سیستم عامل جلسه 7

قطعه بندی Memory Segmentation

دیدگاه کاربرنسبت به حافظه، با حافظه فیزیکی یکسان نیست. این موضوع برای دیدگاه برنامه نویس نسبت به حافظه هم درست است. در واقع، برخورد با حافظه بر حسب ویژگی‌های فیزیکی آن، برای سیستم عامل و برنامه نویس راحت نیست. چه می‌شد اگر سخت افزار می‌توانست یک راهکار حافظه فراهم کند که دیدگاه برنامه نویس را به حافظه فیزیکی واقعی نگاشت (تبدیل-به نحوی عوض کردن) کند

یادداشت:

قطعه‌بندی Segmentation در حافظه سیستم عامل به معنای تقسیم بخش‌های حافظه رایانه به قطعات کوچک‌تر و اختصاص دادن آن‌ها به فرآیندهای مختلف است. هدف از قطعه‌بندی حافظه، جلوگیری از دسترسی ناصحیح یا غیرمجاز به حافظه توسط یک فرآیند دیگر، افزایش کارایی و بهبود عملکرد سیستم عامل و کنترل منابع سیستم است.

در قطعه‌بندی حافظه، هر فرآیند در سیستم، قسمتی از حافظه را به عنوان فضای کاری خود در اختیار دارد و تنها به آن بخش از حافظه دسترسی دارد.

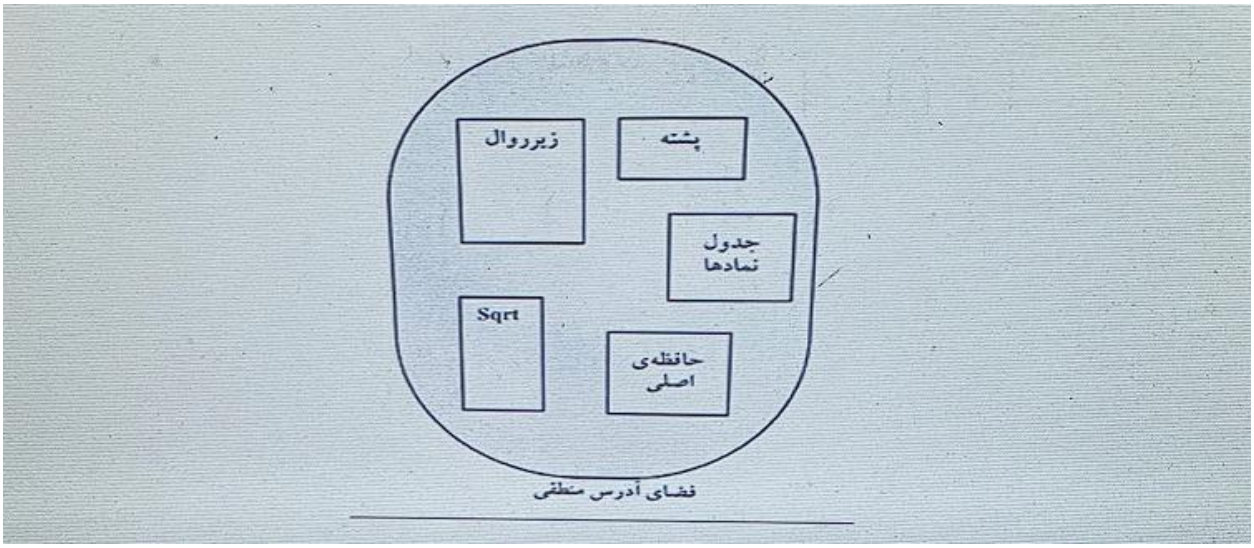
این باعث می‌شود که فرآیندها بتوانند مستقل از یکدیگر کار کنند و مشکلاتی مانند تداخل داده‌ها و تلاقی اطلاعات در حافظه رخ ندهد.

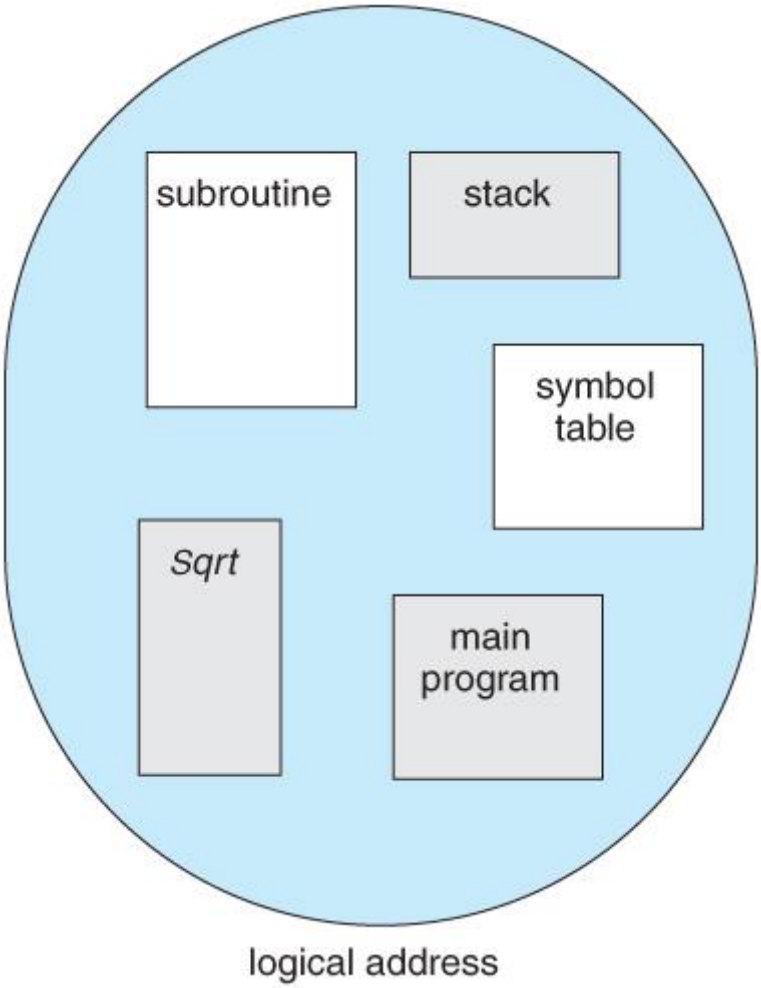
در سیستم عامل، قطعه‌بندی حافظه توسط بخش مدیریت حافظه (Memory Unit Management)MMU کنترل می‌شود. این مدیریت شامل فعالیت‌هایی نظیر تخصیص حافظه به فرآیندها، آزاد سازی حافظه در صورت نیاز و مدیریت حافظه از طریق تکنیک‌هایی مانند Segmentation و Paging است

سیستم آزادی عمل بیشتری برای مدیریت حافظه داشت، در حالی که برنامه نویس یک محیط برنامه نویسی طبیعی‌تر را خواهد داشت. قطعه بندی چنین کاری را انجام می‌دهد.

روش اصلی

برنامه نویس‌ها ترجیح می‌دهند حافظه را به صورت مجموعه ای از قطعاتی با طول متغییر در نظر بگیرند که لازم نیست ترتیبی بین قطعات وجود داشته باشد





هنگام نوشتن برنامه، برنامه نویس آن را به عنوان یک برنامه اصلی با مجموعه ای از متدها، رویه‌ها یا توابع در نظر می‌گیرد. ممکن است شامل ساختمان داده‌های گوناگونی مثل اشیا (Objects)، پشته‌ها (Stacks)، متغییر ها (Variables) و غیره باشد. به هرکدام از این پیمانه‌ها (Module) یا عناصر داده به وسیله نامشان مراجعه می‌شود. برنامه نویس درباره "پشته"، " کتابخانه‌ی توابع ریاضی" و "برنامه اصلی" صحبت می‌کند و به آدرس‌های حافظه ای که این عناصر را در آن جا ذخیره شده اند اشاره نمی‌کند. او دغدغه‌ی این را ندارد که آیا پشته، قبل یا بعد از تابع sqrt() ذخیره شده است یا نه. هر کدام از این قطعات، اندازه‌های متفاوتی دارند. اندازه‌ی آن‌ها به وسیله‌ی اهداف آن‌ها در

برنامه مشخص می‌شود. عناصر موجود در یک قطعه، به وسیله‌ی آفست آن‌ها از آغاز قطعه مشخص می‌شود: مثل اولین دستور برنامه، هفتمین مدخل بر پشته در داخل پشته، پنجمین دستور ()sqrt و غیره...

قطعه بندی (segmentation) یک الگوی مدیریت حافظه است که این دیدگاه برنامه نویس نسبت به حافظه را پشتیبانی می‌کند. فضای آدرس منطقی (Logical Address Space)، مجموعه ای از قطعات است. هر قطعه دارای نام و طول است. آدرس‌ها، نام قطعه و آفستی در داخل قطعه را مشخص می‌کنند.

بنابراین برنامه نویس هر آدرس را با دو کمیت مشخص می‌کند:

نام قطعه و یک آفست (فاصله از مبدا)

برای سهولت پیاده سازی، قطعات شماره گذاری می‌شوند و از طریق این شماره به آن‌ها مراجعه می‌شود (نه نام قطعه). بنابراین این آدرس منطقی شامل یک دوتایی است

<آفست D=displacement و شماره صفحه p=pageNumber >

معمولاً وقتی برنامه ترجمه (Compile - Interpreter) می‌شود، کامپایلر به طور خودکار قطعات برنامه‌ی ورودی را می‌سازد.

سخت افزار قطعه بندی (Segmentation Hardware)

گرچه برنامه نویس می‌تواند از طریق آدرس دو بعدی به اشیای برنامه مراجعه کند ولی حافظه‌ی فیزیکی واقعی، دنباله‌ی یک بعدی ی از بایت‌ها است. لذا باید یک پیاده سازی

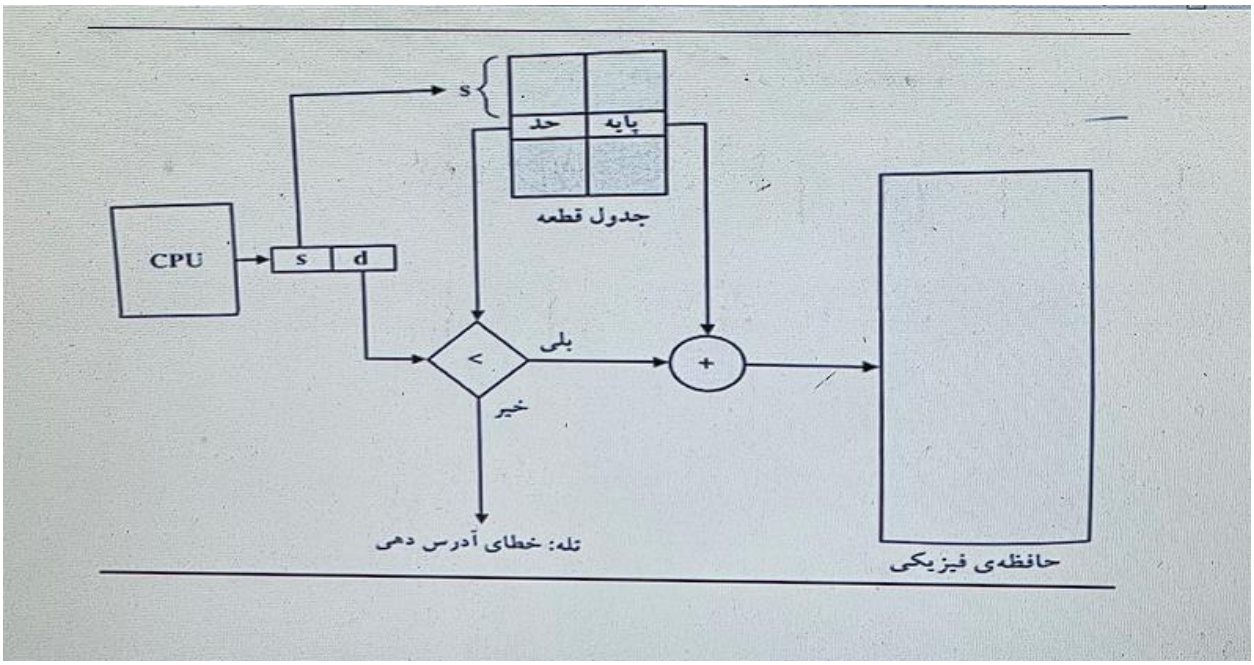
ای را تعریف کنیم که آدرس‌های دو بعدی تعریفی برنامه را به آدرس‌های فیزیکی یک بعدی نگاشت کند.

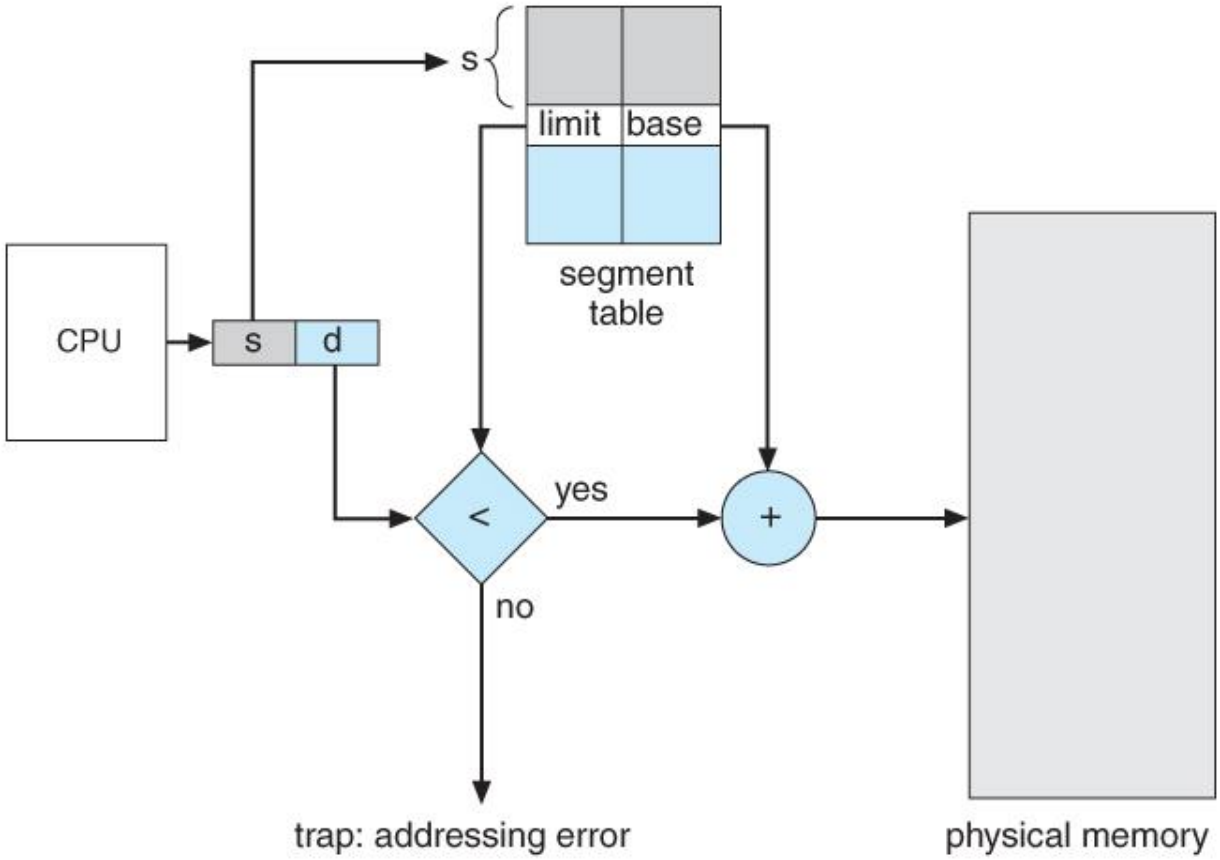
این نگاشت (تبدیل) توسط جدول قطعه (Segment table) انجام می‌شود. هر مدخل جدول قطعه دارای یک پایه‌ی قطعه و یک حد قطعه است

پایه‌ی قطعه شامل آدرس شروع حافظه‌ی فیزیکی است که قطعه در آن جا قرار دارد و حد قطعه، طول قطعه را مشخص می‌کند

آدرس منطقی Logical Address شامل دو بخش است:

شماره‌ی قطعه که با s نمایش داده می‌شود و آفستی که در قطعه که با d نمایش داده می‌شود. شماره‌ی قطعه به عنوان اندیسی در جدول قطعه مورد استفاده قرار می‌گیرد. افست d مربوط به آدرس منطقی باید بین صفر و حد قطعه باشد. اگر نباشد تله ای برای سیستم عامل در نظر می‌گیریم (آدرس منطقی از قطعه خارج شده است). اگر افست معتبر باشد و به پایه‌ی قطعه اضافه می‌شود تا آدرس فیزیکی بایت مطلوب تولید شود. قطعه در واقع آرایه ای از جفت پایه – حد است

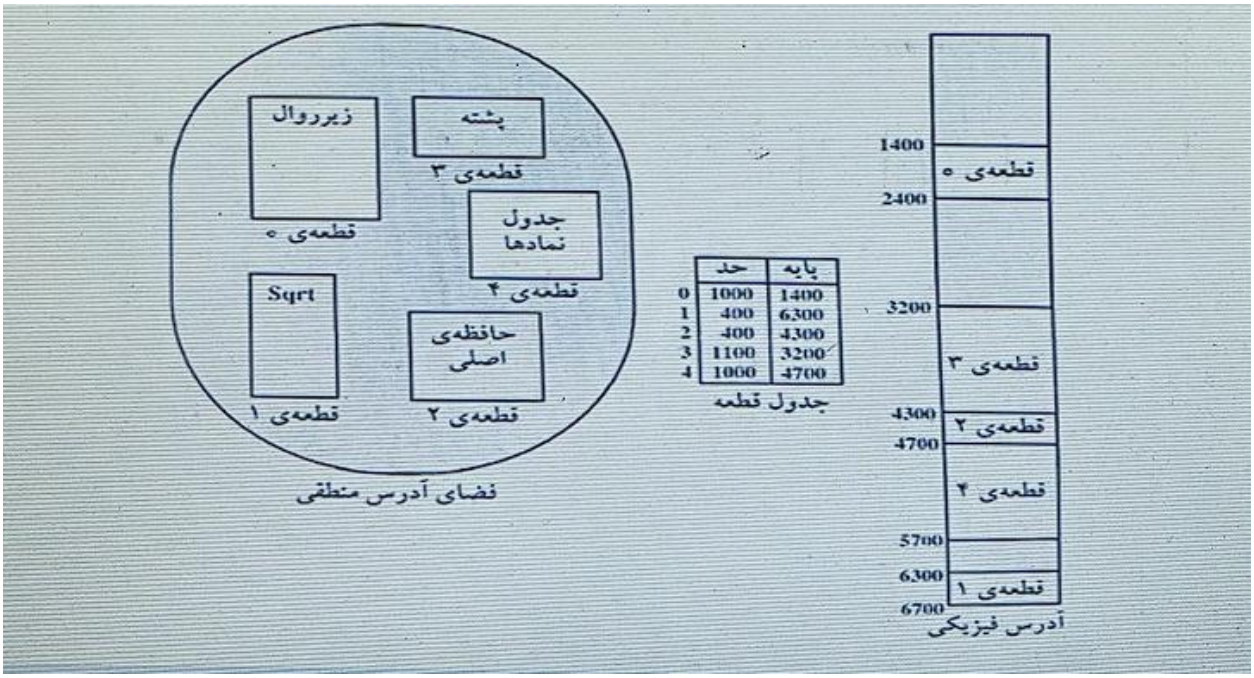

s
حد
پایه
جدول قطعه
CPU
s
d
<
بلی
+
خیر
تله: خطای آدرس دهی
حافظه‌ی فیزیکی

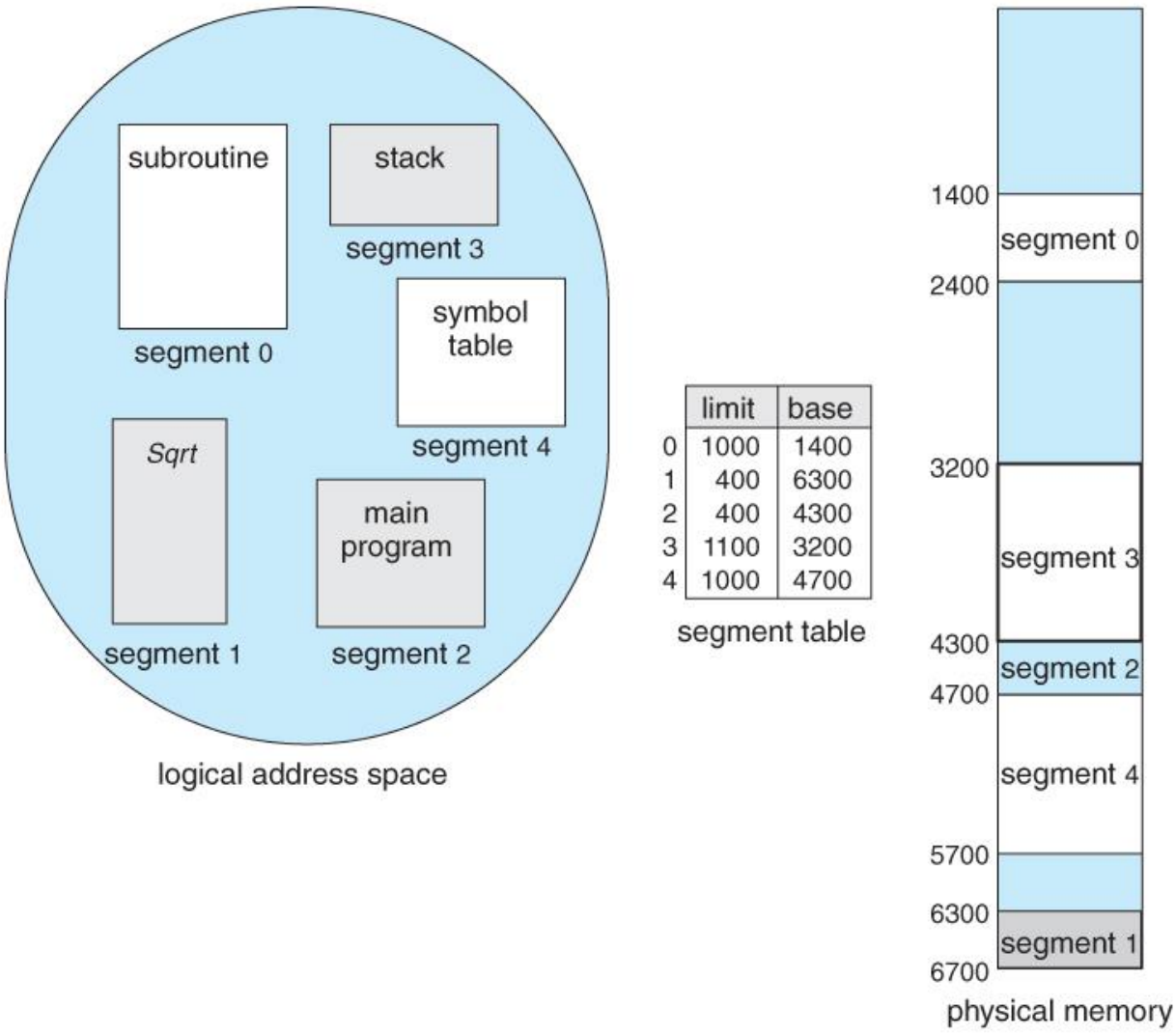

s
limit
base
segment table
CPU
s
d
<
yes
+
no
trap: addressing error
physical memory

در شکل زیر پنج قطعه وجود دارد که از 0 تا 4 شماره گذاری شده اند و این قطعات به صورتی که نشان داده شده است

در حافظه فیزیکی ذخیره شده اند. به ازای هر قطعه یک مدخل در جدول قطعه وجود دارد که آدرس شروع قطعه در حافظه فیزیکی (پایه) و طول قطعه (حد) را مشخص می‌کند.

به عنوان مثال: قطعه‌ی 2 برابر با 400 بایت است که از محل 4300 شروع می‌شود، بنابراین، ارجاع به بایت 53 از قطعه‌ی 2، به محل 4300+53=4353 نگاشت می‌شود. ازجاع به قطعه‌ی 3 و بایت شماره 852 به محل 3200+852=4052 نگاشت می‌شود (3200 پایه‌ی قطعه 3 است). ارجاع به بایت 1222 از قطعه 0 منجر به تله ای به سیستم عامل می‌شود، زیرا طول این قطعه 1000 بایت است.





صفحه بندی (paging)

قطعه بندی اجازه می‌دهد که فضای آدرس یک فرایند، همجوار (پیوسته) نباشد. صفحه بندی، یک طرح دیگر از مدیریت حافظه است که اجازه می‌دهد فضای آدرس فیزیکی فرایند، همجوار نباشد.

صفحه بندی از تکه تکه شدن خارجی و نیاز به فشرده سازی اجتناب می‌کند. در حالی که قطعه بندی نمی‌تواند، علاوه بر این، مساله ی برازش تکه‌هایی با اندازه متغییر را بر روی دیسک ذخیره ساز پشتیبان، حل می‌کند. این مساله به این دلیل به وجود می‌آید

که در صورت نیاز به مبادله‌ی قطعات کد یا داده‌های موجود در حافظه به بیرون از حافظه باید فضایی در ذخیره ساز پشتیبان پیدا شود ذخیره ساز پشتیبان با همان مساله های تکه تکه شدن که در مورد حافظه‌ی اصلی مطرح شد، مواجه است ولی دستیابی به آن کندتر است و در نتیجه فشرده سازی ممکن نیست. به دلیل این امتیازات صفحه بندی نسبت به روش‌های پیشین، شکل‌های مختلف صفحه بندی در اکثر سیستم‌های عامل مورد استفاده قرار می‌گیرد. از سیستم‌های عامل کامپیوترهای بزرگ گرفته تا گوشی‌های هوشمند. صفحه بندی از طریق هماهنگی بین سیستم عامل و سخت افزار کامپیوتر انجام می‌گیرد.

روش اصلی صفحه بندی

یادداشت:

Paging در سیستم عامل، یک روش مدیریت حافظه است که در آن حافظه فیزیکی سیستم به صورت صفحات (Pages) کوچک تقسیم شده و هر صفحه دارای آدرس و شماره صفحه (Page Number) خاصی است. این روش به سیستم‌عامل اجازه می‌دهد تا از حافظه جداگانه از برنامه‌ها استفاده کند و به عنوان یک نقل‌وقوع برای اعمال محدودیت‌ها و مدیریت حافظه استفاده می‌شود./اندازه صفحات معمولا با اندازه fram ها یکسان می باشد

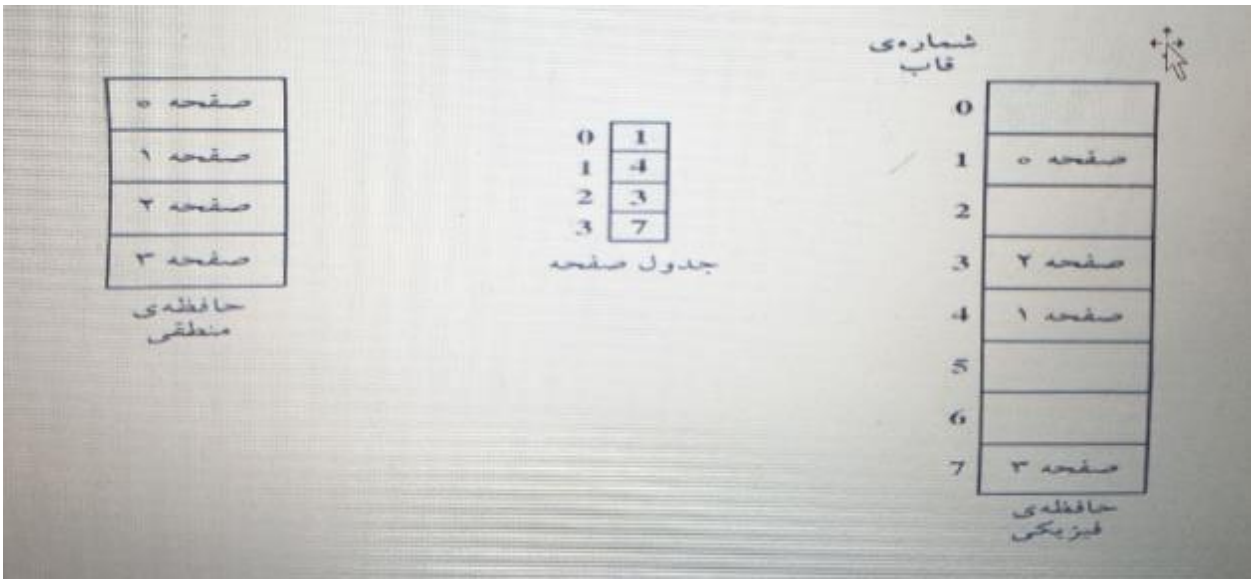
روش اصلی برای پیاده سازی صفحه بندی، تقسیم حافظه‌ی فیزیکی به بلوک‌هایی با اندازه‌ی ثابت به نام قاب (Frames) و تقسیم حافظه‌ی منطقی (Virtual Memory) به بلوک‌هایی با اندازه یکسان به نام صفحات (Pages) است. وقتی فرایندی می‌خواهد

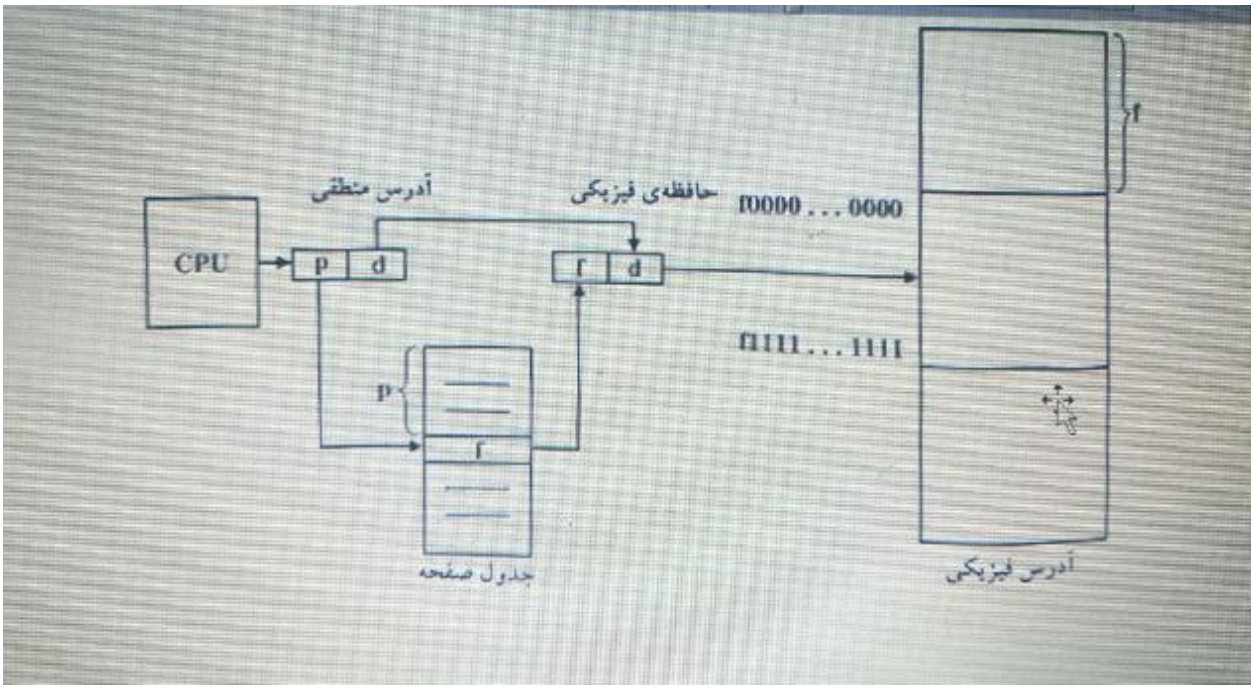
اجرا شود، صفحات آن از سیستم فایل یا ذخیره ساز پشتیبان (Hard Disk - HDD) به قاب‌های (Frame) آزاد حافظه بار می‌شوند. ذخیره ساز پشتیبان به بلوک‌هایی با اندازه‌ی ثابت تقسیم می‌شوند که هم اندازه‌ی حافظه یا خوشه‌هایی از چندین قاب است. این ایده‌ی بسیار ساده، عملکرد فوق العاده و نتیجه‌ی گسترده ای دارد.

هر آدرسی که توسط پردازنده تولید می‌شود به دو بخش تقسیم می‌شود

شماره صفحه (P=PageNumber) و آفست صفحه (D=Displacement)

شماره‌ی صفحه به عنوان اندیسی برای جدول به کار می‌رود. جدول صفحه حاوی آدرس پایه‌ی هر صفحه در حافظه فیزیکی است. این آدرس پایه، با آفست صفحه ترکیب می‌شود تا فضای آدرس فیزیکی تولید و به واحد حافظه ارسال شود.





تفاوت قطعه بندی و صفحه بندی چیست؟

(Pagin vs segmentation)

Segmentation و Paging هر دو به منظور مدیریت حافظه در سیستم‌عامل‌ها استفاده می‌شوند. با این حال، تفاوت‌های زیادی بین این دو روش وجود دارد.

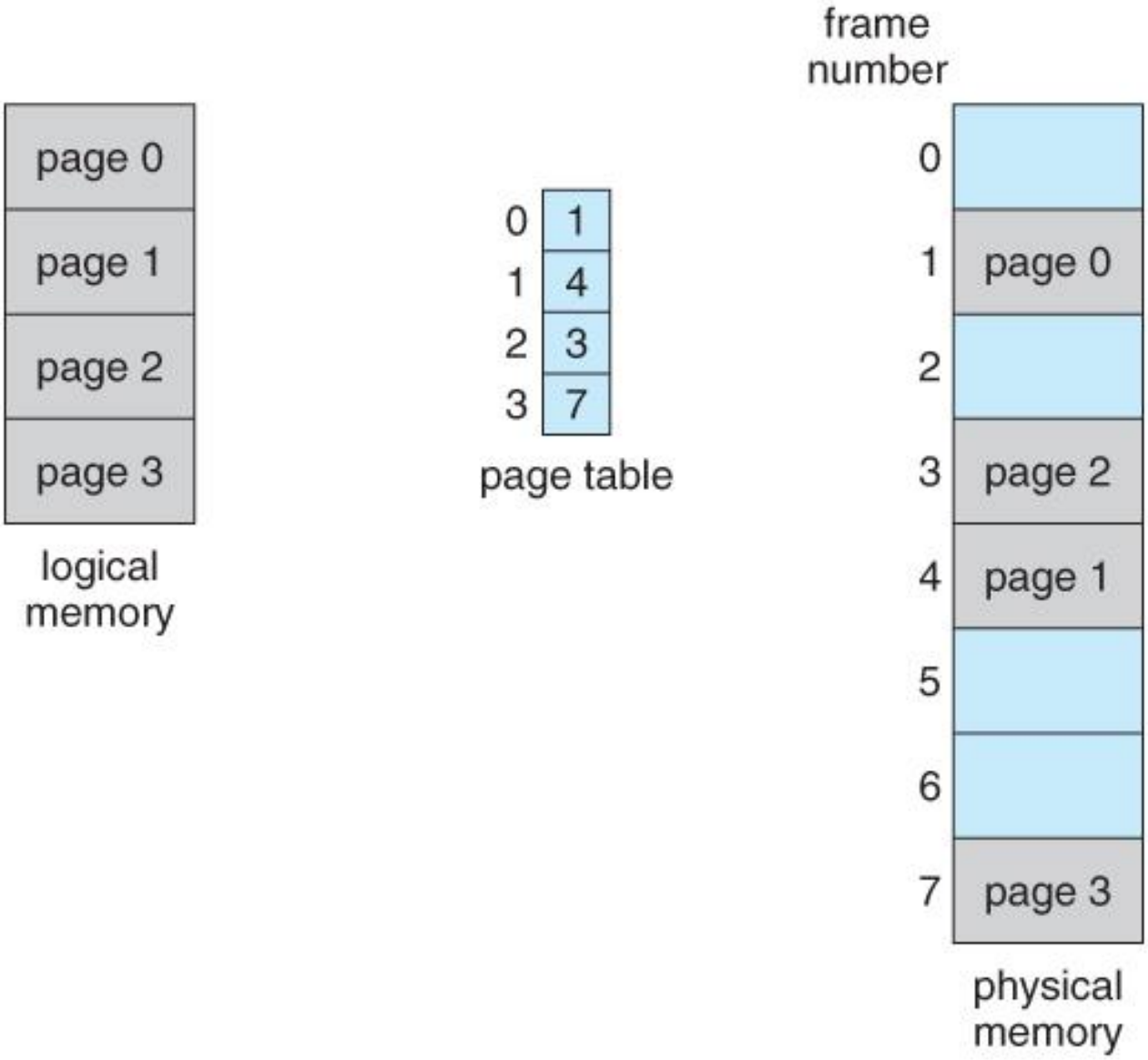
در روش Segmentation، فضای حافظه به بخش‌هایی با اندازه‌های متفاوت تقسیم می‌شود. هر بخش با یک شناسه (segment) تعیین شده است و ابعاد و ظرفیت هر بخش نیز در آن قید می‌شود. به عنوان مثال، یک بخش ممکن است شامل کدهای اجرایی یک برنامه باشد، در حالی که بخش دیگری شامل داده‌های مربوط به آن برنامه است.

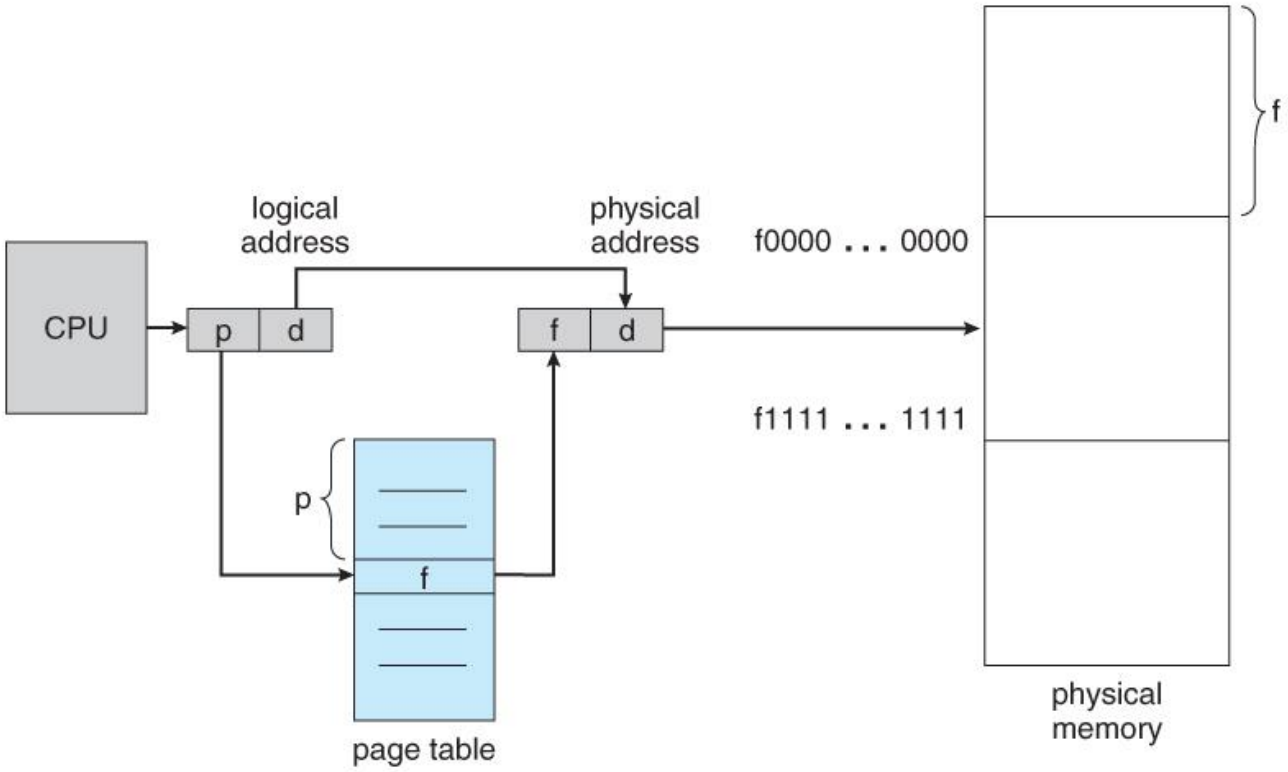
از طرفی، روش Paging فضای حافظه را به صفحات کوچک‌تری تقسیم می‌کند. هر صفحه، با یک شماره صفحه (page number) شناسایی می‌شود و هر صفحه دارای اندازه ثابتی است. در این روش، بخش‌های حافظه به صورت پشت سر هم قرار می‌گیرند و به صورت ناپیوسته استفاده می‌شوند.

با استفاده از روش Paging، سیستم عامل می‌تواند به راحتی صفحات حافظه را به حافظه اصلی (random accress memory – RAM) منتقل کند و از آنها برای دسترسی به داده‌ها و کدهای اجرایی استفاده کند.

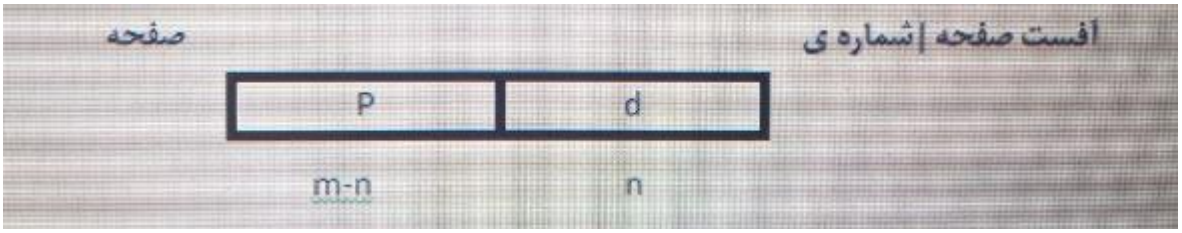
در روش Segmentation، هر بخش به طور جداگانه مدیریت می‌شود و تغییر ظرفیت یک بخش باعث تغییر در فضای حافظه مصرف شده توسط سایر بخش‌ها خواهد شد.

بنابراین، این دو روش از دیدگاه طراحی و مدیریت حافظه از هم متفاوت هستند و هر کدام مزایا و معایب خود را دارند.

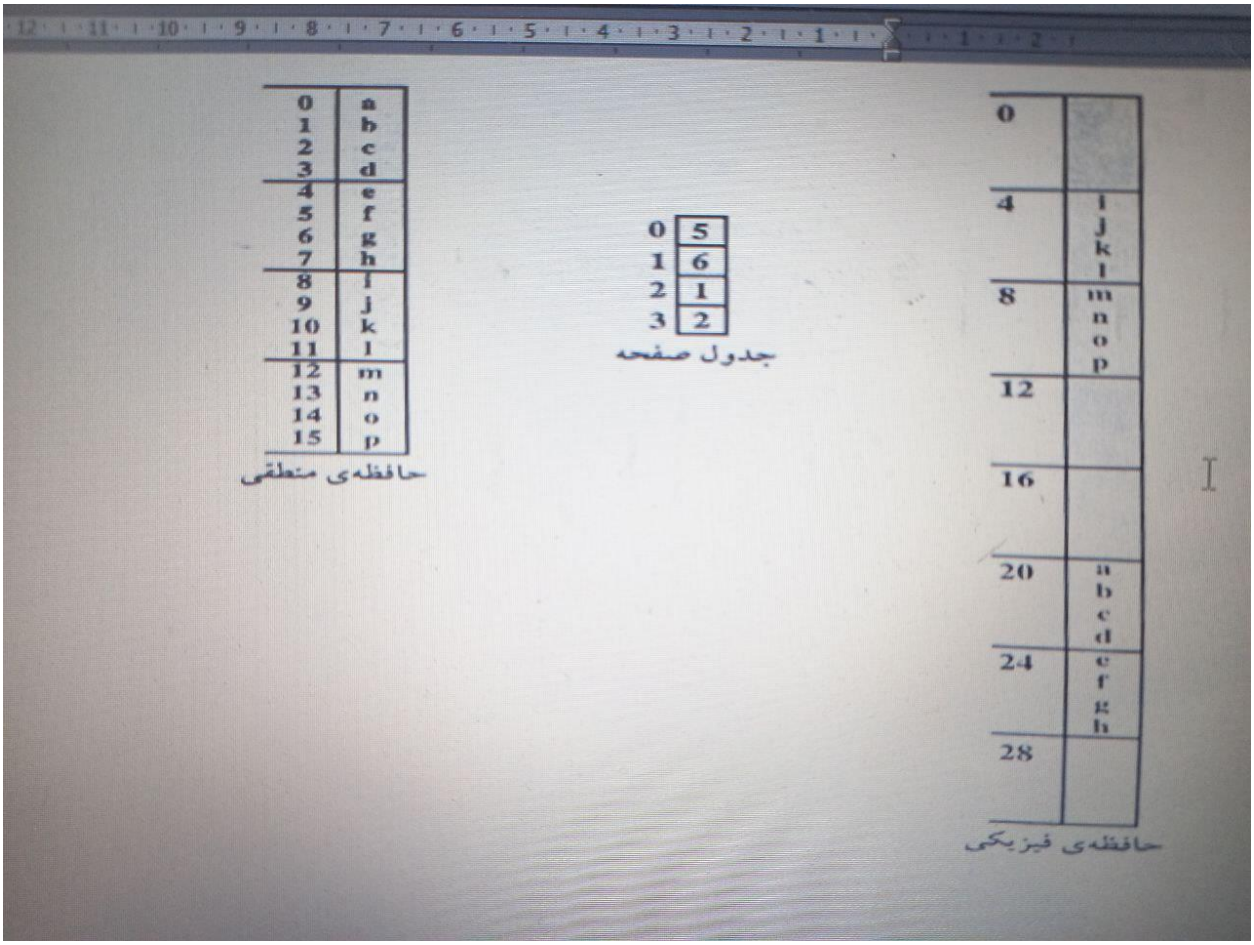

frame
number
page 0
page 1
page 2
page 3
logical
memory
0 1
1 4
2 3
3 7
page table
0
1 page 0
2
3 page 2
4 page 1
5
6
7 page 3
physical
memory

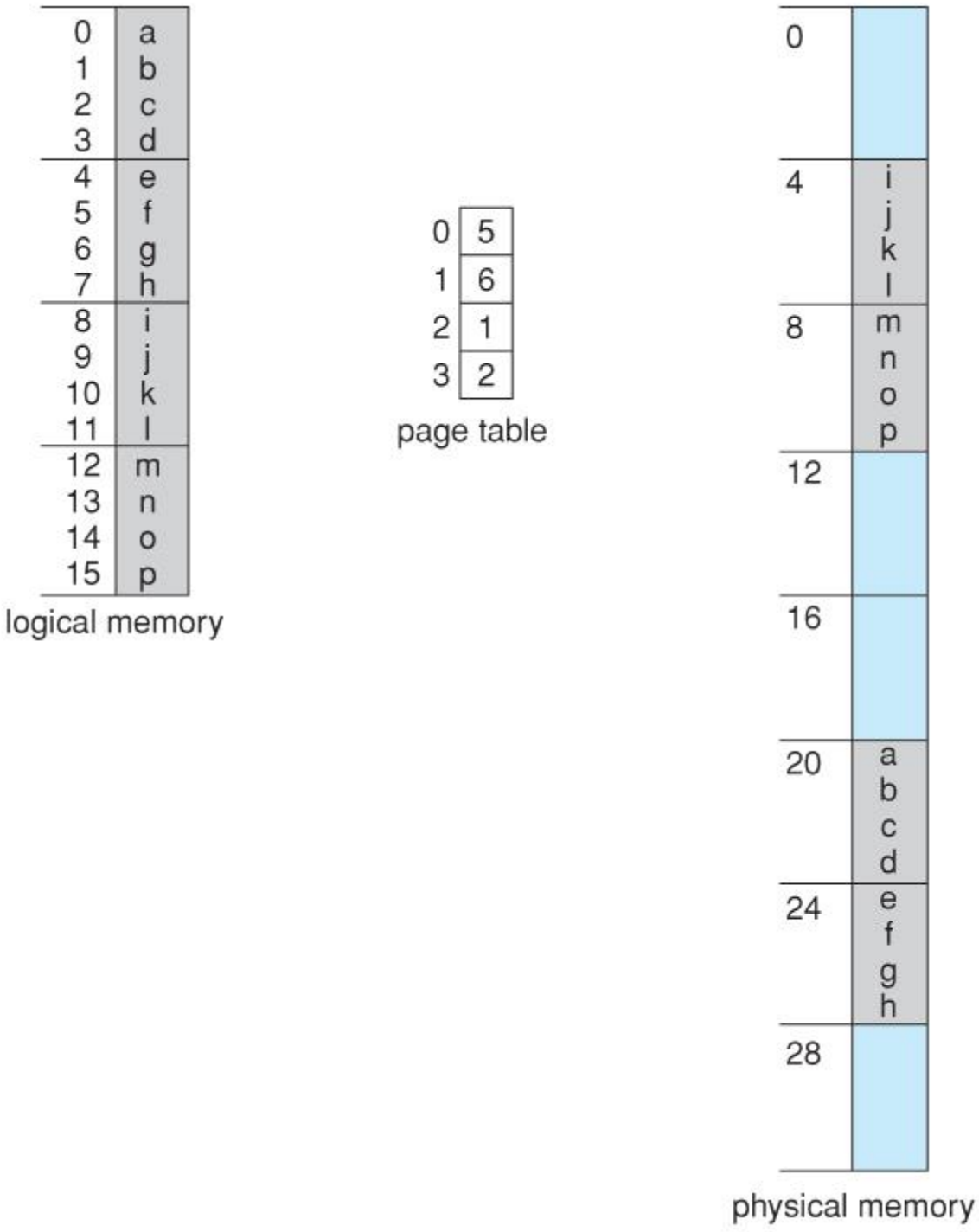


اندازه‌ی صفحه (مثل اندازه‌ی قاب) توسط سخت افزار تعریف ی شود. اندازه‌ی صفحات معمولاً توانی از 2 است. که بر اساس معماری کامپیوتر، بین 512 بایت و 1 گیگابایت به ازای هر صفحه است. چون اندازه صفحه، توان 2 است. ترجمه‌ی آدرس منطقی به شماره صفحه و آفست صفحه، آسان است. اگر اندازه‌ی فضای آدرس منطقی 2^m باشد و اندازه‌ی صفحه برابر با 2^n واحد آدرس (بایت یا کلمه) باشد، آن گاه m-n بیت با ارزش آدرس منطقی، شماره‌ی صفحه و n بیت کم ارزش، آفست صفحه را تعیین می‌کند. بنابراین این آدرس منطقی به صورت زیر است



که در آن p اندیس جدول صفحه و d تفاوت مکان (Displacement) در داخل صفحه است





به عنوان مثال حافظه‌ی شکل بالا را در نظر بگیرید. در این جا $n = 2$ و $m = 4$ است. با استفاده از صفحه‌ی 4 بایتی و حافظه فیزیکی 32 بایتی (8 صفحه)، نشان می‌دهیم که حافظه از دیدگاه کاربر چگونه به حافظه فیزیکی نگاشت می‌شود. آدرس منطقی صفر،

صفحه صفر و آفست صفر است. با مراجعه به جدول صفحه (از طریق اندیس شماره صفحه) متوجه می‌شویم که صفحه‌ی صفر در قاب (Frame) 5 قرار دارد.

لذا آدرس منطقی صفر به آدرس فیزیکی

[= (5x4) + 0] 20 نگاشت می‌شود. آدرس منطقی 3 (صفحه صفر، آفست 3) به آدرس فیزیکی [= (5 x 4) + 3] 23 نگاشت می‌شود و آدرس منطقی 4 نیز صفحه 1 و آفست صفر است. با مراجعه به جدول مشخص می‌شود که صفحه‌ی یک و به قاب 6 نگاشت می‌شود. لذا آدرس منطقی 3 به آدرس فیزیکی [=(6 x 4)+0] 24 نگاشت می‌شود. آدرس منطقی 13 به آدرس فیزیکی 9 نگاشت می‌شود.

آدرس فیزیکی = (طول صفحه x شماره قاب(F)) + offset(d)

Physical address = fream number x freamsize + offset