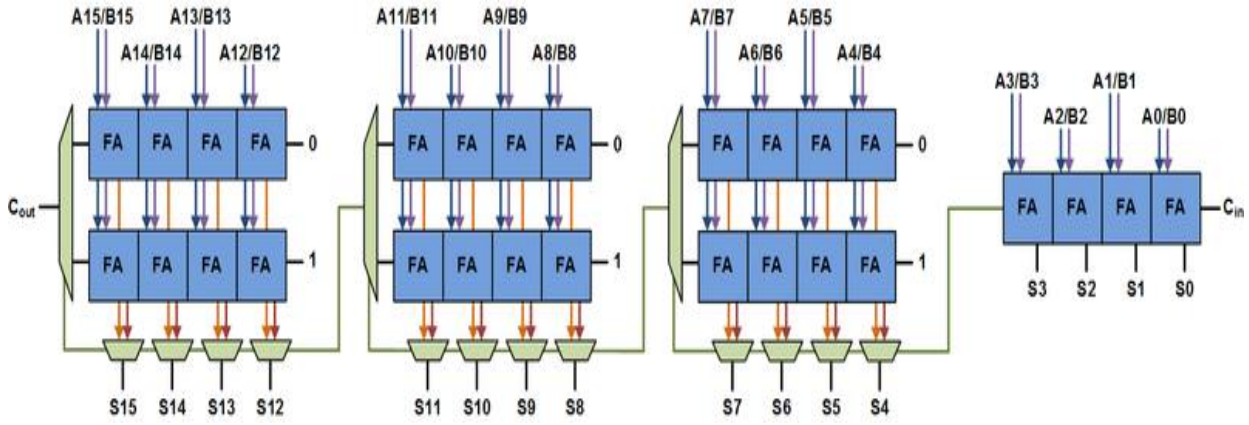
ایده‌ی اصلی طراحی CSA بر اساس این مشاهده است که در مقدار ورودی رقم نقلی فقط دو حالت صفر یا یک قرار خواهد گرفت. بنابراین زمان جمع کردن دو عدد n بیتی، کافی است به ازای هر ورودی یک بار جمع آبشاری با رقم نقلی ۰ و یک بار جمع آبشاری با رقم نقلی ۱ صورت گیرد. این دو محاسبه به طور موازی صورت می‌گیرند و پس از آن نتایج یکی از این دو جمع بر اساس رقم نقلی ورودی، به عنوان خروجی انتخاب می‌شود.

در زیر دو طراحی ساده بر اساس ایده‌ی CSA را مشاهده می‌کنید.



در شکل سمت چپ می‌توان به جای جمع کننده‌ی آبشاری از هر جمع کننده‌ی دیگری استفاده کرد. اما به هر حال با این شیوه طراحی تأخیر بیشتر از حالت ساده خواهد شد. (به دلیل وجود mux). علاوه بر این هزینه‌ی سخت افزاری هم افزایش پیدا کرده است. بنابراین طراحی ارائه شده هیچ مزیتی ندارد.

اما اگر به جای یک CSA از چند CSA که به هم به شکل آبشاری متصل شده اند استفاده کنیم، می‌توانیم تاخیر محاسبات را در شرایط خاص کاهش دهیم. به طور مثال اگر بلوک‌های ۴ بیتی CSA را به صورت آبشاری استفاده کنیم مدار زیر بدست می‌آید.



به این شـیوه که از بلوک‌هایی با تعداد بیت مسـاوی در طراحی CSA اسـتفاده شـود، Uniform Carry Select Adder می‌گویند. تاخیر این نوع طراحی CSA به صورت زیر است:

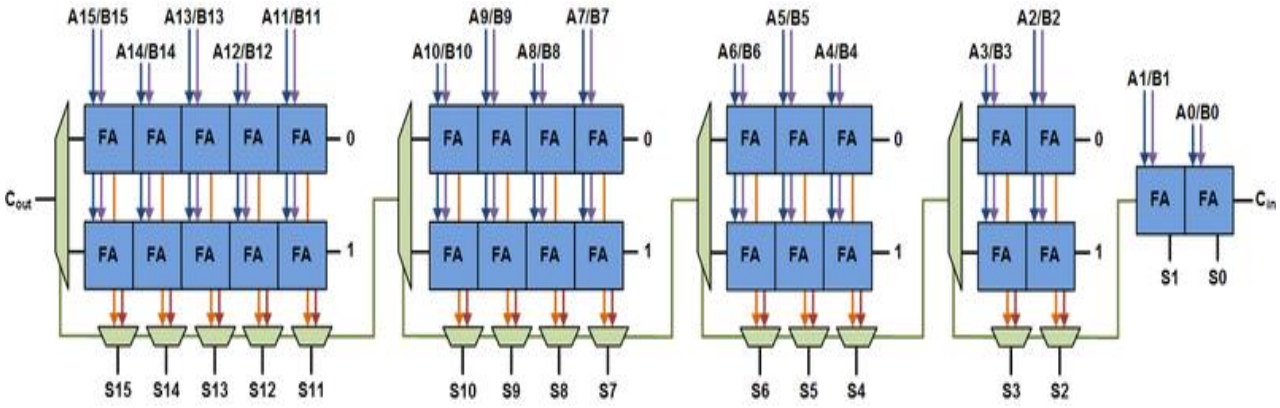
$$Delay = \left(\frac{n}{k}\right) 2d + 3kd$$

که k برابر است با تعداد سطح‌های مدار (در شکل بالا k=4). 3kd تاخیر به دلیل وجود mux هاست و همچنین 2d به دلیل تاخیر Full Adder های به کار رفته است. دقت کنید که انتخاب صحیح k می‌تواند شرایط مختلفی در تاخیر مدار ایجاد کند. به طور مثال اگر k=1، تاخیر برابر 2nd+3d می شود که از تاخیر جمع کننده‌آبشاری (2nd) بیشتر خواهد بود. اگر بخواهیم از جمع کننده آبشاری بهتر باشد خواهیم داشت:

$$\left(\frac{n}{k}\right) 2d + 3kd < 2nd \;\rightarrow 2nd + 3k^2 d < 2nkd \;\rightarrow (-3)k^2 + (2n)k - 2n > 0$$

بنابراین کافی است که نامعادله‌ی فوق را برای بدست آوردن k مناسب حل کنیم، به طوری‌که تاخیر minimum شود.

از معایب این روش این اســت که بلوک (جمع کننده)های آخر مدت زیادی منتظر می‌مانند. یکی از کارهایی که برای افزایش کارایی می‌توان انجام داد، این اســت که تعداد بیت بیشــتری برای جمع کردن به آن‌ها بدهیم. با همین ایده Non-Uniform Carry Select Adder طراحی شــد. به این ترتیب جمع کننده‌ی بهتری خواهیم داشت.



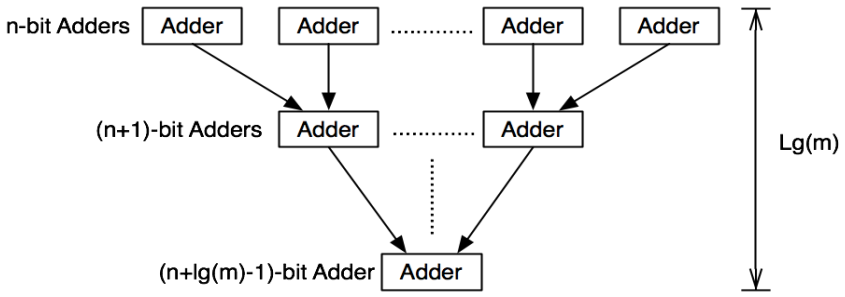
تأخیر در این روش کمتر می‌شود زیرا از تاخیر در مدارهای پایانی، برای جمع کردن ارقام بیشتر استفاده می‌شود و همچنین پهنای آن‌ها کوچکتر می‌شـــود و توان مصـــرفی نیز کاهش پیدا می‌کند. نســـبت به طراحی قبلی (uniform carry select adder) کارآیی بهتری دارد ولی از نظر سخت افزاری تفاوت چندانی ندارند.

Carry Save Adder

فرض کنیم که قصـد داریم m عدد n بیتی را جمع کنیم. اولین روشـی که به ذهن می‌رسـد این اسـت که یک ماتریس m*n رقمی تشکیل دهیم و سطر به سطر جمع کنیم (با جمع کننده‌های آبشاری). در این روش حداقل تعداد جمع کننده‌های آبشاری m-1 تا خواهد بود (بین هر سطر) و هر کدام تأخیری برابر 2*nd خواهند داشت. بدین ترتیب تأخیر کل آن‌ها بسیار زیاد و به شکل زیر خواهد شد:

$$delay = (m-1) * 2nd = \ 2mnd$$

برای بهتر کردن این روش می‌توان جمع کننده‌ها را به صورت درختی قرار داد و به این شکل تعداد طبقات لازم را کاهش داد اما همچنان تأخیر زیادی خواهیم داشت.



$$delay = 2nd * [\log_2 m]$$

$$HW = m * 5ng$$

اما روش بهتر استفاده از جمع کننده‌ی دیگری به نام Carry Save Adder است. ابتدا بایستی به این نکته توجه کنید که یک Full Adder نقش یک جمع کننده‌ی ســـه تایی را ایفا می‌کند (دو ورودی عددی و یک cin). در این روش هر بار به کمک Full Adder ها دســـته‌های ســـه تایی از اعداد را به دســـته‌های دو تایی تبدیل می‌کنیم(C_{out}و حا صل جمع) و این عمل را تکرار می کنیم تا زمانی که به د سته های ۲تایی بر سیم که در این مرحله با یک جمع کننده‌ی دیگر مانند جمع کننده‌ی آبشاری عملیات جمع به پایان می‌رسد.