فصل اول

مقدمه

۱-۱- مقدمه

امروزه پیشرفت در تکنولوژی کامپیوترها، تقریباً همهٔ جنبه‌های جامعه را تحت تأثیر قرار داده است. پیشرفت‌های صورت گرفته در سخت‌افزار، به برنامه‌نویسان اجازه داده است که برنامه‌های واقعاً مفیدی ایجاد کنند. برنامه‌نویسان موفق همیشه به سرعت اجرای برنامه‌های خود توجه می‌کنند چون دادن نتیجه سریع به کاربر در موفقیت یک برنامه نقش حیاتی دارد. در دهه‌های ۱۹۶۰ و۱۹۷۰ اصلی‌ترین محدودیت در سرعت کامپیوترها اندازه حافظه کامپیوترها بود. بعدها پیشرفت‌های ایجاد شده در زمینهٔ طراحی کامپیوترها و تکنولوژی حافظه‌ها واقعاً اهمیت مسأله حافظه کم را کاهش داد. امروزه برنامه-نویسانی که به سرعت اجرای برنامه‌های خود علاقه دارند، باید به مسائلی توجه کنند که جایگزین مسأله کمبود حافظه شده‌اند. از جمله این مسائل می‌توان به ساختار سلسله مراتبی حافظه‌ها و موازات در پردازنده‌ها اشاره نمود.

برنامه‌نویسانی که در جستجوی راههایی هستند که کامپایلرها، سیستم عاملها، برنامه‌های پایگاه داده و حتی application هایی بسازند که قابل رقابت با محصول دیگران باشد، باید دانش خود را در ارتباط با ساختار وسازمان کامپیوتر افزایش دهند. ما در این کتاب توضیح خواهیم داد که داخل یک کامپیوتر چه چیزهایی وجود دارد و شما را با رموز برنامه‌نویسی آشنا خواهیم ساخت. همچنین توضیح داده خواهد شد که ساختار داخلی یک کامپیوتر چگونه بر کارآیی برنامه‌ها تأثیر می‌گذارد. مهمتر از همه اینکه شما یاد خواهید گرفت که چگونه یک کامپیوتر برای خود بسازید.

فصل اول زیر بنای این کتاب است. این بخش به مطالب اساسی و تعاریف می‌پردازد. همچنین بخش-های اصلی نرم‌افزار و سخت‌افزار را بیان می‌نماید. در این بخش مقدمه‌ای در مورد مدارهای فشرده (Integrated circuit) که تکنولوژی مؤثر در پیشرفت کامپیوترها بوده، خواهد آمد.

۱-۲- لایه‌های زیرین برنامه

برای اینکه واقعاً بتوانید با یک ماشین الکترونیکی مانند کامپیوتر ارتباط برقرار کنید باید از علامتهای الکترونیکی استفاده کنید. ساده‌ترین علائم قابل فهم برای این ماشین الکترونیکی، روشن(on) و خاموش (off) می‌باشد و بنابراین الفبای کامپیوتر دارای دو حرف می‌باشد. همان‌طور که ۲۶ حرف موجود در الفبای زبان انگلیسی محدودیتی بر مقدار مطالبی که نوشته می‌شود نمی‌گذارد، الفبای دو حرفی کامپیوتر نیز کارهایی که یک کامپیوتر انجام می‌دهد را محدود نمی‌کند. دو سمبلی که برای این دو حروف استفاده می‌شود ۰ و ۱ است و ما معمولاً زبان کامپیوتر را به صورت عددهایی در مبنای ۲ یا اعداد دودویی می‌شناسیم. به هر کدام از حروف (۰ و ۱) کامپیوتر یک رقم باینری یا بیت گفته می‌شود.

کامپیوترها فقط با فرمانهای ما کار می‌کنند، اسم هر فرمان تکی دستور (instruction) نامیده می‌شود. دستورات که مجموعه‌ای از بیت‌ها (۰ و ۱) می‌باشند که یک کامپیوتر می‌فهمد، می‌توانند به صورت اعداد در نظر گرفته شوند. به طور مثال مجموعهٔ بیت زیر را در نظر بگیرید:

۱۰۰۰۱۱۰۰۱۰۱۰۰۰۰۰

این مجموعه بیت یک دستور می‌باشد که می‌تواند به یک کامپیوتر بگوید که دو عدد را با هم جمع کند. در فصل ۳ توضیح داده خواهد شد که چرا ما برای دستورات و داده‌ها از اعداد استفاده می‌کنیم. استفاده از اعداد برای دستورات و داده‌ها پایه و اساس محاسبات است بنابراین ما فصل ۳ را از دست نخواهیم داد!

اولین برنامه‌نویس‌های کامپیوتر، از طریق اعداد باینری با آن ارتباط برقرار می‌کردند یعنی به زبان ماشین کدنویسی می‌کردند. برنامه نویسی به زبان ماشین کار سختی بود و به همین دلیل به سرعت یک مجموعه علامتها اختراع شدند که به روش فکر کردن انسانها نزدیک بودند و برنامه نویسان از طریق این مجموعه علامت‌ها برنامه نوشتند. اسمی که برای این زبان سمبلیک استفاده می‌شد، امروزه نیز کاربرد دارد و آن زبان اسمبلی است. در ابتدا برنامه نویسان کد نوشته شده با این علائم را به طور دستی به مجموعه ۰ و ۱ ها (زبان ماشین) ترجمه می‌کردند. ترجمه دستی نیز کار مشکلی بود. راهکار بعدی این بود که از خود کامپیوتر برای عمل ترجمه کمک بگیریم. برنامه‌نویسان اولیه برنامه‌هایی را جهت ترجمه از علامتهای نمادین به مجموعهٔ ۰ و ۱ ها نوشتند و این برنامه‌ها را اسمبلر (assembler) نامگذاری کردند. اسمبلرها برنامه‌هایی بودند که یک برنامه را که به زبان اسمبلی نوشته شده بود در ورودی خود دریافت می‌کردند و در خروجی خود مجموعه‌ای از ۰ و ۱ ها تولید می‌کردند.

به طور مثال برنامه‌نویس می‌تواند دستوری را به صورت زیر بنویسد:

Add A, B

و اسمبلر می‌تواند این دستور را به صورت زیر ترجمه نماید:

۱۰۰۰۱۱۰۰۱۰۱۰۰۰۰۰

این دستور به کامپیوتر می‌گوید که دو عدد A و B را با هم جمع کند.

با وجود ایجاد زبان اسمبلی، برنامه‌نویسی هنوز هم کار سختی بود. در زبان اسمبلی برنامه‌نویس باید برای هر دستور زبان ماشین که کامپیوتر می‌تواند آن را انجام دهد یک دستور اسمبلی بنویسد و این باعث می‌شود که برنامه‌نویس مانند یک ماشین فکر کند.

در اینجا یک سؤال ساده مطرح می‌شود:

اگر ما توانستیم یک برنامه‌ای بنویسیم که عمل ترجمه از زبان اسمبلی به زبان ماشین را انجام دهد، چه عاملی باعث می‌شود که نتوانیم برنامه‌ای بنویسیم که عمل ترجمه از نمادهای سطح بالاتر به زبان اسمبلی را انجام دهد.

جواب این است که هیچ چیز.

برنامه‌هایی که این نمادهای سطح بالاتر را قبول می‌کنند، کامپایلر نامیده می‌شوند و زبانهایی که آنها کامپایل می‌کنند، زبانهای برنامه‌نویسی سطح بالا نامیده می‌شوند. کامپایلرها یک برنامه‌نویس را قادر می-سازند که عبارتی مانند عبارت زبان سطح بالای زیر را در برنامه‌های خود بنویسند.

A+B

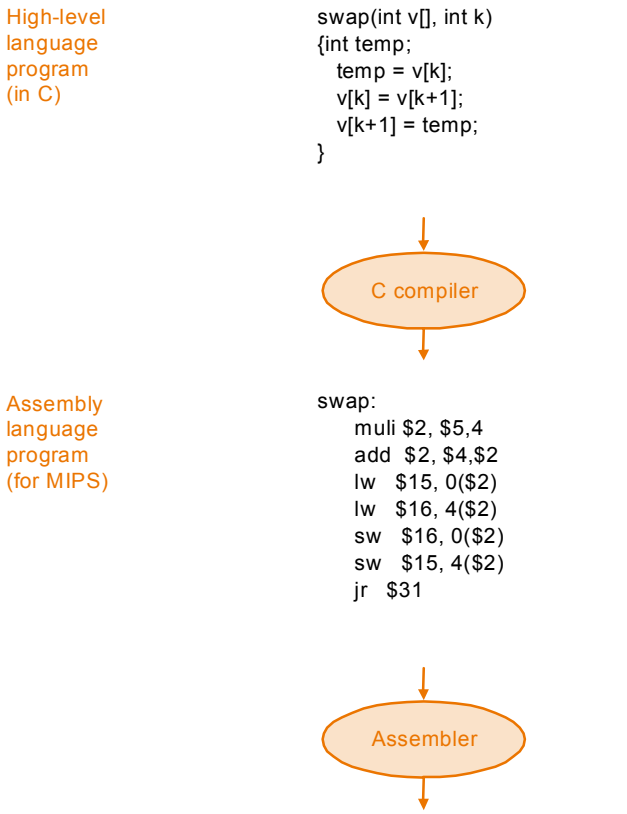
یک کامپایلر ممکن است عبارت فوق را به عبارت زبان اسمبلی زیر کامپایل (ترجمه) نماید.

Add A, B

و یک اسمبلر ممکن است عبارت add A, B را به یک دستور باینری به صورت زیر ترجمه نماید که این دستور به کامپایلر بگوید دو عدد A وB را با هم جمع نماید.

۱۰۰۰۱۱۰۰۱۰۱۰۰۰۰۰

شکل ۱-۱ ارتباط بین این برنامه‌ها و زبانها را نشان می‌دهد.



شکل ۱–۱ ارتباط بین زبانهای برنامه‌نویسی و مبدل‌های آنها.

زبانهای برنامه‌نویسی سطح بالا چندین مزیت مهم دارند. اول اینکه آنها به برنامه‌نویس قابلیت این را می‌دهند که در یک زبان خیلی نزدیک به زبان طبیعی فکر کنند. در این زبانها معمولاً برنامه‌نویس از کلمات و علامت‌های ریاضی زبان انگلیسی استفاده می‌کند. بعلاوه آنها اجازه می‌دهند زبان‌هایی متناظر با کاربرد مورد نظر طراحی شوند. به طور مثال فرترن (Fortran) برای محاسبات ریاضی، کوبول (Cobol) برای پردازش داده‌ها تجاری، لیسپ (Lisp) برای دستکاری سمبل‌ها و ... طراحی شده‌اند.

دومین مزیت زبانهای برنامه‌نویسی سطح بالا افزایش تولیدات برنامه‌نویس است. در زبان‌های سطح بالا حجم کد کم است و تعداد زیادی کار را می‌توان در تعداد کمتری از خط‌های برنامه انجام داد این امر مزیت بزرگی نسبت به زبان اسمبلی که در آن حجم برنامه بسیار بزرگ می‌شود، می‌باشد.

آخرین مزیت این است که زبانهای برنامه‌سازی اجازه می‌دهند که برنامه‌ها از کامپیوتری که بر روی آن طراحی می‌شوند مستقل باشند. این به این دلیل است که کامپایلرها و اسمبلرها می‌توانند برنامه نوشته شده را به زبان ماشین کامپیوتری که می‌خواهد آن را اجرا کند تبدیل می‌کنند.

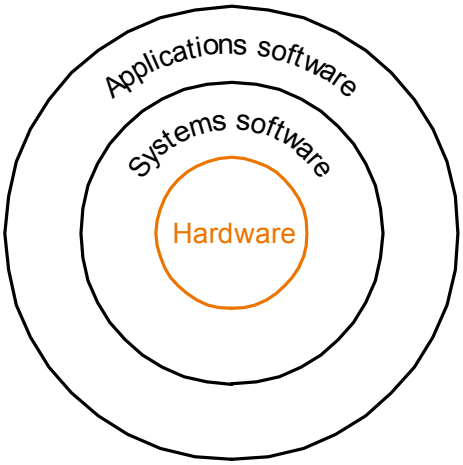
این سه مزیت آنقدر اهمیت دارند که امروزه تعداد برنامه‌های بسیار کمی با زبان اسمبلی نوشته می‌شود. با پیشرفت برنامه‌نویسی، برنامه‌نویسان مشاهده کردند که استفاده مجدد از برنامه‌ها، از اینکه همه چیز را از صفر شروع کنیم خیلی بهتر است. بنابراین برنامه‌نویسان این کار را شروع کردند که روتین‌های پراستفاده را در داخل کتابخانه‌ها (Libraries) جمع کنند. یکی از اولین کتابخانه‌های روتین‌ها، برای خواندن و نوشتن داده بود که به طور مثال شامل روتین‌هایی نظیر روتین‌های کنترل پرینترها بودند که به طور مثال موجود بودن کاغذ داخل پرینتر را قبل از عملیات پرینت کردن چک می‌کنند.

به زودی معلوم گردید که اگر یک برنامه‌ای وجود داشته باشد که اجرا شدن بقیه برنامه‌ها را مدیریت کند یک مجموعه از برنامه‌ها می‌توانند به طور مؤثری اجرا شوند، به محض اینکه اجرای یک برنامه به پایان رسید، برنامهٔ ناظر، برنامهٔ دیگری را از صف برنامه‌های منتظر اجرا، برای اجرا انتخاب می‌کند و بنابراین مانع ایجاد تأخیر می‌شود.

این برنامه ناظر که به زودی کتابخانه روتین‌های ورودی/خروجی نیز به آن اضافه شد، پایه و اساس چیزی بود که امروزه به آن سیستم عامل می‌گوییم. سیستم‌های عامل برنامه‌هایی هستند که منابع یک کامپیوتر را مدیریت می‌کنند تا اینکه برنامه‌های دیگر به طور مؤثری بر روی آن کامپیوتر اجرا شوند.

بعدها نرم‌افزارها بر اساس استفاده آنها دسته‌بندی شدند. نرم‌افزارهایی که معمولاً سرویس‌های مفید در اختیار کاربران قرار می‌دهند، نرم‌افزارهای سیستم(System software) نامیده می‌شوند. سیستم‌عامل‌ها، کامپایلرها و اسمبلرها مثالهایی از نرم‌افزارهای سیستم می‌باشند. در مقابل نرم‌افزارهای سیستمی، نرم-افزارهای کاربردی (applications software) یا همان برنامه‌های کاربردی (application) قرار دارند که نامی است برای برنامه‌هایی که هدفشان کمک کردن به استفاده کننده‌های کامپیوتر است. به طور مثال از این دسته برنامه‌ها می‌توانیم به ادیتورهای متن اشاره کنیم.

شکل ۱-۲ لایه‌های سلسله مراتبی نرم‌افزار و جایگاه آنها نسبت به سخت‌افزار را نشان می‌دهد.



شکل ۱-۲: لایه‌های سلسله مراتبی نرم‌افزار و ارتباط آن با سخت‌افزار.

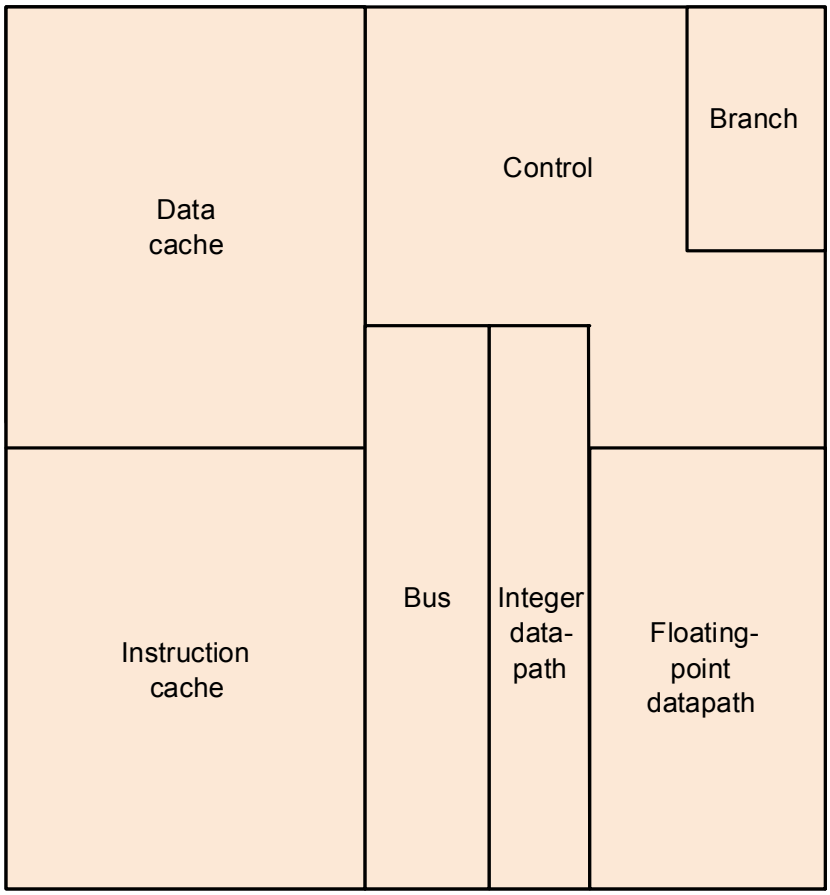
۱-۳- لایه‌های زیرین سخت‌افزار

در بخش قبل ما برای فهمیدن قسمتهای زیرین یک نرم‌افزار به آنچه که در زیر برنامه شما قرار دارد نگاه انداختیم، حال می‌خواهیم چنین کاری را در رابطه با سخت‌افزار انجام دهیم. اگر یک کامپیوتر رومیزی را در نظر بگیریم می‌بینیم که دارای کیبورد، موس، صفحه نمایش و یک کیس که سخت-افزارهای زیادی را در داخل خود نگه‌داری می‌کند، می‌باشد. این کامپیوتر معمولاً دارای یک اتصال شبکه نیز هست که ارتباط کامپیوتر را با کامپیوترهای دیگر، پرینتر و دیسکها برقرار می‌کند. دو ماجول اصلی کامپیوترها قطعات ورودی نظیر کیبورد و قطعات خروجی نظیر صفحه نمایش و پرینترها می-باشند. همان طور که از نامشان پیدا است، با استفاده از ورودی، کامپیوتر تغذیه می‌شود و داده‌ها در اختیار آن قرار می‌گیرد و با استفاده از خروجی‌ها نتایج حاصل از محاسبات کامپیوتر از آن خارج شده و در اختیار کاربر قرار می‌گیرد. در فصل ۸ وسایل ورودی و خروجی با جزئیات بیشتری شرح داده خواهند شد.

اجازه دهید تا یک مرور مقدماتی به سخت‌افزار کامپیوتر داشته باشیم. وقتی که ما کیس یک کامپیوتر را باز می‌کنیم، یک بوردی (board) را مشاهده می‌کنیم که با پلاستیک نازک سبز رنگی پوشیده شده است و بر روی آن تعداد زیادی مستطیل‌های کوچک سیاه و خاکستری قرار دارند. این بورد، مادربورد (mother board) نام دارد. مستطیل‌های کوچک روی مادربورد IC (Integrated circuit) یا چیپ نامیده می‌شوند. این بورد دارای سه قسمت مهم است، قسمتی که به قطعات ID (مثل درایو فلاپی و دیسک سخت) وصل می‌شود، حافظه و پردازنده. تعدادی بورد دیگر نیز به مادربورد وصل می‌شوند که از آن جمله می‌توان به کارت شبکه و کارت گرافیکی اشاره نمود.

حافظه محلی است که برنامه‌ها به هنگام اجرا بر روی آن قرار می‌گیرند. همچنین حافظه داده‌های مورد نیاز برنامه در حال اجرا را نیز نگهداری می‌کند. حافظه معمولاً بر روی دو بورد کوچک که تقریباً در وسط مادربورد نصب می‌شوند، قرار می‌گیرد.

پردازنده قسمت فعال بورد است که دستورات برنامه را اجرا می‌کند. پردازنده اعداد را جمع می‌کند، تست می‌کند، به قطعات IO علامت می‌دهد تا فعال شوند و ... . پردازنده به صورت یک مربع بزرگ بر روی مادربورد قرار دارد. معمولاً پردازنده را به اسم CPU می‌شناسیم که همان واحد پردازش مرکزی Central Processing Unit می‌باشد. شکل ۱-۳ محتویات داخلی یک CPU را نشان می‌دهد.



شکل ۱-۳: محتویات داخلی CPU

پردازنده از دو بخش اصلی تشکیل می‌شود:

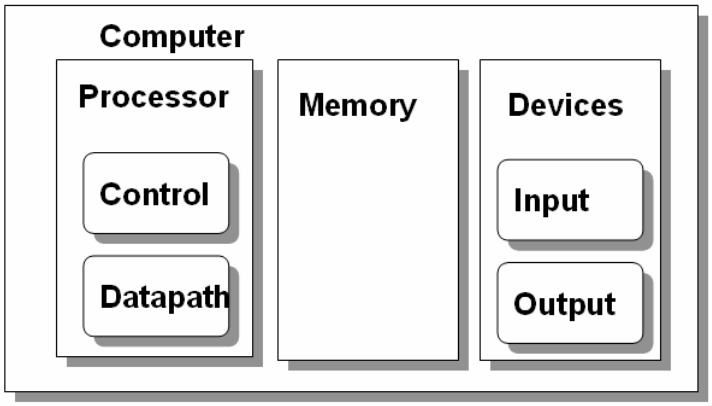
بخشdata path و بخش کنترل (Control) که این دو قدرت و تفکر کامپیوتر را می‌سازند.data path عملیات ریاضی را انجام می‌دهد و بخش کنترل به data path، حافظه و قطعات IO دستور می‌دهد که

چه کاری انجام دهند. بر اساس اینکه دستوری که می‌خواهد انجام شود، چه عملیاتی می‌خواهد انجام دهد، واحد کنترل وظیفهٔ سایر بخش‌ها را مشخص می‌کند.

فصل ۵ در مورد بخش‌های data path و کنترل صحبت خواهد کرد و فصل ۶ در مورد تغییراتی که باید در طراحی این دو بخش صورت بگیرد تا به سرعت بالاتری برسیم صحبت خواهد نمود.

شکل ۱-۳ مربوط به پردازنده پنتیوم اینتل می‌باشد. حجم سیلیکون استفاده شده برای این پردازنده ۹۱ میلی متر مربع بوده و در ساخت آن حدود ۳/۳ میلیون ترانزیستور استفاده شده است. حافظه کش cache این پردازنده تقریباً یک میلیون عدد از ۳ میلیون ترانزیستور را به خود اختصاص داده است. فصل ۷ توضیح خواهد داد که چرا کش منابع زیادی را مصرف می‌کند. قسمتهای دیگر چیپ در فصل-های بعدی توضیح داده خواهد شد. مدار پیش بینی دستور انشعاب (branch prediction) در فصل ۶ و گذرگاه سیستم Bus در فصل ۸ پوشش داده می‌شوند.

نکته مهم: ۵ بخش کلاسیک یک کامپیوتر عبارتند از ورودی، خروجی، حافظهdata path و بخش کنترل که این دو بخش آخری معمولاً ترکیب شده و پردازنده (processor) نامیده می‌شود. شما می-توانید هر بخشی ازهر کامپیوتری را (چه کامپیوترهای قدیمی و چه کامپیوترهای فعلی) در یکی از این دسته‌ها قرار دهید. این پنج بخش در شکل ۱-۴ نشان داده شده‌اند.



شکل ۱-۴: پنج بخش کلاسیک یک کامپیوتر

پردازنده دستورات و داده‌ها را از حافظه دریافت می‌نماید؛ ورودی داده‌ها را در داخل حافظه می‌نویسد و خروجی داده‌ها را از حافظه می‌خواند. بخش کنترل، سیگنالهایی که عملیات data path، حافظه، ورودی و خروجی را مشخص می‌کنند، را تولید می‌نماید.

بر روی مادبورد معمولاً حافظه‌های نوع DRAM قرار دارد. DRAM علامت اختصاری عبارتDynamic Random Access Memory می‌باشد. برای نگهداری داده‌ها و دستورات یک برنامه

چندین DRAM در کنار یکدیگر استفاده می‌شوند. در مقایسه با حافظه‌های با قابلیت دسترسی ترتیبی مانند نوارهای مغناطیسی که زمان دستیابی به قسمت‌های مختلف با هم فرق دارد، قسمت RAM از عبارت DRAM بیانگر این مطلب است که دسترسی به قسمت‌های مختلف این حافظه به یک اندازه طول می‌کشد. حافظهٔ DRAM مانند یک بافر بزرگ یا یک آرایه بزرگ برای ذخیره داده‌ها مورد استفاده قرار می‌گیرد.

۱-۳-۱- معماری مجموعه دستورات

نکته‌ای که در اینجا باید به آن اشاره شود این است که ما هر چقدر وارد جزئیات سخت‌افزار و نرم‌افزار نشویم، به مدلهای سطح بالای ساده‌ای از سخت‌افزار و نرم‌افزار خواهیم رسید. استفاده از این روش (مدلهای لایه‌ای) یک تکنیک اساسی برای طراحی سیستم‌های کامپیوتری بسیار هوشمند می‌باشد. یکی از مهمترین سطوح تجرد[1]، واسط بین سخت‌افزار و نرم‌افزار سطح پایین[2] می‌باشد. به دلیل اهمیت این موضوع یک اسم ویژه به آن اختصاص داده شده است. معماری مجموعه دستورات (ISA[3]) ، یا به طور ساده معماری یک ماشین. ISA شامل همه چیزهایی که برنامه‌نویسان برای نوشتن برنامه‌های باینری زبان ماشین به آنها نیاز دارند می‌باشد. این نیازمندی‌ها شامل دستورات، قطعات I/O و غیره می‌باشد. اجزای مختلف یک معماری در فصل‌های ۳ ، ۴ ، ۷ و ۸ بحث خواهد شد. این واسط استاندارد به طراحان کامپیوتر اجازه می‌دهد تا مستقل از سخت‌افزار در مورد عملکردهای یک کامپیوتر صحبت کنند. به طور مثال ما می‌توانیم در مورد عملکرد یک ساعت دیجیتال (مانند نمایش زمان، زنگ اخطار و ....) مستقل از اینکه سخت‌افزار آن چیست (صفحه نمایش، دکمه‌های پلاستیکی، چرخ دنده‌ها و ...) صحبت کنیم. به همین دلیل طراحان کامپیوتر بین یک پیاده‌سازی[4] و یک معماری تفاوت قائل می‌شوند. یک پیاده‌سازی سخت‌افزاری است که سطوح تجرد معماری را فراهم می‌کند.

نکته مهم: نرم‌افزار و سخت‌افزار هر دو شامل لایه‌های سلسله مراتبی هستند، که لایه‌های زیرین جزئیات را از لایه‌های بالاتر مخفی نگه می‌دارند. پایه و اساس سطح تجرد این است که طراحان سخت‌افزار و نرم‌افزار پیچیدگی طراحی سیستم‌های کامپیوتری را کاهش دهند. یکی از واسط‌های کلیدی بین سطوح تجرد ISA می‌باشد: واسط بین سخت‌افزار و نرم‌افزار سطح پایین. این واسط امکان

---

1 - Abstract
2 - Low level software
Instruction Set Architecture-3
4 - Implementation

ایجاد پیاده‌سازی‌های مختلف با قیمت و سرعت متفاوت که نرم‌افزار یکسانی را اجرا می‌کنند در اختیار قرار می‌دهد.

استفاده‌کننده‌های macintash اثر تغییر ISA را درک می‌کنند. برنامه‌هایی که برای معماری Power PC طراحی شده‌اند بر روی ماشین‌هایی که بر اساس 6800 ساخته شده‌اند اجرا نمی‌شوند، و برنامه‌هایی مبتنی بر 6800 ، به درستی بر روی Power PC اجرا نمی‌شوند. در مقابل خانواده 86*80 شرکت اینتل پیاده‌سازی‌های مختلفی از یک معماری را در اختیار قرار می‌دهند. برنامه‌های نوشته شده برای 8086 اولیه در سال ۱۹۷۸ می‌توانند بر روی Pentium Pro اجرا شوند. Intel در طی سالیان قابلیت‌های زیادی را به سیستم‌های قبلی افزوده است. اما همهٔ پردازنده‌های نسل بعدی برنامه‌های نوشته شده بر روی پردازنده‌های نسل‌های قبلی را اجرا می‌کنند.

۱-۳-۲- محل مطمئن برای نگهداری داده‌ها

اگر تغذیه (برق) کامپیوتر قطع شود، همه چیز از بین می‌رود و این به دلیل این است که حافظه‌های داخل کامپیوتر فرار هستند و با قطع برق اطلاعات آنها از بین می‌رود. در مقام مقایسه یک نوار کاست با قطع برق اطلاعات خود زا از دست نمی‌دهد چون تکنولوژی آنها با حافظه کامپیوتر فرق می‌کند. در نوار کاست از قطعات مغناطیسی استفاده می‌شود.

کامپیوتر معمولاً دارای دو نوع حافظه است: حافظه اصلی و حافظه ثانوی

حافظه اصلی یا اولیه معمولاً از نوع DRAM بوده و اطلاعات آنها با قطع شدن برق از بین می‌رود ولی حافظه جانبی یا ثانویه از نوع مغناطیسی بوده و اطلاعات آنها با قطع شدن برق از بین نمی‌رود. بنابراین برای نگهداری داده‌ای که نباید از بین بروند از حافظه ثانویه استفاده می‌شود. به دلیل اینکه درحافظه-های ثانویه از قطعات مکانیکی استفاده می‌شود، سرعت آنها نسبت به DRAM ها خیلی پایین‌تر است. زمان دسترسی به داده درحافظه ثانویه (هارد دیسک) معمولا بین ۵ تا ۲۰ میلی ثانیه است در حالی که این زمان برای DRAM در حدود ۵۰ تا ۱۰۰ نانو ثانیه است. بنابراین DRAM حدود ۱۰۰۰۰۰ مرتبه سریعتر از هارد دیسک است.

هزینه تولید هارد دیسک‌ها نسبت به DRAM پایین‌تر است بنابراین قیمت آنها نسبت به DRAM کمتر است. در سال ۱۹۹۷ قیمت برای یک مگابایت از دیسک ۵۰ برابر کمتر از DRAM بود.

بنابراین دیسک‌های مغناطیسی و حافظه اصلی سه فرق عمده با هم دارند: دیسک‌ها غیر فرارند، به دلیل اینکه مغناطیسی هستند. سرعت دیسک‌ها کمتر است چون قطعات مکانیکی دارند و قیمت دیسک‌ها کمتر است چون هزینه تولید آنها در مقایسه با DRAM ها پایین‌تر است.

۱-۴- مدارهای مجتمع

تکنولوژی ساخت پردازنده‌ها و حافظه‌ها در طول سالیان همواره در حال پیشرفت بوده است و به همین دلیل پردازنده‌ها وحافظه‌ها همواره در حال بهبود بوده‌اند. جدول ۱-۱ تکنولوژی‌هایی را نشان می‌دهد که در سالهای مختلف مورد استفاده قرار گرفته‌اند. در این جدول یک تخمین از افزایش سرعت به ازای یک هزینه واحد آورده شده است.

جدول ۱-۱: افزایش سرعت در هزینه واحد برای تکنولوژیهای مختلف

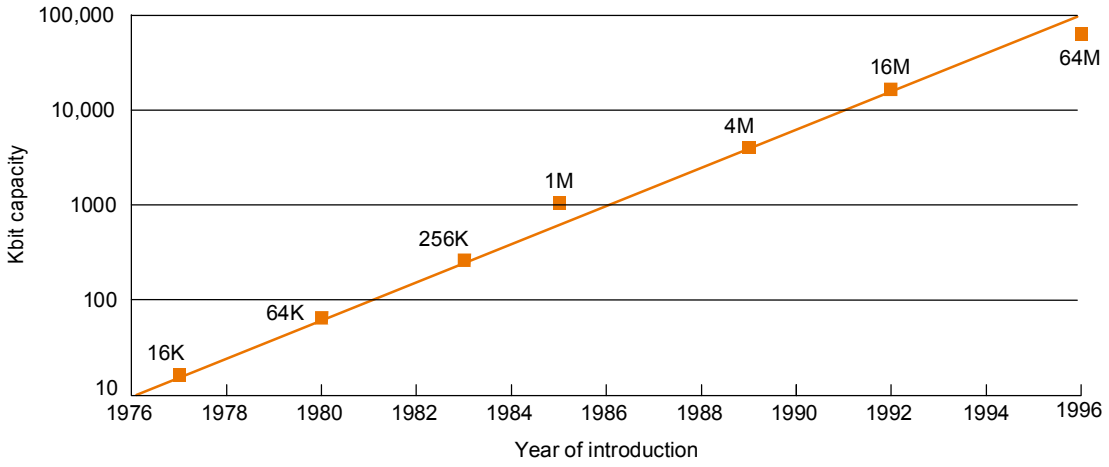
Year	Technology used in computers	Related Performance / unit cost
1951	Vacuum tub	1
1965	Transistor	35
1975	Integrated Circuit	900
1995	Very large-scale Integrated circuit	2,400,000

این بخش در مورد تکنولوژی که سرعت کامپیوتر را از سال ۱۹۷۵ تحت تأثیر قرار داده و در سالهای آتی نیز این اتفاق خواهد افتاد، صحبت خواهد نمود. چون تکنولوژی نشان می‌دهد که کامپیوترها چه کارهایی را می‌توانند انجام دهند و با چه سرعتی در حال پیشرفت می‌باشند، بنابراین ما اعتقاد داریم که همهٔ متخصصین کامپیوتر باید با اصول مدارهای مجتمع آشنا شوند.

یک ترانزیستور به زبان ساده یک سوئیچی است که دارای دو وضعیت روشن و خاموش (on و off) بوده و با جریان برق کنترل می‌شود. یک مدار مجتمع حدوداً ده‌ها تا صدها عدد از این ترانزیستورها را در داخل یک چیپ ترکیب می‌نماید برای نشان دادن نرخ افزایش سرسام‌آور تعداد ترانزیستورها از صدها عدد به میلیون‌ها عدد در داخل یک چیپ به آنها VLSI [1] گفته می‌شود.

شکل ۱-۵ میزان افزایش ظرفیت DRAM ها را از سال ۱۹۷۷ به بعد نشان می‌دهد. همان طور که مشاهده می‌شود ظرفیت DRAMها تقریبا هر سه سال، چهار برابر شده است که در نتیجه در طول بیست سال ظرفیت DRAM ها حدوداً ۱۶۰۰۰ برابر شده است! این میزان افزایش قابل توجه در سرعت و ظرفیت مدارهای مجتمع، طراحی سخت‌افزار و نرم‌افزار را تحت تأثیر قرار می‌دهد و همین عاملی است که فهمیدن مدارهای مجتمع را ضروری می‌سازد.

[1] - Very Large Scale Integrated



شکل ۱–۵: میزان افزایش ظرفیت DRAM ها در طول زمان. واحد محور y ها Kbits می‌باشد.

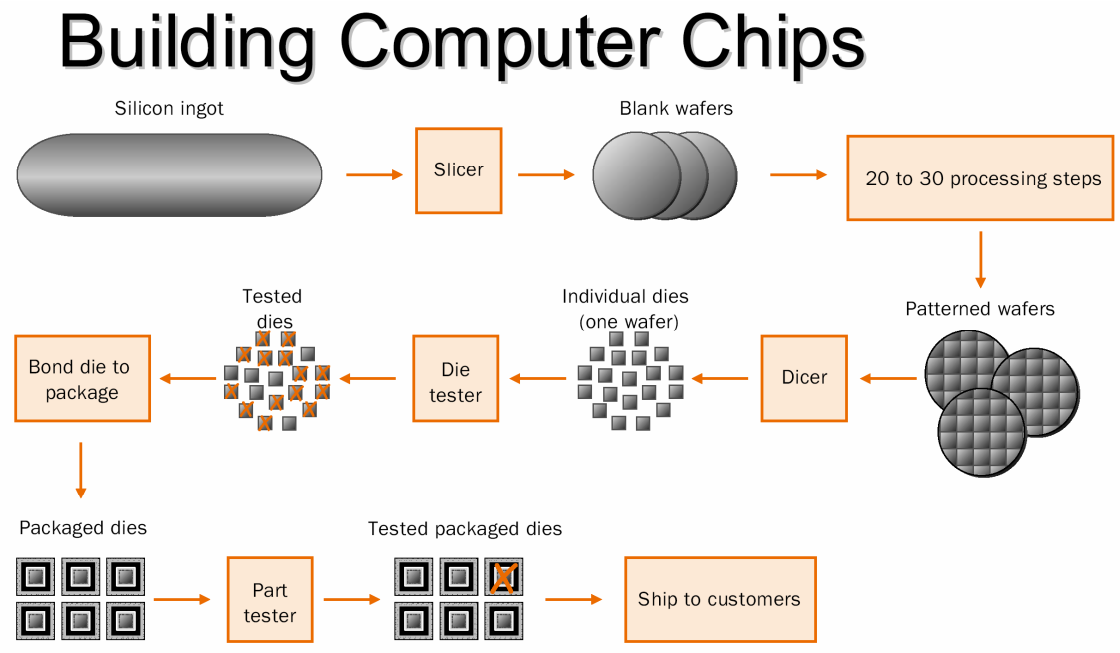
حال که ارتباط افزایش حجم حافظه‌ها را با تکنولوژی متوجه شدیم در این قسمت می‌خواهیم روند ساخته شدن یک چیپ را مورد بررسی قرار دهیم. ساختن یک چیپ از سیلیکون شروع می‌شود (سیلیکون ماده‌ای است که در سنگ یافت می‌شود و جزو عناصر گروه چهارم جدول تناوبی مندلیف می‌باشد). چون سیلیکون جریان الکتریسیته را به خوبی عبور نمی‌دهد به آن عنصر نیمه هادی گفته می‌شود. با استفاده از یک پروسهٔ شیمیایی مخصوص این امکان وجود دارد که موادی به سیلیکون اضافه نمائیم تا در سطح آن محل‌هایی ایجاد کنیم که در آن محل‌ها بتوانیم یکی از سه قطعه زیر را بسازیم:

- مواد هادی جریان الکتریسیته (شبیه به مس و آلومینیوم)
- مواد عایق(مانند پلاستیک و شیشه)
- فضایی (area) که تحت شرایط خاصی می‌تواند هادی و یا عایق (مانند یک کلید) باشد

ترانزیستورها در طبقه آخر قرار می‌گیرند. یک مدار VLSI شامل میلیونها هادی، عایق و سوئیچ می‌باشد که بر روی یک فضای کوچک سیلیکونی ساخته می‌شوند. پروسهٔ ساخت مدارهای مجتمع بر روی هزینه چیپ ها تأثیر زیادی دارد و بنابراین برای طراحان کامپیوتر اهمیت زیادی دارد. شکل ۱–۶ این پروسه ساخت را نشان می‌دهد. این پروسه از یک شمش سیلیکونی شروع می‌شود. سپس یک شمش برش خورده و به قطعات کوچکتر و نازکتری به نام Wafer تقسیم می‌شود. این ویفرها سپس چند مرحله پروسه را طی می‌کنند، که در آن پروسه‌ها یک سری موار شیمیایی بر روی هر ویفر قرار می‌گیرد که ترانزیستورها، هادیها و عایق‌ها را ایجاد می‌کنند، بعد از طی کردن این مراحل ویفرها برش داده می‌شوند و به قطعات کوچکتری به نام die تبدیل می‌شوند. بعد از این مرحله dieهای خراب کنار گذاشته می‌شوند. تعداد die های سالم به کلdieهای یک ویفر yield نامیده می‌شود. بعد از تست die ها، هر کدام از dieهای سالم در داخل یک package، به پین‌های ورودی و خروجی وصل می‌شود این

پروسه bonding نامیده می‌شود. Dieهای packageشده دوباره تست می‌شوند چون ممکن است هنگام package کردن خرابی‌هایی اتفاق بیافتد. پس از تست شدن، pakage به مشتری فروخته می‌شود.

توجه: یکی از مسائل جدید در طراحی کامپیوتر توان تلفاتی می‌باشد. اهمیت توان اتلافی فقط برای کاربردهای قابل حمل (Protable) که طول عمر باتری در آنها مهم است (مانند کامپیوترهای کیفی) نیست بلکه برای کامپیوترهای رومیزی که نیز با افزایش کلاک اهمیت مسأله توان بیشتر می‌شود. پردازنده Alpha 21264 در فرکانس 600 مگاهرتز به طور شگفت‌انگیزی فقط 72 وات توان مصرف می‌کند. توان اتلافی یکی از مسائلی است که می‌تواند سرعت پردازنده‌ها را محدود کند.



شکل ۱-۶: پروسه ساخت چیپ

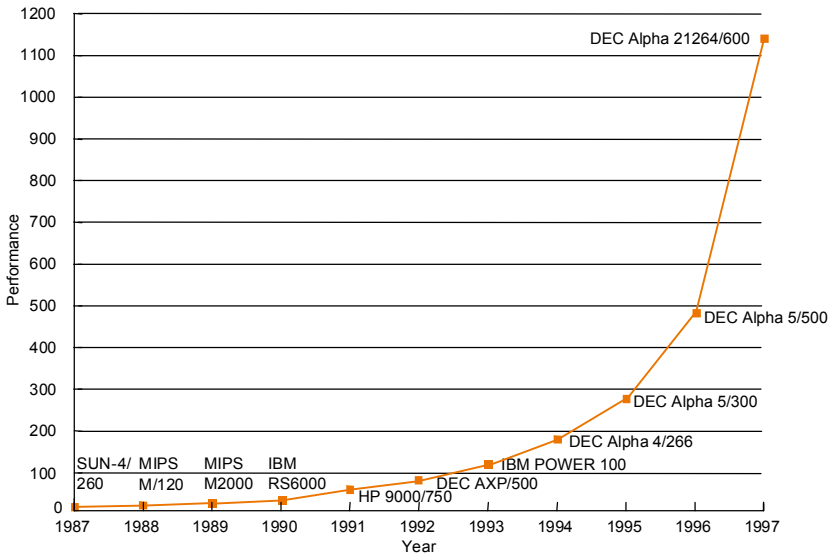
۱-۵- باورهای غلط (fallacies) و اشتباهات (Pitfalls)

هدف از داشتن یک بخش تحت این عنوان که در آخر هر فصل وجود دارد این است که مطالبی را توضیح دهد که ممکن است شما آنها را اشتباه متوجه شده باشید: ما این باور غلط را fallacy نامگذاری خواهیم کرد. وقتی که در مورد یک باور غلط صحبت می‌کنیم برای توضیح دادن آن یک مثال خواهیم آورد. همچنین ما در مورد اشتباهات (Pitfalls) صحبت خواهیم کرد. اغلب این اشتباهات

تعمیم دادن اصولی است که فقط در محدودهٔ کوچکتری درست می‌باشند. هدف از این بخش این است که به شما کمک کند تا جلوی اشتباهات خود را در ماشین‌هایی که استفاده و یا طراحی می‌کنید بگیرید.

اشتباه: چشم پوشی کردن از پیشرفت سخت‌افزار هنگامی که یک ماشین جدید را طراحی می‌کنیم اشتباه است.

فرض کنید که شما می‌خواهید یک ماشین را در مدت ۳ سال تولید کنید و ادعا دارید که این ماشین یک فروشنده فوق‌العاده خواهد بود چون سرعت آن ۳ برابر سرعت ماشین‌های دیگر که امروزه وجود دارند خواهد بود. متأسفانه ماشین شما احتمال دارد فروشنده ضعیفی باشد به دلیل اینکه افزایش سرعت برای صنعت (افزایش سرعت سخت‌افزار) ممکن است به ماشین‌هایی منجر شود که سرعتشان مثل سرعت ماشین شما باشد. به طور مثال فرض کنید ۵۰٪ رشد سالیانه سرعت داشته باشیم (به دلیل پیشرفت تکنولوژی ساخت سخت‌افزار)، در این صورت برای ماشینی که امروزه سرعت آن x می‌باشد می‌توان انتظار داشت که پس از ۳ سال سرعتی در حدود $1.5^3 x = 3.4x$ خواهد داشت. بنابراین ماشین شما مزیت سرعتی نخواهد داشت! بسیاری از پروژه‌ها در شرکتهای کامپیوتری به این دلیل تعطیل شدند که یا این قانون را در نظر نگرفتند (پیشرفت تکنولوژی ساخت سخت‌افزار) و یا اینکه دیرتر از موعد خودشان تمام شدند و سرعت ماشینی که با تأخیر به بازار آمد از میانگین سرعت صنعت پایین‌تر بود. این واقعیت ممکن است در همهٔ صنایع وجود داشته باشد ولی بهبود سریع هزینه بر سرعت (Cost / Performance) در صنعت کامپیوتر آن را به یک امر بسیار مهم تبدیل کرده است. شکل ۱–۷ میزان افزایش سرعت پردازنده‌ها در طول زمان را نشان می‌دهد.



شکل ۱–۷: افزایش سرعت کامپیوترها بین سالهای ۱۹۸۷ تا ۱۹۹۷

۱-۶- موضوع واقعی: ساختن چیپ‌های پنتیوم (Pentium)

در انتهای هر فصل این کتاب، بخشی تحت عنوان موضوع واقعی (Real stuff) وجود دارد که مطالب موجود در همان فصل را به کامپیوترهایی که در عمل از آنها استفاده می‌کنید، مرتبط می‌سازد. این بخش‌ها همیشه تکنولوژی‌هایی را که برای کامپیوترهای IBM PC مورد استفاده قرار می‌گیرد توضیح خواهند داد و اغلب تکنولوژی Apple Macintosh را نیز بحث خواهند کرد. برای این فصل ما مفاهیم مدارهای مجتمع را به چیپ‌هایی که در IBM PC ها استفاده می‌شوند، مرتبط خواهیم نمود.

۱-۷- صحبت پایانی

هر چند پیشگویی این امر مشکل است که واقعاً بهبود هزینه بر سرعت کامپیوترها در آینده چگونه خواهد بود، ولی آنها مطمئناً بهتر از کامپیوترهای فعلی خواهند بود. برای مشارکت در این پیشرفت، طراحان کامپیوتر و برنامه‌نویسان باید مسائل زیادی را بفهمند و یاد بگیرند.

هر جفت طراحان سخت‌افزار و نرم‌افزار سیستم‌های کامپیوتری را به صورت لایه‌ای طراحی می‌کنند که در آن لایه‌های پایین‌تر جزئیات را از دید لایه‌های بالاتر مخفی نگه می‌دارند. این اصول سطح تجرد برای فهمیدن سیستم‌های کامپیوتری، امروزه ضروری است. شاید مهمترین مثال از سطح تجرد، واسط بین سخت‌افزار و نرم‌افزار سطح پایین می‌باشد که ISA نامیده می‌شود. با فرض ثابت نگه داشتن ISA پیاده‌سازی‌های مختلفی از یک معماری با هزینه و سرعت متفاوت را ممکن می‌سازد. همه این پیاده-سازی‌ها نرم‌افزار یکسانی را اجرا می‌کنند. از طرف دیگر ممکن است این معماری مانع پیشرفت گردد و نیاز به این داشته باشد که واسط مورد نظر تغییر کند. تکنولوژی‌های کلیدی برای پردازنده‌های مدرن، سیلیکون و کامپایلر می‌باشند. به طور واضح شما باید ویژگیهای سیلیکون و کامپایلر را درک نمائید. چیزی که اهمیت آن کمتر از فهمیدن مفهوم مدارهای مجتمع نیست فهمیدن سرعت تغییرات مورد انتظار تکنولوژی‌های ساخت IC می‌باشد. یک مثال مرتبط با این مسأله این است که سرعت DRAM ها هر سه سال ۴ برابر می‌شود. در حالی که سیلیکون عامل مؤثری در پیشرفت سریع سخت‌افزار می-باشد، ایده‌های جدید در طراحی کامپیوترها، هزینه بر سرعت کامپیوترها را بهبود داده است. دو مورد از ایده‌های کلیدی، به کارگیری موازی سازی در کامپیوترها و سلسله مراتب حافظه می‌باشد. نوعاً موازی سازی از طریق پایپلاین و به کارگیری مفهوم دسترسی محلی به حافظه از طریق حافظه‌های کش می-باشد.

راهنمای فصول این کتاب

همان طور که توضیح داده شد پنج قسمت کلاسیکی برای کامپیوتر وجود دارد: datapath، بخش کنترل، حافظه، ورودی و خروجی این پنج قسمت به عنوان یک قالب کاری (frame work) برای قسمت‌های باقی مانده این کتاب مورد استفاده قرار می‌گیرد:

- مسیر داده (datapath): فصول ۴و ۵ و۶
- بخش کنترل: فصول ۵ و۶
- حافظه: فصل ۷
- ورودی: فصل ۸
- خروجی: فصل ۸

فصل ۶ توضیح خواهد داد که چگونه پایپلاین در پردازنده موازی‌سازی انجام می‌دهد و فصل ۷ در مورد اینکه چگونه سلسله مراتب حافظه خاصیت محلی بودن را بکار می‌گیرد، بحث خواهد نمود. بقیه فصل‌ها مقدمات و مباحث تکمیلی در مورد این موضوعات را مطرح می‌کنند. فصل ۲ در مورد سرعت پردازنده‌ها صحبت خواهد کرد و بنابراین توضیح می‌دهد که چگونه یک کامپیوتر را ارزیابی کنیم. فصل ۳ در مورد مجموعه دستورات که همان واسط بین کامپایلرها و ماشین است صحبت خواهد کرد و تأکید خواهد نمود که کامپایلرها و زبانهای برنامه‌نویسی چه نقشی در استفاده از خواص مجموعه دستورات خواهند داشت.

فصل ۹ این کتاب با بحثی در مورد سیستم های چند پردازنده‌ای این کتاب را به پایان خواهد برد.

۱-۸- چشم‌انداز تاریخی و مطالعه اضافی

در این کتاب در آخر هر فصل بخشی تحت این عنوان وجود دارد. در این بخش‌ها ما ممکن است توسعهٔ یک ایده را از طریق یک‌سری از ماشین‌ها بررسی کنیم یا بعضی از پروژه‌های مهم را توضیح دهیم. ما در هر مورد مراجع را معرفی خواهیم نمود و اگر علاقمند بودید می‌توانید به آنها مراجعه کنید و بیشتر آنها را ارزیابی کنید. در حالت کلی مطالب این بخش در مورد مطالب همان فصل مربوطه خواهد بود و مباحث تاریخی و تکمیلی را در مورد مطالب آن فصل بیان خواهد نمود.

۱-۹- کلمات و اصطلاحات کلیدی

لیستی از کلمات و اصطلاحات در آخر هر فصل و در پیوست آورده شده است. این کلمات ایده‌های کلیدی بحث شده در آن فصل و پیوست را منعکس می‌کند. اگر شما معنای کلمه و اصطلاح گفته شده

در این قسمت را هنوز نفهمیده‌اید، می‌توانید به واژگان کلمات در آخر کتاب مراجعه کنید. همهٔ کلمات و اصطلاحات کلیدی در این قسمت توضیح داده شده است.

فصل دوم

ارزیابی کارآیی پردازنده‌ها

۲-۱- مقدمه

این فصل توضیح خواهد داد که چگونه کارآیی[6] (سرعت) یک کامپیوتر را اندازه بگیریم و آن را گزارش کنیم. در این فصل سعی خواهیم نمود که به سؤالات زیر جواب دهیم:

- چرا کارآیی مهم است؟
- چگونه ما می‌توانیم کارآیی را به دقت تعریف کنیم؟
- چگونه طراحی سخت‌افزار کارآیی نرم‌افزار را تحت تأثیر قرار می‌دهد؟
- چگونه در دنیای واقعی کارآیی اندازه گرفته شود؟
- چرا بعضی از سخت‌افزارها بهتر از بقیه عمل می‌کنند؟
- کدام یک از فاکتورهای کارآیی به سخت‌افزار وابسته‌اند؟
- ما برای اجرا شدن سریع‌تر یک برنامه به یک ماشین جدید نیاز داریم یا نیازمند یک سیستم-عامل جدید هستیم؟
- مجموعه دستورات یک ماشین چگونه کارآیی را تحت تأثیر قرار می‌دهند؟

در این فصل عاملهای تعیین کننده در کارآیی کامپیوترها مورد بحث قرار خواهد گرفت. بررسی کردن کارآیی به این دلیل مهم است که کارآیی سخت‌افزار اغلب کلیدی‌ترین اثر را در کل سیستم متشکل از سخت‌افزار و نرم‌افزار برعهده دارد. ما معمولاً در این فصل کارآیی و سرعت را به جای هم استفاده می‌کنیم. ولی در بعضی از موارد این دو عبارت کاملاً معادل هم نیستند. به عنوان مثال اگر هدف ما استفاده از ماشینی باشد که تعداد کار بیشتری را در واحد زمان انجام دهد، ماشینی که سریعتر است بعضی مواقع نمی‌تواند هدف ما را برآورده کند و ما مجبور هستیم در این مواقع از ماشینی استفاده کنیم که قابلیت‌های بیشتری داشته باشد (کاراتر باشد). ما در این کتاب اغلب از اصطلاح کارآیی استفاده خواهیم کرد.

اظهار نظر کردن در مورد کارآیی یک سیستم واقعاً کار مشکلی است. سیستم‌های بزرگ نرم‌افزاری با جزئیات زیاد به همراه بازهٔ وسیعی از تکنیک‌های افزایش سرعت که توسط طراحان سخت‌افزار به کار گرفته می‌شوند، از جمله مواردی هستند که صحبت کردن دربارهٔ کارآیی را مشکل می‌نمایند.

تقریبا کار غیر ممکنی است که برگه راهنمایی از مجموعه دستورات یک کامپیوتر را بردارید و با یک برنامه مشخص، تعیین نمایید که آن کامپیوتر برنامه شما را چقدر سریعتر اجرا خواهد کرد. در واقع

[6] - Performance

برای برنامه‌های کاربردی مختلف، ممکن است مترهای مختلفی مناسب باشد و قسمتهای مختلف یک کامپیوتر ممکن است نقش متفاوتی در تعیین سرعت کل سیستم داشته باشند.

البته در انتخاب بین کامپیوترهای مختلف، کارآیی تقریباً همیشه یک مشخصه مهم به حساب می‌آید. اندازه‌گیری و مقایسه ماشین‌های مختلف برای خریداران و در نتیجه برای طراحان یک امر حیاتی محسوب می‌شود که این مطلب را فروشندگان کامپیوترها به خوبی درک می‌کنند. اغلب، فروشندگان دوست دارند که شما کامپیوترهای آنها را خیلی ساده نگاه کنید و این نگاه کردن ممکن است بدون توجه به نیاز خریدار، که برنامه‌های کاربردی خودش را بر روی آن اجرا خواهد کرد انجام بگیرد. در بعضی مواقع ادعاهایی در مورد کامپیوترها می‌شود که دید مفیدی برای بعضی از برنامه‌های کاربردی در اختیار قرار نمی‌دهند. بنابراین فهمیدن چگونگی اندازه گیری کارآیی یک کامپیوتر و محدودیتهای اندازه‌گیری در انتخاب یک ماشین، بسیار مهم می‌باشند.

علاقمندی ما نسبت به کارآیی کامپیوتر ماورای این است که فقط کارآیی کامپیوتر را از بیرون آن اندازه بگیریم. ما نیاز به این داریم که بفهمیم چه چیزهایی کارآیی یک کامپیوتر را تحت تأثیر قرار می‌دهند. فهمیدن اینکه چرا بعضی قسمتهای نرم‌افزار همان طور که ما انتظار داریم کار نمی‌کنند، چرا یک مجموعه از دستورات می‌تواند به گونه‌ای پیاده سازی شود که بهتر عمل کند، و اینکه چگونه بعضی از قابلیتهای سخت‌افزار کارآیی را تحت تأثیر قرار می‌دهند، همگی از مسائلی است که ما علاقمند به پیدا کردن جواب مناسب برای آنها هستیم.

به طور مثال برای بهبود سرعت یک سیستم نرم‌افزاری، ممکن است احتیاج داشته باشیم که فاکتورهای جدید سخت‌افزاری را که تأثیر بر کل سیستم دارند را بشناسیم و میزان اهمیت این فاکتورها را درک نمائیم. این فاکتورها ممکن است شامل موارد زیر باشد: برنامه‌های کاربران کدام دستورات ماشین را بیشتر استفاده می‌کنند، سخت‌افزار چگونه این مجموعه دستورات را پیاده‌سازی می‌کند و سیستم‌های IO و حافظه چقدر خوب کار می‌کنند؟ فهمیدن اینکه چگونه تأثیر این فاکتورها را بر کارآیی تعیین کنیم، برای درک انگیزه‌های پشت صحنه طراحی بعضی از قابلیت‌های ویژه ماشین که ما آنها را در فصل‌های آتی خواهیم دید، خیلی مهم می‌باشند. این فصل روشهای اندازه‌گیری کارآیی را توضیح خواهد داد. در بخش ۲-۲ ما معیارهای اندازه‌گیری کارآیی را از دو دیدگاه استفاده کننده کامپیوتر و طراح سخت افزار بیان خواهیم کرد. در بخش ۲-۳ مشاهده خواهیم کرد که چگونه این معیارها به هم وابستگی دارند و فرمولی جهت اندازه‌گیری کارآیی ارائه خواهد شد که در سراسر این کتاب از آن استفاده خواهد شد. بخش‌های ۲-۴ و ۲-۵ در مورد نحوه انتخاب benchmark هایی که برای ارزیابی

کارآیی کامپیوترها استفاده می‌شود صحبت خواهد کرد. همچنین یاد می‌گیریم که چگونه سرعت اجرای دسته‌ای از برنامه‌ها را به طور دقیق خلاصه و جمع‌بندی نمائیم و گزارش تهیه کنیم. بخش ۲-۶ یک دسته از benchmark های معمول استفاده شده برای CPUها را توضیح خواهد داد و با استفاده از این benchmark ها سرعت چند پردازنده اینتل را اندازه خواهد گرفت. سرانجام در فصل ۲-۷ ما به دسته-ای از کج فهمی‌ها و دام‌هایی که معمولاً طراحان و تحلیل کننده‌های کارآیی را گرفتار می‌کنند، خواهیم پرداخت.

تعریف کارآیی

وقتی که گفته می‌شود کارآیی این کامپیوتر بالاتر از آن یکی است، منظور چیست؟ هر چند این سؤال ساده به نظرمی‌رسد ولی یک مقایسه با هواپیماهای مسافربری نشان می‌دهد که این سؤال چقدر قابل تأمل است.

شکل ۲-۱ تعدادی از هواپیماهای مسافربری را به همراه سرعت متوسط آنها، برد مسافتی و ظرفیتشان نشان می‌دهد. اگر بخواهیم ببینیم که کدام یک از هواپیماهای موجود در جدول کارآیی بالاتری دارد، اول باید کارآیی را تعریف کنیم. به طور مثال، با در نظر گرفتن اندازه‌گیری‌های مختلف، می‌بینیم که هواپیمایی که بالاترین سرعت متوسط را دارد concorde، و هواپیمایی که بیشترین ظرفیت را دارد 747 می‌باشد.

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
747	6.5 hours	610 mph	470	286,700
Concorde	3 hours	1350 mph	132	178,200

شکل ۲-۱: مقایسه کارآیی هواپیماهای مختلف

بنابراین اگر ما بخواهیم کارآیی این دو هواپیما را مقایسه کنیم. به دو صورت می‌توانیم این کار را انجام دهیم. می‌توانیم به این صورت تعریف کنیم که هواپیمای کاراتر هواپیمایی است که یک مسافر را از نقطه‌ای به نقطهٔ دیگر در کمترین زمان جابجا می‌کند. پس اگر شما علاقمند باشید که زودتر به مقصد برسید واضح است که concorde کاراتر می‌باشد. روش دوم مقایسه، برای این مثال تعداد مسافر جابجا شده در واحد زمان است. بر اساس این معیار همان طور که در ستون آخر جدول نیز نشان داده شده است 747 کاراتر می‌باشد.

به طور مشابه ما می‌توانیم کارآیی کامپیوترها را به طرق مختلف تعریف نمائیم. اگر شما یک برنامه را بر روی دو کامپیوتر مختلف اجرا کنید، خواهید گفت که کامپیوتری سریعتر است که آن برنامه را سریعتر اجرا می‌کند. ولی اگر برنامه خود را بر روی یک ترمینال در مرکز کامپیوتری اجرا کنید که دارای دو کامپیوتر است که به طور مشترک توسط چندین نفر استفاده می‌شوند در این صورت خواهید گفت که کامپیوتری سریعتر است که تعداد زیادتری برنامه را در روز اجرا کند. همان طور که یک کاربر دوست دارد که زمان پاسخ (response time) یا همان زمان اجرا (execution time) که مدت زمان بین شروع و خاتمه یک کار است، بر روی یک کامپیوتر برای برنامه او کمتر شود، در یک مرکز کامپیوتر، مدیر مرکز اغلب دوست دارد که تعداد کار انجام گرفته در واحد زمان (throughput) افزایش پیدا کند.

مثال

برای نشان دادن نحوهٔ استفاده از ایده‌های جدید، مثالهای ویژه‌ای در سراسر این کتاب آورده شده است. ما ابتدا مثال را می‌آوریم و بعد به آن پاسخ می‌دهیم. سعی کنید که مثال را خودتان جواب دهید، اگر نتوانستید و یا به پاسخ خود اطمینان نداشتید جواب را مشاهده نمائید. مثالهایی که آورده می‌شود شبیه به مسائلی است که در آخر هر فصل آورده می‌شود. این اولین مثال از این کتاب است:

مثال: آیا تغییرات انجام شده زیرین بر روی یک سیستم کامپیوتری، throughput را زیاد می‌کند؟ زمان پاسخ دهی را کاهش می‌دهد؟ و یا هر دو؟

۱. جایگزین کردن پردازنده سیستم با یک نسخه سریعتر
۲. اضافه کردن یک پردازنده جدید به سیستمی که از چندین پردازنده برای کارهای مختلف استفاده می‌کند. به طور مثال سیستمی که عملیات رزواسیون یک خطوط هواپیمایی را انجام می‌دهد.

جواب: کم کردن زمان پاسخ دهی تقریباً همیشه throughput را بهبود می‌دهد. بنابراین در مورد اول، زمان پاسخ دهی و throughput هر دو بهتر می‌شوند. در مورد ۲، برنامهٔ هیچ کسی سریعتر انجام نمی-گیرد، بنابراین فقط throughput بهتر می‌شود. اگر در مورد ۲، نیاز به زمان پاسخ دهی سریع همانند افزایش throughput اهمیت داشته باشد، سیستم ممکن است مجبور کند که درخواستها در داخل یک صف قرار بگیرند. در این حالت افزایش throughput، زمان پاسخ دهی را نیز کاهش خواهد داد چون زمان انتظار را در داخل صف کاهش خواهد داد. بنابراین در تعداد زیادی از سیستم‌های واقعی کامپیوتری، تغییر زمان پاسخ دهی یا throughput، اغلب دیگری را نیز تحت تأثیر قرار می‌دهد.

ما در مبحث کارآیی کامپیوترها، در ابتدا در طول چند فصل بر روی زمان پاسخ دهی تمرکز خواهیم کرد. در فصل ۸ که در ارتباط با سیستم‌های ورودی/ خروجی می‌باشد ما اندازه‌گیری‌های مربوط به throughput را نیز صحبت خواهیم کرد.

برای اضافه کردن سرعت (کارآیی)، ما می‌خواهیم که زمان پاسخ دهی یا زمان اجرا را کاهش دهیم، بنابراین می‌توان زمان اجرا و کارآیی را برای یک ماشین x به صورت زیر مرتبط کرد:

$$\text{کارآیی ماشین x} = \frac{1}{\text{زمان اجرا بر روی ماشین x}}$$

فرمول فوق به صورت زیر نیز نوشته می‌شود:

$$performance_x = \frac{1}{(ExecutionTime)_x}$$

این به این معنی است که برای دو ماشین x و y ، اگر کارآیی x بیشتر از کارآیی y باشد داریم:

$$performance_x > performance_y$$
$$\Rightarrow \frac{1}{(ExecutionTime)_x} > \frac{1}{(ExecutionTime)_y}$$
$$\Rightarrow (ExecutionTime)_y > (ExecutionTime)_x$$

یعنی اینکه اگر x سریعتر از y باشد زمان اجرا بر روی y بیشتر از زمان اجرا بر روی x خواهد بود. در مبحث طراحی کامپیوتر، ما اغلب سرعت دو کامپیوتر مختلف را به صورت عددی مقایسه می‌کنیم. مثلاً می‌گوئیم ماشینx ، n بار سریعتر از ماشین y می‌باشد و می‌نویسیم.

$$\frac{performance_x}{performance_y} = n$$

اگرx، n بار سریعتر از y باشد در این صورت زمان اجرا بر رویy، n بار بیشتر از زمان اجرا بر روی x خواهد شد:

$$\frac{performance_x}{performance_y} = \frac{(ExecutionTime)_y}{(ExecutionTime)_x} = n$$

مثال:اگر ماشین A یک برنامه را در ۱۰ ثانیه و ماشین B آن را در ۱۵ ثانیه اجرا کند، ماشین A چند برابر از B سریعتر است؟

جواب: طبق فرمول بالایی می‌دانیم که اگر ماشین A ، n بار سریعتر از B باشد می‌توان نوشت:

$$\frac{performanc\,e_A}{performanc\,e_B} = \frac{(ExecutionTime)_B}{(ExecutionTime)_A} = n$$

بنابراین نسبت کارآیی به صورت زیر حساب می‌شود:

$$n = \frac{15}{10} = 1.5$$

بنابراین A ، 1.5 برابر سریعتر از B می‌باشد.

در مثال بالایی می‌توان گفت که ماشین B ، ۱/۵ برابر کندتر از ماشین A عمل می‌کند. چون داریم:

$$\frac{performanc\,e_A}{performanc\,e_B} = 1.5 \Rightarrow \frac{performanc\,e_A}{1.5} = performanc\,e_B$$

برای سادگی در مقایسه کارآیی کامپیوترها ما از عبارت سریعتر بودن استفاده خواهیم کرد. به دلیل اینکه کارآیی و زمان اجرا معکوس هم می‌باشند، افزایش کارآیی نیازمند این است که زمان اجرا را کاهش دهیم. برای جلوگیری از گیج شدن بین جمله‌های افزایش و کاهش، ما معمولاً وقتی که منظورمان "افزایش کارآیی" و "کاهش زمان اجراست"، می‌گوئیم "بهبود کارآیی"[۷] یا "بهبود زمان اجرا"[۸].

۲-۲- اندازه‌گیری کارآیی

اکنون زمان اندازه‌گیری کارآیی است، کامپیوتری که همان مقدار کار را در زمان کمتری انجام می‌دهد سریعتر است. زمان اجرای برنامه با ثانیه اندازه گرفته می‌شود. اما زمان می‌تواند بسته به اینکه ما چه چیزی را شمارش می‌کنیم، به گونه‌ای دیگر تعریف گردد. سرراست‌ترین تعریف زمان، زمان پاسخ‌دهی یا زمان انقضا می‌باشد[۹]. این اصطلاحات به معنی کل زمان لازم برای خاتمه یک کار است، که این زمان شامل دسترسی به حافظه دیسک، دسترسی به حافظه اصلی، دسترسی به IOها، سربار اجرای سیستم-عامل و هر زمان دیگری است.

---

[7] - Performance improvement
[8] - Execution time improvement
[9] - Elapsed time

کامپیوترها اغلب به صورت اشتراک زمانی استفاده می‌شوند و یک پردازنده ممکن است بر روی چندین برنامه به صورت همزمان کار کند. در این صورت سیستم ممکن است تلاشش در این راستا باشد که به جای کاهش زمان اجرای یک برنامه، throughput را کاهش دهد. بنابراین ما می‌خواهیم که بین زمان انتظار و زمانی که cpu فقط برای ما کار می‌کند تفاوت قائل شویم. زمان اجرای cpu (cpu execution time) یا به طور ساده cpu time مدت زمانی است که cpu صرف کار ما می‌کند و شامل زمان انتظار IO یا زمان اجرای برنامه‌های دیگر نمی‌شود. (به خاطر بیاورید که زمان پاسخ‌دهی به کاربر زمان انقضای برنامه است نه زمان cpu time).

cpu time می‌تواند به دو بخش تقسیم گردد: زمانی از cpu که صرف اجرای خود برنامه می‌شود (user cpu time) و زمانی از cpu که صرف اجرای سیستم‌عامل در ارتباط با این برنامه می‌شود. (معمولاً system cpu time).

سیستم عامل unix دستوری به نام time دارد که می‌تواند زمان انقضا را نشان دهد به طور مثال این دستور می‌تواند نتیجه‌ای مانند عبارت زیر را برگرداند:

90.7u 12.95 2:39 65%

در عبارت فوق زمان user cpu time 90.7 ثانیه، زمان system cpu time 12.9 ثانیه، زمان انقضا 2 دقیقه و 39 ثانیه (۱۵۹ ثانیه) و درصدی از زمان انقضا که مربوط به زمان cpu time می‌باشد به صورت زیر است:

$$\frac{90.7+12.9}{159}=0.65$$

یا 65 درصد. در این مثال بیش از $\frac{1}{3}$ از زمان انقضا صرف انتظار برای IO، اجرای برنامه‌های دیگر یا هر دو شده است.

بعضی مواقع وقتی صحبت از زمان اجرای cpu می‌شود ما معمولاً از زمان system cpu time چشم-پوشی می‌کنیم و این معمولاً به دلیل غیر دقیق بودن اندازه‌گیری سیستم‌عامل که خودش را اندازه می-گیرد و نیز به دلیل بی‌عدالتی در مواقعی است که می‌خواهیم system cpu time دو ماشین که سیستم-عاملهای مختلفی را اجرا می‌کنند مقایسه کنیم. از طرف دیگر، کدهای سیستمی در بعضی ماشینها، جزو کدهای کاربری ماشین دیگر می‌باشد. تقریباً هیچ برنامه‌ای نمی‌تواند بدون حضور سیستم‌عامل بر روی سخت‌افزار اجرا شود، بنابراین زمان اجرای برنامه را می‌توان مجموع user cpu time و system cpu time در نظر گرفت.

ما بین کارآیی محاسبه شده بر اساس زمان انقضا و براساس زمان cpu execution time فرق قائل خواهیم شد. ما عبارت کارآیی سیستم (system performance) را برای زمان انقضا برای یک سیستم خالی از برنامه به کار می‌بریم و عبارت cpu performanc را در ارتباط با user cpu time به کار خواهیم برد. ما در این فصل بر روی کارآیی cpu (cpu performance) تمرکز خواهیم کرد، هر چند بحث ما برای سرعت می‌تواند برای زمان انقضا و برای cpu time نیز مورد استفاده قرار گیرد.

هر چند که ما مانند کاربران کامپیوتر به زمان دقت می‌کنیم، وقتی ما جزئیات یک ماشین را بررسی می-کنیم راحت‌تر این است که معیارهای دیگری را برای سرعت در نظر بگیریم. علی الخصوص طراحان کامپیوتر ممکن است در مورد یک ماشین به این صورت فکر کنند که با استفاده از یک معیاری نشان دهند که سخت‌افزار آن کامپیوتر عملیات پایه‌ای را چقدر سریعتر انجام می‌دهد. تقریباً همهٔ کامپیوترها در ساختارشان از یک کلاک (clock) استفاده می‌کنند که این کلاک سیگنالی است که به صورت پریودیک تولید می‌شود و مشخص می‌کند اتفاقات در سخت‌افزار کی رخ دهند. به هر پریود کلاک یک سیکل کلاک (cycles , clocks , clock periods , clock ticks , ticks , clock cycle) گفته می‌شود. طراحان معمولاً طول پریود کلاک را به صورت زمان یک سیکل کامل کلاک (مانند ۲ نانو ثانیه) و یا به صورت نرخ کلاک (مانند ۵۰۰ مگاهرتز یا 500 mhz) که معکوس پریود کلاک است به کار می‌برند. در بخش بعدی ما ارتباط بین سیکل کلاک و زمان اجرای برنامه کاربر را توضیح خواهیم داد.

۲-۳- ارتباط بین معیارها (مترها)

کاربران و طراحان برای اندازه‌گیری سرعت از معیارهای مختلفی استفاده می‌کنند. اگر ما بتوانیم این معیارها را به هم مرتبط سازیم در این صورت می‌توانیم اثر یک تغییر در طراحی بر روی سرعت کامپیوتر را که توسط کاربر مشاهده می‌شود، مشخص نمائیم. فرمول زیر نحوه محاسبه زمان اجرای cpu را نشان می‌دهد. این فرمول ارتباط کلاک با زمان cpu را نشان می‌دهد.

زمان اجرای cpu = تعداد کلاک‌های لازم برای اجرای برنامه × پریود کلاک

چون فرکانس کلاک (clock rate) و پریود کلاک معکوس هم هستند فرمول بالایی را می‌توان به صورت زیر نیز نمایش داد:

$$\text{زمان اجرای cpu} = \frac{\text{تعداد کلاک‌های لازم برای اجرای برنامه}}{\text{فرکانس کلاک}}$$

این فرمول به وضوح نشان می‌دهد که طراح سخت‌افزار می‌تواند با کاهش طول پریود کلاک یا تعداد کلاک‌های لازم برای اجرای یک برنامه، سرعت اجرای برنامه را افزایش دهد (سرعت را بهبود دهد). ما در ادامه این فصل و در فصول ۵ و۶ و۷ خواهیم دید که طراح نمی‌تواند همزمان هم طول پریود کلاک را کاهش دهد و هم تعداد کلاکهای لازم برای برنامه را کاهش دهد و اغلب باید یک بالانس بین آنها برقرار کند (trade off). اکثر تکنیکهایی که تعداد کلاکها را کاهش می‌دهند معمولاً طول پریود کلاک را افزایش می‌دهند.

مثال: برنامه دلخواه ما بر روی کامپیوتر A که فرکانس کلاک آن ۴۰۰ مگاهرتز است در ۱۰ ثانیه اجرا می‌شود. ما می‌خواهیم به یک طراح کامپیوتر کمک کنیم که یک ماشین به نام B بسازد که برنامه ما را در ۶ ثانیه انجام دهد. طراح کامپیوتر به ما گفته است که افزایش فرکانس کلاک امکان‌پذیر است ولی این افزایش فرکانس طراحی بخش‌های دیگر cpu را تحت تأثیر قرار خواهد داد و باعث خواهد شد که ماشین B برای اجرای این برنامه تعداد کلاکهایی در حدود ۱/۲ برابر ماشین A نیاز داشته باشد. ما چه فرکانس کلاکی را برای رسیدن به این هدف از طراح بخواهیم؟

جواب: در ابتدا تعداد کلاکهای لازم برای اجرای برنامه بر روی ماشین A را حساب می‌کنیم:

$$\text{زمان اجرای برنامه بر روی A} = \frac{\text{تعداد کلاک‌های لازم برای اجرای برنامه بر روی A}}{\text{فرکانس کلاک ماشین A}}$$

$$10\ \text{s} = \frac{\text{تعداد کلاک‌های لازم برای اجرای برنامه بر روی A}}{400 \times 10^6}$$

$$\text{تعداد کلاک‌های لازم برای اجرای برنامه بر روی A} = 4000 \times 10^6$$

زمان اجرای برنامه توسط ماشین B به صورت زیر است:

$$\text{زمان اجرای برنامه بر روی B} = \frac{\text{تعداد کلاک‌های لازم برای اجرای برنامه بر روی B}}{\text{فرکانس کلاک ماشین B}}$$

$$6\ \text{s} = \frac{1.2 \times (\text{تعداد کلاک‌های لازم برای اجرای برنامه بر روی A}) \times 1.2}{} = \frac{1.2 \times 4000 \times 10^6}{}$$

فرکانس کلاک ماشینB فرکانس کلاک ماشین B

فرکانس کلاک ماشینB = 800 MHz

بنابراین ماشین B برای اینکه برنامه ما را در ۶ ثانیه اجرا کند باید فرکانسش دو برابر فرکانس ماشین A باشد.

تلاقی سخت‌افزار و نرم‌افزار (Hardware Software Interface)

در سراسر این کتاب شما بخشهایی تحت این عنوان خواهید دید. این بخش‌ها تأثیر متقابل بعضی ویژگی‌های نرم‌افزار (نرم‌افزار می‌تواند یک برنامه، یک کامپایلر و یا یک سیستم‌عامل باشد) و بعضی ویژگی‌های سخت‌افزار بر همدیگر را بیان خواهد نمود و آن را به صورت کاملاً واضح و روشنی توضیح خواهد داد. این بخشها باعث خواهند شد که ما به خاطر بسپاریم که طراحی سخت‌افزار و نرم-افزار در بسیاری از موارد تأثیر متقابلی بر هم دارند.

در معادلات موجود در مثالهای قبلی جایی برای تعداد دستورات مورد نیاز یک برنامه در نظر گرفته نشده بود. اما به هر حال چون کامپایلر برای اجرا شدن برنامه دستورهایی به زبان ماشین تولید می‌کند و ماشین نیز برای اجرای برنامه باید آن دستورات را اجرا نماید، بنابراین زمان اجرای یک برنامه باید به تعداد دستورات تولید شده برای یک برنامه وابسته باشد. یکی از راههایی که می‌توان برای محاسبه زمان اجرا در نظر گرفت این است که بگوئیم زمان اجرا مساوی است با تعداد دستورات یک برنامه ضربدر زمان متوسط لازم برای اجرای یک دستور. بنابراین تعداد سیکلهای کلاک مورد نیاز برای یک برنامه می‌تواند به صورت زیر محاسبه شود:

(تعداد کلاک متوسط لازم برای اجرای هر دستور) × (تعداد دستورات برنامه) = تعداد کلاک‌های لازم برای اجرای یک برنامه

عبارت تعداد کلاکهای متوسط مورد نیاز برای هر دستور[10] اغلب CPI نامیده می‌شود. چون دستورات مختلف بسته به اینکه چه کاری انجام می‌دهند، ممکن است زمان اجرای متفاوتی داشته باشند، CPI متوسط زمان اجرای همهٔ دستوراتی است (زمان اجرا به کلاک) که در داخل یک برنامه قرار دارند. CPI روشی برای مقایسه دو پیاده‌سازی مختلف از یک مجموعه دستورات را نیز در اختیار قرار می‌دهد چون در این دو پیاده‌سازی تعداد دستورات مورد نیاز برنامه مطمئناً یکی خواهد بود.

[10] - clock cycles per instruction

مثال: فرض کنید ما دو پیاده‌سازی از یک مجموعه دستورات واحد در اختیار داشته باشیم. ماشین A برای یک برنامه دارای پریود کلاک ۱ نانوثانیه بوده و CPI آن ۲ می‌باشد و ماشین B دارای پریود کلاک ۲ نانوثانیه بوده و CPI آن ۱/۲ می‌باشد. برای این برنامه کدام ماشین سریعتر است و چقدر؟

جواب: می‌دانیم تعداد دستوراتی که هر کدام از ماشین‌ها برای این برنامه اجرا می‌کنند یکی است. ما این تعداد دستورات را I می‌نامیم. در ابتدا تعداد کلاکهای پردازنده را برای هر کدام از ماشین‌ها حساب می‌کنیم:

تعداد کلاک‌های لازم برای اجرای برنامه بر روی ماشین A = I × 2.0

تعداد کلاک‌های لازم برای اجرای برنامه بر روی ماشین B = I × 1.2

حال زمان اجرا برنامه را برای هر کدام از ماشین‌ها حساب می‌کنیم:

زمان اجرای برنامه بر روی ماشین A = تعداد کلاک‌های لازم برای اجرای برنامه × پریود کلاک

$$= I \times 2.0 \times 1\,ns = 2 \times I\,ns$$

همین طور برای ماشین B داریم:

$$\text{زمان اجرای برنامه بر روی ماشین B} = I \times 1.2 \times 2\,ns = 2.4 \times I\,ns$$

واضح است که ماشین A سریعتر است چون زمان اجرای پایین‌تری دارد.. برای اینکه بدانیم A چقدر سریعتر از B است به صورت زیر عمل می‌کنیم:

$$\frac{\text{کارآیی ماشین A}}{\text{کارآیی ماشین B}} = \frac{\text{زمان اجرای برنامه بر روی ماشین B}}{\text{زمان اجرای برنامه بر روی ماشین A}} = \frac{2.4 \times I\,ns}{2 \times I\,ns} = 1.2$$

بنابراین ماشین A برای این برنامه، 1.2 برابر سریعتر ار ماشین B می‌باشد.

مثال: آیا موارد زیر می‌توانند به تنهایی نشان دهندهٔ کارآیی باشند؟

۱. تعداد کلاکهای لازم برای یک برنامه
۲. تعداد دستورات یک برنامه
۳. پریود کلاک
۴. تعداد کلاک لازم برای اجرای یک دستور
۵. تعداد متوسط دستوراتی که در یک ثانیه اجرا می‌شوند.

جواب: هیچکدام از موارد نمی‌توانند به تنهایی نشان دهنده کارآیی باشند. کارآیی با زمان اجرا ارتباط معکوس دارد و باید زمان اجرا به نوعی در معیار مشخص شده باشد. در مورد اول تعداد کلاک معیار

مناسبی نیست چون مشخص نیست که پریود کلاک چقدر است و بنابراین نمی‌توانیم زمان اجرا را بدست آوریم. در مورد دوم تعداد دستورات نیز معیار مناسبی نیست چون نوع و زمان اجرای دستورات مشخص نیست و نمی‌توانیم زمان اجرای برنامه را بدست آوریم. دستورات مختلف زمان اجرای متفاوتی دارند به عنوان مثال یک دستور ضرب نسبت به یک دستور جمع زمان اجرای بالاتری دارد و دستورات ممیز شناور که با داده‌های ممیزدار کار می‌کنند نسبت به دستوراتی که از داده‌های نوع صحیح استفاده می‌کنند، زمان اجرای بالاتری دارند. در مورد سوم پریود کلاک نیز معیار مناسبی نیست چون مشخص نیست که برنامه به چند کلاک برای اجرا نیاز دارد و نمی‌توانیم زمان اجرای برنامه را مشخص کنیم. در مورد چهارم تعداد کلاک لازم برای اجرای هر دستور نیز معیار مناسبی نیست چون تعداد دستورات و پریود کلاک مشخص نشده و نمی‌توانیم زمان اجرا را حساب کنیم. در مورد پنجم تعداد متوسط دستوراتی که در یک ثانیه اجرا می‌شود نیز به تنهایی معیار مناسبی نیست چون نوع دستورات و تعداد کلاک لازم برای هر دستور و فرکانس کلاک مشخص نیست و نمی‌توانیم زمان اجرا را بدست آوریم.

مثال: اگر دو ماشین ISA یکسان داشته باشد کدام یک از موارد زیر در دو ماشین همیشه برابر می‌باشد؟

۱. فرکانس کلاک
۲. CPI
۳. زمان اجرا
۴. تعداد دستورات
۵. MIPS

جواب: اگر دو ماشین مختلف وجود داشته باشند که ISA یکسانی را پیاده سازی کرده باشند یعنی مجموعه دستوراتی که پشتیبانی می‌کنند مثل هم باشد، در این صورت حتماً تعداد دستورات یک برنامه که برای این دو ماشین کامپایل بشود در هر دو کامپایل یکی خواهد بود.

مثال: یک طراح کامپایلر می‌خواهد بین دو قطعه کد برای یک ماشین مشخص یکی را انتخاب نماید. بر اساس پیاده سازی سخت افزار، سه کلاس مختلف از دستورات وجود دارد: کلاس A، کلاس B و کلاس C که به ترتیب برای اجرا شدن نیاز به ۱، ۲ و ۳ کلاک دارند. قطعه کد اول دارای ۵ دستور است: ۲ مورد از کلاس A، ۱ مورد از کلاس B و ۲ مورد از کلاس C و قطعه کد دوم دارای ۶ دستور است: ۴ مورد از کلاس A، ۱ مورد از کلاس B و ۱ مورد از کلاس C.

الف) کدام قطعه کد سریعتر اجرا می‌شود و چقدر؟

ب) CPI را برای هر قطعه کد حساب کنید؟

جواب: الف) برای بدست آوردن تعداد کلاک از فرمول زیر استفاده می‌کنیم:

$$\text{تعداد کلاکهای لازم} = \sum_{i=1}^{n} (CPI_i \times C_i)$$

که در آن CPI_i تعداد کلاکهای لازم برای اجرای دستورات از نوع کلاس i می‌باشد و C_i تعداد دستورات کلاس i می‌باشد.

تعداد کلاکهای لازم برای قطعه کد ۱ = $2 \times 1 + 1 \times 2 + 2 \times 3$ = ۱۰ کلاک

تعداد کلاکهای لازم برای قطعه کد ۲ = $4 \times 1 + 1 \times 2 + 1 \times 3$ = ۹ کلاک

چون هر دو قطعه کد بر روی یک ماشین اجرا می‌شوند و در نتیجه پریود کلاک برای هر دو یکی است بنابراین قطعه کدی سریعتر است که تعداد کلاک کمتری لازم داشته باشد. بنابراین در این مثال با اینکه قطعه کد ۲ تعداد دستورات بیشتری دارد ولی سریعتر اجرا می‌شود.

ب) برای بدست آوردن CPI از فرمول زیر استفاده می‌کنیم:

$$\text{CPI} = \frac{\text{تعداد کلاکها}}{\text{تعداد دستورات}}$$

$$\text{قطعه کد اول} \quad \text{CPI} = \frac{۱۰}{۵} = ۲$$

$$\text{قطعه کد دوم} \quad \text{CPI} = \frac{۹}{۶} = ۱/۵$$

مثال: فرض کنید یک ماشین داشته باشیم که با سرعت MHZ ۵۰۰ کار کند و فرض کنید که این ماشین دارای سه کلاس دستور از نوع A و B و C باشد که به ترتیب برای اجرا شدن به یک، دو و سه کلاک نیاز داشته باشند. می‌خواهیم دو کامپایلر را بر روی این دو ماشین امتحان کنیم. این کامپایلرها برای تولید کد از یک برنامه بزرگ مورد استفاده قرار می‌گیرند.

کامپایلر اول به تعداد ۵ میلیارد دستور از نوع A، یک میلیارد دستور از نوع B و یک میلیارد دستور از نوع C استفاده می‌کند. و کامپایلر دوم به تعداد ۱۰ میلیارد دستور از نوع A، یک میلیارد دستور از نوع B و یک میلیارد دستور از نوع C استفاده می‌کند.

الف) کدام قطعه کد از لحاظ زمان اجرا سریعتر است؟

ب) کدام قطعه کد از لحاظ تعداد MIPS سریعتر است؟

جواب: MIPS به عنوان یک معیار برای مقایسه کارآیی پردازنده‌ها مورد استفاده قرار می‌گیرد. MIPS که مخفف عبارت Millions of Instruction Per second می‌باشد برای یک کامپیوتر به این صورت تعریف می‌شود: تعداد میلیون دستوری که یک کامپیوتر در یک ثانیه انجام می‌دهد. این مثال نشان می-دهد که MIPS معیار مناسبی برای کارآیی نیست چون در بعضی مواقع نمی‌تواند زمان اجرا را به درستی نشان بدهد.

الف) برای زمان اجرا از فرمول زیر استفاده می‌کنیم:

$$\text{زمان اجرا برای قطعه کد اول} = \left(\sum_{i=1}^{n} CPI_i \times C_i\right) \times \text{پریود کلاک}$$

$$= (۵ \times ۱ + ۱ \times ۲ + ۱ \times ۳) \times ۱۰^۹ \times \text{پریود کلاک}$$

$$= \frac{۱۰ \times ۱۰^۹}{۵۰۰ \times ۱۰^۶} = \text{پریود کلاک} \times ۶ \times ۱۰ \times ۱۰ = ۲۰\ s$$

$$\text{زمان اجرای قطعه کد دوم} = \left(\sum_{i=1}^{n} CPI_i \times C_i\right) \times \text{پریود کلاک}$$

$$= (۱۰\times۱ + ۱\times۲ + ۱\times۳) \times ۱۰^۹ \times \text{پریود کلاک} = \frac{۱۵ \times ۱۰^۹}{۵۰۰ \times ۱۰^۶} = ۳۰\ s$$

پس از نظر زمان اجرا قطعه کد اول سریعتر است.

ب) برای MIPS از فرمول زیر استفاده می‌کنیم:

$$MIPS = \frac{\text{تعداد دستورات}}{}$$

$$10^6 \times \text{زمان اجرا}$$

$$\text{MIPS قطعه کد اول} = \frac{(۵ + ۱ + ۱) \times ۱۰^۹}{۲۰ \times ۱۰^۶} = ۳۵۰$$

$$\text{MIPS قطعه کد دوم} = \frac{(۱۰ + ۱ + ۱) \times ۱۰^۹}{۳۰ \times ۱۰^۶} = ۴۰۰$$

بنابراین از نظر MIPS قطعه کد دوم سریعتر است.

مثال: تعداد دستورات برنامه زیر مشخص کنید.

```
Li   $a0, 1000        // a0 = 1000
Loop: sub  $a0, $a0,1      //  a0 = a0 - 1
bne $a0, $0, Loop  // if a0 != 0  go to loop
```

جواب: تعداد دستورات یک برنامه را می‌توان به دو صورت شمارش نمود: استاتیکی و دینامیکی. تعداد دستورات استاتیکی در حقیقت همان تعداد خطوط یا تعداد دستورات ظاهری برنامه است ولی تعداد دستورات دینامیکی تعداد دستوراتی است که در عمل اجرا می‌شوند. بنابراین برای این مثال داریم:

تعداد دستورات استاتیک = ۳

چون تعداد دستورات ظاهری برنامه ۳ می‌باشد.

تعداد دستورات دینامیک = ۲۰۰۱

توضیحات هر دستور در کنار همان دستور بعد از علامت // آورده شده است. اگر این برنامه را trace کنیم می‌بینیم که حلقه برنامه به تعداد ۱۰۰۰ مرتبه اجرا می‌شود. چون تعداد دستورات داخل حلقه ۲ است پس تعداد کل دسوراتی که در این حلقه اجرا خواهد شد ۲۰۰۰ مورد می‌شود. توجه داریم که یک دستور نیز در خارج حلقه قرار دارد. پس در کل تعداد دستورات دینامیکی برنامه که در عمل اجرا می‌شوند، ۲۰۰۱ مورد خواهد شد.

مثال: دو پروسسور زیر را که دارای ISA یکسان یا سازگار (ISA compatible) می‌باشند از نظر کارآیی باهم مقایسه کنید؟

۱. یک پردازنده ۸۰۰ مگاهرتز AMD Duron که CPI آن ۱/۲ است

۲. یک پردازنده ۱ گیگا هرتز پنتیوم III (P3) که CPI آن ۱/۵ است

جواب: چون دو پردازنده ISA یکسانی دارند پس برای یک برنامه فرضی تعداد دستورات برای هر جفت آنها یکی است و داریم:

پریود کلاک × CPI × تعداد دستورات = زمان اجرای برنامه روی AMD

$= I \times ۱/۲ \times ۱/۲۵ \text{ ns} = ۱/۵ \text{ ns} \times I$

زمان اجرای برنامه روی P3 $= I \times ۱/۵ \times ۱ \text{ ns} = ۱/۵ \text{ ns} \times I$

چون زمان اجرای برنامه فرضی بر روی دو پردازنده یکی است پس کارآیی آنها یکسان است!

مثال: دو کامپیوتر زیر را که کلاک مختلف ولی ISA یکسان دارند با هم مقایسه کنید.

۱. P4 2.5GHz

۲. P4 3GHz

جواب: چون دو کامپیوتر ISA یکسانی دارند پس تعداد دستورات یک برنامه فرضی بر روی آنها برابر است و همچنین چون دو کامپیوتر از یک نسل می‌باشند (هر دو از نسل P4 هستند) از نظر ساختاری مانند هم بوده و CPI آنها نیز یکی است و داریم:

پریود کلاک × CPI × تعداد دستورات = زمان اجرا

زمان اجرای ۱ = CPI × ۰/۴ ns × تعداد دستور

زمان اجرای ۲ = CPI × ۰/۳۳ ns × تعداد دستور

$$\frac{\text{کارآیی ۲}}{\text{کارآیی ۱}} = \frac{\text{زمان اجرای ۱}}{\text{زمان اجرای ۲}} = \frac{۰/۴}{۰/۳۳} = ۱/۲$$

پس پردازنده P4 3GHz به اندازه ۱/۲ برابر سریعتر از پردازنده P4 2.5GHz است.

مثال: مقایسه دو کامپایلر برای یک ماشین

فرض کنید دو کامپایلر برای یک ماشین داشته باشیم: کامپایلر معمولی و کامپایلر بهینه. کامپایلری که بهتر عمل می‌کند کدهایی تولید می‌کند که تعداد دستورات آن ۵ ٪ (پنج درصد) کمتر است و همچنین CPI آنها ۱۰٪ پایین‌تر است. این دو کامپایلر را مقایسه کنید.

جواب: فرض کنید کامپایلر معمولی را کامپایلر پایه در نظر گرفته و تعداد دستورات تولید شده توسط آن را برای یک برنامه فرضی با I ، CPI کد تولید شده توسط آن را با CPI_B و پریود کلاک ماشین را با T نشان دهیم در این صورت داریم:

زمان اجرای کد تولید شده توسط کامپایلر پایه = تعداد دستورات پایه × CPI_B × پریود کلاک

$$= I \times CPI_B \times T$$

زمان اجرای کد تولید شده با کامپایلر بهینه = تعداد دستورات × CPI × T

$$= (0.95 \times I) \times (0.9 \times CPI_B) \times T$$

$$= 0.86 \times (I \times CPI_B \times T)$$

زمان اجرای کامپایلر پایه × ۰/۸۶ =

$$\frac{\text{زمان اجرای پایه}}{\text{زمان اجرای بهینه}} = \frac{\text{کارآیی کامپایلر بهینه}}{\text{کارآیی کامپایلر پایه}} = \frac{۱/۰۰}{۰/۸۶} = ۱/۱۷$$

توجه: با توجه به مثالهای متنوعی که حل کردیم می‌توان نتیجه گرفت که روشهای متعددی برای افزایش سرعت پردازنده‌ها وجود دارد. بعضی از این روشها مربوط به تکنیک‌های کامپایلری و نرم-افزاری است که می‌توان توسط این تکنیکها کدهای با تعداد دستورات کم و با CPI پایین تولید نمود (CPI می‌تواند بر اساس نوع و تعداد دستورات از کدی به کد دیگر تغییر کند در واقع ترکیب دستورات[11] CPI را عوض می‌کند). بعضی دیگر از این روشها سخت افزاری هستند و می‌توان سخت-افزارهایی طراحی نمود که پریود کلاک پایینی داشته باشند و تعداد کلاک لازم برای اجرای هر دستور نیز پایین باشد (در واقع با طراحی سخت افزارهای مناسب می‌توان CPI را کم کرد).

قانون امدال

قانون امدال در معماری کامپیوتر قانونی است که می‌توان به کمک آن ارزش کارهایی که قرار است انجام شوند را از قبل پیش بینی نمود. این کارها عمدتاً مربوط به بهبودهایی است که در سخت‌افزار پردازنده‌ها داده می‌شود. این قانون به شرح زیر است:

[11] - Instruction mix

قانون امدال: مقدار بهبودهایی که انجام می‌شود به میزان مؤثر بودن آنها محدود می‌شود.

برای توضیح این قانون سیستم واحدی دانشگاه را در نظر می‌گیریم. در این سیستم بعضی دروس یک واحدی، بعضی دو واحدی، بعضی سه واحدی و بعضی نیز چهار واحدی‌اند. همان‌طور که می‌دانیم در محاسبه معدل ترم، دروسی بیشتر تأثیر دارند که تعداد واحد آنها بالاتر است. بنابراین در این سیستم ارزش دروس چهار واحدی بیشتر از سه واحدی، ارزش دروس سه واحدی بیشتر از دو واحدی، ارزش دروس دو واحدی نیز بیشتر از یک واحدی است. بنابراین ما بهتر است بیشترین وقت و سرمایه‌گذاری را بر روی دروسی انجام دهیم که بیشترین تأثیر را در معدل دارند. در معماری کامپیوتر قانون امدال می‌گوید: قبل از اینکه بهبودی را انجام دهی یا کاری را انجام دهی باید به میزان مؤثر بودنش نیز توجه کنی و آن کار را وقتی انجام دهی که تأثیرش در مقابل کاری که انجام می‌دهی قابل توجه باشد. در قالب فرمول قانون امدال به صورت زیر بیان می‌شود:

$$\text{زمان اجرای برنامه بعد از بهبود} = \frac{\text{زمان اجرایی که تحت تأثیر قرار نمی‌گیرد}}{\text{میزان بهبود}} + \text{زمان اجرایی که تحت تأثیر قرار می‌گیرد}$$

مثال: فرض کنید که تصمیم گرفته باشیم سخت افزار پردازنده را تغییر دهیم تا سرعت اجرای دستورات floating point دو برابر شود. دو برابر کردن سرعت یک ایده خوب است ولی باید ببینیم در قبال کاری که انجام می‌دهیم چقدر بهبود در کل زمان اجرای یک برنامه به وجود می‌آید. به عبارتی باید ببینیم دستورات floating point چه میزان در برنامه ما نقش دارند. فرض کنید برای یک برنامه فرضی فقط ۱۰ درصد زمان اجرای برنامه (T) شامل دستورات floating point باشد. برای این برنامه تأثیر بهبود سخت‌افزار را بررسی کنید.

جواب: با استفاده از قانون امدال می‌توان نوشت:

$$\text{زمان اجرای برنامه بعد از بهبود} = \frac{۰/۱۰\ T}{۲} + ۰/۹۰\ T = ۰/۹۵\ T$$

یعنی زمان اجرای کل برنامه فقط ۵ درصد بهبود خواهد داشت. این میزان بهبود خیلی بالا نیست بنابراین بر طبق قانون امدال دو برابر کردن سرعت اجرای دستورات floating point به امید بهبود زمان اجرای برنامه در این مثال ارزش فنی ندارد.

نتیجه قانون امدال: قسمتهایی از برنامه را بهبود دهیم که بیشتر استفاده می‌شوند چون بیشترین تأثیر را در کل زمان اجرای برنامه دارند. قانون امدال بیان می‌کند که موارد پر استفاده را بهبود دهیم و یا به عبارتی می‌گوید: Make the common case fast.

۲-۴- انتخاب برنامه‌ها برای اندازه گیری کارآیی

کاربر کامپیوتری که هر روز مجموعه برنامه‌های ثابتی را اجرا می‌کند، یک کاندیدای خوب برای ارزیابی سرعت یک کامپیوتر جدید است. مجموعه برنامه‌هایی که توسط این کاربر اجرا می‌شوند، یک workload نامیده می‌شوند. برای مقایسه کارآیی دو سیستم کامپیوتری این کاربر به سادگی زمان اجرای یک workload را بر روی دو ماشین مقایسه می‌کند. اما مشکلی که در اینجا وجود دارد این است که اکثر کاربران در این شرایط قرار ندارند و باید روشهای دیگری را برای ارزیابی کارآیی استفاده کنند و امیدوار باشند که این روشها بتوانند نشان دهنده کارآیی workload های واقعی باشند. این روشها معمولاً مبتنی بر استفاده از مجموعه‌ای از benchmark ها می‌باشند. Benchmark ها مجموعه‌ای از برنامه‌ها هستند که که برای اندازه‌گیری کارآیی انتخاب می‌شوند. Benchmark ها یک workload را تشکیل می‌دهند که کاربر امیدوار است که نشان دهنده کارآیی یک workload واقعی باشد.

یک benchmark خوب باید بتواند کارآیی برنامه‌های واقعی را منعکس نماید و علی‌الخصوص benchmark باید دارای ترکیب واقعی از دستورات باشد (یعنی شامل ترکیبی از دستورات باشد که در برنامه‌های واقعی ظاهر می‌شوند). امروزه ثابت شده است که بهترین benchmark ها برنامه‌های واقعی هستند. تعدادی از benchmark های معمول عبارتند از:

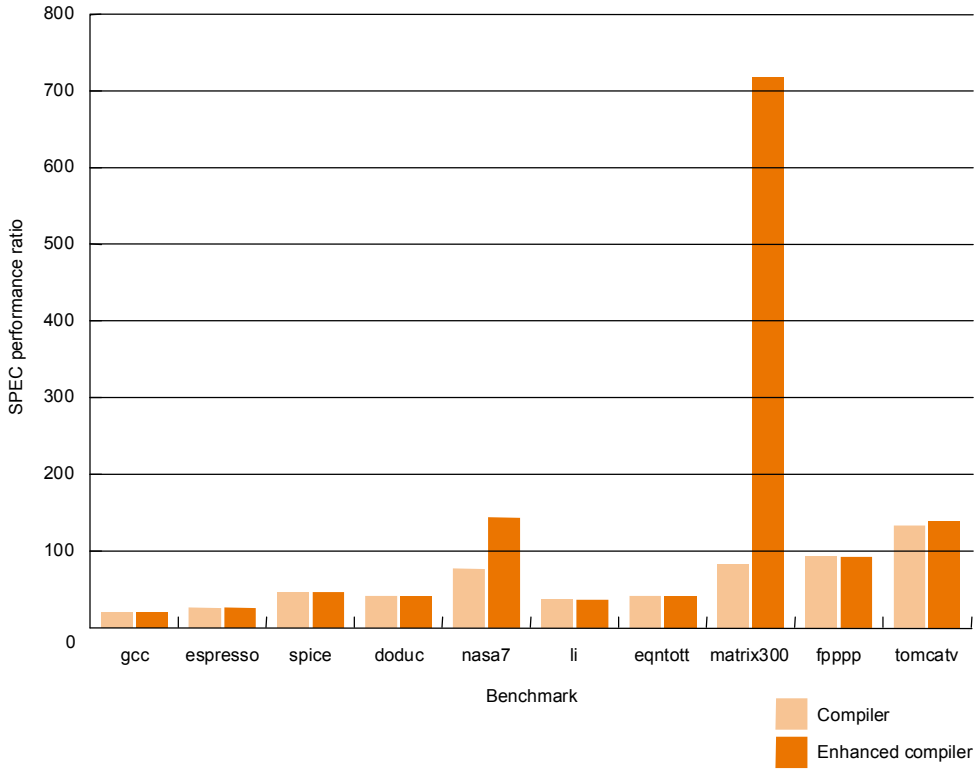
- Adobe Photoshop for image processing
- BABCO SYSmark for office applications
- Unreal Tournament 2003 for 3D games
- SPEC (System Performance Evaluation Cooperative)

امروزه برخی از سازنده‌های کامپیوتر بر اساس اطلاعاتی که از کاربران و benchmark های آنان دارند، روشهای نادرستی را در پیش می‌گیرند و اطلاعات غلطی را به کابران می‌دهند. این سازندگان یا از تکنیکهای کامپایلری استفاده می‌کنند که برای یک benchmark مشخص کدهای بهینه‌ای تولید می‌کنند که بر روی سخت افزار سریعتر اجرا می‌شوند و یا اینکه از تکنیکهای سخت‌افزاری استفاده می‌کنند که یک ترکیب مشخص از دستورات را که مربوط به یک benchmark مشخص است را با سرعت بالاتری اجرا می‌کنند. استفاده از برنامه‌های واقعی به عنوان benchmark استفاده از روشهای فوق برای افزایش

سرعت اجرا شدن benchmark ها را تقریباً غیر ممکن می‌سازد حتی اگر بهبودی نیز حاصل شود این بهبود چون مربوط به برنامه‌های واقعی است، قابل استفاده خواهد بود.

استفاده کردن از benchmark هایی که اندازه آنها کوچک است و یا کارآیی آنها به قسمت کوچکی از کد برنامه بستگی دارد خطر بهینه سازی‌های کامپایلری یا سخت افزاری را به همراه خواهد داشت. به طور مثال مجموعه benchmark های SPEC در ابتدا برای این منظور انتخاب شدند که از برنامه‌های واقعی برای اندازه‌گیری کارآیی استفاده کنند. متأسفانه اولین نسخه از مجموعه SPEC که در سال ۱۹۸۹ وارد بازار شد، دارای یک benchmark به نام matrix300 بود که فقط دارای یک سری عملیات ضرب ماتریسی بود. در واقع ۹۹ درصد از زمان اجرا مربوط به یک خط از این برنامه بود. این واقعیت که زمان خیلی زیادی از اجرای برنامه مربوط به یک خط از برنامه بود که در آن یک خط، یک عملیات مشخص انجام می‌گرفت، باعث شد تا چند کمپانی کامپیوتری کامپایلرهایی را تولید و یا خریداری کنند تا زمان اجرای این benchmark را بهینه سازی کنند. شکل ۲-۳ نسبت کارآیی[۱۲] (معکوس زمان اجرا) را برای یک ماشین با دو کامپایلر مختلف نشان می‌دهد. برنامه‌های این شکل مجموعه benchmark های SPEC89 می‌باشند. در این شکل برای هر برنامه دو میله وجود دارد که میله سمت راستی مربوط به کامپایلر بهینه‌سازی شده برای matrix300 می‌باشد. همان طور که مشاهده می‌شود کامپایلر بهینه-سازی شده برای matrix300 اثری بر روی ۸ مورد از ۱۰ مورد از برنامه‌های مجموعه benchmark های SPEC89 ندارد، اما کارآیی matrix300 را بیش از ۹ برابر بهبود داده است. کاربری که براساس matrix300 قضاوت می‌کند حسابی دچار اشتباه بزرگی خواهد شد به دلیل اینکه بهبود داده شده فقط مربوط به matrix300 بوده و برای برنامه‌های دیگر کاربردی نخواهد داشت.

[12] - Performance ratio



شکل ۲-۳: نسبت کارآیی مجموعه benchmark های SPEC89 برای یک ماشین با دو کامپایلر مختلف

ضعف مربوط به matrix300 باعث شد که این benchmark در نسخه بعدی مجموعه SPEC یعنی SPEC92 کنار گذاشته شود.

حال سؤال این است که چرا همه مردم از برنامه‌های واقعی برای اندازه‌گیری کارآیی استفاده نمی‌کنند؟ یکی از دلایل این است که benchmark های کوچک به علت کوچک بودن به راحتی کامپایل و شبیه-سازی می‌شوند (حتی بعضی مواقع دستی کامپایل می‌شوند) و به همین دلیل در شروع طراحی جذابیت‌های خاصی دارند. مخصوصاً چون در شروع طراحی یک ماشین هنوز کامپایلری برای آن نوشته نشده است، این benchmark ها جایگاه خاصی دارند. دلیل دیگری که برای استفاده از benchmark های کوچک وجود دارد این است که آنها راحت‌تر از برنامه‌های بزرگ استاندارد سازی می‌شوند و بنابراین نتایج منتشر شده زیادی برای benchmark های کوچک وجود.

اگرچه استفاده از benchmark های کوچک در مراحل اولیه طراحی قابل توجیه است، اما هیچ دلیل قانع کننده‌ای وجود ندارد که از آنها برای ارزیابی کارآیی سیستم‌های کامپیوتری که وارد بازار شده‌اند استفاده کنیم. در گذشته یکی از دلایلی که از benchmark های کوچک به جای برنامه‌های بزرگ استفاده می‌شد این بود که آنها بر روی ماشین‌های مختلف به راحتی پورت می‌شدند و از طریق آنها می‌شد سیستم‌های مختلف را مقایسه نمود. اما امروزه این دلیل خیلی درست نیست و استفاده از

benchmark های کوچک فقط در مراحل اولیه طراحی مجاز است و بعد از آن باید از برنامه‌های واقعی برای ارزیابی کارآیی ماشین‌ها استفاده شود.

پس از آنکه ما مجموعه مناسبی از benchmark ها را انتخاب نموده و توسط آنها کارآیی را اندازه‌گیری کردیم، می‌توانیم یک گزارش برای کارآیی تهیه کنیم. توضیحاتی که در گزارش آورده می‌شود باید طوری باشد که اندازه‌گیری ما قابل تکرار باشد. گزارش ما باید شامل یک لیست از همه چیزهایی باشد که ممکن است یک فرد دیگر برای تکرار آزمایشهای ما به آنها نیاز داشته باشد. این لیست باید شامل نسخه سیستم عامل، کامپایلرها، ورودی‌ها و همچنین نحوه پیکربندی ماشین باشد. به طور مثال توصیف سیستمی[۱۳] ماشینی که برای بدست آوردن نتایج شکل ۲-۳ از آن استفاده شده است در شکل ۲-۴ نشان داده شده است

Hardware	
Model number	Powerstation 550
CPU	41.67 MHz POWER 4164
FPU	Integrated
Number of CPUs	1
Cache size per CPU	64K data / 8K instruction
Memory	64 MB
Disk subsystem	2 400-MB SCSI
Network interface	NA
Software	
OS type and rev	AIX v3.1.5
Compiler rev	AIX XL C/6000 Ver. 1.1.5 AIX XL Fortran Ver. 2.2
Other software	None
File system type	AIX
Firmware level	NA
System	
Tuning parameters	None
Background load	None
System state	Multiuser (single-user-login)

شکل ۲-۴: توصیف سیستمی ماشین استفاده شده برای اندازه‌گیری‌های شکل ۲-۳

[13] - System description

Benchmark های Synthetic

یک synthetic benchmark، یک برنامه است که فقط برای منظور اندازه‌گیری کارآیی نوشته شده است و معمولاً اندازه آن کوچک بوده و به راحتی بر روی CPU های مختلف پورت می‌شود. بنابراین برای مقایسه سیستم‌ها راحت‌تر می‌باشند. هدف از طراحی synthetic benchmark ها این بوده که یک برنامه واحد داشته باشیم که ویژگیهای تعداد زیادی از برنامه‌ها را داشته باشد و تعداد تکرار دستورات در داخل برنامه متناسب با تعداد تکرار دستورات در تعداد زیادی از benchmark ها باشد. معروفترین benchmark های synthetic عبارتند از Whetstone و Dhrystone .

چون این نوع benchmark ها واقعی نیستند نشان دهنده هیچ چیز خاصی نیستند. به طور مثال هیچ خروجی در اختیار کاربر قرار نمی‌دهند که کاربر بداند برنامه به درستی اجرا شده است و نیز اگر بر روی یک ماشین کارآیی آنها بهبود پیدا کرد هیچ دلیلی وجود ندارد که برای برنامه‌های واقعی نیز چنین باشد و کارآیی آن ماشین برای سایر برنامه‌ها نیز بهبود یابد. اندازه این benchmark ها کوچکتر است و بنابراین به راحتی تکنیکهای بهینه سازی کامپایلر و سخت‌افزار برای آنها قابل اعمال است.

نکته: بنا به دلایل فوق synthetic benchmark ها برای اندازه گیری کارآیی انتخاب مناسبی نیستند.

۲-۵- مقایسه کردن و خلاصه سازی کارآیی

حال که یاد گرفتیم برنامه‌هایی را به عنوان benchmark انتخاب نمائیم و برای مقایسه از معیارهای زمان اجرا و throughput استفاده کنیم، ممکن است تصور کنید که مقایسه کردن کارآیی کار سر راستی باشد. اما معمولاً به جای یک benchmark از چند benchmark استفاده می‌شود و ما باید یاد بگیریم که چگونه کارآیی یک گروه از benchmark ها را اندازه‌گیری کنیم. اگر چه خلاصه کردن یک مجموعه از اندازه‌گیری‌ها اطلاعات کمی در اختیار قرار می‌دهد، ولی بازار و حتی کاربران اغلب ترجیح می‌دهند که یک عدد ساده از کارآیی داشته باشند و با استفاده از آن مقایسه‌های خود را انجام دهند. حال سؤال کلیدی این است که این عدد چگونه محاسبه می‌شود؟ شکل ۲-۵ زمان اجرای دو برنامه را بر روی دو ماشین مختلف نشان می‌دهد.

	Computer A	Computer B
Program 1 (seconds)	1	10
Program 2 (seconds)	1000	100
Total time (seconds)	1001	110

شکل ۲-۵: زمان اجرای دو برنامه بر روی دو ماشین مختلف

طبق تعریفی که ما از سرعت داشتیم، عبارتهای زیر در مورد شکل ۲-۵ صادق هستند:

- ماشین A برای برنامه ۱، ۱۰ مرتبه سریعتر از ماشین B عمل می‌کند.
- ماشین B برای برنامه ۲، ۱۰ مرتبه سریعتر از ماشین A عمل می‌کند.

این دو جمله خیلی گیج کننده هستند. در نهایت کدام ماشین سریعتر است؟

ساده‌ترین راه برای خلاصه‌سازی کارآیی در این موارد که بیش از چند برنامه داریم این است که مجموع زمانهای اجرا را نسبت به هم مقایسه کنیم. بنابراین:

$$\frac{\text{کارآیی ماشین B}}{\text{کارآیی ماشین A}} = \frac{\text{مجموع زمان اجرای برنامه‌ها بر روی ماشین A}}{\text{مجموع زمان اجرای برنامه‌ها بر روی ماشین B}} = \frac{۱۰۰۱}{۱۱۰} = ۹/۱$$

یعنی اینکه ماشین B برای مجموعه برنامه‌های ۱ و۲ ، ۹/۱ مرتبه سریعتر از ماشین A است.

این نوع خلاصه سازی مستقیماً متناسب با زمان اجرا می‌باشد. اگر workload ما به صورتی باشد که برنامه‌های ۱ و ۲ را به یک اندازه اجرا کنیم، رابطه فوق می‌تواند برای مقایسه ماشین‌های مختلف استفاده شود.

بعضی مواقع به جای رابطه فوق میانگین زمان اجرا که آن هم متناسب با زمان اجراست را به کار می-برند. این میانگین، میانگین حسابی[۱۴] یا AM نامیده می‌شود و از فرمول زیر بدست می‌آید:

$$AM = \frac{1}{n}\sum time_i$$

که در آن $time_i$ زمان اجرای برنامه i ام موجود در workload متشکل از n برنامه است. هر چقدر AM کوچکتر باشد، کارآیی ماشین بالاتر خواهد بود.

AM زمانی قابل استفاده است که برنامه‌های موجود در workload به یک اندازه اجرا شوند، اگر غیر از این باشد می‌توانیم برای هر برنامه یک وزنی اختصاص دهیم که نشان دهنده تعداد تکرار برنامه در آن workload باشد. به طور مثال اگر ۲۰ درصد کارها در workload برنامه ۱ و ۸۰ درصد کارها در workload برنامه ۲ باشد در این صورت وزنها به صورت ۰/۲ و ۰/۸ خواهند بود. با استفاده از جمع کردن حاصل‌ضرب فاکتورهای وزن در زمان اجرا فرمول میانگین حسابی وزن‌دار[۱۵] یا WAM بدست می‌آید:

$$WAM = \sum_{i=1}^{n} weight_i \times time_i$$

[14] - Arithmetic Mean
[15] - Weighted Arithmetic Mean

فرمول میانگین حسابی حالت خاصی از فرمول میانگین حسابی وزن‌دار می‌باشد که در آن وزن همه برنامه‌ها مساوی است.

۲-۶- موضوع واقعی: benchmark های SPEC95 و کارآیی پردازنده‌ها

SPEC معروفترین مجموعه benchmark های CPU می‌باشد. SPEC که مخفف عبارت System Performance Evaluation Cooperative (مجموعه ارزیابی کننده کارآیی سیستم) می‌باشد، در سال ۱۹۸۹ توسط تعدادی از کمپانی‌های کامپیوتر برای این منظور ایجاد شد که اندازه‌گیری و خلاصه سازی کارآیی از طریق یک پروسه اندازه‌گیری کنترل شده، بهبود داده شود و همچنین benchmark های واقعی‌تری برای اندازه‌گیری کارآیی مورد استفاده قرار گیرند. تاکنون چهار نسخه از SPEC وارد بازار شده است: SPEC89، SPEC92، SPEC95 و SPEC2000. مجموعه SPEC95 نسخه سال ۹۵ SPEC می‌باشد که شامل ۸ برنامه اعداد صحیح و ۱۰ برنامه ممیز شناور می‌باشد. مجموعه برنامه‌های SPEC95 در شکل ۲-۶ نشان داده شده است.

Benchmark	Description
go	Artificial intelligence; plays the game of Go
m88ksim	Motorola 88k chip simulator; runs test program
gcc	The Gnu C compiler generating SPARC code
compress	Compresses and decompresses file in memory
li	Lisp interpreter
ijpeg	Graphic compression and decompression
perl	Manipulates strings and prime numbers in the special-purpose programming language Perl
vortex	A database program
tomcatv	A mesh generation program
swim	Shallow water model with 513 x 513 grid
su2cor	quantum physics; Monte Carlo simulation
hydro2d	Astrophysics; Hydrodynamic Naiver Stokes equations
mgrid	Multigrid solver in 3-D potential field
applu	Parabolic/elliptic partial differential equations
trub3d	Simulates isotropic, homogeneous turbulence in a cube
apsi	Solves problems regarding temperature, wind velocity, and distribution of pollutant
fpppp	Quantum chemistry
wave5	Plasma physics; electromagnetic particle simulation

شکل ۲-۶: مجموعه benchmark های SPEC95

معمولاً دو گزارش جداگانه برای مجموعه برنامه‌های صحیح و برنامه‌های ممیز شناور داده می‌شود. روند اندازه‌گیری با استفاده از برنامه‌های مجموعه SPEC به این صورت است که در ابتدا عددهای اندازه‌گیری شده برای زمان اجرا با تقسیم کردن زمان اجرای برنامه بر روی ماشین SUN sparc 10/40 به زمان اجرای بدست آمده بر روی ماشینی که می‌خواهیم کارآیی آن را اندازه بگیریم، به صورت نرمال

شده درآورده می‌شود. سپس این عدد نرمال شده به یک معیار برای اندازه‌گیری کارآیی منجر می‌شود که نسبت SPEC ratio نامیده می‌شود. هر چقدر عدد بدست آمده برای SPEC ratio بزرگتر باشد نشان دهنده کارآیی بالاتر می‌باشد (در واقع SPEC ratio با زمان اجرا نسبت معکوس دارد). خلاصه-سازی‌های SPECint95 و SPECfp95 با استفاده از میانگین هندسی SPEC ratio ها بدست می‌آید که اولی مربوط به برنامه‌های صحیح و دومی مربوط به برنامه‌های ممیز شناور مجموعه SPEC می‌باشد. فرمول میانگین هندسی[16] به صورت زیر است:

$$\sqrt[n]{\prod_{i=1}^{n}(execution\ time\ ratio)_i}$$

که در آن $(execution\ time\ ratio)_i$ زمان اجرای نرمال شده بر اساس یک ماشین مرجع برای برنامه i- ام مجموعه benchmark ها می‌باشد.

همان طور که قبلاً نیز توضیح داده شده است برای یک ISA مفروض، افزایش کارآیی CPU می‌تواند به یکی از سه روش زیر صورت بگیرد:

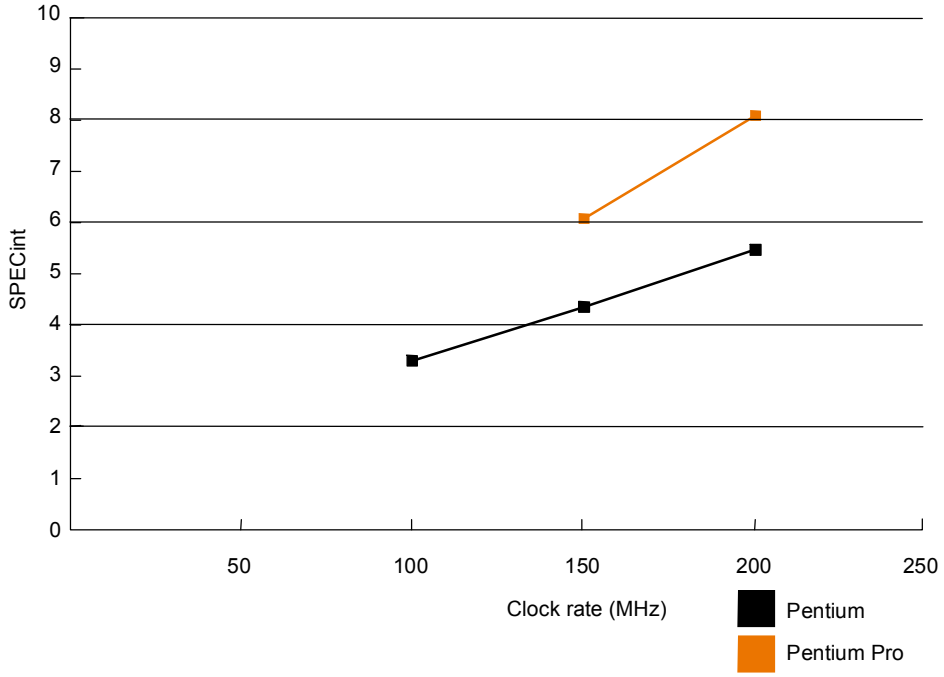
۱- افزایش فرکانس کلاک

۲- بهبود ساختار پردازنده که باعث کم شدن CPI شود

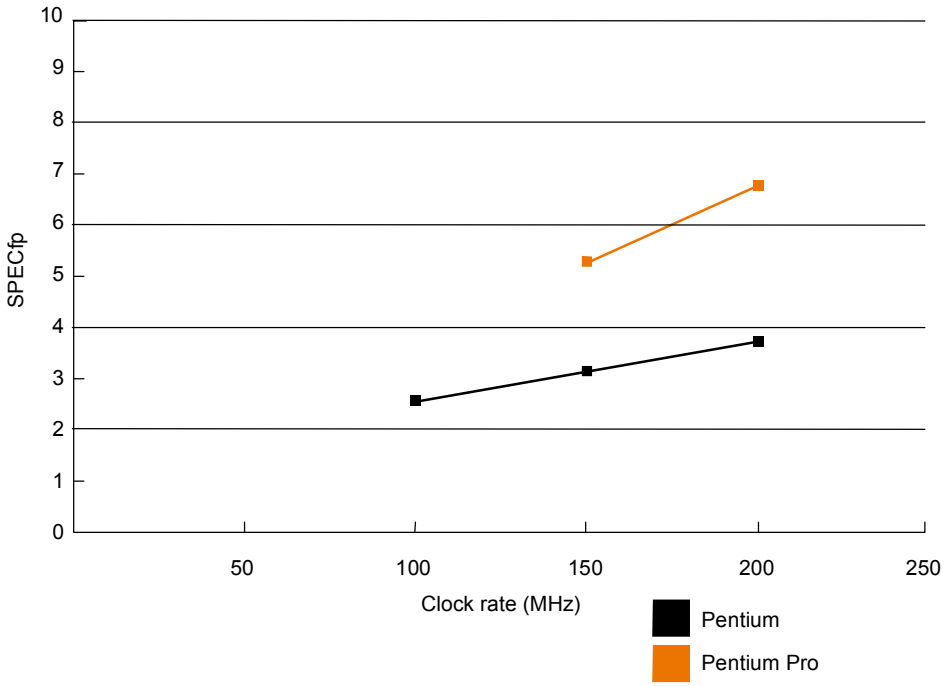
۳- بهبودهای کامپایلری که باعث کم شدن تعداد دستورات یا تولید دستوراتی با میانگین CPI پایین‌تر می‌شود

برای نشان دادن بهبودهای فوق بر روی کارآیی، شکل‌های ۲-۷ و ۲-۸ به ترتیب اندازه‌گیری‌های SPECint95 و SPECfp95 را برای پردازنده‌های Pentium و Pentium pro نشان می‌دهند. چون SPEC نیاز به این دارد که حتماً روی سخت‌افزار واقعی اجرا شود و سیستم حافظه تأثیر زیادی بر روی کارآیی دارد، سیستم‌های دیگری که بر اساس این پردازنده‌ها ساخته می‌شوند ممکن است مقداری با این شکلها متفاوت باشند چون سیستم حافظه آنها ممکن است قدری متفاوت باشد. ماشین‌های اینتل که در اینجا اندازه‌گیری شده‌اند سیستم‌های حافظه و کامپایلر قوی‌تری استفاده کرده‌اند و این بدان معناست که اکثر ماشین‌های دیگری که با استفاده از این پردازنده ساخته می-شوند کارآیی پایین‌تری برای benchmarks های SPEC خواهند داشت.

[16] - Geometric Mean



شکل۲–۷: SPECint محاسبه شده برای پردازنده‌های Pentium و Pentium pro در فرکانس‌های مختلف



شکل۲–۸: SPECfp محاسبه شده برای پردازنده‌های Pentium و Pentium pro در فرکانس‌های مختلف

چند برداشت مهم از شکلهای ۲–۷ و ۲–۸ قابل مشاهده است. مهمترین آنها بهبود کارآیی پردازنده Pentium pro نسبت به Pentium می‌باشد. در یک فرکانس فرضی SPECint نشان می‌دهد که

Pentium pro، ۱/۴ تا ۱/۵ برابر سریعتر از Pentium می‌باشد و SPECfp95 نشان می‌دهد که Pentium pro ، ۱/۷ تا ۱/۸ برابر سریعتر می‌باشد. اگرچه بهبودهای کامپایلری مشخص برای هر پردازنده‌ای وجود دارد، اکثر بهبودهای کارآیی برای Pentium pro از طریق بهبودهای ساختاری (سخت‌افزاری) ایجاد شده است.

برداشت مهم دیگری که از شکلها به دست می‌آید این است که وقتی که فرکانس کلاک با استفاده از ضریب مشخص اضافه می‌شود، کارآیی پردازنده با ضریب کمتری اضافه می‌شود. به طور مثال وقتی که فرکانس کلاک Pentium از ۱۰۰ مگاهرتز به ۲۰۰ مگاهرتز افزایش پیدا می‌کند (دو برابر می‌شود)، کارآیی SPECint 95 فقط ۱/۷ و کارآیی SPECfp95 فقط ۱/۴ برابر بهبود می‌یابد. دلیل این امر به سیستم حافظه پردازنده بر می‌گردد. در کل، چون سرعت حافظهٔ اصلی تغییر چندانی نمی‌کند، اضافه شدن سرعت پردازنده در گلوگاه سیستم حافظه گرفتار می‌شود. این تأثیر در benchmark های ممیز شناور بیشتر مشهود است چون آنها نسبت به برنامه‌های integer بزرگتر هستند. این رفتار مثال خوبی برای قانون امدال می‌باشد که بیانگر میزان بهبود بر اساس میزان مؤثر بودن آن می‌باشد.

در مقام مقایسه، میزان افزایش کارآیی پردازنده Pentium pro، با اضافه شدن فرکانس کلاک بهتر از Pentium می‌باشد. هر چند که این میزان افزایش برابر ضریب افزایش کلاک نیست. به طور مثال، بهبود کارآیی SPECfp95 از فرکانس ۱۵۰ به ۲۰۰ مگاهرتز در Pentium pro، ۱/۲۴ می‌باشد در حالی که پردازنده Pentium بهبودی در حدود ۱/۱۸ را برای این بازه نشان می‌دهد. در فصل ۷ در مورد تأثیر سیستم حافظه در کارآیی پردازنده صحبت خواهد شد و اینکه چرا پردازنده Pentium pro سیستم حافظه بهتری دارد.

۲-۸ نکات پایانی

ما در این فصل بر روی performance یا همان کارآیی سیستم‌ها و نحوه اندازه‌گیری آن تمرکز کردیم. تمرکز کردن تنها بر روی کارآیی بدون در نظر گرفتن هزینه[۱۷] خیلی منطقی نیست و ما باید بین این دو پارامتر تعادل برقرار کنیم.

[17] - Cost

ما در این فصل دیدیم که با استفاده از زمان اجرای برنامه‌های واقعی بر روی پردازنده‌ها، یک روش قابل اطمینان برای اندازه‌گیری و گزارش کردن کارآیی وجود دارد. این زمان اجرا وابسته به تعدادی فاکتور مهم می‌باشد که این وابستگی در فرمول زیر مشاهده می‌شود:

زمان اجرا = تعداد دستورات × CPI × پریود کلاک

ما به وفور از این فرمول و فاکتورهای آن بهره خواهیم گرفت. همان طور که توضیح داده شد هیچ کدام از این فاکتورها به تنهایی نمی‌توانند نشان دهنده کارآیی باشند و فقط حاصل‌ضرب آنها که مساوی زمان اجراست قابل استفاده است. مطمئناً دانستن این فرمول به تنهایی برای طراحی کردن یا ارزیابی کردن یک کامپیوتر نمی‌تواند کافی باشد. ما باید بفهمیم که چگونه جنبه‌های مختلف طراحی، این فاکتورهای مهم را تحت تأثیر قرار می‌دهند. این نوع دانش شامل موارد متعددی می‌شود از جمله: تأثیر معماری مجموعه دستورات بر تعداد دستورات دینامیکی، تأثیر pipelining و سیستم حافظه بر روی CPI، و تأثیر تکنولوژی و سازمان یک کامپیوتر بر روی فرکانس کلاک. هنر طراحی کامپیوتر فقط اعمال کردن عدد در فرمول کارآیی نیست بلکه مشخص کردن این موضوع هست که طراحی‌های مختلف چگونه کارآیی و هزینه را تحت تأثیر قرار می‌دهند.

اکثر کاربران کامپیوتر به هر دو مورد هزینه و کارآیی دقت می‌کنند. فهمیدن ارتباط بین جنبه‌های مختلف طراحی و کارآیی آن کار مشکلی است و مشخص کردن هزینه قابلیتهای مختلف یک طرح نیز کار به مراتب مشکل‌تری است. هزینه یک ماشین فقط به هزینه قسمتهای تشکیل دهنده آن محدود نیست، بلکه هزینه‌هایی همچون هزینه‌های نیروی انسانی برای جمع‌آوری سیستم[۱۸] ، هزینه تحقیقات، هزینه توسعه سیستم، تبلیغات بازار و ... را نیز باید به آن اضافه نمود. امروزه به دلیل پیشرفت سریعی که در تکنولوژی‌های پیاده سازی ایجاد می‌شود، اهمیت دادن بیش از حد به مورد هزینه در طول مدت‌هایی حدود ۶ ماه یا یک سال خیلی منطقی به نظر نمی‌رسد.

و نکته آخر اینکه معماری‌های مختلف کامپیوتر توسط کارآیی و هزینه سنجیده می‌شوند و پیدا کردن تعادل بین این دو همیشه به عنوان یک هنر در طراحی کامپیوتر مطرح بوده است.

مطالب مهم این فصل:

- Performance یا همان کارآیی یک معیار مهم برای مقایسه سیستم‌هاست.
- دو روش معتبر برای اندازه‌گیری کارآیی وجود دارد: محاسبه زمان اجرا و محاسبه throughput
- فرمول اصلی کارآیی: زمان اجرا = تعداد دستورات × CPI × پریود کلاک

[18] - Assemble

- بهترین راه برای اندازه‌گیری کارآیی این است که از برنامه‌های واقعی استفاده کنیم.
- Benchmark ها برنامه‌هایی هستند که برای اندازه‌گیری کارآیی انتخاب می‌شوند و باید به اندازه کافی بزرگ باشند که اجرای آنها توسط کامپایلر و یا سخت افزار بهینه سازی نگردد و تا حد ممکن از برنامه‌های واقعی انتخاب شوند. مهمترین مجموعه benchmark های شناخته شده مجموعه SPEC می‌باشد.
- قانون امدال بیان می‌کند که ما از یک بهبود داده شده در طرح چقدر می‌توانیم انتظار بهبود کارآیی را داشته باشیم. میزان بهبود کارآیی به میزان مؤثر بودن تغییر داده شده بستگی دارد.
- اگر به جای یک برنامه چند برنامه برای ارزیابی کارآیی استفاده شود و بخواهیم کارآیی را خلاصه سازی کنیم از فرمول‌های میانگین حسابی و یا میانگین حسابی وزن‌دار استفاده می-کنیم. میانگین هندسی نیز قابل استفاده است ولی میانگین هندسی دارای ضعف‌هایی است که نمی‌توان آن را به عنوان یک معیار خوب قبول کرد.
- گزارش تهیه شده برای کارآیی باید طوری باشد که توسط دیگران قابل تکرار باشد. بنابراین باید همه چیزهایی که برای تکرار یک آزمایش لازم است در گزارش آورده شود.