

سیستم عامل جلسه 7

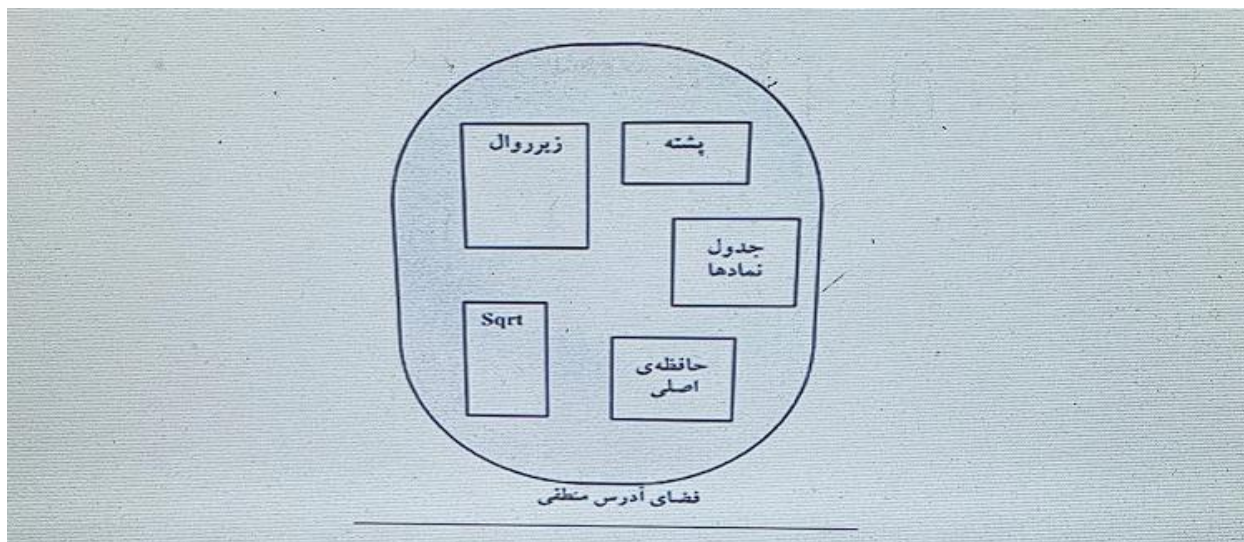
قطعه بندی

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 ویژگی های فیزیکی آن، برای سیستم عامل و برنامه نویس راحت نیست. چه می شد اگر سخت افزار می توانست یک راهکار حافظه فراهم کند که دیدگاه برنامه نویس را به حافظه فیزیکی واقعی نگاشت کند

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

روش اصلی

برنامه نویس ها ترجیح می دهند حافظه را به صورت مجموعه ای از قطعاتی با طول متغیر 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، رویه ها یا توابع در نظر می گیرد. ممکن است شامل ساختمان داده های گوناگونی مثل اشیاء، پشته ها، متغیرها و غیره باشد. به هر کدام از این پیمانه ها یا عناصر داده به وسیله نامشان مراجعه می شود. برنامه نویس درباره "پشته"، "کتابخانه ی توابع ریاضی" و "برنامه اصلی" صحبت می کند و به آدرس های حافظه ای که این عناصر را در آن جا ذخیره شده اند اشاره نمی کند. او دغدغه ی این را ندارد که آیا پشته، قبل یا بعد از تابع $\text{sqrt}()$ ذخیره شده است یا نه. هر کدام از این قطعات، اندازه های متفاوتی دارند. اندازه ی آن ها به وسیله ی اهداف آن ها در برنامه مشخص می شود. عناصر موجود در یک قطعه، به وسیله ی آفست آن ها از آغاز قطعه مشخص می شود: مثل اولین دستور برنامه، هفتمین مدخل بر پشته در داخل پشته، پنجمین دستور $\text{sqrt}()$ و غیره...

قطعه بندی (segmentation) یک الگوی مدیریت حافظه است که این دیدگاه برنامه نویس نسبت به حافظه را پشتیبانی می کند. فضای آدرس منطقی، مجموعه ای از قطعات است. هر قطعه دارای نام و طول است. آدرس ها، نام قطعه و آفستی در داخل قطعه را مشخص می کنند.

بنابراین برنامه نویس هر آدرس را با دو کمیت مشخص می کند:

نام قطعه و یک آفست (فاصله از مبدا)

برای سهولت پیاده سازی، قطعات شماره گذاری می شوند و از طریق این شماره به آن ها مراجعه می شود (نه نام قطعه). بنابراین این آدرس منطقی شامل یک دوتایی است

<آفست و شماره صفحه>

معمولا وقتی برنامه ترجمه می شود، کامپایلر به طور خودکار قطعات برنامه ی ورودی را می سازد.

سخت افزار قطعه بندی

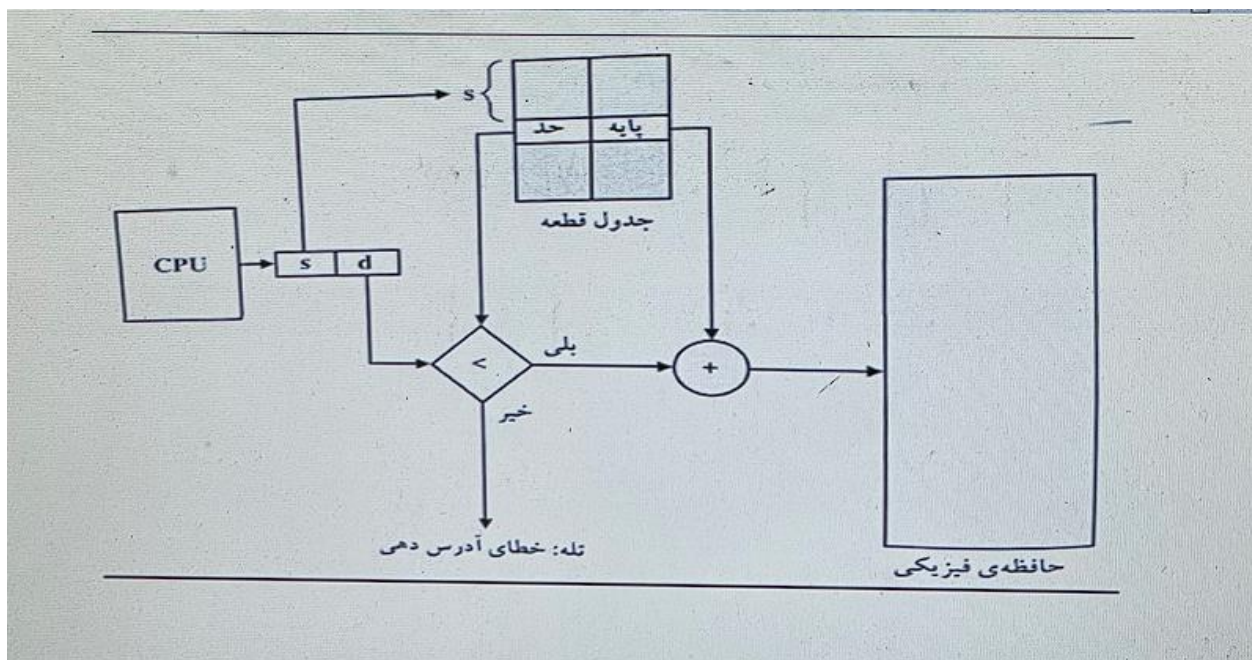
گرچه برنامه نویس می تواند از طریق آدرس دو بعدی به اشیای برنامه مراجعه کند ولی حافظه ی فیزیکی واقعی، دنباله ی یک بعدی ی از بایت ها است. لذا باید یک پیاده سازی ای را تعریف کنیم که آدرس های دو بعدی تعریفی برنامه را به آدرس های فیزیکی یک بعدی نگاشت کند.

این نگاشت توسط جدول قطعه (Segment table) انجام می شود. هر مدخل جدول قطعه دارای یک پایه ی قطعه و یک حد قطعه است

پایه ی قطعه شامل آدرس شروع حافظه ی فیزیکی است که قطعه در آن جا قرار دارد و حد قطعه ، طول قطعه را مشخص می کند

آدرس منطقی شامل دو بخش است:

شماره ی قطعه که با s نمایش داده می شود و آفستی که در قطعه که با d نمایش داده می شود. شماره ی قطعه به عنوان اندیسی در جدول قطعه مورد استفاده قرار می گیرد. آفست d مربوط به آدرس منطقی باید بین صفر و حد قطعه باشد. اگر نباشد تله ای برای سیستم عامل در نظر می گیریم (آدرس منطقی از قطعه خارج شده است). اگر آفست معتبر باشد و به پایه ی قطعه اضافه می شود تا آدرس فیزیکی بایت مطلوب تولید شود. قطعه در واقع آرایه ای از جفت پایه - حد است

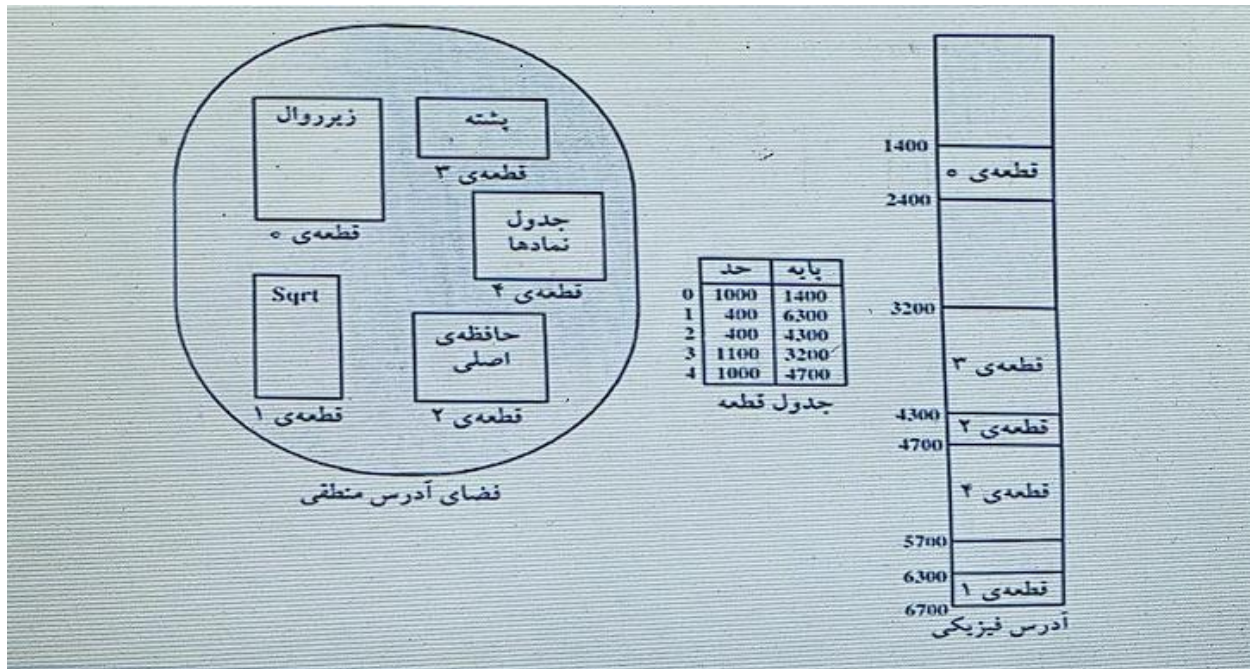


در شکل زیر پنج قطعه وجود دارد که از 0 تا 4 شماره گذاری شده اند و این قطعات به صورتی که نشان داده شده است در حافظه فیزیکی ذخیره شده اند. به ازای هر قطعه یک مدخل در جدول قطعه وجود دارد که آدرس شروع قطعه در حافظه فیزیکی (پایه) و طول قطعه (حد) را مشخص می کند.

به عنوان مثال «

قطعه ی 2 برابر با 400 بایت ست که از محل 4300 شروع می شود ، بنابراین، ارجاع به بایت 53 از قطعه ی 2، به محل $4353 = 53 + 4300$ نگاشت می شود. ارجاع به

قطعه ی 3 و بایت شماره 852 به محل $4052 = 852 + 3200$ نگاشت می شود (3200 پایه ی قطعه 3 است). ارجاع به بایت 1222 از قطعه 0 منجر به تله ای به سیستم عامل می شود، زیرا طول این قطعه 1000 بایت است.



صفحه بندی (paging)

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

صفحه بندی از تکه تکه شدن خارجی و نیاز به فشرده سازی اجتناب می کند. در حالی که قطعه بندی نمی تواند، علاوه بر این، مساله ی برازش تکه هایی با اندازه متغیر را بر روی دیسک ذخیره ساز پشتیبان، حل می کند. این مساله به این دلیل به وجود می آید که در صورت نیاز به مبادله ی قطعات کد یا داده های 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 حافظه ی اصلی مطرح شد، مواجه است ولی دستیابی به آن کندتر است و در نتیجه فشردن سازی ممکن نیست. به دلیل این امتیازات صفحه بندی نسبت به روش های پیشین ، شکل های مختلف صفحه بندی در اکثر سیستم های عامل مورد استفاده قرار می گیرد. از سیستم های عامل کامپیوتر های بزرگ گرفته تا گوشی های هوشمند. صفحه بندی از طریق هماهنگی بین سیستم عامل و سخت افزار کامپیوتر انجام می گیرد.

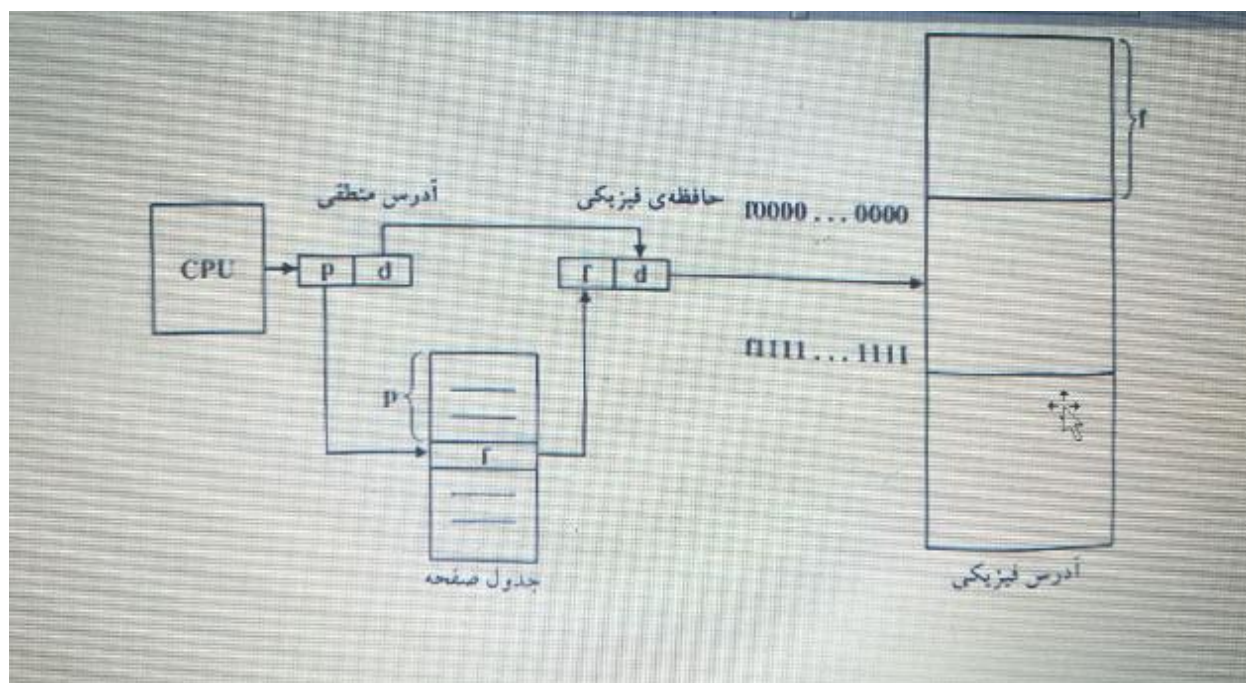
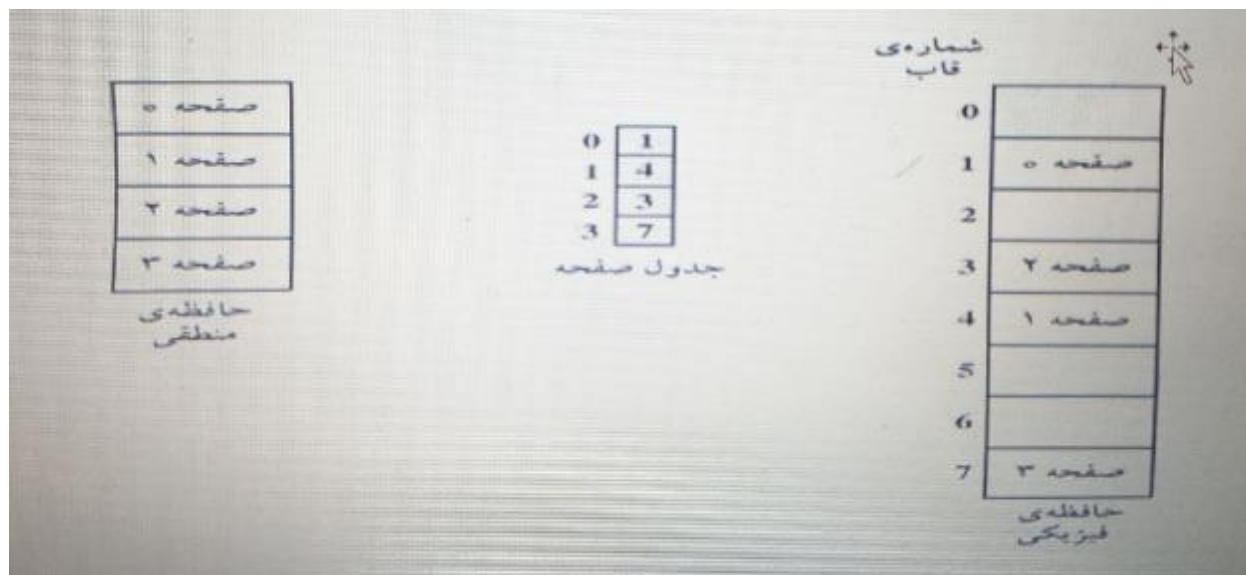
روش اصلی صفحه بندی

روش اصلی برای پیاده سازی صفحه بندی، تقسیم حافظه ی فیزیکی به بلوک هایی با اندازه ی ثابت به نام قاب (Frames) و تقسیم حافظه ی منطقی به بلوک هایی با اندازه یکسان به نام صفحات (Pages) است. وقتی فرایندی می خواهد اجرا شود، صفحات آن از سیستم فایل یا ذخیره ساز پشتیبان (Hard Disk - HDD) به قاب های آزاد حافظه بار می شوند. ذخیره ساز پشتیبان به بلوک هایی با اندازه ی ثابت تقسیم می شوند که هم اندازه ی حافظه یا خوشه هایی از چندین قاب است. این ایده ی بسیار ساده، عملکرد فوق العاده و نتیجه ی گسترده ای دارد.

هر آدرسی که توسط پردازنده تولید می شود به دو بخش تقسیم می شود

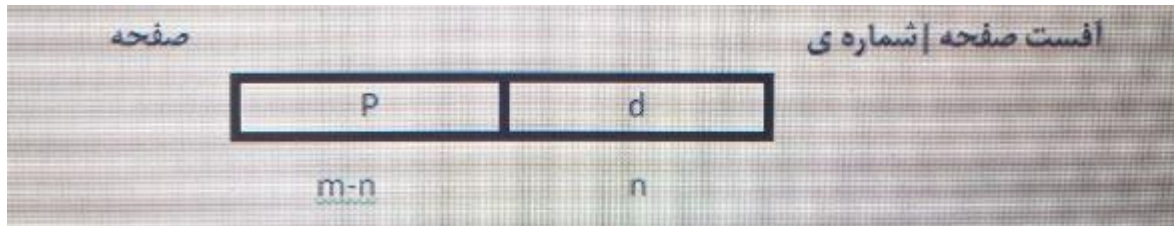
شماره صفحه (P) و آفست صفحه (D)

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

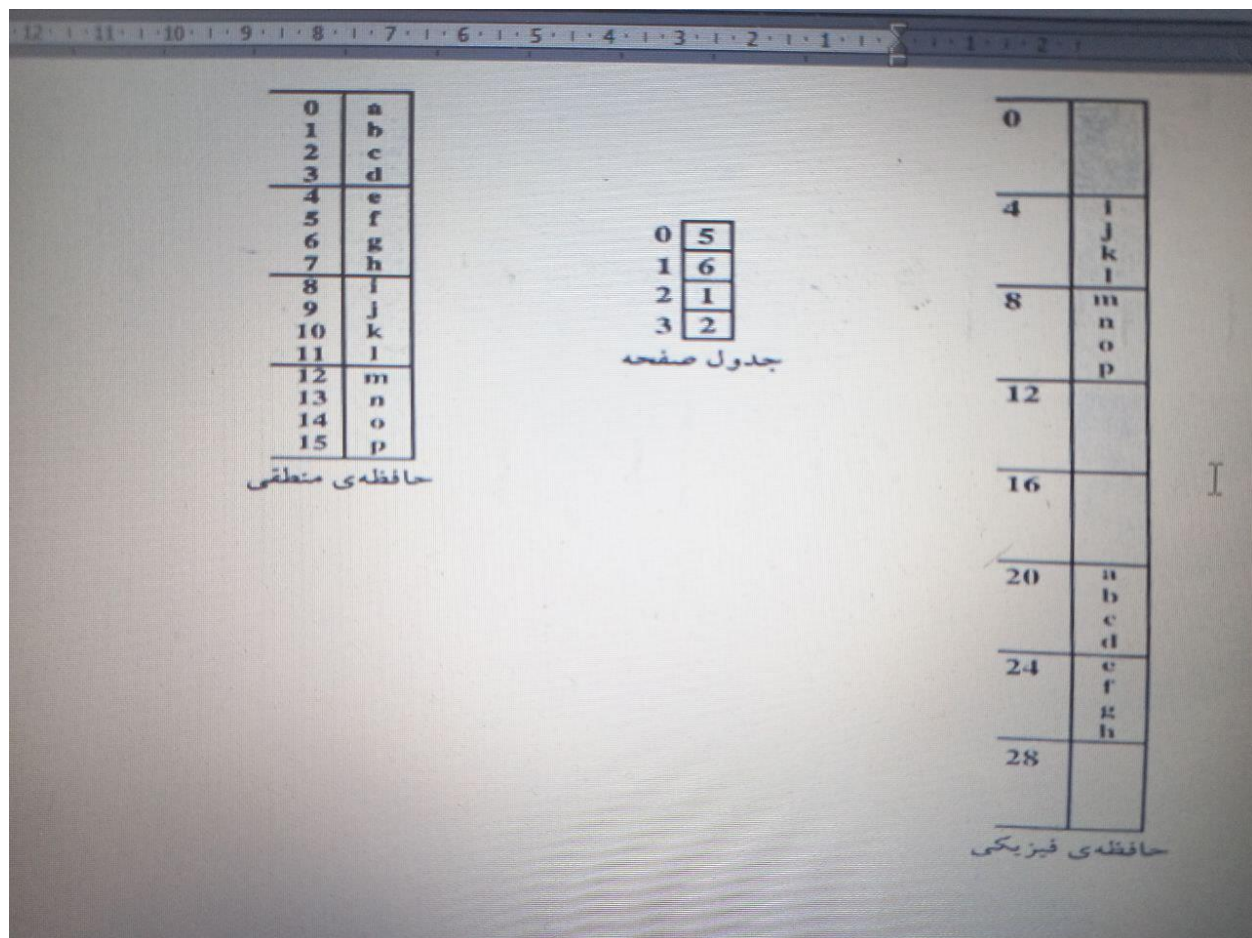


اندازه ی صفحه (مثل اندازه ی قاب) توسط سخت افزار تعریف ی شود. اندازه ی صفحات معمولاً توانی از 2 است. که بر اساس معماری کامپیوتر، بین 512 بایت و 1 گیگابایت به ازای هر صفحه است. چون اندازه صفحه، توان 2 است. ترجمه ی آدرس منطقی به شماره صفحه و آفست صفحه، آسان است. اگر اندازه ی فضای آدرس منطقی 2^m باشد و اندازه ی صفحه برابر با 2^n واحد آدرس (بایت یا کلمه) باشد، آن گاه $m-n$

بیت با ارزش آدرس منطقی، شماره ی صفحه و n بیت کم ارزش، آفست صفحه را تعیین می کند. 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 حافظه ی شکل بالا را در نظر بگیرید. در این جا $n = 2$ و $m = 4$ است. با استفاده از صفحه ی 4 بایتی و حافظه فیزیکی 32 بایتی (8 صفحه)، نشان می دهیم

که حافظه از دیدگاه کاربر چگونه به حافظه فیزیکی نگاشت می شود. آدرس منطقی صفر، صفحه صفر و آفست صفر است. با مراجعه به جدول صفحه (از طریق اندیس شماره صفحه) متوجه می شویم که صفحه ی صفر در قاب (Frame) 5 قرار دارد.

لذا آدرس منطقی صفر به آدرس فیزیکی

$20 [= (5 \times 4) + 0]$ نگاشت می شود. آدرس منطقی 3 (صفحه صفر، آفست 3) به آدرس فیزیکی $23 [= (5 \times 4) + 3]$ نگاشت می شود و آدرس منطقی 4 نیز صفحه 1 و آفست صفر است. با مراجعه به جدول مشخص می شود که صفحه ی یک و به قاب 6 نگاشت می شود. لذا آدرس منطقی 3 به آدرس فیزیکی $24 [= (6 \times 4) + 0]$ نگاشت می شود. آدرس منطقی 13 به آدرس فیزیکی 9 نگاشت می شود.

$$\text{آدرس فیزیکی} = \text{offset}(d) + (\text{طول صفحه} \times \text{شماره قاب}(F))$$