جزوه جامع

مبانی کامپیوتر و برنامه‌سازی

از مفاهیم پایه تا C و C++

مولف: استاد ارسطو خلیلی فر

انتشارات بابان
BABAN.IR

موسسه بابان

راهیان ارشد و دکتری انتشارات بابان

مبانی کامپیوتر و برنامه‌سازی

فصل اول: مبانی کامپیوتر (درسنامه)

(ویژه کنکور ارشد و دکتری ۱۴۰۶)

ویژه‌ی داوطلبان کنکور ارشد و دکتری مهندسی کامپیوتر و IT

براساس کتب مرجع

پل جی. دیتل و هاروی ام. دیتل ، جری آر. هنلی و الیوت بروس کافمن

برایان ویلسون کرنیگان و دنیس مک‌آلیستر ریچی

ارسطو خلیلی‌فر

کلیه‌ی حقوق مادی و معنوی این اثر در سازمان اسناد و کتابخانه‌ی ملی ایران به ثبت رسیده است.

۱
مبانی کامپیوتر

خلاصه فصل: داخل کامپیوتر بخشی وجود دارد به نام CPU که داخل آن، بخش ارزشمندی قرار دارد به نام ALU که قدرت انجام سریع **عملیات ریاضی** (جمع، تفریق، ضرب، تقسیم) و **عملیات منطقی** (NOT, XOR, OR, AND) دارد. برای استفاده از این قدرت کامپیوتر باید برنامه کامپیوتری (نرم‌افزار) بنویسیم، سپس سیستم عامل، حافظه (RAM) و پردازنده (CPU) را به برنامه کامپیوتری اختصاص می‌دهد و پردازنده آنرا اجرا می‌کند. در این درس برنامه‌ها را به زبان ساخت‌یافته C و شیءگرا C++ می‌نویسیم.

کلمات کلیدی: کامپیوتر، برنامه، الگوریتم، حافظه، پردازنده و سیستم عامل

مقدمه

کامپیوتر (Computer) در مفهوم لغوی به معنی محاسبه‌کننده و شمارشگر است.

سلام عرض می‌کنم خدمت دوستان عزیز. به دوره مبانی کامپیوتر و برنامه‌سازی خوش آمدید. من ارسطو خلیلی‌فر هستم و در این دوره همراه شما خواهم بود.

اجازه بدهید درس را با یک سؤال ساده شروع کنیم: کامپیوتر می‌تواند در هر ثانیه میلیاردها عملیات انجام دهد؛ اما از کجا می‌فهمد چه کاری را باید انجام دهد؟ پاسخ این است: **از طریق برنامه‌ای که انسان برای آن می‌نویسد.**

هدف ما در این آموزش فقط یادگرفتن چند دستور برنامه‌نویسی نیست. ابتدا می‌خواهیم با اجزای اصلی کامپیوتر، نحوه پردازش اطلاعات و چگونگی اجرای برنامه‌ها آشنا شویم. سپس یاد می‌گیریم چگونه یک مسئله را تحلیل کنیم، برای آن الگوریتم بنویسیم و راه‌حل خود را با استفاده از زبان C به یک برنامه قابل‌اجرا تبدیل کنیم.

مطالب را از پایه و به‌صورت مرحله‌به‌مرحله پیش می‌بریم؛ بنابراین اگر پیش‌زمینه‌ای در برنامه‌نویسی ندارید، جای نگرانی نیست. در کنار هر مبحث نیز مثال‌ها و تمرین‌هایی بررسی می‌کنیم تا مفاهیم کاملاً قابل‌درک شوند.

در فصل اول، ابتدا با مفهوم کامپیوتر، اجزای اصلی آن و نقش سخت‌افزار و نرم‌افزار آشنا می‌شویم و سپس به‌تدریج وارد دنیای برنامه‌نویسی خواهیم شد. در ادامه درس را شروع می‌کنیم.

۱- کامپیوتر ماشینی است که، عملیات واگذار شده به آن را، در صورتی که از قبل جزئیات کار برایش مشخص باشد و به آن دستور داده شود انجام می‌دهد.

۲- کامپیوتر به ماشینی اطلاق می‌شود که پردازنده (CPU) و حافظه (RAM) دارد و برنامه‌پذیر است و می‌تواند **عملیات ریاضی** و **عملیات منطقی** را با **سرعت** و **دقتی** بسیار بیشتر از انسان انجام دهد.

تفاوت عمده کامپیوتر با سایر ماشین‌ها، برنامه‌پذیر بودن آن است. به عنوان مثال، کامپیوتر می‌تواند نمرات یک دانشجو را از ما تحویل بگیرد و معدل وی را محاسبه کند و به ما تحویل دهد. اما دقت داشته باشید که کامپیوتر، معنای معدل را نمی‌داند، بلکه باید قبلاً به آن بگوییم که شیوه محاسبه معدل چگونه است، تا براساس آن، معدل را محاسبه کرده، به ما تحویل دهد.

دستورالعمل‌هایی که به کامپیوتر داده می‌شود تا کامپیوتر براساس آن، کاری را انجام دهد، برنامه کامپیوتری نام دارد. بنابراین، کامپیوتر برای اجرای هر کاری نیاز به برنامه دارد. برنامه‌ها معمولاً تحت یک زبان برنامه‌نویسی، نظیر C و C++ نوشته می‌شوند.

امروزه رایانه‌های شخصی می‌توانند در هر ثانیه میلیاردها دستور را اجرا کنند و توان ابررایانه‌ها از این مقدار نیز بسیار بیشتر است. با وجود این قدرت پردازشی، کامپیوتر بدون برنامه نمی‌تواند کاری انجام دهد؛ زیرا برنامه مشخص می‌کند داده‌ها چگونه پردازش شوند. **ساز بدون نوازنده، کاربردی ندارد.** هر برنامه مجموعه‌ای از دستورالعمل‌هاست که برنامه‌نویس آن‌ها را می‌نویسد تا نوع و ترتیب عملیات کامپیوتر را تعیین کند.

چرا باید درس مبانی کامپیوتر و برنامه‌سازی را بخوانیم؟

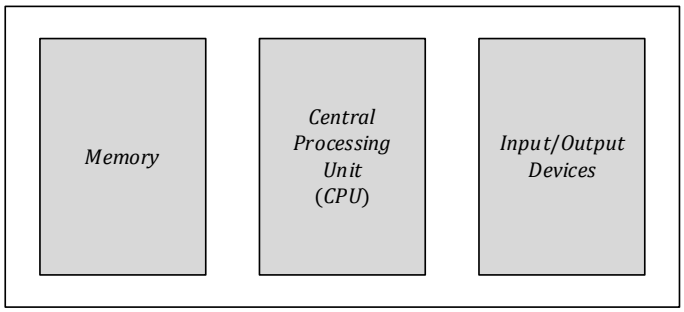
برای اینکه بتوانیم از قدرت انجام سریع **عملیات ریاضی** (جمع، تفریق، ضرب، تقسیم) و **عملیات منطقی** (NOT, XOR, OR, AND) در کامپیوتر استفاده کنیم باید برنامه کامپیوتری بنویسیم در این شرایط CPU خط به خط و به ترتیب دستورهای برنامه را می‌خواند و اجرا می‌کند تا نتیجه مورد نظر تولید شود. با یادگیری مبانی کامپیوتر و برنامه‌سازی، می‌آموزیم چگونه این برنامه‌ها را طراحی و ایجاد کنیم.

سیستم کامپیوتری

به طور کلی هر **سیستم کامپیوتری** از دو بخش اصلی **(۱) سخت‌افزار** و **(۲) نرم‌افزار** تشکیل شده است:

۱- سخت‌افزار: (Hardware)

بخش فیزیکی سیستم کامپیوتری است. اجزای فیزیکی و قابل لمس کامپیوتر مانند پردازنده (CPU)، حافظه (RAM)، صفحه‌کلید، نمایشگر و تجهیزات ذخیره‌سازی. پیشرفت سریع فناوری‌های سخت‌افزاری، هزینه ساخت و استفاده از کامپیوترها را به‌طور چشمگیری کاهش داده است. کامپیوترهایی که در گذشته فضای یک اتاق بزرگ را اشغال می‌کردند و میلیون‌ها دلار قیمت داشتند، امروزه در قالب تراشه‌های سیلیکونی بسیار کوچک و با هزینه‌ای بسیار کمتر ساخته می‌شوند. این پیشرفت باعث شده است رایانه‌ها به محصولاتی قدرتمند، ارزان و فراگیر تبدیل شوند. یکی از مهم‌ترین مفاهیم مرتبط با پیشرفت سخت‌افزار، **قانون مور** است که گوردون مور آن را در دهه ۱۹۶۰ میلادی مطرح کرد. براساس این قانون، تعداد ترانزیستورهای قابل‌جاسازی روی یک تراشه تقریباً هر دو سال دو برابر می‌شود. افزایش تعداد ترانزیستورها معمولاً موجب **افزایش قدرت پردازش پردازنده‌ها** و **ظرفیت حافظه‌ها** می‌شود، بدون آنکه هزینه تولید آن‌ها به همان نسبت افزایش یابد. البته قانون مور نمی‌تواند تا ابد ادامه یابد؛ زیرا ترانزیستورها در حال نزدیک‌شدن به ابعاد اتمی هستند و کوچک‌ترکردن آن‌ها دشوارتر و پرهزینه‌تر می‌شود. بااین‌حال، فناوری‌هایی مانند تراشه‌های سه‌بعدی با قراردادن لایه‌ها روی یکدیگر، امکان افزایش قدرت پردازشی و ظرفیت حافظه را فراهم می‌کنند. یک کامپیوتر در حالت انتزاعی، از سه بخش اصلی زیر تشکیل شده است:

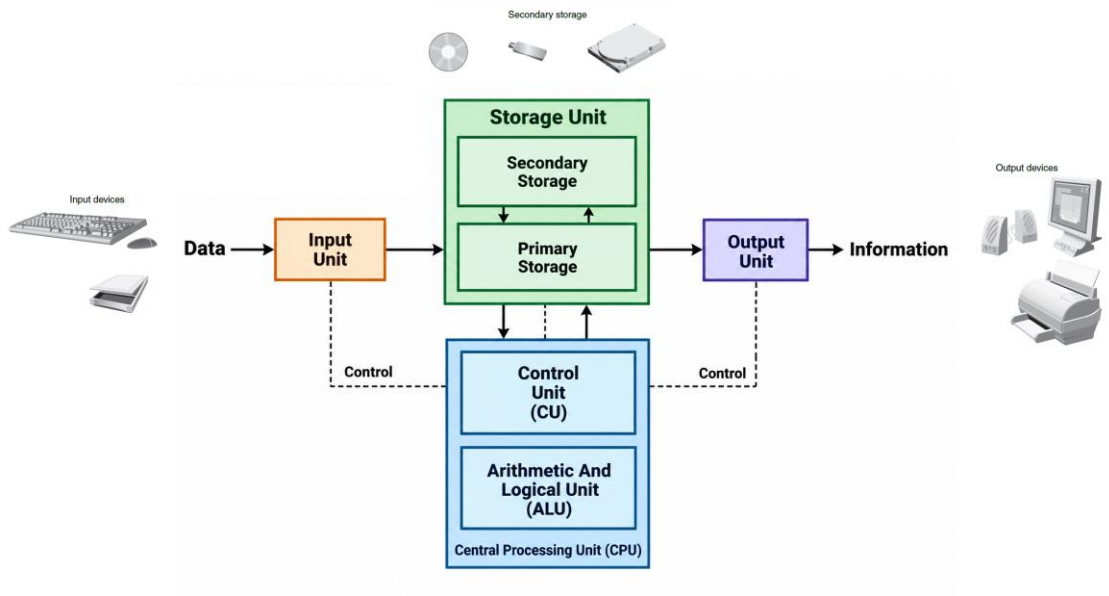


اجزای کلی یک کامپیوتر

این سه جزء که با یکدیگر در تعاملند، وظایف مشخصی را برای عملکرد کلی سیستم بر عهده دارند:

- **واحد پردازش مرکزی $(Central\ Processing\ Unit - CPU)$**
- **حافظه $(Memory)$**
- **دستگاه‌های ورودی/ خروجی $(Input/Output\ Devices - I/O)$**

یک کامپیوتر در حالت غیرانتزاعی، از بخش‌های زیر تشکیل شده است:



بخش‌های درونی CPU

بخش‌های داخل CPU به صورت زیر است:

- رجیسترها $(Registers)$.
- واحد ریاضی و منطقی $(Arithmetic\ and\ Logic\ Unit - ALU)$.
- واحد کنترل (CU).

بخش‌های بیرونی CPU

بخش‌های خارج از CPU به صورت زیر است:

- حافظه اصلی
- دستگاه‌های ورودی و خروجی

واحد پردازش مرکزی (CPU)

واحد پردازش مرکزی (CPU)، وظیفه اصلی اجرای دستورات برنامه‌ها را بر عهده دارد.
در ادامه به بررسی بخش‌های داخلی CPU و خارج از آن و ارتباط بین آنها باهم می‌پردازیم:

رجیسترها یا ثبات‌ها (Register)

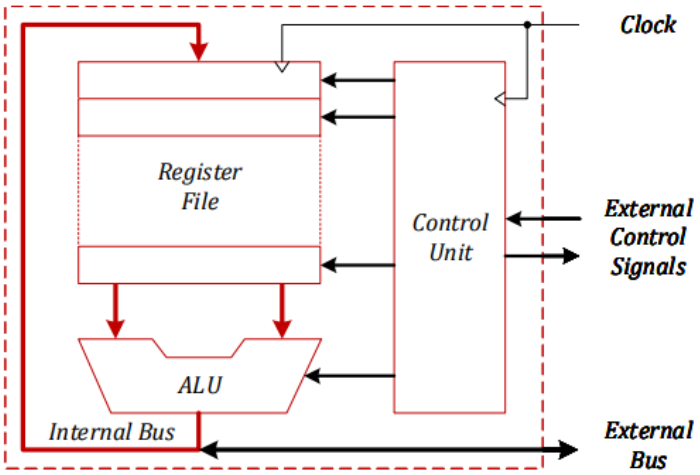
رجیستر یک حافظه بسیار کوچک و بسیار سریع در داخل CPU است که داده‌ها، دستورها و نتایجی را که CPU در همان لحظه به آن‌ها نیاز دارد، موقتاً نگهداری می‌کند. رجیستر سریع‌ترین و کوچکترین حافظه موجود در یک سیستم کامپیوتری است.

واحد ریاضی و منطقی (ALU)

وظایف این واحد به‌صورت زیر است:

- عملیات ریاضی (جمع، تفریق، ضرب، تقسیم)
- عملیات منطقی (NOT, XOR, OR, AND)

مدار درونی این واحد، شامل مجموعه‌ای از مدارات عملیات ریاضی و منطقی است. ALU دارای دو ورودی برای دریافت داده‌ها است، به صورت زیر:



نوع عملیات ریاضی مثل جمع یا منطقی مثل *AND* توسط پایه‌های کنترلی انتخاب عمل، تعیین می‌شود.

رجیستر داده (*Data Register – DR*): این رجیستر مسئول ذخیره‌سازی موقت داده‌هایی است که از حافظه یا دستگاه‌های ورودی و خروجی خوانده می‌شوند، تا در مراحل بعدی (مانند عملیات ریاضی یا منطقی) مورد استفاده قرار گیرند.

مثال: برای محاسبه جمع دو عدد a با مقدار 2 و b با مقدار 3 یعنی c = a + b ، مقادیر a و b موقتاً در دو رجیستر *DR* قرار می‌گیرد، *ALU* آن‌ها را جمع می‌کند و نتیجه 5 نیز موقتاً در یک رجیستر *DR* قرار می‌گیرد.

واحد کنترل (CU)

با در نظر گرفتن یک *CPU*، واحد کنترل کارهای زیر را انجام می‌دهد:

- **مرحله واکشی دستور (*Instruction Fetch – IF*)**
- **مرحله رمزگشایی دستور (*Instruction Fetch – ID*)**
- **مرحله اجرای دستور (*Instruction Execute – EX*)**

علاوه بر کارهای فوق، نیاز است واحد کنترل کارهای زیر را نیز در حین واکشی، رمزگشایی و اجرا انجام دهد.

- **صدور فرمان‌های کنترلی:** توسط سیگنال، فرمان‌های کنترلی (فعال شدن) را به پایه‌های کنترلی بخش‌های داخلی *CPU* مثل (رجیسترها و *ALU*) و بخش‌های خارج از *CPU* مثل (حافظه و I/O) ارسال می‌کند تا بخش مورد نظر برای انجام عملیات **فعال** شود.
- **هماهنگی:** فعالیت بین تمامی بخش‌های مختلف سیستم کامپیوتری را هماهنگ می‌کند. این هماهنگی تضمین می‌کند که اجزای مختلف سیستم به درستی و به موقع با یکدیگر همکاری کنند تا وظایف محوله به صورت کارآمد انجام شوند.

در ادامه به بررسی مراحل واکشی، رمزگشایی و اجرا می‌پردازیم:

- **مرحله واکشی دستور (*Instruction Fetch – IF*)** : واحد کنترل، توسط پایه‌های کنترلی **فرمان فعال شدن** حافظه برای عمل خواندن دستور از حافظه (*Read*) و **فرمان فعال شدن** رجیستر *IR* را می‌دهد. و همچنین **هماهنگ** می‌کند کد دستورالعمل جاری از حافظه وارد رجیستر *IR* شود.

رجیستر دستور (*Instruction Register – IR*): کد دستوری که از حافظه واکشی می‌شود، لازم است به صورت موقت درون *CPU* ذخیره شود تا رمزگشایی و سپس اجرا شود. به همین دلیل رجیستری برای ذخیره کد دستورالعمل واکشی شده در نظر گرفته می‌شود که آن را *IR* می‌نامند.

توجه: در ادامه جهت درک مفاهیم به مثال‌های انتزاعی زیر دقت کنید، البته در عمل، جزئیات بیشتری دارد که جهت ساده‌سازی از بیان جزئیات آن صرف نظر می‌کنیم و در درس معماری کامپیوتر به آن می‌پردازیم.

مثال: برای محاسبه جمع دو عدد a با مقدار 2 و b با مقدار 3 دستور c = a + b در مرحله واکشی دستور در رجیستر *IR* قرار می‌گیرد.

- **مرحله رمزگشایی دستور (*Instruction Fetch – ID*)**: در این مرحله **واحد کنترل** با دریافت «**کد دستور**

Instruction Code» از رجیستر IR آن را رمزگشایی می‌کند.

به صورت دقیق رمزگشایی یک دستور عبارتست از:

- مشخص کردن نوع عملیات ریاضی یا منطقی
- تعیین داده‌های مورد نیاز برای اجرای عملیات

مثال: برای محاسبه جمع دو عدد a با مقدار 2 و b با مقدار 3 دستور c = a + b در مرحله واکشی در رجیستر IR قرار می‌گیرد، سپس در مرحله رمزگشایی، مشخص می‌شود که **عمل جمع** باید انجام شود و داده‌های آن a و b است.

- **مرحله اجرای دستور (*Instruction Execute – EX*)**: در این مرحله **واحد کنترل** با توجه به دستور رمزگشایی شده و تشخیص نوع عملیات، فرمان‌های کنترلی لازم را برای بخش‌های درونی و بیرونی CPU صادر می‌کند.

حالت عمل خواندن داده از دستگاه ورودی: در این حالت، **واحد کنترل** توسط پایه‌های کنترلی **فرمان فعال شدن** دستگاه ورودی برای عمل خواندن داده ($Read$) از دستگاه ورودی و **فرمان فعال شدن** حافظه را می‌دهد و همچنین **هماهنگ** می‌کند داده‌ها از دستگاه ورودی به حافظه منتقل شود.

مثال: برای محاسبه جمع دو عدد a و b یعنی c = a + b ، مقدار 2 برای a و مقدار 3 برای b از ورودی خوانده می‌شود و در حافظه قرار می‌گیرد. فرض کنید مراحل واکشی و رمزگشایی از قبل انجام شده است.

حالت عملیات ریاضی (جمع، ضرب، تفریق و تقسیم) : در این حالت، **واحد کنترل** توسط پایه‌های کنترلی **فرمان فعال شدن** حافظه برای عمل خواندن داده از حافظه ($Read$) و **فرمان فعال شدن** رجیستر DR را می‌دهد، همچنین **هماهنگ** می‌کند مقدار دو داده از حافظه به دو رجیستر DR منتقل شود، سپس **واحد کنترل**، توسط پایه‌های کنترلی **فرمان فعال شدن** ALU برای **عملیات ریاضی** (جمع) روی دو رجیستر DR را می‌دهد، سپس نتیجه محاسبات داخل **رجیستر نتیجه** DR قرار می‌گیرد. و در نهایت **واحد کنترل** توسط پایه‌های کنترلی **فرمان فعال شدن** حافظه برای عمل نوشتن داده در حافظه ($Write$) را می‌دهد، همچنین **هماهنگ** می‌کند مقدار رجیستر نتیجه DR از رجیستر به حافظه منتقل شود.

مثال: برای محاسبه جمع دو عدد a و b یعنی c = a + b ، مقدار 2 برای a و مقدار 3 برای b از قبل از ورودی خوانده شده و در حافظه قرار دارد، مقادیر a و b موقتاً در دو رجیستر DR قرار می‌گیرد، ALU توسط مدار جمع، آن‌ها را جمع می‌کند و نتیجه 5 نیز موقتاً در یک **رجیستر نتیجه** DR قرار می‌گیرد و در نهایت مقدار **رجیستر نتیجه** DR از رجیستر به حافظه منتقل می‌شود.

حالت عمل نوشتن داده در دستگاه خروجی: در این حالت، **واحد کنترل** توسط پایه‌های کنترلی **فرمان فعال شدن** دستگاه خروجی برای عمل نوشتن داده ($Write$) در دستگاه خروجی و **فرمان فعال شدن** حافظه را می‌دهد و همچنین **هماهنگ** می‌کند داده‌ها از حافظه به دستگاه خروجی منتقل شود.

مثال: در حال حاضر مقدار 5 در حافظه قرار دارد و درنهایت از حافظه به دستگاه خروجی جهت نمایش منتقل می‌شود.

> ***توجه:*** واحد کنترل (CU) ستاد فرماندهی و هماهنگی کل قوا را برعهده دارد و ALU مجری **عملیات ریاضی** (جمع، تفریق، ضرب و تقسیم) و **عملیات منطقی** است، همانند رابطه مدیر (CU) و کارمند (ALU). واحد کنترل تعیین می‌کند چه زمانی داده‌ها از واحد ورودی به حافظه منتقل شود، چه زمانی ALU عملیات ریاضی و منطقی را انجام دهد و چه زمانی نتایج به واحد خروجی ارسال شود.

در ادامه بخش‌های دیگر سخت‌افزار بررسی می‌شود:

دستگاه‌های ورودی (*Input Devices*)

این دستگاه‌ها برای **دریافت داده‌ها** و دستورات از کاربر یا محیط بیرونی و انتقال آن‌ها به کامپیوتر **جهت پردازش** استفاده می‌شوند. نمونه‌های بارز آن شامل صفحه‌کلید (برای ورود متن و فرمان)، ماوس (برای کنترل مکان‌نما و انتخاب)، میکروفون (برای ورود صدا) ، دوربین (برای ورود تصاویر و ویدئوهای محیط واقعی) و اسکنر (برای ورود اسناد و تصاویر کاغذی) هستند.

دستگاه‌های خروجی (Output Devices)

این دستگاه‌ها وظیفه‌ی **انتقال داده‌های پردازش‌شده** از کامپیوتر به دنیای خارج را بر عهده دارند تا کاربر بتواند آن‌ها را مشاهده، شنیده یا لمس کند. صفحه‌نمایش (برای نمایش بصری)، بلندگو (برای خروجی صدا) و چاپگر (برای چاپ اسناد) از جمله دستگاه‌های خروجی محسوب می‌شوند.

دستگاه‌های ورودی و خروجی (Input / Output Devices)

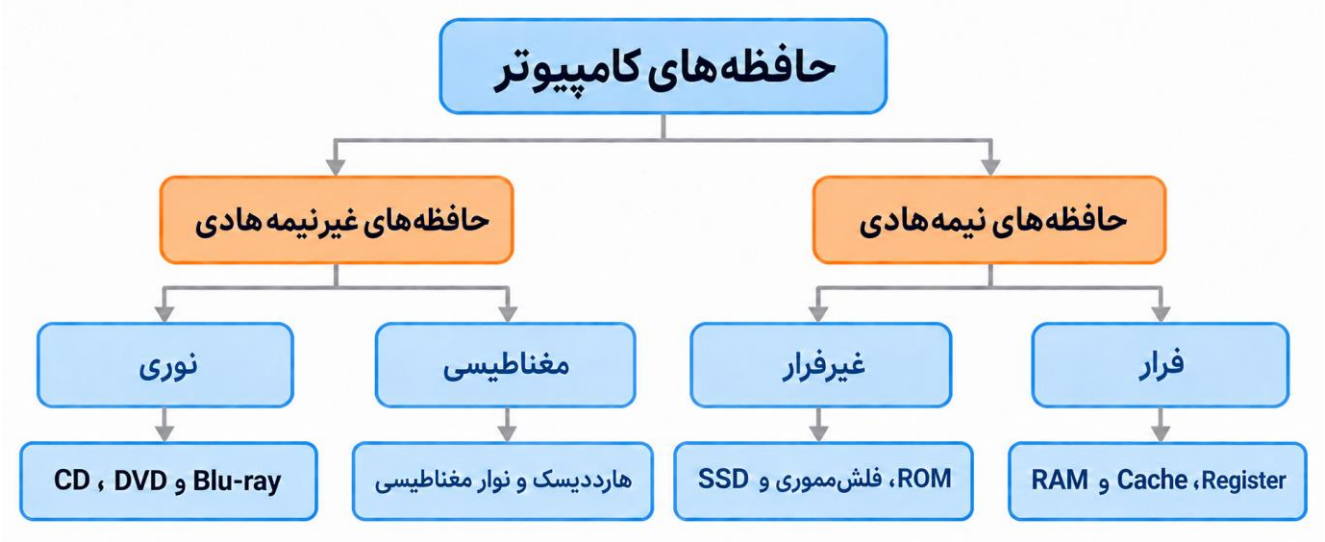
برخی از دستگاه‌ها قابلیت هم دریافت و هم ارسال اطلاعات را دارند و به همین دلیل به آن‌ها دستگاه‌های ورودی و خروجی می‌گویند. به عنوان مثال، نمایشگرهای لمسی، هم اطلاعات ورودی (لمس کاربر) را دریافت می‌کنند و هم اطلاعات خروجی (تصویر) را نمایش می‌دهند. همچنین، مودم‌ها و کارت‌های شبکه نیز از این دسته هستند که هم اطلاعات را دریافت و هم ارسال می‌کنند.

حافظه‌های کامپیوتر (Computer Memory)

حافظه یکی از ارکان اصلی هر سیستم کامپیوتری است که برای **نگهداری داده‌ها و دستورهای برنامه** به‌صورت موقت یا بلندمدت به کار می‌روند. این حافظه‌ها از نظر سرعت، ظرفیت، هزینه، ماندگاری اطلاعات و فناوری ساخت با یکدیگر تفاوت دارند.

انواع حافظه

حافظه‌های کامپیوتر از نظر **فناوری ساخت** به دو دسته **نیمه‌هادی** و **غیرنیمه‌هادی** تقسیم می‌شوند.



حافظه نیمه‌هادی

حافظه‌های نیمه‌هادی با استفاده از مدارهای مجتمع و قطعات الکترونیکی ساخته می‌شوند. این حافظه‌ها به دو دسته **فرّار** و **غیرفرّار** تقسیم می‌شوند.

حافظه‌های فرّار (Volatile Memory)

حافظه‌های فرّار برای نگهداری اطلاعات به برق نیاز دارند و با قطع برق، اطلاعات آن‌ها از بین می‌رود. امکان خواندن و نوشتن مکرر اطلاعات در این حافظه‌ها وجود دارد.

مهم‌ترین حافظه فرّار، حافظه با دسترسی تصادفی (RAM) است. منظور از دسترسی تصادفی این است که پردازنده می‌تواند با زمانی تقریباً یکسان به محل‌های مختلف حافظه دسترسی پیدا کند.

حافظه‌های *RAM* به دو نوع اصلی تقسیم می‌شود:

- **حافظه ایستا (Static RAM – SRAM)**
- **حافظه پویا (Dynamic RAM – DRAM)**

در SRAM، اطلاعات تا زمانی که برق برقرار باشد، بدون نیاز به تازه‌سازی حفظ می‌شوند. اما سلول‌های حافظه در DRAM به‌تدریج بار

الکتریکی خود را از دست می‌دهند؛ بنابراین اطلاعات آن باید در فاصله‌های زمانی کوتاه، با عملیات **تازه‌سازی (Refresh)** حفظ شود.

خود «اطلاعات» مستقیماً محو نمی‌شوند؛ در واقع **بار الکتریکی خازن‌های DRAM کاهش می‌یابد** و در نتیجه ممکن است اطلاعات از بین بروند. حتی اگر برق برقرار باشد ولی تازه‌سازی انجام نشود، بار خازن‌ها کاهش می‌یابد و اطلاعات از بین می‌روند. با قطع برق نیز اطلاعات DRAM به‌طور کامل از بین می‌رود. DRAM هم به برق دائمی نیاز دارد و هم باید مرتب، **تازه‌سازی (Refresh)** شود. خازن با بار الکتریکی، یک بیت را نگه می‌دارد، بار خازن به‌مرور نشت می‌کند، اگر خازن‌ها مرتب **تازه‌سازی (Refresh)** نشوند، اطلاعات از بین می‌رود. کنترل‌گر حافظه، این تازه‌سازی را خودکار و در فاصله‌های بسیار کوتاه انجام می‌دهد. هر سلول DRAM معمولاً از یک ترانزیستور و یک خازن و هر سلول SRAM معمولاً از شش ترانزیستور تشکیل شده است. DRAM اطلاعات را با بار الکتریکی خازن نگه می‌دارد، اما SRAM اطلاعات را با وضعیت یک مدار ترانزیستوری حفظ می‌کند. تا وقتی برق برقرار باشد، مدار ترانزیستوری وضعیت صفر یا یک را حفظ می‌کند و به تازه‌سازی نیاز ندارد.

برنامه‌های کاربردی و داده‌های مرتبط با آن در حافظه‌ای ذخیره می‌شوند که به آن اصطلاحا «**حافظه اصلی (*RAM*)**» می‌گویند.

حافظه اصلی کامپیوتر معمولا از نوع **DRAM** و **حافظه نهان پردازنده** معمولا از نوع **SRAM** است.

به حافظه RAM، حافظه با دسترسی تصادفی ***Random Access Memory – RAM*** نیز گفته می‌شود، زمان دسترسی «**Access Time**» به داده‌ها مستقل از مکان فیزیکی آن‌ها در حافظه است.

حافظه نهان (Cache Memory)

اختلاف زیاد میان سرعت پردازنده و حافظه اصلی (RAM)، می‌تواند باعث انتظار پردازنده و کاهش عملکرد سیستم شود. برای کاهش این اختلاف سرعت، از حافظه‌ای کوچک و بسیار سریع به نام حافظه نهان (Cache) استفاده می‌شود.

حافظه نهان براساس **اصل محلی‌بودن**، داده‌ها و دستورهایی را نگهداری می‌کند که احتمال می‌رود پردازنده در آینده نزدیک به آن‌ها نیاز داشته باشد. هنگامی که پردازنده داده‌ای را درخواست می‌کند، ابتدا حافظه نهان بررسی می‌شود. در این حالت، یکی از دو وضعیت زیر رخ می‌دهد:

اصابت (برخورد) حافظه نهان (*Cache Hit*)

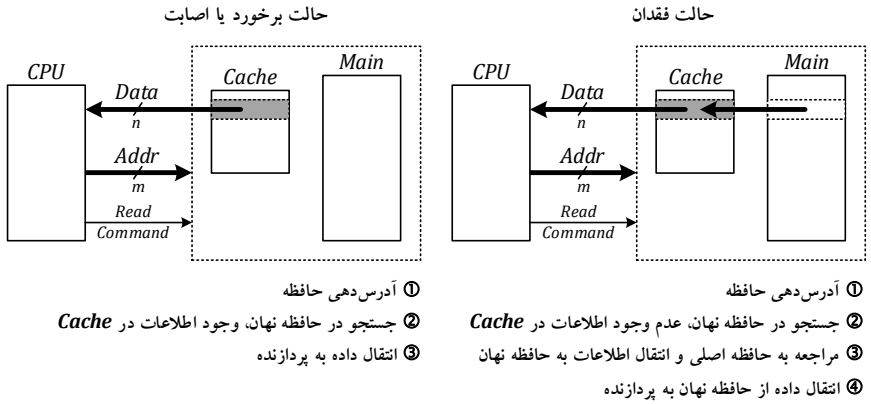
اگر داده موردنظر در حافظه نهان موجود باشد، پردازنده با سرعت زیادی به آن دسترسی پیدا می‌کند. به این وضعیت اصابت حافظه نهان می‌گویند. این حالت مطلوب است و باعث افزایش سرعت اجرای برنامه‌ها می‌شود.

عدم اصابت حافظه نهان (*Cache Miss*)

اگر داده موردنظر در حافظه نهان وجود نداشته باشد، عدم اصابت حافظه نهان رخ می‌دهد. در این حالت، مراحل زیر انجام می‌شود:

۱) داده مورد نیاز از حافظه اصلی خوانده می‌شود.
۲) داده به همراه بلوکی از داده‌های مجاور، براساس اصل محلی‌بودن، در حافظه نهان قرار می‌گیرد.
۳) داده در اختیار پردازنده قرار می‌گیرد تا پردازش ادامه یابد.

بنابراین، حافظه نهان با نگهداری داده‌ها و دستورهای پرکاربرد در نزدیکی پردازنده، متوسط زمان دسترسی به اطلاعات را کاهش می‌دهد و تا حد زیادی اختلاف سرعت میان پردازنده و حافظه اصلی را جبران می‌کند.



نمایش انتقال داده مورد نیاز به پردازشگر مرکزی، در دو حالت وجود داده (اصابت) و عدم وجود داده (عدم اصابت) در حافظه نهان

حافظه نهان به‌عنوان یک **واسط سریع میان پردازنده و حافظه اصلی** عمل می‌کند. ازآنجاکه سرعت حافظه اصلی بسیار کمتر از سرعت پردازنده است، نبود حافظه نهان باعث می‌شود پردازنده برای دریافت داده‌ها **مدت بیشتری منتظر** بماند. بنابراین، حافظه نهان با نگهداری داده‌ها و دستورهای موردنیاز پردازنده، زمان انتظار را کاهش می‌دهد و موجب افزایش کارایی سیستم می‌شود.

رجیسترها یا ثبات‌ها (Register) : رجیستر یک حافظه بسیار کوچک و بسیار سریع در داخل CPU است که داده‌ها، دستورها و نتایجی را که CPU در همان لحظه به آن‌ها نیاز دارد، موقتاً نگهداری می‌کند.

حافظه‌های غیرفرّار (Non – Volatile Memory)

حافظه‌های غیرفرّار برای حفظ اطلاعات به برق نیاز ندارند؛ بنابراین با قطع برق، اطلاعات آن‌ها از بین نمی‌رود. ROM ، فلش‌مموری و SSD از حافظه‌های نیمه‌هادی غیرفرّار هستند.

حافظه SSD

SSD مخفف **Solid-State Drive** به معنی **درایو حالت جامد** است. **SSD** نوعی **حافظه جانبی، نیمه‌هادی، دائمی و غیرفرّار** است که اطلاعات را حتی بعد از خاموش شدن کامپیوتر نگه می‌دارد. برخلاف **HDD**، قطعه مکانیکی و متحرک ندارد و اطلاعات را روی **تراشه‌های حافظه فلش** ذخیره می‌کند؛ به همین دلیل سریع‌تر، کم‌صداتر و مقاوم‌تر است.

حافظه ROM

ROM مخفف **Read Only Memory** به معنی **حافظه فقط خواندنی** است. **ROM** نوعی حافظه **نیمه‌هادی** و **غیرفرّار** است که معمولاً برای نگهداری برنامه‌ها و اطلاعاتی استفاده می‌شود که نباید به‌طور مکرر تغییر کنند؛ مانند برنامه‌های موردنیاز برای راه‌اندازی اولیه کامپیوتر. درROM های اولیه، اطلاعات هنگام ساخت نوشته می‌شد و امکان تغییر آن وجود نداشت؛ اما برخی انواع جدیدتر آن، مانند EEPROM قابلیت پاک‌شدن و برنامه‌ریزی مجدد دارند.

حافظه‌های غیرنیمه‌هادی

حافظه‌های غیرنیمه‌هادی برای ذخیره‌سازی اطلاعات از فناوری‌های مغناطیسی یا نوری استفاده می‌کنند. این حافظه‌ها غیرفرّارند؛ یعنی اطلاعات را بدون نیاز به برق حفظ می‌کنند.

- **حافظه‌های مغناطیسی:** هارد دیسک (HDD) و نوار مغناطیسی
- **حافظه‌های نوری:** CD، DVD و Blu-ray

هارد دیسک و دیسک‌های نوری دارای دسترسی مستقیم هستند؛ اما نوار مغناطیسی دارای دسترسی ترتیبی (***Sequential Access Memory – SAM***) است. بنابراین، برای دسترسی به داده‌ای مشخص در نوار مغناطیسی، باید داده‌های پیش از آن پیمایش شوند.

حافظه ثانویه یا جانبی (Secondary Memory) : حافظه‌ای **غیرفرّار** است که برای نگهداری دائمی برنامه‌ها و اطلاعات استفاده می‌شود و با قطع برق، اطلاعات آن از بین نمی‌رود. این حافظه می‌تواند **نیمه‌هادی**، مانند SSD و فلش‌مموری، یا **غیرنیمه‌هادی**، مانند هارد دیسک (HDD) و دیسک‌های نوری، باشد. حافظه ثانویه در مقایسه با حافظه اصلی، معمولاً ظرفیت بیشتر، سرعت کمتر و هزینه ذخیره‌سازی پایین‌تری دارد.

سلول‌های حافظه (*Memory Cells*)

سلول‌های حافظه، عناصر اصلی ذخیره‌سازی اطلاعات هستند و هر سلول می‌تواند یک بیت، یعنی مقدار **0 یا 1**، را نگهداری کند. مجموعه‌ای از این سلول‌ها، فضای لازم برای ذخیره‌سازی داده‌ها و دستورهای برنامه را تشکیل می‌دهد. کارایی سیستم کامپیوتری تنها به سرعت **واحد پردازش مرکزی (CPU)** وابسته نیست، بلکه ساختار **سلسله‌مراتب حافظه (Memory Hierarchy)** نیز در آن نقش مهمی دارد. سلسله‌مراتب حافظه از **ثبات‌ها**، **حافظه نهان (Cache)** ، **حافظه اصلی (RAM)** و **حافظه ثانویه مانند HDD و SSD** تشکیل شده و با هدف ایجاد تعادل میان **سرعت**، **ظرفیت و هزینه** طراحی شده است. حافظه نهان با نگهداری داده‌ها و دستورهای پرکاربرد در نزدیکی پردازنده، زمان دسترسی به حافظه را کاهش می‌دهد و موجب افزایش کارایی سیستم می‌شود. آشپز ابتدا وسیله یا ماده موردنیازش را کنار دست خود جست‌وجو می‌کند؛ اگر آنجا نباشد، به سراغ میز بزرگ آشپزخانه می‌رود. پردازنده نیز ابتدا اطلاعات را در Cache جست‌وجو می‌کند و اگر پیدا نشود، آن را از RAM می‌گیرد.

۲ – نرم‌افزار: (Software)

بخش منطقی سیستم کامپیوتری است. مجموعه برنامه‌ها و دستوراتی که مشخص می‌کنند سخت‌افزار چه کاری را انجام دهد. بنابراین، سخت‌افزار به‌تنهایی نمی‌تواند نیاز موردنظر انسان را برطرف کند؛ بلکه برای استفاده از قدرت انجام سریع **عملیات ریاضی** و **عملیات منطقی** **سخت‌افزار** کامپیوتر به نرم‌افزار نیاز داریم. نرم‌افزار واسط بین کاربر نهایی و سخت‌افزار است. **ساز بدون نوازنده، کاربردی ندارد.** هر برنامه مجموعه‌ای از دستورالعمل‌هاست که برنامه‌نویس آن‌ها را می‌نویسد تا نوع و ترتیب عملیات کامپیوتر را تعیین کند.

نتیجه: همکاری سخت‌افزار و نرم‌افزار در یک سیستم کامپیوتری، منجر به برآورده‌سازی نیاز انسان می‌شود.

انواع نرم‌افزارها

در یک دسته‌بندی کلی برای نرم‌افزارهای موجود، می‌توان آنها را به دو گروه اصلی زیر تقسیم نمود:

نرم‌افزارهای کاربردی

برنامه‌هایی که برای رفع نیازهای کاربران نهایی کامپیوتر نوشته می‌شود، به بیان دیگر نرم‌افزارهای کاربردی به‌طور **مستقیم به انسان** سرویس می‌دهند. مانند نرم‌افزارهای مالی و حسابداری، حقوق و دستمزد، واژه‌پرداز، ویرایش تصویر، فرهنگ لغات و ماشین‌حساب.

نرم‌افزارهای سیستمی

برنامه‌هایی که برای بهره‌برداری از سخت‌افزار یا سرویس‌دادن به سایر برنامه‌ها نوشته شده‌اند. به‌بیان دیگر نرم‌افزارهای سیستمی به طور مستقیم به نرم افزارهای دیگر و به طور **غیرمستقیم به انسان** سرویس می‌دهند. اصلی‌ترین نرم‌افزار سیستمی، **سیستم عامل** است.

توجه مهم: در ادامه جهت نگاه **الگوریتمی** از عبارت **برنامه کامپیوتری** به جای **نرم‌افزار کاربردی** استفاده می‌کنیم.

برنامه کامپیوتری

یک برنامه کامپیوتری ماهیتی منطقی دارد و از دو بخش زیر تشکیل شده است:

۱– **ساختار داده:** محل نگهداری داده‌ها

۲– **دستورات اجرایی:** کدهای قابل اجرا که باعث انجام وظایف مورد نظر می‌شود

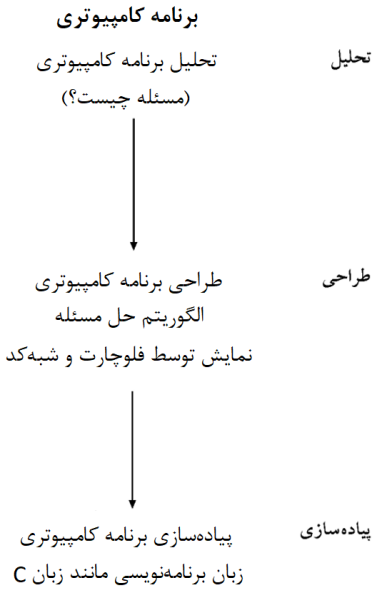
مراحل ایجاد برنامه‌کامپیوتری

یک برنامه کامپیوتری در طی سه مرحله زیر ایجاد می‌شود:

۱- تحلیل (مسئله چیست؟)

۲- طراحی (الگوریتم حل مسئله و نمایش توسط فلوچارت یا شبه‌کد)

۳- پیاده‌سازی (کدنویسی به زبان C)



مثال: سیستم جمع دو عدد و نمایش آن در خروجی را **تحلیل**، **طراحی** و **پیاده‌سازی** نمایید.

تحلیل

در مرحله تحلیل، مشخص می‌کنیم **مسئله چیست؟** و برنامه باید چه کاری انجام.

مثال: جمع دو عدد و نمایش نتیجه در خروجی.

طراحی

پس از تحلیل، نوبت به طراحی می‌رسد. در طراحی، **الگوریتم** برنامه مشخص می‌شود. طراحی، برنامه را در سطحی از انتزاع تصویر می‌کند که به کد نزدیک است. طراحی، به عنوان نقشه راهی دقیق، و نزدیک به زبان پیاده‌سازی، پیاده‌سازی برنامه را ساده‌تر می‌کند و احتمال بروز خطا را کاهش می‌دهد.

الگوریتم (Algorithm)

الگوریتم، مجموعه‌ای از **مراحل** مرتب برای حل یک مسئله است.

مثال: الگوریتم جمع دو عدد و نمایش نتیجه در خروجی، به صورت زیر است:

۱- شروع

۲- سه متغیر a، b و c را تعریف کن.

۳- دو عدد از ورودی خوانده و مقادیر آنها را در متغیرهای a و b قرار بده.

۴- مقادیر دو متغیر a و b را جمع کن و مقدار حاصل را در متغیر c قرار بده.

۵- مقدار متغیر c را در خروجی نمایش بده.

۶- پایان

ترتیب مراحل اهمیت زیادی دارد؛ زیرا حتی اگر هر مرحله به طور مستقل و به تنهایی درست باشد، ترتیب نادرست مراحل منجر به نتیجه نادرست خواهد شد. یک الگوریتم هنگام پیش‌روی و حرکت از بالا به پایین سه حالت را تجربه می‌کند، **(۱)** یا مراحل به‌ترتیب و یکی پس از دیگری اجرا می‌شود (توالی) ، **(۲)** یا مراحل سر دوراهی می‌ماند (انتخاب) **(۳)** یا مراحل دچار تکرار می‌شود (تکرار) به این سه

حالت، **جریان کنترل برنامه** (Program Control Flow) گفته می‌شود، به صورت زیر:

- **توالی (Sequence)**: اجرای مراحل به‌ترتیب و یکی پس از دیگری
- **انتخاب (Selection) یا شرط (Condition)**: انتخاب یکی از مسیرهای اجرا براساس یک شرط
- **تکرار (Iteration)**: اجرای چندباره یک یا چند مرحله تا برقراری یک شرط

روش‌های نمایش الگوریتم

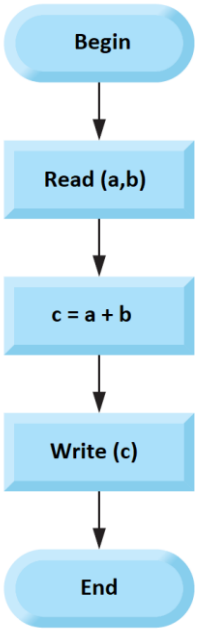
الگوریتم را به دو روش می‌توان نمایش داد:

- **فلوچارت: (Flowchart)** نمایش گرافیکی الگوریتم
- **شبه‌کد: (Pseudocode)** نمایش نوشتاری الگوریتم

۱- فلوچارت (Flowchart)

فلوچارت، **نمایش گرافیکی الگوریتم** است. در فلوچارت، هر مرحله با نمادی مشخص نشان داده می‌شود و پیکان‌ها، ترتیب اجرای مراحل را تعیین می‌کنند. در فلوچارت می‌توان از ساختارهای اصلی کنترل برنامه، یعنی **توالی، انتخاب و تکرار** به صورت گرافیکی، استفاده کرد. برای نمایش **شروع** و **پایان** الگوریتم از **مستطیل با گوشه‌های گرد** استفاده می‌شود که به‌ترتیب عبارت‌های Begin و End در آن‌ها نوشته می‌شود. هر کنش در برنامه، مانند عملیات محاسبات، عملیات دریافت ورودی و نمایش خروجی، با **مستطیل** و شرط یا تصمیم‌گیری با **لوزی** نشان داده می‌شوند.

مثال: نمایش الگوریتم جمع دو عدد توسط **فلوچارت** به صورت زیر است:



۲- شبه‌کد یا زبان طراحی برنامه (Pseudocode)

شبه‌کد، **نمایش نوشتاری الگوریتم** است. به شبه‌کد زبان طراحی برنامه (Program Design Language یا PDL) نیز گفته می‌شود. شبه‌کد برای نمایش نوشتاری الگوریتم، پیش از کدنویسی، به کار می‌رود.

واژه Pseudocode که «سودوکد» تلفظ می‌شود، از ترکیب دو واژه Pseudo به‌معنای «شبه» و Code به‌معنای «کد» ساخته شده است؛ بنابراین، سودوکد یا شبه‌کد به متنی گفته می‌شود که شبیه کد برنامه‌نویسی است، اما مستقیماً اجرا نمی‌شود.

شبه‌کد ترکیبی از واژه‌های یک زبان طبیعی، مانند فارسی یا انگلیسی، و ساختارهایی شبیه زبان‌های برنامه‌نویسی است. در شبه‌کد می‌توان از ساختارهای اصلی کنترل برنامه، یعنی **توالی، انتخاب و تکرار**، استفاده کرد. شبه‌کد ممکن است در نگاه اول مانند یک زبان برنامه‌نویسی به نظر برسد؛ اما **زبان برنامه‌نویسی نیست** و کامپیوتر نمی‌تواند شبه‌کد را مستقیماً **کامپایل** یا **اجرا** کند.

همچنین شبه‌کد، برخلاف زبان‌های برنامه‌نویسی قواعد نحوی ثابت و سخت‌گیرانه‌ای ندارد؛ بنابراین، لازم نیست تمام جزئیات مانند نوع دقیق متغیرها، شکل کامل دستورها یا علامت ";" در پایان دستورها در آن نوشته شود. بااین‌حال، شبه‌کد باید روشن، منظم و بدون ابهام باشد تا بتواند مراحل یک الگوریتم را به‌درستی نمایش دهد.

شبه‌کد نسبت به کد برنامه در سطح بالاتری از انتزاع قرار دارد؛ یعنی به‌جای پرداختن به جزئیات یک زبان برنامه‌نویسی، بر منطق حل مسئله، عملیات موردنیاز و ترتیب اجرای آن‌ها تمرکز می‌کند. به همین دلیل، طراح می‌تواند پیش از پیاده‌سازی، منطق الگوریتم را بررسی کند و خطاهای احتمالی آن را برطرف سازد. پس از بررسی شبه‌کد، برنامه‌نویس آن را به کد یک زبان برنامه‌نویسی تبدیل می‌کند.

در شبه‌کد می‌توان برای بیان الگوریتم از دستورهایی شبیه زبان C استفاده کرد، بدون آنکه رعایت تمام قواعد نحوی زبان C، مانند قراردادن علامت ";" در پایان دستورها، الزامی باشد.

برخی ابزارها می‌توانند گونه‌های مشخص و قاعده‌مند شبه‌کد را به قالب اولیه یا اسکلت یک برنامه تبدیل کنند. کد تولیدشده به این روش معمولاً کامل نیست و باید توسط برنامه‌نویس تکمیل شود.

مثال: نمایش الگوریتم جمع دو عدد توسط **شبه کد به زبان فارسی** به صورت زیر است:

```
شروع
   دو عدد a و b را بخوان
   c = a + b
   مقدار c را نمایش بده
پایان
```

مثال: نمایش الگوریتم جمع دو عدد توسط **شبه کد به زبان انگلیسی** به صورت زیر است:

```
Begin
   integer a , b , c
   Read (a , b)
   c = a + b
  write (c)
End
```

توجه: تعریف متغیرها، دستور اجرایی الگوریتم محسوب نمی‌شود، بلکه یک دستور تعریف یا اعلان است که نام و نوع متغیر را مشخص می‌کند. به همین دلیل در **فلوچارت** و **شبه‌کد** می‌توان تعریف متغیرها را بیان کرد یا نکرد. برای مثال، عبارت int a , b , c ; در زبان C فقط نام و نوع متغیر را برای کامپایلر مشخص می‌کند و سیستم‌عامل نیز حافظه موردنیاز برنامه را فراهم می‌سازد.

مثال:

```
int a , b , c ;
```

اعلان بالا، فقط سه متغیر را تعریف می‌کند و دستور اجرایی نیست؛ اما دستور زیر:

```
c = a + b ;
```

یک دستور اجرایی است، زیرا عمل جمع و انتساب را انجام می‌دهد.

پیاده سازی

پس از طراحی نوبت به پیاده‌سازی می‌رسد. در پیاده‌سازی، الگوریتم با استفاده از یک زبان برنامه‌نویسی، مانند زبان C ، به کد تبدیل می‌شود.

مثال: پیاده‌سازی الگوریتم جمع دو عدد توسط **زبان C** به صورت زیر است:

```
#include <stdio.h>

int  main (void)
{
   int a , b , c ;

   scanf ("%d %d" , &a , &b) ;
   c = a + b ;
  printf ("%d" , c) ;

  return 0;
}
```

توجه: دقت کنید که می‌توان تحلیل و طراحی را به روش و ابزارهای ساخت‌یافته انجام داد ولی فعالیت پیاده‌سازی را توسط یک زبان برنامه‌نویسی شی‌گرا مثل C++ انجام داد و از امکانات شی‌گرایی زبان استفاده نکرد اما عکس این مطلب امکان‌پذیر نیست، یعنی نمی‌توان تحلیل و طراحی را به روش شیءگرا انجام داد ولی فعالیت پیاده‌سازی را توسط یک زبان ساخت‌یافته مثل C انجام داد. زیرا زبان ساخت‌یافته مثل C، امکانات شی‌گرایی (کلاس، وارثت و چندریختی) را پشتیبانی نمی‌کند. زبان C++ یک فوق مجموعه از C است.

انتزاع یا تجرید (abstraction)

با نگاهی سطح به سطح به حل یک مساله، سطوح مختلفی از انتزاع را خواهیم داشت. در بالاترین سطح انتزاع، راه حل به صورت کلی به زبان محیط مساله بیان می‌گردد. در سطوح پایین‌تر، راه حل به جزئیات پیاده‌سازی نزدیک‌تر می‌شود و در پایین‌ترین سطح انتزاع راه حل به صورتی بیان می‌شود تا مستقیما قابل پیاده‌سازی باشد.

مثال: مکالمه محاوره‌ای زیر را در نظر بگیرید:

سوال: کجا هستی؟

پاسخ: بیرون

سوال: کی می‌یای؟

پاسخ: میام

همانطورکه مشاهده می‌کنید مکالمه فوق کاملا انتزاعی و با حذف جزئیات از طرف پاسخ‌دهنده انجام شده است. و برای سوال‌کننده همچنان سوالاتی نظیر کجای بیرون دقیقاً و چه زمانی دقیقاً، باقی‌مانده است، چون پاسخ‌ها کاملاً انتزاعی از طرف پاسخ‌دهنده بیان شده است. انسان موجودی انتزاعی و ماشین موجودی غیرانتزاعی است.

توجه: در مورد هر موضوعی هرچه‌قدر جزئیات **بیشتری** بدانیم، درجه انتزاع ذهن ما نسبت به آن موضوع **کمتر** است و بالعکس در مورد هر موضوعی هرچه‌قدر جزئیات **کمتری** بدانیم درجه انتزاع ذهن ما نسبت به آن موضوع **بیشتر** است.

برنامه‌نویسی ساخت‌یافته

به‌طور معمول، دستورهای یک برنامه به همان ترتیبی که نوشته شده‌اند، یکی پس از دیگری اجرا می‌شوند. به این شیوه، **اجرای ترتیبی**

گفته می‌شود. بااین‌حال، در بسیاری از برنامه‌ها لازم است براساس شرایط مختلف، مسیر اجرای دستورها تغییر کند. به تغییر مسیر عادی اجرای برنامه، **انتقال کنترل** گفته می‌شود. در دهه ۱۹۶۰، استفاده بی‌رویه از دستور goto که امکان پرش به بخش‌های مختلف برنامه را فراهم می‌کرد، باعث پیچیدگی برنامه‌ها و دشوارشدن درک، اشکال‌زدایی و نگهداری آن‌ها می‌شد. به همین دلیل، برنامه‌نویسان به‌تدریج استفاده از goto را محدود کردند و به‌سمت روشی منظم‌تر به نام **برنامه‌نویسی ساخت‌یافته** حرکت کردند.

برنامه‌نویسی ساخت‌یافته روشی برای طراحی و نوشتن برنامه‌های منظم، قابل‌فهم و قابل‌نگهداری است. در این روش، مسیر اجرای برنامه با استفاده از سه ساختار اصلی **توالی**، **انتخاب** و **تکرار** کنترل می‌شود و از پرش‌های نامنظم میان بخش‌های مختلف برنامه با دستور goto پرهیز می‌گردد.

اساس این روش، **تجزیه برنامه به مجموعه‌ای از تابع‌های کوچک‌تر** است. هر تابع بخشی از مسئله را حل می‌کند و همکاری تابع‌ها با یکدیگر، عملکرد کلی برنامه را شکل می‌دهد.

هر تابع می‌تواند شامل یک یا چند بلوک باشد. بلوک مجموعه‌ای از تعریف‌ها و دستورهاست که در زبان C درون آکولادهای { } قرار می‌گیرد و مانند یک دستور مرکب در نظر گرفته می‌شود. بلوک‌ها می‌توانند به‌صورت تودرتو، یعنی درون بلوک‌های دیگر، قرار گیرند. استفاده از تابع‌ها، بلوک‌ها و ساختارهای کنترل منظم، خواندن، آزمایش، اشکال‌زدایی و نگهداری برنامه را آسان‌تر می‌کند.

متغیری که درون یک تابع یا بلوک تعریف می‌شود، **محلی (Local)** است و فقط در همان محدوده قابل‌استفاده است. متغیری که بیرون از همه تابع‌ها تعریف شود، **سراسری (Global)** است و تابع‌های مختلف برنامه می‌توانند به آن دسترسی داشته باشند. استفاده از متغیرهای محلی، احتمال تداخل ناخواسته میان بخش‌های مختلف برنامه را کاهش می‌دهد.

در برنامه‌نویسی ساخت‌یافته می‌توان از روش بازگشتی (Recursive) نیز استفاده کرد؛ یعنی یک تابع می‌تواند خود را فراخوانی کند. البته استفاده از بازگشت جزء الزامی برنامه‌نویسی ساخت‌یافته نیست.

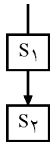
زبان C یکی از شناخته‌شده‌ترین زبان‌هایی است که امکان نوشتن برنامه‌های ساخت‌یافته را فراهم می‌کند. برنامه‌نویسی ساخت‌یافته برای بسیاری از برنامه‌ها مناسب است؛ اما با افزایش اندازه و پیچیدگی نرم‌افزار، مدیریت تعداد زیادی تابع و داده‌های مرتبط با آن‌ها دشوارتر می‌شود. برای سازماندهی بهتر برنامه‌های بزرگ، روش‌هایی مانند **برنامه‌نویسی شیءگرا** گسترش یافتند. شیءگرایی در فصل مربوطه بررسی می‌شود.

مثال:

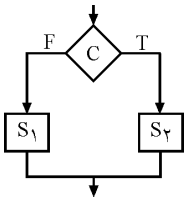
```c
#include <stdio.h>

int main (void)
{
   int a , b , c ;

   scanf ("%d %d" , &a , &b) ;
   c = a + b ;
   printf ("%d" , c) ;

   return 0;
}
```

برنامه فوق یک نمونه از برنامه‌نویسی ساخت‌یافته است؛ زیرا دستورها در یک بلوک مشخص قرار دارند و به‌ترتیب، یکی پس از دیگری اجرا می‌شوند. هر مجموعه دستور داخل { } یک بلوک است. مسیر اجرای برنامه مشخص و منظم است و از goto استفاده نشده است. متغیرهای a ، b و c محلی هستند. این برنامه از بین ساختارهای توالی، انتخاب و تکرار فقط ساختار توالی را نشان می‌دهد و برای معرفی اولیه برنامه‌نویسی ساخت‌یافته ساده‌تر است.

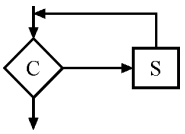
ساختار ترتیب یا توالی(Sequence): اجرای پشت سر هم دستورات بدون پرش.



ساختارهای انتخاب یا شرط (Condition): امکان ایجاد انشعاب بر اساس مقدار شرط مورد نظر.



ساختارهای تکرار(Iteration): امکان ایجاد تکرار برای یک مجموعه خاص از دستوارات بر اساس مقدار شرط تکرار مورد نظر.



توجه: ساختارهای کنترل توالی ، شرط و تکرار در فصل **کنترل برنامه** بررسی خواهد شد.

پیمانه‌سازی (modularity)

وقتی با یک مساله برنامه‌نویسی به زبان ساخت‌یافته مواجه می‌شوید، برای بالابردن خوانایی و کنترل بهتر برنامه این پرسش مطرح است که چگونه این مساله به توابع (پیمانه‌ها یا ماژول‌ها) تقسیم شود، که تعیین تعداد توابع به عهده مهندسان نرم‌افزار است. پیمانه‌ای کردن یعنی تقسیم نرم‌افزار به چند مولفه جداگانه و متمایز که این مولفه‌ها به عنوان پیمانه‌هایی هستند که از اجتماع آن‌ها، خواسته‌های مساله برآورده می‌شود. با این عمل مدیریت مفهومی یک برنامه حاصل می‌شود و قابلیت خوانایی نرم‌افزار چندین برابر می‌گردد. درک و فهم نرم‌افزار یکپارچه (monolithic) که تنها از یک پیمانه (ماژول) تشکیل شده است، بسیار مشکل است، زیرا تعداد متغیرها، حجم ارجاعات، تعداد مسیرهای کنترلی و پیچیدگی سراسری آن بسیار زیاد است. بنابراین اگر بتوان نرم‌افزار را به پیمانه‌هایی مناسب تقسیم نمود، پیچیدگی و هزینه کلی آن کاهش و قابلیت خوانایی آن افزایش خواهد یافت.

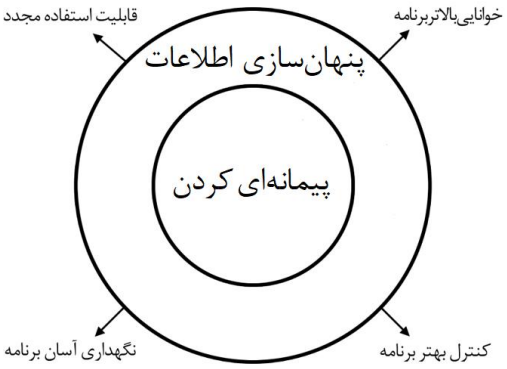
توجه: وقتی با یک مساله برنامه‌نویسی به زبان شیءگرا مواجه می‌شوید، مانند روش‌های ساخت‌یافته دیگر این پرسش مطرح نیست که چگونه این مساله به چند تابع (پیمانه) تقسیم می‌شود بلکه این پرسش مطرح است که چگونه این مساله به چند **کلاس** (پیمانه) تقسیم می‌شود.

پنهان‌سازی اطلاعات

در ساخت‌یافتگی، شرط لازم برای برقراری پنهان‌سازی اطلاعات (information hiding)، پیمانه‌ای کردن است و شرط کافی برای برقراری پنهان‌سازی اطلاعات تعریف متغیرهای **محلی (Local)** و دستورالعمل‌های مرتبط با تابع است. در یک بیان ساده پنهان‌سازی اطلاعات می‌گوید بخشی از نرم‌افزار در داخل یک پیمانه (تابع) محصور شود. هدف از پنهان‌سازی اطلاعات، پنهان کردن روال انجام دستورات تابع و متغیرهای محلی در پس واسط یا بلاک تابع است. استفاده‌کنندگان پیمانه‌ها (توابع) نیازی به دانستن جزئیات داخلی پیمانه‌ها (توابع) ندارند.

توجه: پیمانه‌ای‌کردن و برقراری پنهان‌سازی اطلاعات به عنوان اصول و معیارهای طراحی مطلوب، منجر به **خوانایی بالاتر برنامه، کنترل بهتر برنامه، قابلیت استفاده مجدد** پیمانه‌های (توابع) برنامه جاری در برنامه‌های آتی و **نگهداری آسان برنامه** جاری می‌گردد.

شکل زیر گویای مطلب است:



زمان و هزینه‌ی فرآیند تولید نرم‌افزار با استفاده از قطعات آماده (قابلیت استفاده مجدد) کاهش چشمگیری دارد. در واقع یک قطعه با قابلیت استفاده‌ی مجدد یک بار ساخته می‌شود و بارها و بارها استفاده می‌شود و این یعنی سودآوری!

در حیطه‌ی ساخت‌یافته، مولفه (پیمانه) یک قطعه عملیاتی مبتنی بر تابع است که بر دو نوع می‌باشد:

۱) تابع کاربردی: مانند تابع جمع که برنامه‌نویس آن را می‌نویسد. اگر تابع کاربردی، شرایط قابل حمل بودن (مانند تعریف متغیر درون تابع به صورت محلی و صریح و عدم استفاده از متغیرهای سراسری) را داشته باشد می‌تواند به عنوان یک قطعه‌ی آماده‌ی قابل استفاده‌ی مجدد در پروژه‌های بعدی مورد استفاده مجدد قرار گیرد.

۲) تابع سیستمی: مانند تابع sin برای عمل سینوس، تابع scanf برای خواندن از ورودی و تابع printf برای نوشتن در خروجی که کامپایلر زبان C تعاریف آن را فراهم می‌کند و می‌تواند به عنوان یک قطعه‌ی آماده و قابل استفاده‌ی مجدد، در پروژه‌ها مورد استفاده مجدد قرار گیرد.

زبان‌های برنامه‌نویسی

زبان وسیله‌ای برای برقراری ارتباط است. همان‌طور که انسان‌ها از زبان‌هایی مانند فارسی و انگلیسی برای ارتباط با یکدیگر استفاده می‌کنند، برنامه‌نویسان نیز از **زبان‌های برنامه‌نویسی** برای بیان دستورهای موردنیاز کامپیوتر استفاده می‌کنند. برای اینکه کامپیوتر بتواند کاری را انجام دهد، باید دستورهای لازم را در قالب یک برنامه به آن بدهیم. زبان برنامه‌نویسی مجموعه‌ای از **نمادها** و **قواعد** برای نوشتن این دستورهاست. قواعد مربوط به ساختار و شیوه نوشتن دستورها، **نحو (Syntax)** نامیده می‌شود و برنامه‌نویس باید آن‌ها را رعایت کند. نحو (Syntax) در زبان برنامه‌نویسی، مانند دستور زبان (Grammar) در زبان‌های طبیعی مثل فارسی و انگلیسی است. همان‌طور که برای ساختن یک جمله درست باید قواعد زبان انگلیسی را رعایت کنیم، برای نوشتن یک برنامه درست نیز باید قواعد نحوی زبان برنامه‌نویسی را رعایت کنیم. مثلاً در زبان C :

```
printf ("Hello") ;
```

از نظر نحوی درست است؛ اما اگر ; را حذف کنیم، برنامه دارای **خطای نحوی (Syntax Error)** می‌شود:

```
printf ("Hello")
```

انواع زبان‌های برنامه‌نویسی

زبان‌های برنامه‌نویسی از نظر شیوه بیان دستورها و میزان نزدیکی به سخت‌افزار با یکدیگر تفاوت دارند. بعضی زبان‌ها دستورها را به‌صورت صفر و یک بیان می‌کنند، برخی از نمادهای اختصاری استفاده می‌کنند و برخی دیگر به شیوه بیان انسان نزدیک‌ترند. به طور کلی زبان‌های برنامه‌نویسی به سه گروه زیر تقسیم می‌شوند:

۱ – زبان ماشین (Machine Language)

۲ – زبان اسمبلی (Assembly Language)

۳ – زبان‌های سطح بالا (High-Level Languages)

زبان ماشین (Machine Language)

ساختار کامپیوتر طوری طراحی شده‌است که بجز مقدارهای صفر و یک، مقادیر دیگری را درک نمی‌کند. زبان ماشین در پایین‌ترین سطح

زبان‌های برنامه‌نویسی قرار گرفته‌است. دستورهای آن به‌صورت **صفر و یک** بیان می‌شوند و پردازنده آن‌ها را **مستقیماً و بدون نیاز به ترجمه** اجرا می‌کند. این دستورها کارهایی مانند جمع، مقایسه و انتقال داده را مشخص می‌کنند.

زبان ماشین به **نوع پردازنده** وابسته است؛ یعنی ترکیب یکسانی از صفر و یک‌ها ممکن است در دو پردازنده مختلف، معنای متفاوتی داشته باشد. بنابراین، برنامه نوشته‌شده برای یک نوع پردازنده، لزوماً روی پردازنده‌ای با معماری متفاوت اجرا نمی‌شود و اصطلاحا قابل حمل نیست.

انسان‌ها مفاهیم را با کلمات و نمادهای آشنا بهتر درک می‌کنند، درحالی‌که پردازنده فقط دستورهای زبان ماشین را به‌صورت صفر و یک اجرا می‌کند. ازاین‌رو، نوشتن، خواندن و **یافتن خطاهای برنامه** در زبان ماشین برای انسان طاقت‌فرسا و زمان‌بر است؛ به همین دلیل، برنامه‌نویسان معمولاً برنامه‌ها را با زبان‌های دیگری می‌نویسند و آن‌ها را مستقیماً با صفر و یک بیان نمی‌کنند.

مثال: قطعه‌کد زیر در زبان ماشین یک کامپیوتر فرضی، دو مقدار ذخیره‌شده در متغیرهای x و y را جمع می‌کند و نتیجه را در متغیر z ذخیره می‌کند. مطابق فرض، در هر دستور، چهار بیت اول نوع عملیات و چهار بیت دوم آدرس متغیر در حافظه را مشخص می‌کند.

دستور	چهار بیت اول: عملیات
0001	خواندن مقدار مورد نظر از حافظه و قرار دادن در رجیستر
0010	جمع‌کردن مقدار مورد نظر از حافظه با مقدار رجیستر
0011	ذخیره‌کردن مقدار رجیستر در حافظه مورد نظر

آدرس متغیرهای x ، y و z نیز به ترتیب 0100 ، 0101 و 0110 است. بنابراین، قطعه کد محاسبه z = x + y به صورت زیر است:

```
0001  0100
0010  0101
0011  0110
```

دستور	چهار بیت اول: عملیات	چهار بیت دوم: آدرس متغیر
0001 0100	خواندن مقدار مورد نظر از حافظه و قرار دادن در رجیستر	از آدرس 0100 مقدار x را بخوان.
0010 0101	جمع‌کردن مقدار مورد نظر از حافظه با مقدار رجیستر	از آدرس 0101 مقدار y را بخوان.
0011 0110	ذخیره‌کردن مقدار رجیستر در حافظه مورد نظر	در آدرس 0110 مقدار z را بنویس.

دستور اول، مقدار متغیر x را در رجیستر قرار می‌دهد. دستور دوم، مقدار متغیر y را به آن اضافه می‌کند و دستور سوم، نتیجه را در متغیر z ذخیره می‌کند.

نکته آموزنده: یک ترکیب صفر و یک می‌تواند بسته به جایگاهش، کد دستور یا آدرس باشد. همچنین در دو پردازنده با معماری مختلف می‌تواند معنای متفاوتی داشته باشد.

زبان اسمبلی (Assembly Language)

زبان اسمبلی برای آسان‌تر کردن برنامه‌نویسی نسبت به زبان ماشین معرفی شد. اسمبلی یک زبان برنامه‌نویسی **سطح پایین** است که در آن، به‌جای نوشتن دستورها به‌صورت صفر و یک، از **نمادهای کوتاه** استفاده می‌شود. به این ترتیب، دستورها نسبت به زبان ماشین به زبان انسان نزدیک‌تر و برای برنامه‌نویس خواناتر شدند. برای مثال، دستور ADD برای جمع‌کردن، دستور SUB برای تفریق و دستور MOV برای انتقال داده به کار می‌رود. بنابراین، نوشتن و خواندن برنامه در زبان اسمبلی از زبان ماشین آسان‌تر است.

زبان اسمبلی نیز مانند زبان ماشین، به معماری پردازنده وابسته است؛ بنابراین، برنامه‌ای که برای یک معماری نوشته شده است، معمولاً روی معماری دیگری اجرا نمی‌شود و **قابل‌حمل نیست**. برنامه‌نویس نیز باید با رجیسترها و دستورهای پردازنده آشنا باشد. اگرچه برنامه‌نویسی به زبان اسمبلی از زبان ماشین آسان‌تر است، اما همچنان نوشتن و اشکال‌زدایی برنامه‌های بزرگ در این زبان دشوار و زمان‌بر است.

مثال: قطعه‌کد زیر در زبان اسمبلی پردازنده خانواده x86، دو مقدار ذخیره‌شده در متغیرهای x و y را جمع می‌کند و نتیجه را در متغیر z ذخیره می‌کند.

```
MOV AX , [x]
ADD AX , [y]
MOV [z] , AX
```

دستور اول، مقدار متغیر x را در رجیستر AX قرار می‌دهد. دستور دوم، مقدار متغیر y را به آن اضافه می‌کند و دستور سوم، نتیجه را در متغیر z ذخیره می‌کند. AX یک محل کوچک (Register) برای نگهداری اطلاعات داخل پردازنده (CPU) است.

توجه: پردازنده **فقط** دستورهای **زبان ماشین** را می‌تواند مستقیماً اجرا کند و به تبع نمی‌تواند دستورهای زبان اسمبلی را مستقیماً اجرا کند؛ بنابراین پیش از اجرای یک برنامه به زبان اسمبلی، نرم‌افزاری به نام **اسمبلر (Assembler)** نیاز است تا دستورهای زبان اسمبلی را به **زبان ماشین** ترجمه کند تا پردازنده بتواند آن‌ها را اجرا کند.

اسمبلر (Assembler) : یک نرم‌افزار مترجم است که کد یک برنامه نوشته‌شده به زبان اسمبلی را به زبان ماشین تبدیل می‌کند تا پردازنده بتواند آن را اجرا کند.

واژه **Assembly** از فعل انگلیسی **Assemble** می‌آید؛ یعنی «کنار هم گذاشتن اجزا». در برنامه‌نویسی، **اسمبلر**؛ کد عملیات، رجیسترها و آدرس‌ها را به شکل صفر و یک تبدیل می‌کند و کنار هم قرار می‌دهد تا **دستورهای زبان ماشین** ساخته شود. این فرایند را «اسمبلی» می‌نامند.

زبان‌های سطح بالا (High-Level Languages)

زبان سطح بالا به زبان انسان نزدیک است. زبان‌های سطح بالا برای آسان‌ترکردن برنامه‌نویسی و افزایش سرعت توسعه نرم‌افزار طراحی شدند. در این زبان‌ها، از واژه‌ها و ساختارهایی نزدیک به زبان انگلیسی و نمادهای متداول ریاضی استفاده می‌شود؛ بنابراین، نوشتن، خواندن و یافتن و رفع خطاهای برنامه از زبان ماشین و اسمبلی آسان‌تر است.

مثال: قطعه‌کد زیر در زبان C، دو مقدار ذخیره‌شده در متغیرهای x و y را جمع می‌کند و نتیجه را در متغیر z ذخیره می‌کند.

```
z = x + y;
```

این دستور، مقدار متغیرهای x و y را جمع می‌کند و نتیجه را در متغیر z قرار می‌دهد.

توجه: زبان اسمبلی هم از واژه‌های قابل‌فهم برای انسان استفاده می‌کند. پس تفاوت فقط انگلیسی بودن دستورها نیست؛ تفاوت در شیوه بیان کار است.

در C می‌گوییم: «دو مقدار را جمع کن و نتیجه را در z بگذار.»

```
z = x + y;
```

در اسمبلی باید جزئیات کار با پردازنده را هم مشخص کنیم:

```
MOV AX , [x]
ADD AX , [y]
MOV [z] , AX
```

یعنی مقدار متغیر x را در رجیستر AX قرار بده، مقدار متغیر y را به آن اضافه کن و نتیجه را در z ذخیره کن. پس زبان C به شیوه بیان مسئله نزدیک‌تر است؛ زبان اسمبلی به دستورهای پردازنده.

در زبان‌های سطح بالا مثل C، معمولاً یک دستور مثل `z = x + y;` می‌تواند کاری را انجام دهد که در زبان اسمبلی به چندین دستور نیاز دارد. در زبان سطح بالا، برنامه‌نویس کمتر درگیر جزئیات سخت‌افزاری، مانند ثبات‌ها و دستورهای پردازنده، می‌شود و بیشتر بر **حل**

مسئله و طراحی برنامه تمرکز می‌کند.

زبان‌های سطح بالا وابستگی کمتری به معماری پردازنده دارند و **قابل‌حمل‌تر** هستند؛ یعنی برنامه‌های نوشته‌شده با آن‌ها معمولاً بدون تغییر یا با تغییرات اندک و با استفاده از ابزارهای مناسب، روی سیستم‌های مختلف اجرا می‌شوند. زبان‌های C ، C++، Java و Python از نمونه‌های زبان‌های سطح بالا هستند.

توجه: پردازنده **فقط** دستورهای **زبان ماشین** را می‌تواند مستقیماً اجرا کند و به تبع نمی‌تواند دستورهای زبان سطح بالا مثل زبان C را مستقیماً اجرا کند؛ بنابراین پیش از اجرای یک برنامه به زبان سطح بالا، نرم‌افزاری به نام **کامپایلر** یا **مفسر** نیاز است تا دستورهای زبان سطح بالا را به **زبان ماشین** ترجمه کند تا پردازنده بتواند آن‌ها را اجرا کند.

کامپایلر (Compiler) : کامپایلر، نرم‌افزار مترجمی است که کل کد منبع یک برنامه نوشته‌شده به زبان سطح بالا را به‌صورت یکجا به زبان ماشین یا کد میانی تبدیل می‌کند تا پردازنده بتواند آن را اجرا کند. در روش کامپایلری مرحله ترجمه و اجرا از هم جداست. در زبان‌هایی مانند C ، پس از ترجمه کد منبع و پیوند با بخش‌های مختلف برنامه، یک فایل اجرایی تولید می‌شود. این فایل در ویندوز اغلب پسوند exe. دارد. برنامه کامپایل‌شده معمولاً سریع‌تر از اجرای همان برنامه به روش تفسیری است. زبان C و C++ کامپایلری هستند.

توجه کنید که خروجی کامپایلر همیشه مستقیماً فایل اجرایی نیست؛ ممکن است ابتدا **کد شیء** تولید شود و پس از عملیات پیوند، فایل اجرایی ساخته شود. **کد منبع (.c) ← کامپایلر ← کد شیء (.obj) ← پیونددهنده ← فایل اجرایی(.exe)**

کد شیء شامل دستورهای ماشین است، اما معمولاً هنوز به‌تنهایی قابل اجرا نیست؛ زیرا باید با سایر کدهای شیء و کتابخانه‌های موردنیاز پیوند داده شود.

زبان C قابل‌حمل (Portable) است؛ یعنی کد مبدأ (Source Code) آن را می‌توان بدون تغییر یا با تغییرات اندک، به سیستم‌های مختلف منتقل و برای هر سیستم **مجدداً کامپایل** کرد تا کد ماشین مخصوص آن سیستم تولید شود. پس کد مبدأ C قابل‌حمل است، نه فایل اجرایی تولیدشده.

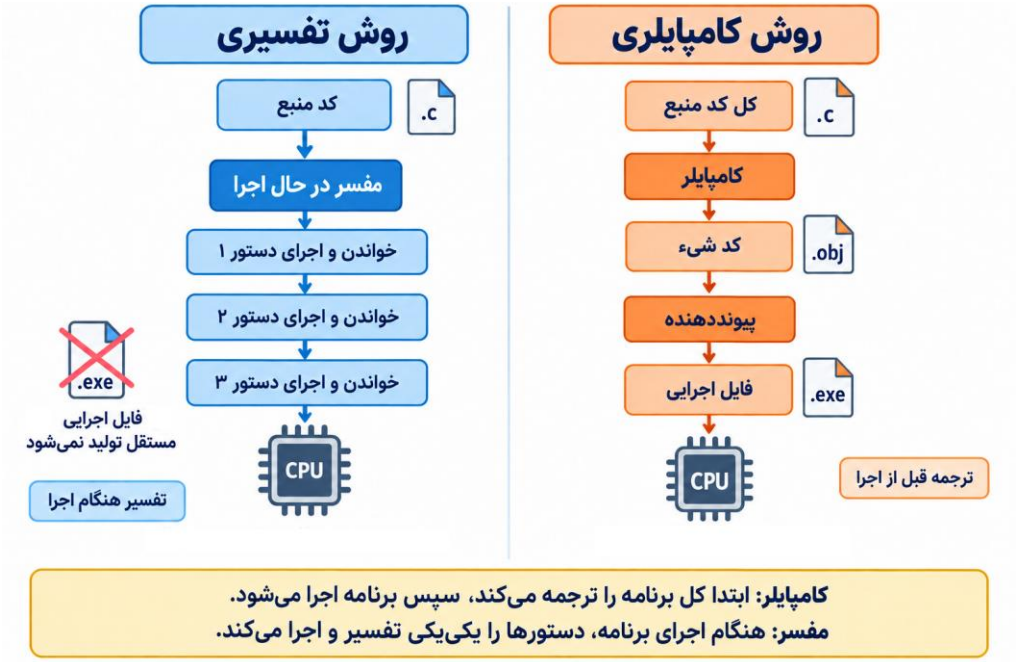
خلاصه: کل برنامه ابتدا ترجمه و سپس اجرا می‌شود.

مفسر (Interpreter) : مفسر، نرم‌افزار مترجمی است که هنگام اجرای برنامه، دستورهای برنامه را یکی‌یکی تفسیر و اجرا می‌کند. در روش مفسری مرحله ترجمه و اجرا به صورت دستوربه‌دستور پیش می‌رود. در این روش فایل اجرایی مستقلی تولید نمی‌شود؛ بنابراین چون دستورها هنگام اجرای برنامه تفسیر می‌شوند، سرعت اجرا معمولاً کمتر از برنامه‌ای است که از قبل به زبان ماشین کامپایل شده است. نسخه‌های اولیه زبان BASIC به روش تفسیری اجرا می‌شدند.

زبان‌های مفسری قابل‌حمل (Portable) هستند؛ یعنی کد مبدأ (Source Code) آن‌ها را می‌توان بدون تغییر یا با تغییرات اندک به سیستم‌های مختلف منتقل کرد و با **مفسر مخصوص همان سیستم** اجرا نمود. بنابراین کد مبدأ قابل‌حمل است، نه نرم‌افزار مفسر؛ زیرا برای هر سیستم‌عامل و معماری پردازنده، نسخه سازگار مفسر لازم است.

خلاصه: برنامه هنگام اجرا، دستوربه‌دستور تفسیر و اجرا می‌شود.

مثال: برای درک تفاوت کامپایلر و مفسر، یک کنفرانس خبری را در نظر بگیرید. اگر مترجم ابتدا تمام سخنان گوینده را دریافت کند و سپس ترجمه کامل آن را ارائه دهد، عملکرد او شبیه **کامپایلر** است. اما اگر مترجم پس از هر جمله، همان جمله را ترجمه کند تا شنوندگان هم‌زمان صحبت‌ها را دنبال کنند، عملکرد او شبیه **مفسر** است.



روش ترکیبی (Hybrid Approach) : برخی زبان‌ها مانند جاوا، از ترکیب **کامپایل** و **مفسر** استفاده می‌کنند، در این روش، کامپایلر کد منبع را یک بار به **بایت‌کد (Bytecode)** تبدیل می‌کند و همان بایت‌کد، با کمک **نسخه مخصوص مفسر برای هر سیستم‌عامل** اجرا می‌شود. برای مثال، یک بایت‌کد جاوا می‌تواند با مفسر جاوا در ویندوز، لینوکس و مک اجرا شود؛ بنابراین نیازی نیست برنامه برای هر سیستم دوباره کامپایل شود. مزیت روش ترکیبی به روش مفسر خالص این است که **بایت‌کد** نوعی کد میانی است که از کد اصلی برنامه ساده‌تر است؛ بنابراین مفسر آن را راحت‌تر و سریع‌تر پردازش می‌کند. برای هر سیستم‌عامل، نسخه مخصوصی از مفسر وجود دارد؛ بنابراین همان بایت‌کد می‌تواند بدون کامپایل مجدد، روی ویندوز، لینوکس و مک اجرا شود. این ویژگی، **قابلیت حمل برنامه** را افزایش می‌دهد. روش ترکیبی بیشتر همان روش مفسری است که کد منبع آن توسط یکبار کامپایل خلاصه‌تر شده‌است.

زبان‌های سطح میانی: زبان C به‌عنوان یک زبان سطح میانی (middle-level language) نیز شناخته می‌شود. بدین معنا که علاوه بر داشتن قابلیت‌های زبان‌های سطح بالا ، قابلیت‌های زبان‌های سطح پایین (مثل اسمبلی) را نیز داراست.

سیستم عامل (Operating System)

سیستم عامل نرم‌افزار اصلی کامپیوتر است که در دسته نرم‌افزارهای سیستمی قرار دارد. سیستم عامل مدیریت نرم‌افزاری تمامی عملیات داخل یک کامپیوتر را بر عهده دارد. سیستم‌عامل، وظیفه **مدیریت منابع سخت‌افزاری** مانند پردازنده، حافظه و دستگاه‌های ورودی و خروجی، **مدیریت سیستم فایل (File System)** و **مدیریت اجرای فرآیندهای کاربر** را برعهده دارد تا برنامه‌ها بتوانند به‌صورت **ایمن، کارآمد و هم‌زمان** اجرا شوند. همچین سیستم عامل امکان ارتباط آسان کاربر نهایی و برنامه‌های کاربردی با سخت‌افزار را فراهم می‌سازد.
به طور کلی سیستم‌عامل دو نقش اصلی دارد:

۱- ماشین توسعه‌یافته (Extended Machine) یا ماشین مجازی (Virtual Machine)

در این حالت، سیستم عامل **به‌طور مستقیم** نیاز انسان را مرتفع می‌کند. سیستم‌عامل پیچیدگی‌های سخت‌افزار و سیستم فایل را از دید کاربر (انسان) پنهان می‌کند و امکاناتی ساده مانند محیط دسکتاپ، منوی Start ، New Folder ، Copy & Paste ، File Explorer و اجرای برنامه Paint در محیط دسکتاپ ویندوز را در اختیار او قرار می‌دهد. کاربر از طریق پوسته، برنامه Paint را اجرا می‌کند، اما Paint برای استفاده از پردازنده، حافظه و دستگاه‌های ورودی و خروجی، درخواست‌های خود را به هسته می‌فرستد. البته پس از اجرای Paint ، کاربر مستقیماً با رابط خود Paint نیز تعامل می‌کند.
برای مثال دیگر، هنگام New Folder ، کاربر نیازی ندارد بداند اطلاعات آن چگونه در دیسک ذخیره می‌شود؛ سیستم‌عامل تمام موارد لازم برای ایجاد یک Folder جدید را انجام می‌دهد. در این شرایط سیستم عامل نقش ماشین مجازی را برای کاربر ایفا می‌کند.
کاربر از طریق **پوسته (Shell)** به عنوان رابط متنی یا گرافیکی به امکانات سیستم‌عامل دسترسی پیدا می‌کند. بنابراین، پوسته واسط بین کاربر نهایی و سیستم‌عامل است؛ پوسته درخواست‌های کاربر را دریافت می‌کند و برای اجرا در اختیار **هسته (Kernel)** قرار می‌دهد؛

سپس نتیجه را به کاربر نمایش می‌دهد.

محیط دسکتاپ و File Explorer ویندوز نوعی **پوسته گرافیکی** هستند.

محیط Command Prompt و PowerShell نوعی **پوسته متنی** هستند و فقط هنگام اجرا وارد RAM می‌شوند.

پوسته، یک نرم‌افزار سیستمی است که ارتباط میان کاربر و سیستم‌عامل را فراهم می‌کند. پوسته هنگام شروع محیط کاربری یا اجرای خط فرمان در حافظه قرار می‌گیرد و برخلاف هسته، می‌تواند بسته یا دوباره اجرا شود.

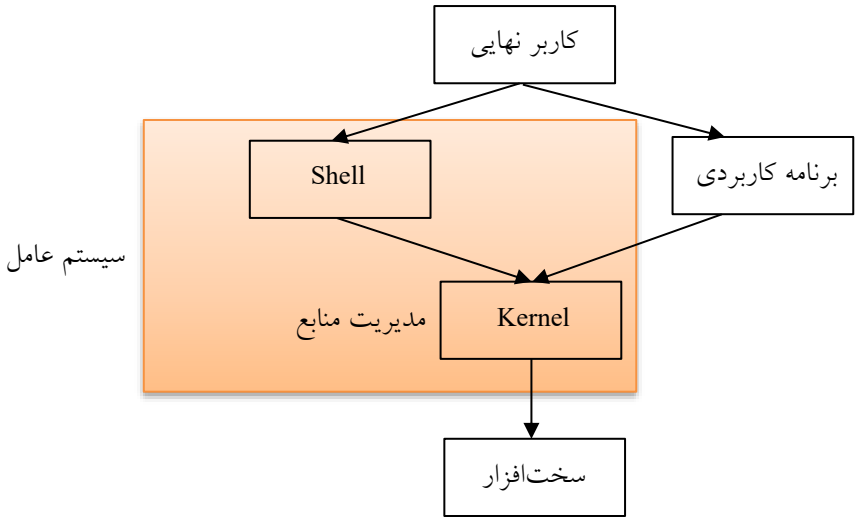
۲ – مدیر منابع (Resource Manager)

در این حالت، سیستم عامل **به‌طور غیرمستقیم** نیاز انسان را مرتفع می‌کند. چون اول سیستم عامل **به‌طور مستقیم** نیازهای برنامه کاربری مثل **پردازنده، حافظه** و **دستگاه‌های ورودی و خروجی** را مرتفع می‌کند، و دوم برنامه کاربردی مثل نرم‌افزار Word نیاز انسان را مرتفع می‌کند. بخش اصلی سیستم‌عامل، **هسته (Kernel)** نام دارد. هسته وظیفه **مدیریت منابع سخت‌افزاری** مانند **پردازنده، حافظه** و **دستگاه‌های ورودی و خروجی، مدیریت سیستم فایل** و **مدیریت اجرای فرآیندها** را برعهده دارد.

برنامه‌های کاربردی برای استفاده از منابع کامپیوتر، درخواست‌های خود را از طریق **فراخوانی‌های سیستمی (System Calls)** به هسته ارسال می‌کنند. ولی هسته، مستقیماً با سخت‌افزار ارتباط برقرار می‌کند. برای‌مثال، برنامه کاربردی Word برای ذخیره‌کردن یک فایل، درخواست خود را از طریق فراخوان سیستمی به هسته می‌فرستد و هسته عملیات دسترسی به حافظه ذخیره‌سازی را مدیریت می‌کند.

همچنین کاربر نهایی نیز مستقیماً با هسته ارتباط برقرار نمی‌کند؛ بلکه کاربر درخواست خود را از طریق **پوسته** یا **برنامه‌های کاربردی** اعلام می‌کند. برای‌مثال اگر در File Explorer گزینه New Folder توسط کاربر نهایی صادر شود، درخواست از طریق پوسته گرافیکی به هسته می‌رسد.

هسته سیستم عامل هنگام راه‌اندازی کامپیوتر در حافظه اصلی (RAM) بارگذاری می‌شود و تا زمان خاموش‌شدن سیستم فعال می‌ماند.



سیستم عامل بین **کاربر نهایی** و **برنامه کاربردی** و سخت‌افزار کامپیوتر قرار می‌گیرد تا کاربر نهایی و برنامه کاربردی بتوانند به آسانی با سخت‌افزار کار کنند.

- پوسته (**Shell**) رابط میان کاربر نهایی و سیستم‌عامل
- هسته (**Kernel**) مدیریت منابع

روش‌های مرتفع‌سازی نیاز انسان در سیستم‌های کامپیوتری

به طور کلی نیاز انسان‌ها در سیستم‌های کامپیوتری به دو روش زیر مرتفع می‌گردد:

۱- توسط سیستم عامل

در این حالت نیازهای انسان توسط سیستم عامل مرتفع می‌گردد. انسان درخواست‌هایش را تقدیم سیستم عامل می‌کند و سیستم عامل انجام می‌دهد بدون آنکه نیاز باشد انسان از جزئیات سخت‌افزار باخبر باشد. در واقع در این حالت سیستم عامل یک ماشین مجازی به شکلی کاملا انتزاعی (بیانی) و نزدیک به زبان انسان برای انسان فراهم کرده‌است. مانند درخواست نمایش لیست فایل‌ها، کپی و انتقال فایل‌ها توسط کاربران نهایی.

۲ – توسط همکاری برنامه‌های کاربردی و سیستم عامل

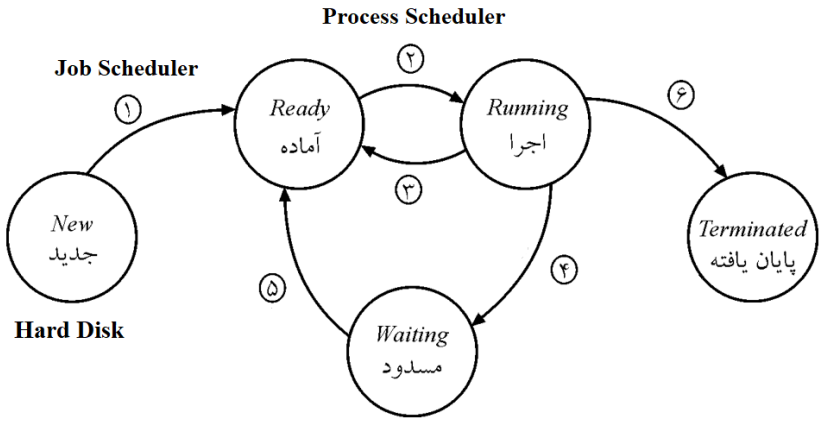
در این حالت نیازهای انسان توسط همکاری برنامه‌های کاربردی و سیستم عامل مرتفع می‌گردد. لازم به ذکر است که نیازهای انسان توسط برنامه‌های کاربردی وقتی مرتفع می‌گردد که نیازهای برنامه‌های کاربردی جهت انجام توسط سیستم عامل مرتفع گردد که نتیجه آن برآورده شدن نیاز انسان است. بنابراین وظیفه تخصیص منابع به برنامه‌های کاربردی همچون **حافظه اصلی** (RAM)، **پردازنده** (CPU) و **دستگاه‌های ورودی و خروجی** برعهده سیستم عامل است.

چرخه حیات فرآیند

یک برنامه به خودی خود فرآیند نیست. زیرا برنامه یک موجودیت غیرفعال (Passive) است که مثلاً در قالب یک فایل با پسوند *.exe و بر روی هارد دیسک ذخیره شده است. اما فرآیند یک موجودیت فعال (Active) می‌باشد که در حال پیشروی است.

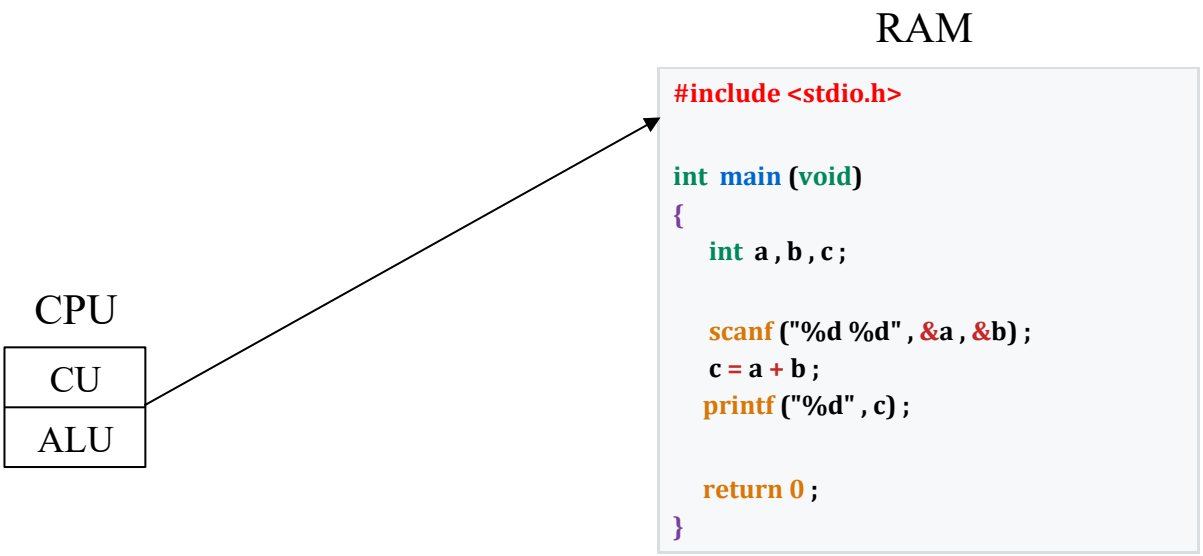
برنامه جمع دو عدد پس از پیاده‌سازی و ایجاد فایل کد منبع sum.c ، کد اجرایی exe آن توسط کامپایلر C به صورت sum.exe ایجاد می‌شود و روی هارد دیسک ذخیره می‌شود. اگر روی فایل sum.exe در سیستم عامل ویندوز کلیک شود، Job به Process تبدیل می‌شود و در حالت آماده (Ready) قرار می‌گیرد. این انتقال توسط زمانبند کار (Job Scheduler) انجام می‌شود.

از لحظه‌ای که یک Job توسط زمانبند کار انتخاب و وارد گردونه‌ی مراحل مختلف می‌شود، تا لحظه‌ای که اجرای آن به طور کامل خاتمه یافته و از سیستم خارج می‌شود، فرآیند (Process) نام دارد.



مثال: **برنامه جمع دو عدد توسط زبان C به عنوان فرآیند P1 در حاظه به صورت زیر است:**

کلمات کلیدی: کامپیوتر، برنامه، الگوریتم، حافظه، پردازنده و سیستم عامل



پس از قرار گرفتن فرآیند P1 در حافظه، CPU خط به خط آنرا واکشی، رمزگشایی و اجرا می‌کند.

یک فرآیند طی مدت زمانی که در سیستم وجود دارد، می‌تواند حالت‌های زیر را داشته باشد:

جدید (New) : فرآیند در حال ایجاد شدن است.

آماده (Ready) : فرآیند درون حافظه قرار دارد و منتظر است تا نوبت CPU به آن برسد.

در حال اجرا (Running) : فرآیند درون حافظه قرار دارد و CPU هم دارد و در حال اجراست.

در حال انتظار (Waiting) : در این حالت فرآیند منتظر وقوع یک رویداد است (Event)، برای مثال منتظر تکمیل شدن یک عمل I/O است. در واقع وقتی فرآیندی در حالت انتظار است، نیازی به CPU ندارد. به این حالت مسدود یا Blocked نیز می‌گویند.

خاتمه یافته (Terminated) : اجرای فرآیند پایان یافته است.

نمونه‌هایی از سیستم‌عامل‌ها و تاریخچه آن‌ها

از سیستم‌عامل‌های رایج می‌توان به ویندوز، سیستم‌عامل‌های لینوکسی و macOS برای رایانه‌های شخصی و اندروید، iOS و iPadOS برای گوشی‌های هوشمند و تبلت‌ها اشاره کرد. برای آشنایی با روند شکل‌گیری این سیستم‌عامل‌ها، تاریخچه آن‌ها را به‌ترتیب بررسی می‌کنیم.

یونیکس (Unix) : توسعه یونیکس در سال ۱۹۶۹ در آزمایشگاه‌های بل شرکت AT&T ، توسط **کن تامپسون و دنیس ریچی** آغاز شد. زبان C نیز در اوایل دهه ۱۹۷۰ توسط **دنیس ریچی** در همین آزمایشگاه ساخته شد. تامپسون و ریچی در سال ۱۹۷۳ هسته و بسیاری از اجزای اصلی یونیکس را با زبان C بازنویسی کردند و تنها بخش‌های وابسته به سخت‌افزار را به زبان اسمبلی نگه داشتند. این کار، انتقال یونیکس به سخت‌افزارهای مختلف را آسان‌تر کرد. یونیکس با پشتیبانی از چند کاربر و اجرای هم‌زمان چند برنامه، تأثیر مهمی بر طراحی سیستم‌عامل‌های بعدی گذاشت.

داس (MS-DOS) و ویندوز (Windows) : مایکروسافت در سال ۱۹۸۱ سیستم‌عامل MS-DOS را برای رایانه‌های شخصی عرضه کرد. کاربران در این سیستم‌عامل، کارهای خود را با واردکردن دستورهای متنی انجام می‌دادند. نخستین نسخه ویندوز در سال ۱۹۸۵ به‌عنوان محیطی گرافیکی مبتنی بر DOS عرضه شد و با استفاده از پنجره‌ها، منوها و آیکون‌ها، کار با رایانه را آسان‌تر کرد. ویندوز امروزی سیستم‌عاملی مستقل و اختصاصی است که توسط مایکروسافت توسعه داده می‌شود .

سیستم‌عامل مک (macOS) : اپل در سال ۱۹۸۴ رایانه مکینتاش را همراه با سیستم‌عاملی گرافیکی عرضه کرد. در سال ۲۰۰۱، سیستم‌عامل جدید Mac OS X با **پایه‌ای مبتنی بر یونیکس** عرضه شد. این خانواده بعدها macOS نام گرفت. بنابراین، macOS امروزی از خانواده یونیکس است، اما سیستم‌عامل اولیه مکینتاش مبتنی بر یونیکس نبود.

لینوکس (Linux) : لینوکس هسته‌ای **متن‌باز و شبه‌یونیکس (Unix-like)** است که لینوس تورووالدز توسعه آن را در سال **۱۹۹۱** آغاز کرد. بخش اصلی هسته لینوکس با زبان **C** نوشته شده است؛ هرچند در بخش‌های نزدیک به سخت‌افزار از زبان اسمبلی نیز استفاده می‌شود. شبه‌یونیکس‌بودن یعنی طراحی و شیوه کار آن از یونیکس الهام گرفته، اما هسته آن به‌صورت مستقل نوشته شده است. در کاربرد رایج، سیستم‌عامل‌های مبتنی بر این هسته، مانند **اوبونتو**، نیز «لینوکس» نامیده می‌شوند. این سیستم‌عامل‌ها علاوه بر هسته، ابزارها، کتابخانه‌ها و رابط کاربری دارند و در سرورها، رایانه‌های شخصی و سیستم‌های نهفته استفاده می‌شوند.

در نرم‌افزارهای متن‌باز، کد منبع در دسترس است و افراد و شرکت‌ها می‌توانند مطابق مجوز نرم‌افزار در بررسی، اصلاح و توسعه آن مشارکت کنند. بسیاری از این نرم‌افزارها رایگان‌اند، اما متن‌بازبودن لزوماً به معنای رایگان‌بودن نیست.

سیستم‌عامل‌های همراه اپل : اپل در سال ۲۰۰۷ آیفون را همراه با سیستم‌عاملی عرضه کرد که بعدها iOS نام گرفت. امروزه iOS برای آیفون و iPadOS برای آیپد توسعه داده می‌شود. این سیستم‌عامل‌ها با macOS فناوری‌های پایه مشترکی دارند، اما متناسب با دستگاه‌های همراه طراحی شده‌اند.

اندروید (Android) : نخستین نسخه تجاری اندروید در سال ۲۰۰۸ عرضه شد. اندروید سیستم‌عاملی مبتنی بر هسته لینوکس است که با هدایت گوگل و مشارکت توسعه‌دهندگان و شرکت‌های مختلف توسعه می‌یابد. بخش پایه آن متن‌باز است، اما برخی برنامه‌ها و خدمات همراه آن اختصاصی‌اند. اندروید به‌طور گسترده در گوشی‌های هوشمند، تبلت‌ها و دستگاه‌های دیگر استفاده می‌شود .

از سال ۲۰۰۸ تاکنون، توسعه سیستم‌عامل‌ها ادامه یافته است. نسخه‌های جدید ویندوز، سیستم‌عامل‌های لینوکسی، macOS، اندروید و iOS با هدف افزایش امنیت، بهبود کارایی و پشتیبانی از سخت‌افزارهای جدید عرضه شده‌اند. همچنین، اپل در سال **۲۰۱۹** سیستم‌عامل **iPadOS** را برای آیپد معرفی کرد. در این دوره، پشتیبانی از پردازنده‌های چندهسته‌ای، خدمات ابری، دستگاه‌های همراه و قابلیت‌های هوش مصنوعی اهمیت بیشتری پیدا کرده است.