دانشگاه آزاد اسلامی واحد بافت

خلاصه کتاب طراحی ، معماری

و

سازمان کامپیوتر

واسط سخت افزار / نرم افزار

تهیه کننده :

کاظم فریدی

دانشجوی کارشناسی ارشد مهندسی معماری سیستمهای کامپیوتری

با سلام

دوستان عزیز موارد خلاصه شده فوق چیزهایی است که استاد گرامی در کلاسهایی که برگزار گردید توسط بنده برداشت شده است و این خلاصه نویسی فقط برای راهنمایی استفاده شما در خواندن مطالب کتاب می باشد اما در صورت داشتن وقت سعی نمایید کلیه مطالب فصول کتاب را برای فهم بیشتر خوانده و فرابگیرید . با تشکر فریدی

فصول یک ، دو وسه توضیحاتی برای فراگیری و یادآوری دروس گذشته دوره لیسانس میباشد که صفحات قابل خواندن و درک کردن مشخص گردیده که میتوانید برای فراگیری و یادآوری دوباره حتما مطالعه نمایید .

خلاصه فصل اول

از صفحه ۱۹: (رده های گوناگون کاربردهای کامپیوی و شاخص های آنها) تا صفحه ۲۱

از صفحه ۲۴: (۱-۲ فراسوی برنامه های شما از زبان سطح بالا به زبان سخت افزار)تا پایان صفحه ۲۷

صفحه ۲۸: دورنما

صفحه ۲۹ شکل ۱-۴

صفحه ۳۱ پاراگراف دوم (هرتصویر متشکل از ماتریسی ....)

صفحه ۳۳ پاراگراف دوم (حافظه مکانی برای ذخیره تا پایان پاراگراف)

صفحه ۳۴ از شکل ۱-۹ تا پایان صفحه

صفحه ۳۵ دورنما

صفحه ۳۷ پاراگراف دوم تا آخر پاراگراف (با این اوصاف ....)

خلاصه فصل دوم

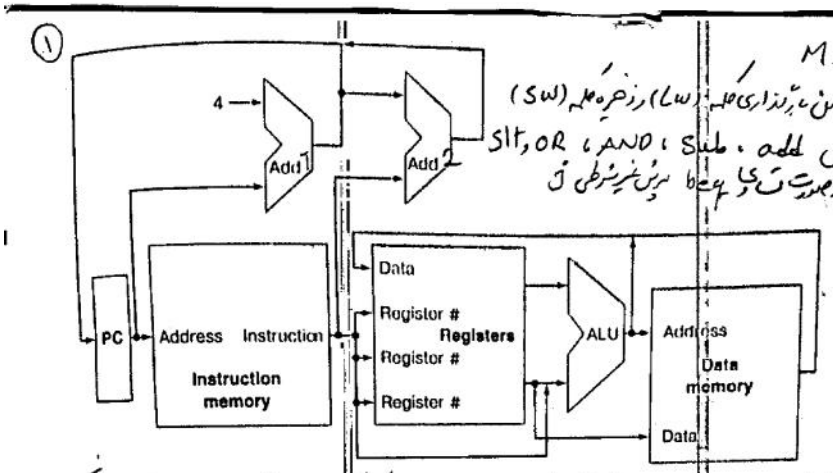
از صفحه (۱۰۹ فیلدهای دستورالعمل MIPS) تا پایان صفحه ۱۱۲ دورنما

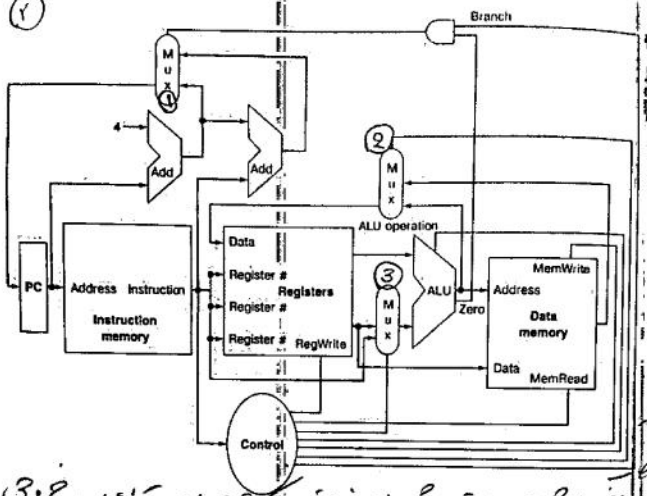
صفحه ۱۱۶ دستورالعمل های تصمیم گیری تا حلقه ها صفحه ۱۱۸

صفحه ۱۴۳ جمع بندی سبک های آدرس دهی در MIPS تا بالای صفحه ۱۴۴

فصل سوم هیچ موردی تدریس نگردید

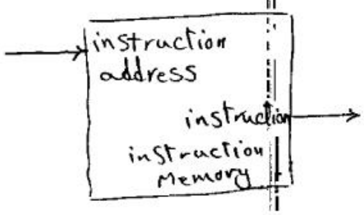
فصل چهارم و پنجم و هفتم نیز بصورت خلاصه شده با توضیحات اشکال در زیر برای شما اسکن شده است که امیدوارم مورد استفاده شما دوستان گرامی قرار بگیرد و بتوانید نمره خوبی را در امتحان پایان ترم بدست آورید .


4
Add
Add
PC
Address
Instruction
Instruction memory
Data
Register #
Registers
Register #
Register #
ALU
Address
Data memory
Data



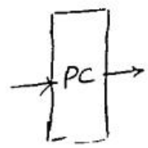
③

a) واحد حافظه برای ذخیره‌سازی دستورالعمل‌ها در برنامه نیاز داریم.
سپس کار هر دستورالعمل آدرس را تدارک می‌بینیم



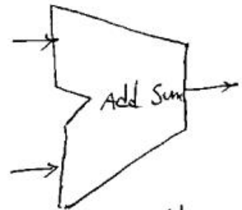
a) instruction memory

b) یک شمارنده برنامه Program Counter موسوم به PC
یک ثبات Regester است که آدرس دستورالعمل جاری
را نگه می‌دارد.



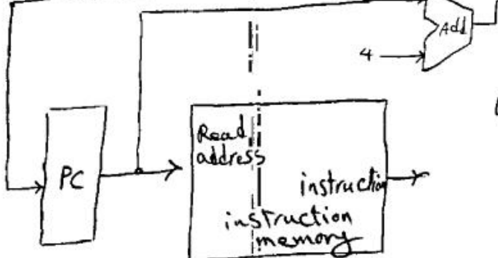
b) program counter

c) جمع‌کننده. تا بتوانیم برای مشخص کردن آدرس دستورالعمل بعدی به مقدار موجود در PC بیفزاییم
این جمع‌کننده مداری تماماً ترکیبی است.



c: Adder

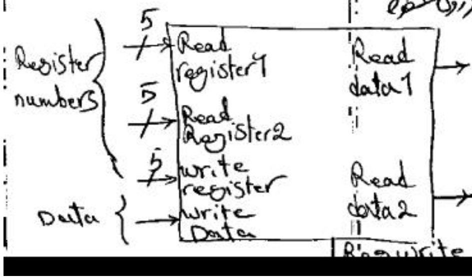
توضیح: برای اجرای یک دستورالعمل در نخستین گام باید آن دستورالعمل را از حافظه واکشی fetch کنیم همچنین برای
اجرای دستورالعمل بعدی باید مقدار کنونی PC را به اندازه ٤ واحد افزایش داد تا به دستورالعمل بعدی اشاره کند.

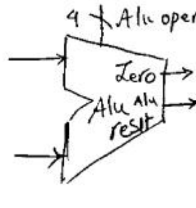


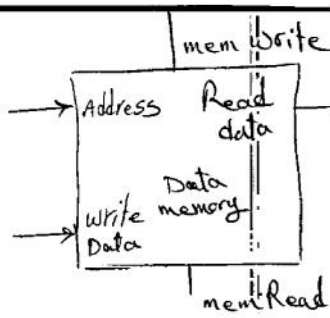
دستورالعمل‌ها نوع R: تمامی دستورالعمل‌های این رده دو ثبات
را خوانده و یکی از عملیات Alu (محاسباتی یا منطقی) را بر روی آنها
انجام می‌دهد و در نهایت نتیجه را در ثبات دیگر می‌نویسند. دستورات
های این رده and - add - sub - OR و slt است.

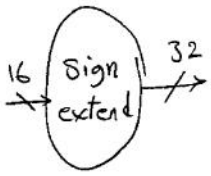
add st1, st2, st3 ← برای اجرا باید ثبات st2 و st3 را خوانده و نتیجه باید در ثبات st1 نوشت.
در مؤلفه مورد نیاز برای پیاده‌سازی عملیات محاسباتی / منطقی از نوع ثبات R-type عبارتند از یک بانک ثبات و یک Alu
بانک ثبات مشتمل بر تمام ثبات‌ها و دو درگاه (پورت) کار خواندن و یک درگاه برای نوشتن است.

بانک ثبات: همیشه محتوای ثبات‌هایی را در خروجی‌های خود ظاهر می‌سازد که شماره آن ثبات‌ها در روی خطوط ورودی Read1/Read2
مشخص شده است. و هیچ سیگنال کنترلی دیگری برای خواندن نیاز نیست. برای نوشتن درون یکی از ثبات‌ها باید
شماره ثبات را بر روی خطوط write مشخص شده باشد و افزون بر فعال کردن
سیگنال کنترلی Regwrite داده‌ها نیز باید در لبه سیگنال ساعت بر روی خطوط
write Data پایدار شده باشد.



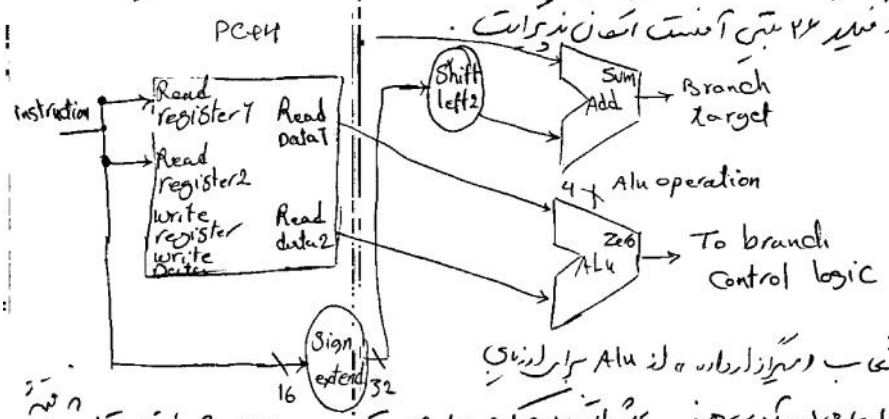


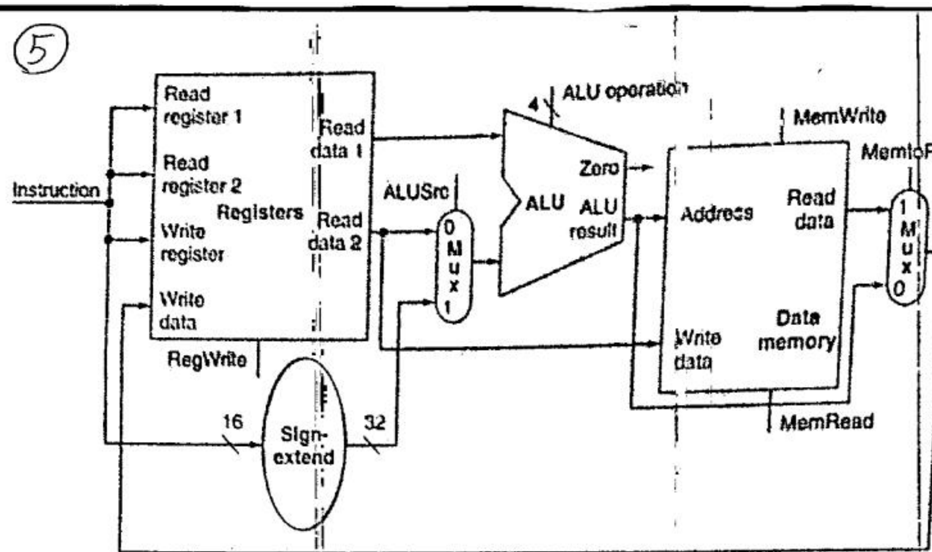




a) واحد حافظه داده

b) واحد گسترش علامت





مثال:

مسیرگذر داده برای دستورالعمل‌های حسابی - منطقی که همگی از نوع R و مبتنی بر ثبات هستند و دستورالعمل‌های مراجعه به حافظه نسبتاً شبیه است. چگونه می‌توان یک مسیرگذر داده برای بخش عملیاتی دستورالعمل‌های مراجعه به حافظه و همچنین دستورالعمل‌های حسابی - منطق ثبات به قسمی که بطور اشتراکی از یک بایگانی ثبات Register File و یک Alu واحد برای اجرای هر دو نوع این دستورالعمل‌ها بهره گرفت هر کجا لازم باشد از مالتی پلکسر بهره گیرید.

جواب: در اینجا دیگر مسیرگذر داده که بررسی آن تنها یک بایگانی ثبات و یک ALU وجود داشته باشد در ورودی Alu از دو دسته ورودی که منبع متفاوتی دارند پشتیبانی کنیم. به همین ترتیب برای خطوط ورودی به بایگانی ثبات (که به منظور نوشتن در یک ثبات تعبیه شده‌اند) باید از دو دسته ورودی که از مبادی متفاوت می‌آیند پشتیبانی شود. بدین سان باید مالتی پلکسر در ورودی Alu و یک مالتی پلکسر دیگر در ورودی بایگانی ثبات قرار داده شود.

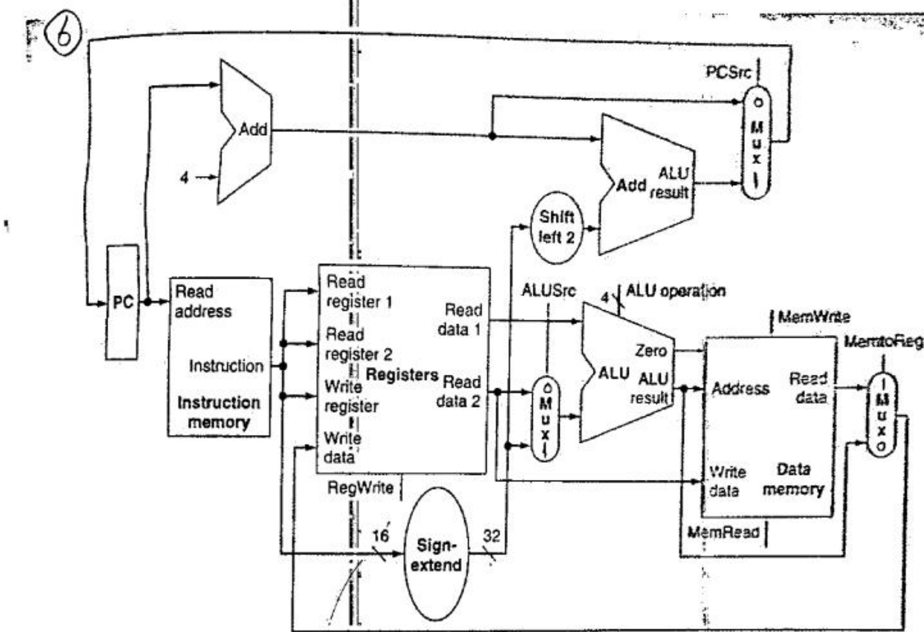
طراحی واحد کنترل مرکزی:

Address 15:0

0	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0

* فیلد موسوم به opcode که در آن کد عمل درج می‌گردد همیشه در موقعیت بیت‌های 26 تا 31 قرار دارند به این فیلد با نام op[5:0] اشاره خواهیم کرد.
* دو ثباتی که بایستی خوانده شوند همیشه در فیلدهای موسوم به rs و rt مشخص می‌گردند که به ترتیب در موقعیت بیت‌های 21 و 25 و 16 تا 20 قرار دارند. این موضوع برای تمام دستورالعمل‌های از نوع R پرش شرطی beq و دستورالعمل ذخیره در حافظه صدق می‌کند.
* شانزده بیت آفستی که در بطن دستورالعمل‌های پرش شرطی و دستورالعمل‌های ذخیره و بارگذاری وجود دارند در موقعیت بیت‌های 0 تا 15 قرار می‌گیرند.
* ثبات پایه برای دستورالعمل‌های ذخیره و بارگذاری همیشه در موقعیت بیت‌های 21 تا 25 موسوم به فیلد rs قرار دارند.
* ثبات مقصد یکی از دو موقعیت مشخص خواهد شد. در دستورالعمل بارگذاری Load در موقعیت بیت‌های 16 تا 20 فیلد موسوم به rt قرار دارد در حالی که در دستورالعمل‌های نوع R یعنی دستورالعمل‌های از نوع ثبات همین فیلد در موقعیت بیت‌های 11 تا 15 در فیلد rd واقع است. به همین دلیل برای مشخص کردن شماره ثبات مقصد که باید درون آن نوشته شود به مالتی پلکسر نیاز داریم.

6

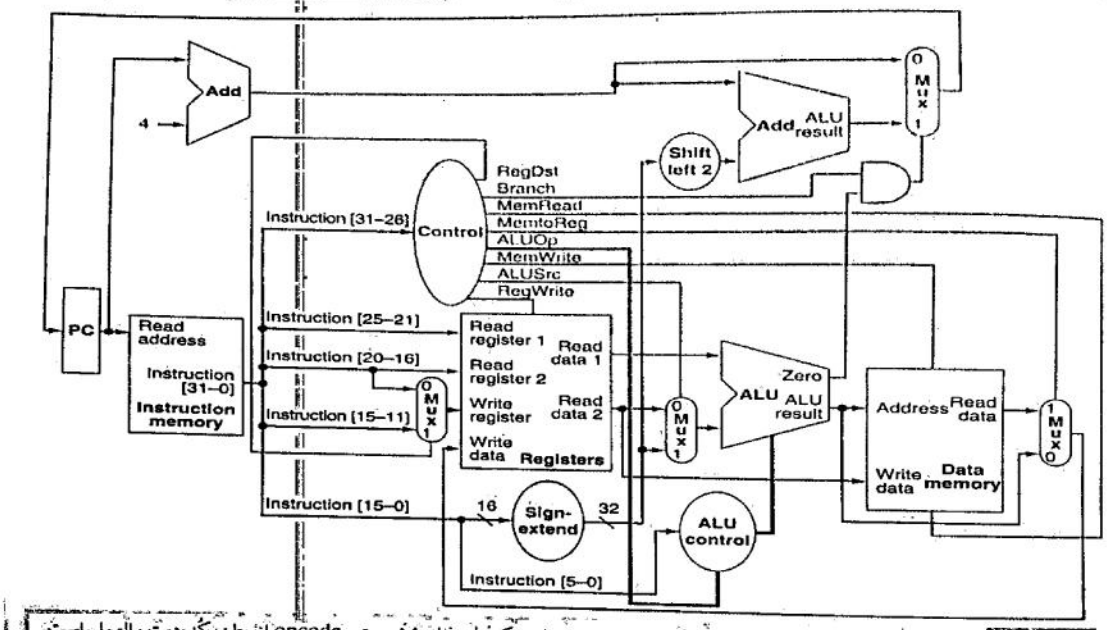


PCSrc : این خط کنترل، در دو شرط فعال می‌شود. اول آنکه دستورالعمل در حال اجرا «دستور پرش شرطی یعنی» beq باشد (تصمیمی که واحد کنترل می‌تواند براساس فیلد opcode اتخاذ کند) و دوم آنکه خروجی Zero از Alu (که برای مقایسه تساوی دو عملوند از آن استفاده می‌گردد) مؤکداً فعال باشد. بنابراین برای تولید سیگنال PcSrc نیاز خواهیم داشت تا سیگنال خروجی از واحد کنترل موسوم به سیگنال Branch و سیگنال Zero از خروجی Alu را با یکدیگر And کنیم.

(مربوط به صفحه ۷)

مثال ۲ : دستورالعمل (st2) offset , st1 را شرح دهید.

۱- دستورالعمل مورد نظر واکشی Fetch شده و شمارنده برنامه PC افزایش می‌یابد.
۲- مقدار یک ثبات (در این مثال st2) از درون بانک ثبات خوانده می‌شود.
۳- Alu حاصل جمع مقداری که از بانک ثبات خوانده شده و ثبات کردن ۱۶ بیت کم ارزش دستورالعمل (یعنی مقدار offset) را پس از گسترش علامت Sign-extension حاصل می‌کند.
۴- حاصل جمع بدست آمده از Alu بعنوان آدرس داده در حافظه مورد استفاده قرار می‌گیرد.
۵- داده‌ای که از حافظه بدست می‌آید درون یکی از ثبات‌های بانک ثبات نوشته خواهد شد. ثبات مقصد در بیت 16 تا 20 از دستورالعمل مشخص شده است. (در این مثال st1)



شکل ۴-۱۷ «مسیر گذار داده» به همراه واحد کنترل. ورودی واحد کنترل، فیلد شش بیتی opcode از بطن کد دستورالعمل است. خروجی‌های واحد کنترل عبارتند از ۳ سیگنال کنترلی که مالتی‌پلکسرها را کنترل می‌کنند (سیگنال‌های RegDst ، ALUSrc و MemtoReg)، سه تای دیگر نیز برای کنترل و راهبری عملیات خواندن و نوشتن از «هایگانی ثبات» (register file) و «حافظه» تولید می‌شود (سیگنال‌های RegWrite ، MemRead و MemWrite). سیگنال تک‌بیتی Branch مشخص می‌کند که آیا یک دستورالعمل انشعاب در حال اجراست؟ دو بیت سیگنال کنترلی ALUOp برای کنترل ALU مورد استفاده قرار خواهد گرفت. یک گیت AND برای کنترل مقدار بعدی PC سیگنال تولید می‌کند: اگر دستورالعمل از نوع انشعاب باشد و مقادیر عملوندها مساوی بودند مقدار بعدی PC از طریق مالتی‌پلکسر با «آدرس هدف پرش» جایگزین می‌شود.

مثال: عملکرد مسیر داده datapath برای دستورالعمل نوع R مانند add $t1, $t2, $t3.

همهٔ عملیات در یک سیکل ساعت اتفاق می‌افتد ولی می‌توان در اجرای دستورالعمل چهار مرحله را تشخیص کرد.

۱- دستورالعمل واکشی شده و مقدار شمارندهٔ برنامه PC افزایش خواهد یافت

۲- دو ثبات $t2 و $t3 از درون مانگانی ثبات register File خوانده می‌شود. افزون بر این واحد کنترل رهنمود لازم مرحلهٔ تنظیم خطوط کنترل را هم صادر خواهد کرد.

۳- Alu بر روی داده‌های خوانده شده از هایگانی ثبات و بر اساس کد عملیات (یعنی فیلد شش بیتی Funct در بیت‌های صفر تا ۵ از کد دستورالعمل)، عمل مطلوب را انجام داده و نتیجه را در خروجی تولید می‌کند.

۴- بر اساس بیت‌های ۱۱ تا ۱۵ از دستورالعمل که نگاره ثبات مقصد را تعیین می‌کنند. در این مثال $t1 نتیجهٔ خروجی از Alu در هایگانی ثبات نوشته می‌شود.

مثال ۲: beq $t1, $t2, offset

۱- دستورالعمل واکشی Fetch شده و شمارندهٔ برنامه PC افزایش می‌یابد.

۲- دو ثبات مبدأ، در این مثال $t1 و $t2 از درون هایگانی ثبات خوانده می‌شوند

۳- Alu مقادیر خوانده شده از ثبات‌ها را تفریق می‌کند. همزمان مقدار PC+4 با مقدار ثبت شده کم اندیس از دستورالعمل یعنی مقدار offset پس از گسترش علامت و شیفت چپ یا ارزیابی دو بیت جمع خواهد شد. حاصل جمع آدرس هدف پرش خواهد بود

۴- از نتیجه حاصل بردی سیگنال Zero از Alu برای تصمیم‌گیری در خصوص آنکه حاصل کرامین جمع کشته و (بدون PC زخیره شود

(۸)

اجرای دستورالعمل‌ها به شیوه خط لوله‌ای «PiPeline»

۱- واکشی دستورالعمل از حافظه (Fetch)

۲- خواندن ثبات‌ها در حالی که دستورالعمل در حال کدگشایی و تولید سیگنال‌های کنترلی است. قالب دستورالعمل‌های MIPS

این اجازه را می‌دهد که خواندن از ثبات‌ها هم‌زمان با کدگشایی دستورالعمل اتفاق بیفتد

۳- عمل درخواستی اجرا می‌شود یا آدرس محاسبه می‌شود

۴- به یک عملوند در حافظه داده در data memory مراجعه می‌شود

۵- نتیجه عمل در یک ثبات نوشته می‌شود.

فرمول ضریب افزایش سرعت در پیاده‌سازی مبتنی بر خط لوله:

$$\text{خط لوله (زمان بین اجرای دستورالعمل‌ها)} = \frac{\text{غیرخط لوله (زمان بین اجرای دستورالعمل‌ها)}}{\text{تعداد مراحل لوله}}$$

نکته: شگرد خط لوله کارایی Performance و سرعت اجرا را با افزایش ظرفیت گذردهی throughput دستورالعمل‌ها بهبود می‌بخشد

موانع و خطرات خط لوله: «گاهی اوقات در سیر خط لوله شرایطی پیش می‌آید که دستورالعمل بعدی را نمی‌توان در سیکل ساعت آتی اجرا کرد

به چنین رخدادهایی اصطلاحاً خطرات hazards یا به عبارتی تفصیلی «خطرات و موانع سد راه در خط لوله» گفته می‌شود.

سه نوع داریم: ۱- خطرات و موانع ساختاری Structural hazard ← سخت‌افزار قادر به پشتیبانی از ترکیب خاصی از دستورالعمل‌هایی که می‌خواهیم آنها را در یک سیکل واحد اجرا کنیم نیست.

۲- خطرات و موانع ناشی از داده (Data hazard) زمانی رخ می‌دهد که خط لوله باید در انتظار تکمیل یکی از مراحل قبلی از حرکت بایستد. در خط لوله کامپیوترها خطرات و موانع ناشی از داده زمانی رخ می‌دهد که یک دستورالعمل به دستورالعمل پیشین خود که هنوز در خط لوله قرار دارد وابسته است.

۳- خطرات و موانع کنترلی (Control hazard) از آنجا ناشی می‌شود که باید براساس نتیجه به دست آمده از یک دستورالعمل در حال اجرای دیگری تصمیم‌گیری شود.

مسیرگذار داده و واحد کنترل در خط لوله:

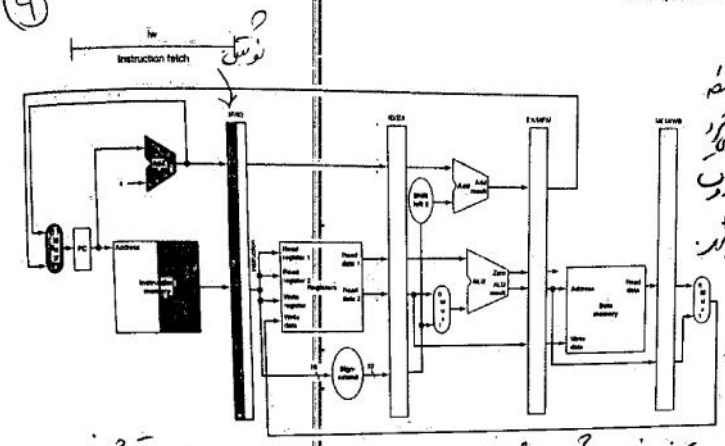
IF : مرحله واکشی دستورالعمل　instruction Fetch

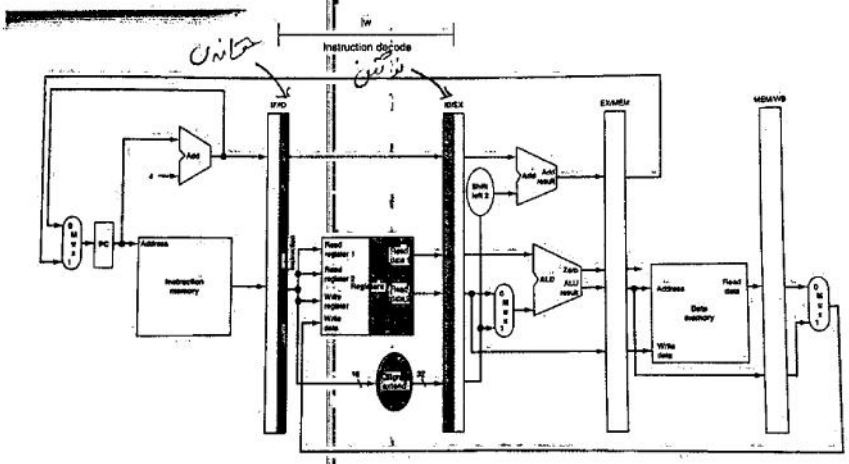
ID : مرحله کدگشایی و خواندن از بانک ثبات　instruction Decode

EX : اجرا یا محاسبه آدرس　Execution

MEM : مراجعه به حافظه　Memory Access

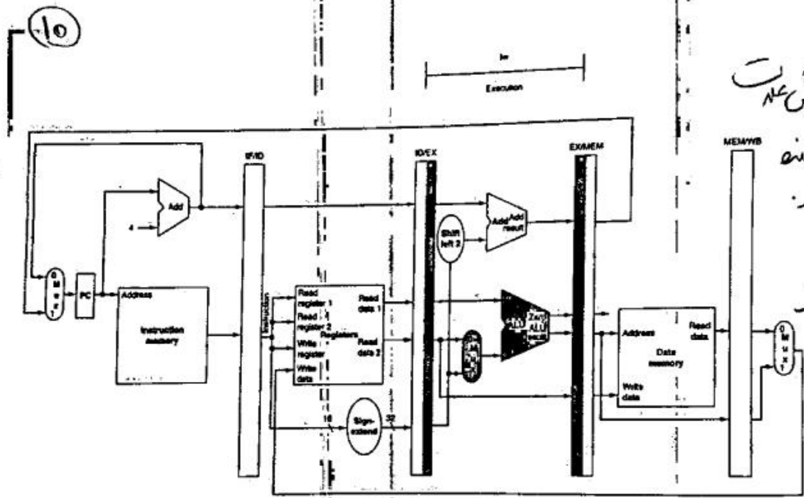
WB : بازنویسی　Write Back


Instruction fetch
IF/ID
ID/EX
EX/MEM
MEM/WB
PC
Address
Instruction memory
Data memory
ALU
Add

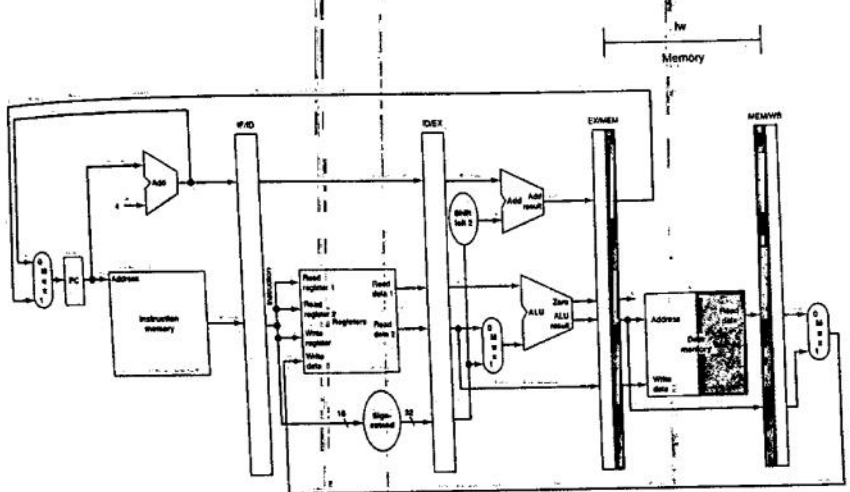

Instruction decode
IF/ID
ID/EX
EX/MEM
MEM/WB
PC
Address
Instruction memory
Data memory
ALU
Add

(۱۵)

۳ اجرا یا محاسبه آدرس :
محتوای ثبات اول و انضمام مقدار گسترش علامت یافته فیلدی را بسط از ثبات ID/EX خوانده و برای عمل جمع به سوی ALU روانه می‌گردند. در نهایت مقدار حاصل جمع در ثبات EX/MEM از خط لوله ذخیره خواهد شد.

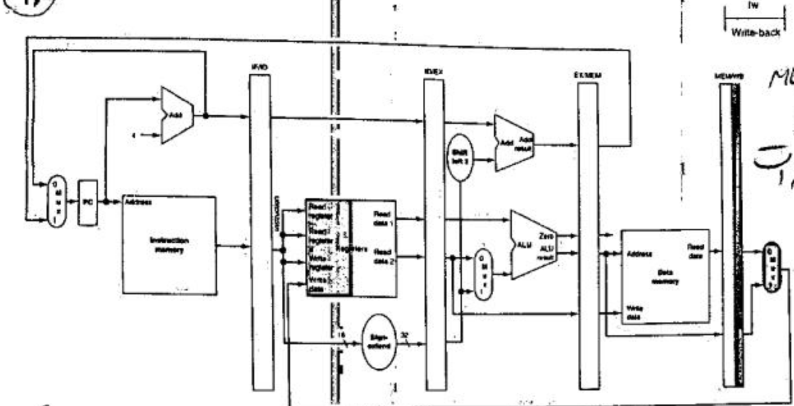


نکته : برای گذر دادن هر چیزی از مرحله ابتدایی به مراحل آتی یک لوله، اطلاعات باید درون ثبات‌های خط لوله قرار داده شود و گرنه به محض آنکه دستورالعمل بعدی وارد مرحله‌ای از خط لوله شود اطلاعات فعلی آن مرحله از دست خواهد رفت.



۴ - مراجعه به حافظه : با استفاده از آدرس موجود در ثبات EX/MEM از خط لوله، محتوای کلمه‌ای از حافظه خوانده می‌گردد. و کلمه خوانده شده در ثبات MEM/WB از خط لوله ذخیره خواهد شد.

11

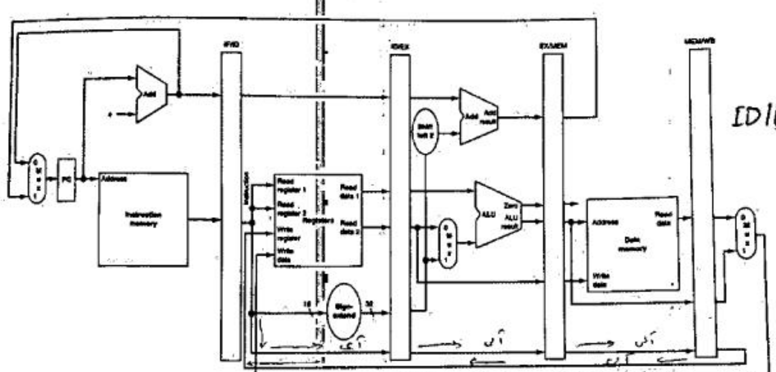


۵. بازنویسی :
داده ای که در مرحله قبل در ثبات MEM/WB از خط لوله ذخیره شده، خوانده می‌شود تا آن را درون بانک ثبات که در وسط شکل قرار گرفته است بنویسید

سوال: به نظر شما در آخرین مرحله از دستورالعمل بارگذاری شده، محتوای کدام ثبات باید تغییر کند؟ به بیان روشن‌تر، شماره ثباتی که باید درون آن نوشته شود از بطن کدام دستورالعمل تعیین می‌شود؟

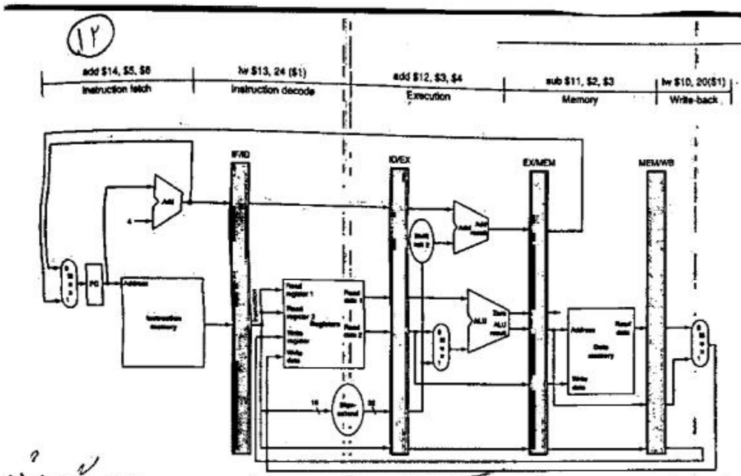
جواب: دستورالعملی که در لحظه نوشتن داده درون ثبات IF/ID قرار دارد حاوی شماره ثبات مقصد هست ولیکن مقدار آن متعلق به دستورالعملی است که خیلی بعدتر از دستورالعمل بارگذاری بر روی خط لوله قرار گرفته است. بنابراین ما بایستی شماره ثبات مقصد که در بطن دستورالعمل بارگذاری مشخص شده تا آخرین مرحله خط لوله حفظ شود.

راه حل: (چرا باید شماره ثبات مقصد را حفظ کنیم) آن است که برای استفاده اشتراکی از یک مسیر داده خط لوله‌ای، در مرحله واکشی و خواندن دستورالعمل (مرحله IF) دستورالعملی خوانده می‌شود و سپس هر یک از ثبات‌های مختص خط لوله موظف اند اطلاعاتی را که در مراحل بعد مورد نیاز است ذخیره کرده و در اختیار بگذارند.



نسخه تصحیح شده «مسیر داده». رای شماره ثبات مقصد که در آن شماره ثبات مقصد ابتدا به ثبات ID/EX منتقل می‌شود، بعد به ثبات ID/EX، سپس به ثبات EX/MEM منتقل و در نهایت درون ثبات MEM/WB ذخیره خواهد شد. شماره ثبات مقصد در مرحله بازنویسی (مرحله WB) برای تعیین ثباتی که باید درون آن نوشته شود مورد استفاده قرار خواهد گرفت.

(۱۲)



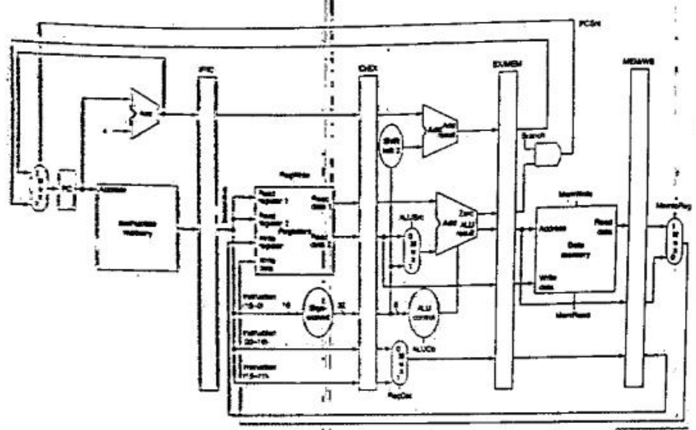
نمودار تک سیکل ساعتی
Sigle - clock - cycle diagram

Lw $10, 20($1)
Sub $11, $2, $3
add $12, $3, $4
Lw $13, 24($1)
add $14, $5, $6

در این نمودار زمان از سمت چپ به راست
صفحه جلو رفته و دستورالعمل‌ها به ترتیب از بالا به پایین صفحه در مسیر اجرا پیش روی می‌کنند | هر مرحله موسوم به Stage نامگذاری میشود.

۴ـ خطوط کنترلی مرتبط به دسترسی به حافظه: خطوط کنترلی که در این مرحله به مقدار دهی و تنظیم شوند عبارتند از Branch، MemRead و MemWrite این سیگنالها به ترتیب توسط دستورالعمل در پرش شرط برابر یعنی beq، بارگذاری و ذخیره مقدار دهی و تنظیم خواهند شد.

۵ـ خطوط کنترلی مرتبط با بازنویسی. دو سیگنال کنترلی برای این مرحله از خط لوله عبارتند از MemtoReg و RegWrite. سیگنال MemtoReg از بین دو مقدار خروجی Alu و مقدار مستخرج از حافظه یکی از آنها را برای نوشته شدن در بانک ثبات انتخاب می‌کند. سیگنال RegWrite مقدار انتخاب شده را درون ثبات می‌نویسد.



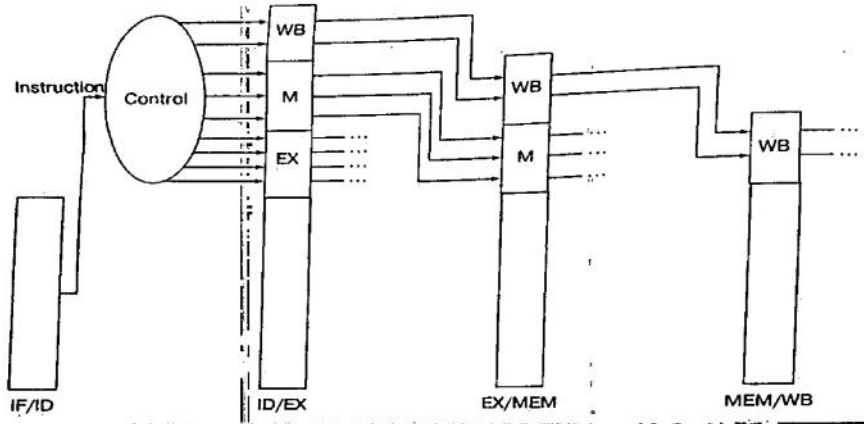
واحد کنترل خط لوله:
مسیر داده خط لوله ای Pipelined datapath
به همراه سیگنال کنترلی

۱ـ خطوط کنترلی مرتبط با واکشی دستورالعمل:
از آنجایی که خطوط کنترل لازم برای خواندن از حافظه دستورالعمل و برای نوشتن در PC همیشه متواتراً فعال است بنابراین چیز خاصی در این مرحله خط لوله نداریم که نیاز به کنترل داشته باشد.

۲ـ خطوط کنترل مرتبط به کدگشایی دستورالعمل و خواندن از بانک ثبات: دقیقاً همانند مرحله قبل در این مرحله نیز همان شرایط حاکم است و خط کنترلی خاصی که گزینه‌های انتخابی وجود ندارد که نیاز به مقدار دهی داشته باشد.

۳ـ خطوط کنترلی مرتبط با اجرا یا محاسبه آدرس: سیگنالهایی که در این مرحله می‌بایست مقدار دهی و تنظیم شوند عبارتند از: RegDst، Aluop، AluSrc این سیگنالها به ترتیب ثبات مقصد Destination Register، نوع عمل مورد انتظار از Alu و مبداء عملوند دیگر Alu را گزینش و کنترل می‌کنند. سیگنال AluSrc با انتخاب بین مقدار خوانده شده دوم از بانک ثبات، یا مقدار ثابت علامت دار که فیلد بلافصل دستور است یکی را به عنوان عملوند دیگر Alu انتخاب می‌کند تا Alu همراه آن عمل کند.

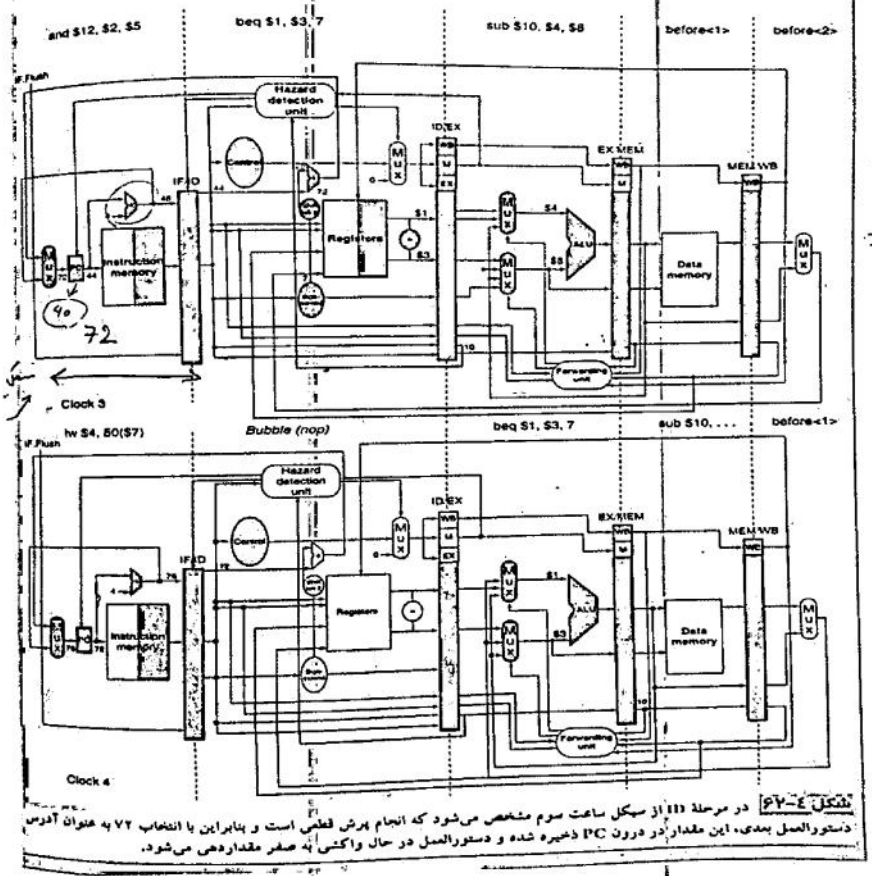
(۱۳)



شکل ٤-٥٩ خطوط کنترلی برای سه مرحلهٔ آخر از خط لوله. دقت کنید که ۳ سیگنال از ۹ سیگنال کنترلی در مرحلهٔ EX مورد استفاده قرار می‌گیرند در حالی که ۵ سیگنال باقیمانده از طریق ثبات EX/MEM به مراحل بعدی تحویل داده می‌شوند. از این ۵ سیگنال ۲ تای آنها در مرحلهٔ MEM بهره‌برداری شده و دوتای باقیمانده یکبار دیگر از طریق ثبات MEM/WB به آخرین مرحله از خط لوله یعنی «مرحلهٔ بازنویسی» (WB Stage) تحویل خواهد شد.

مثال: شکل زیر را با اجرای دستورالعمل‌های مناسب توضیح دهید.

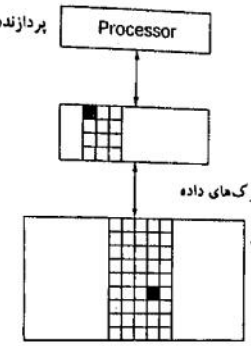
پاسخ: شکل ۴-۶۲ تمام وقایعی را که در صورت انجام پرش اتفاق خواهد افتاد به تصویر کشیده است برخلاف نمودار شکل ۴-۶۱، در این شکل تنها یک «حباب» بر روی خط لوله ظاهر می‌شود.



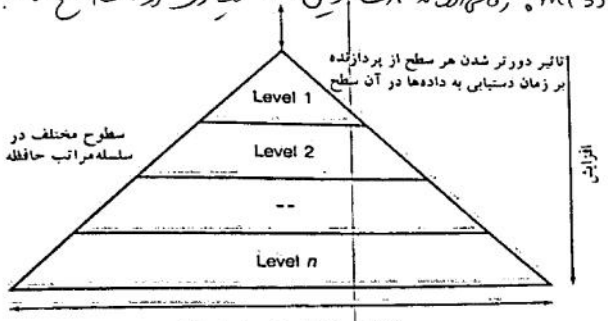
36 sub $10, $4, $8
40 beq $1, $3, 7
44 and $12, $2, $5
48 or $13, $2, $6
52 add $14, $4, $2
56 slt $15, $6, $7
72 lw $4, 50($7)

شکل ٤-٦٢ در مرحلهٔ ID از سیکل ساعت سوم مشخص می‌شود که انجام پرش قطعی است و بنابراین با انتخاب ۷۲ به عنوان آدرس دستورالعمل بعدی، این مقدار در درون PC ذخیره شده و دستورالعمل در حال واکشی به صفر مقداردهی می‌شود.

فصل پنجم



شکل ۵-۲ می‌توان تصور کرد که هر جفت از سطوح سلسله‌مراتب حافظه دارای یک سطح مافوق و یک سطح مادون است و داده‌ها بین سطوح هم‌جوار منتقل می‌شوند. در هر سطح، کوچکترین واحد اطلاعات که یا در آن سطح حاضر و یا غایب است اصطلاحاً «بلوک» یا «سطر» نامیده می‌شود. عموماً وقتی چیزی را بین دو سطح مجاور منتقل و کپی می‌کنیم یک بلوک کامل انتقال می‌یابد.



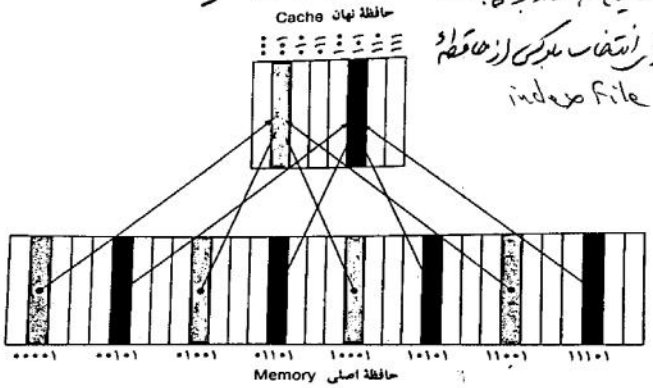
شکل ۵-۳ این نمودار ساختار یک حافظهٔ سلسله‌مراتبی را به تصویر کشیده است: هر چه از پردازنده دورتر شویم ظرفیت آن افزایش خواهد یافت؛ این ساختار با فرض بهره‌گیری از مکانیزم‌های عملیاتی مناسب، این امکان را فراهم می‌آورد تا زمان دستیابی پردازنده به حافظه در حدود زمان دستیابی به داده‌ها در سطح نخست باشد ولی کماکان ظرفیتی معادل با آخرین سطح (سطح nام) را در اختیار بگذارد. حفظ چنین پنداشتی موضوع مباحث این فصل است. اگرچه دیسک‌های محلی بطور معمول در پایین‌ترین سطح از سلسله‌مراتب حافظه قرار می‌گیرند ولیکن در برخی از سیستم‌ها در پایین‌ترین سطح، از نوارهای مغناطیسی یا سرویس‌دهنده‌های فایل موجود بر روی شبکهٔ محلی استفاده می‌کنند و آنها را نیز به عنوان سطح بعدی از سلسله‌مراتب حافظه به حساب می‌آورند.

(۲)

در انگاری « نگاشت مستقیم » direct mapped ما با در دست داشتن یک آدرس دقیقاً می‌دانیم که باید در کدام بلوک از حافظهٔ نهان به دنبال آن بگردیم زیرا بیت‌های کم (بیش) آدرس، دقیقاً شماره یک بلوک یکتا از حافظه نهان را تعیین می‌کنند.

فیلد برچسب tag field : که به منظور مقایسه با مقدار برچسب از حافظهٔ نهان به کار می‌آید.

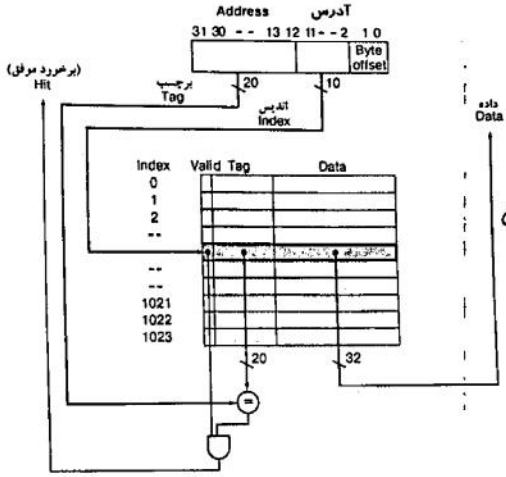
اندیس حافظهٔ نهان : که برای انتخاب بلوکی از حافظهٔ نهان کاربرد دارد. index file



شکل ۵-۵ یک حافظهٔ نهان مبتنی بر «نگاشت مستقیم» با ظرفیت هشت دروازه. این شکل نشان می‌دهد که چگونه کلمات واقع در آدرس ۰ تا ۳۱ از حافظهٔ اصلی در موقعیت‌های مشابهی از حافظهٔ نهان نگاشته می‌شوند. از آنجایی که ظرفیت حافظهٔ نهان فقط هشت کلمه است بنابراین آدرس مفروض X مستقیماً به کلمه‌ای با آدرس X *mod* ۸ از حافظهٔ نهان نگاشته خواهد شد. بنابراین سه بیت کم‌ارزش از آدرس هر کلمه در حافظهٔ اصلی، محل قرارگیری آن کلمه در حافظهٔ نهان را تعیین می‌کند. با این تفاسیر تمام کلمات واقع در آدرس‌های 00001_{two} و 01001_{two} و 10001_{two} و 11001_{two} همگی به موقعیت مشابهی از حافظهٔ نهان نگاشته می‌شوند.

* آدرس‌های ۳۲ بیتی برای هر بایت حافظه
* حافظه نهان به گونه نگاشت مستقیم
* حافظه نهان متشکل از 2^n بلوک و طبیعتاً برای اندیس هر بلوک n بیت نیاز است.
* طول هر بلوک 2^m کلمه چهار بایتی (معادل 2^{m+2} بایت) است فلذا m بیت برای آدرس دهی کلمات درون هر بلوک حافظه نهان و دو بیت برای آدرس دهی هر بایت از یک کلمه نیاز داریم.

با این اوصاف نهانهای فیلد برچسب tag معادل $32-(n+m+2)$ بیت است. و تعداد کل بیت‌های مورد نیاز برای حافظه نهان به شرح ذیل است.



شکل ۵-۷ در این حافظهٔ نهان بخش کم‌ارزش از آدرس برای انتخاب یکی از دروازه‌ها به کار گرفته می‌شود؛ دروازه‌ای که حاوی کلمهٔ داده و برچسب است. این حافظهٔ نهان ۱۰۲۴ کلمه (معادل ۱ کیلوبایت) را در خود نگه می‌دارد. در سراسر این فصل آدرس‌ها ۳۲ بیتی فرض شده‌اند. مقدار برچسب از دروازهٔ انتخاب‌شده، توسط بخش کم‌ارزش آدرس باید با بخش پرارزش از آدرس کلمهٔ درخواستی مقایسه شود. از آنجا که این حافظهٔ نهان ۱۰۲۴ کلمه داره و هر بلوک تک‌کلمه‌ای است بنابراین ۱۰ بیت از آدرس به عنوان اندیس بکار می‌رود و بنابراین باید ۲۰=۳۲-۱۰۰۲ بیت از آدرس با برچسب مقایسه شود. اگر این ۲۰ بیت منطبق باشند و بیت اعتبار نیز روشن باشد کلمهٔ موجود در حافظهٔ نهان تحویل پردازنده خواهد شد و در غیر این‌صورت «فقدان» رخ داده است.

از آنجایی که طول هر بلوک 2^m کلمه (معادل 2^{m+5} بیت) است و برای فیلد اعتبار به یک بیت نیاز داریم. تعداد کل بیت‌های لازم برای این حافظه نهان

$$2^n \times (\text{block size} + \text{Tag Size} + \text{Valid Field Size})$$

است و برابر است با

$$2^n \times (2^m \times 32 + (32-(n+m+2))+1) = 2^n \times (2^m \times 32 + 31 - n - m)$$

(۳)

مثال: با فرض آدرس ها ۳۲ بیتی: کل تعداد بیت های مورد نیاز برای پیاده سازی یک حافظه نهان با الگوی نگاشت مستقیم و ظرفیت ۱۶ کیلوبایت که در آن هر بلوک حاوی ۴ کلمه چهار بایتی است، چقدر است؟

$\underset{\text{کیلوبایت}}{16\,KB} = 4\,\underset{\text{کیلوکلمه}}{KW}$ (2^{12} کلمه) بلوک های به اندازه ۴ کلمه (2^2) در مجموع ۱۰۲۴ بلوک وجود دارد.

هر بلوک شامل بر $4\times32=128$ بیت داده یک برچسب ۱۸ بیتی ($32-10-2-2$) و یک بیت که بیت اعتبار

حجم کل حافظه نهان

$$2^{10}\times(4\times32+(32-10-2-2)+1)=2^{10}\times147=147\,kbits=18816\,Byte$$

$$=18{,}375\,kByte$$

نتیجه برای یک حافظه ۱۶ کیلوبایتی ۱۸٫۴ کیلوبایت نیاز است.

مثال: حافظه نهان ۶۴ بلوک و هر بلوک مشتمل بر ۱۶ بایت. بایتی که در آدرس ۱۲۰۰ قرار دارد به کدامین بلوک از حافظه نهان نگاشته خواهد شد؟

رابطه نگاشت: (تعداد بلوک ها در حافظه نهان) Mod (آدرس بلوک)

آدرس هر بلوک:

$$\frac{\text{Byte Address (آدرس بایت)}}{\text{Byte per block (تعداد بایت در هر بلوک)}}$$

$$\left[\frac{\text{Byte address}}{\text{Byte per block}}\right]\times \text{Byte per block}$$

$$\left[\frac{\text{Byte address}}{\text{Byte per block}}\right]\times \text{Byte per block}+(\text{Byte per block}-1)$$

بنابراین با فرض ۱۶ بایت در هر بلوک، آدرس ۱۲۰۰ به بلوک $\frac{1200}{16}=75$ در حافظه اصلی و $(75\ Mod\ 64)=11$ در حافظه نهان نگاشته خواهد شد. در حقیقت این بلوک دربرگیرنده تمام آدرس های بین ۱۲۰۰ تا ۱۲۱۵ است و بالطبع تمام این آدرس ها به همین بلوک نگاشت می شود.

مراحل که در هنگام رخداد فقدان دستورالعمل یا "instruction miss" دنبال کرد:

۱- مقدار واقعی PC (یعنی مقدار واقعی شمارنده برنامه پس از کسر ۴ واحد) را به سوی حافظه اصلی ارسال کنید.

۲- به حافظه اصلی دستور خواندن صادر کنید. و تا تکمیل فرآیند دسترسی و استخراج کلمه منتظر باشید.

۳- داده مستخرج از حافظه را در خانه داده از درایه متناظر با آن در حافظه نهان cach entry بنویسید و بیت های بالاتر آدرس حاصل از خروجی Alu را درون خانه برچسب از درایه مربوطه و در آخر بیت اعتبار valid bit را فعال کنند. ۴- عملیات اجرای دستورالعمل را در نخستین گام آغاز کنید تا دستورالعمل بار دیگر واکشی شود. با این تفاوت که

[illegible]

(۴)

مدیریت عملیات نوشتن:

بازسازی اصلی ذخیره Store داده‌ای را درون حافظه نهان داده data cache نوشته ایم درون آنها چیزی در حافظه اصلی تغییر کند؛ بنابراین پس از عمل نوشتن در حافظه نهان، حافظه اصلی حاوی مقداری متفاوت از آنچه که درون حافظه نهان ذخیره شده است خواهد بود. در همین حالتی اصطلاحاً گفته میشود که حافظه نهان و حافظه اصلی ناهمسان هستند (inconsistent) ساده‌ترین روش برای آنکه حافظه نهان و حافظه اصلی را همسان و همگون نگه داریم آن است که همیشه داده ها را در هر دوی حافظه های نهان و اصلی بنویسیم. چنین هر آلگوریتمی اصطلاحاً نوشتن تمام عیار write-through می گویند.

بافر نوشتن "write buffer" در زمانی که داده در انتظار نوشته شدن در حافظه اصلی است می‌باید آن را در خود نگه می دارد.

یک الگوی جایگزین برای نوشتن تمام عیار "write-through" الگوی بازنویسی "write-back scheme" است در آنجا بازنویسی وقتی در حافظه عمل نوشتن صادر می شود. تقدا جدید فقط در بلوک متناظر از حافظه نهان نوشته خواهد شد و این بلوک تغییر یافته تنها زمانی در حافظه سطح پایین تر بازنویسی میشود که قرار باشد آن بلوک بلوک دیگری جایگزین شود. الگوی بازنویسی میتواند کارایی و سرعت کلی بازنشت را بهبود ببخشد.

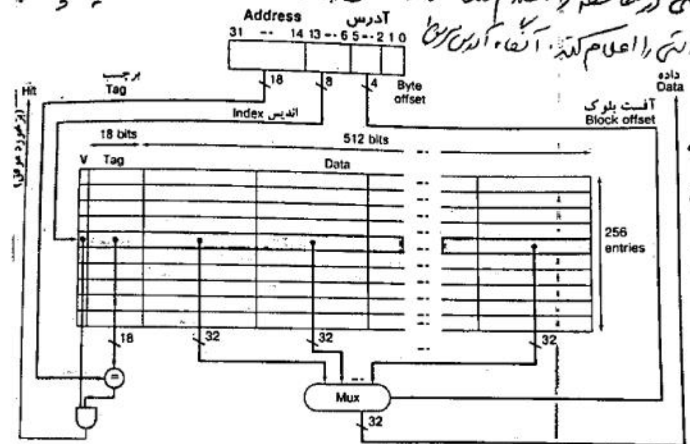
⑤

مراحل تقاضا برای خواندن از هر یک از این حافظه‌های نهان به روال زیر خواهد بود:

۱- آدرس مورد نظر به سوی حافظهٔ نهان متناظر با درخواست (حافظه داده یا حافظه دستورالعمل) ارسال می‌شود. این آدرس یا از طریق PC (شمارنده برنامه) و به منظور واکشی دستورالعمل بعدی و یا از طریق Alu به منظور درخواست یک عملوند، آماده و ارسال می‌شود.

۲- اگر حافظه نهان وجود کلمه درخواستی در حافظه را اعلام کند مقدار آن کلمه پس از بررسی خطوط داده در اختیار پردازنده خواهد بود.

۳- اگر حافظه نهان فقدان کلمه درخواستی را اعلام کند، آنگاه آدرس مربوط را به حافظه اصلی می‌فرستیم. به محض آنکه حافظه اصلی کلمه داده را اعاده کرد آن را در حافظه نهان نوشته و سپس برای بار دوم درخواست مربوطه یک بار دیگر آن را می‌خواند.



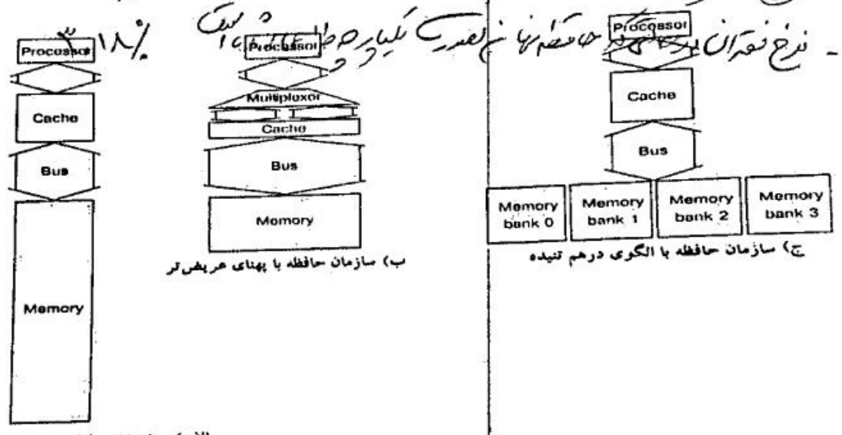
شکل ۵-۹ حافظهٔ نهان ۱۶ کیلوبایتی در پردازندهٔ Intrinsity FastMATH دارای ۲۵۶ بلوک و هر بلوک شامل ۱۶ کلمهٔ ۴ بایتی است. فیلد برچسب ۱۸ بیتی و فیلد اندیس ۸ بیتی است؛ فیلد چهاربیتی (بیت‌های دوم تا پنجم از آدرس) به عنوان اندیس یکی از کلمات از هر بلوک توسط یک مالتی‌پلکسر انتخاب می‌کند.

شرح: نرخ فقدان در حافظه نهان در پردازنده برای دو وضعیت کنونی آن و حافظه نهان یکپارچه نشان داده می‌شود.

۱- حجم حافظه نهان ۳۲ کیلوبایت

۲- نرخ مؤثر فقدان در حالتی که حافظه به صورت دوگانه طراحی شده است ۳٫۲۴٪

۳- نرخ فقدان در حافظه نهان به صورت یکپارچه طراحی شده است ۳٫۱۸٪



الف) سازمان حافظه به پهنای یک کلمه

ب) سازمان حافظه با پهنای عریض‌تر

ج) سازمان حافظه با الگوی درهم تنیده

شکل ۵-۱۱: نخستین روش برای ارتقاء پهنای باند حافظه، افزایش پهنای فیزیکی یا منطقی سیستم حافظه است. در این شکل پهنای باند سیستم سادهٔ حافظه در سمت چپ به دو روش ارتقاء یافته است: استفاده از گذرگاه عریض‌تر در شکل وسط و استفاده از الگوی «درهم‌تنیده» (interleaved) در سمت راست شکل؛ سیستم حافظه در الگوی «درهم‌تنیده» با آرایشی عریض‌تر سازمان‌دهی و پهنای باند آن افزایش یافته است ولیکن گذرگاه ارتباطی بین پردازنده و حافظه عریض‌تر نشده است.

(۶)

حافظه‌های نهان چند سطحی Multilevel cache : با افزودن یک سطح اضافه‌تر به سلسله مراتب حافظه جریمه فقدان کمتر می‌شود.

زمان cpu "cpu Time" : ۱- سیکل‌های ساعتی که cpu صرف اجرای برنامه می‌کند.
۲- سیکل‌های ساعتی که cpu در انتظار سیستم حافظه در تعلیق می‌گذراند.

$$\text{cpu Time} = (\text{Cpu execution clock cycles} + \text{Memory-Stall clock cycles}) \times \text{Clock cycle Time}.$$

سیکل‌های تعلیق ناشی از سیستم حافظه را می‌توان به صورت مجموع سیکل‌های تعلیق در عملیات خواندن و سیکل‌های تعلیق در عملیات نوشتن تعریف کرد:

$$\text{Read-Stall cycles} = \frac{\text{Reads}}{\text{Program}} \times \text{Read miss rate} \times \text{Read miss Penalty}$$

در الگوی نوشتن تمام عیار write through تعلیق از دو عامل منشاء می‌گیرد:

۱- فقدان‌هایی که باید نوشته شود. write miss ← معمولاً در چنین وضعیتی قبل از ادامه عمل نوشتن نیاز داریم تا بلوک متناظر را واکشی کنیم.

۲- تعلیق‌های ناشی از بافر نوشتن write buffer ← در صورت پر بودن بافر و درخواست دیگر برای نوشتن، پردازنده را ناچار به تعلیق می‌کند.

سیکل‌های تعلیقی ناشی از عملیات نوشتن به صورت مجموع زیر قابل محاسبه است:

$$\text{write-Stall cycles} = \left(\frac{\text{writes}}{\text{Program}} \times \text{write miss rate} \times \text{write miss penalty}\right) + \text{write buffer Stalls}$$

نوشتن‌ها و خواندن‌های برنامه را با تعریف یک مقدار واحد برای نرخ فقدان و جریمه فقدان به صورت زیر ترکیب کنیم

$$\text{Memory-Stall clock cycles} = \left(\frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss Penalty}\right)$$

$$\Rightarrow \text{Memory-Stall clock cycles} = \left(\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{instruction}} \times \text{Miss penalty}\right)$$

(۷)

محاسبه کارایی حافظهٔ نهان

CPI = حالتی که برای هر دستورالعمل نیاز داریم.

مثال: نرخ فقدان در حافظهٔ نهان دستورالعمل ۲٪

نرخ فقدان حافظهٔ نهان داده ۴٪

بدون تعلیق حافظه Memory Stalls دارای $CPI = 2$

جریمه معادل ۱۰۰ سیکل ساعت باشد چه میزان سریعتر کار خواهد کرد.

فراوانی دستورالعمل‌ها در ضبط و بارگذاری را ۳۶٪ فرض کنید. (خواندن و نوشتن)

فقدان دستورالعمل برحسب تعداد دستورالعمل $= I$

$$\text{instruction miss cycles} = I \times 2\% \times 100 = 2I$$

جریمه (نبودن) — I معادل خواندن دستورالعمل‌ها است.

فقدان داده

$$\text{Data miss cycles} = I \times 36\% \times 4\% \times 100 = 1.44I$$

تعداد کل سیکل‌های تعلیقی ناشی از حافظه عبارت است از $2I + 1.44I = 3.44I$

یعنی به ازای هر دستورالعمل بیش از سه سیکل تعلیق خواهیم داشت.

با احتساب تعلیق های ناشی از حافظه مقدار $CPI = 2 + 3.44 = 5.44$

نسبت زمان اجرای cpu کارایی‌ها:

$$\frac{\text{cpu time with Stalls}}{\text{cpu time with perfect cache}} = \frac{I \times CPI_{stall} \times \text{clock cycle}}{I \times CPI_{perfect} \times \text{clock cycle}} = \frac{CPI_{stall}}{CPI_{perfect}}$$

$$= \frac{5.44}{2} = 2.72$$

با این اوصاف کارایی پردازنده با یک حافظهٔ نهان ایده‌آل به میزان ۲٫۷۲ بهتر خواهد بود. (قرمز نوشته بودم)

(۸)
برای آنکه تأثیر توأمان زمان دسترسی به داده‌ها چه در صورت برخورد موفق hit به طور درخواستی و چه در صورت فقدان miss ، در محاسبات کارایی سیستم دخالت داده شود طراحان اغلب از معیاری موسوم به AMAT میانگین زمان دسترسی به حافظه average memory access time برای تحلیل و بررسی طراحی حافظه بهره می‌گیرند.

AMAT= Time for a hit + miss rate × Miss penalty

AMAT= جریمه فقدان × نرخ فقدان + زمان یک برخورد موفق

مثال: محاسبه میانگین زمان دسترسی به حافظه:

مقدار AMAT را برای پردازنده‌ای با مشخصات ذیل حساب کنید:

زمان هر سیکل ساعت 1ns ، جریمه فقدان معادل ۲۰ سیکل ساعت ، نرخ فقدان ۰٫۰۵ در هر دستورالعمل

زمان دسترسی به حافظه نهان معادل ۱ سیکل ساعت.

فرض: جریمه فقدان هم برای خواندن و هم برای نوشتن یکسان است.

حل: میانگین زمان دسترسی به حافظه به ازای هر دستورالعمل عبارت است از

AMAT = جریمه فقدان × نرخ فقدان + زمان یک دسترسی موفق

$= 1 + 0.05 \times 20$

$= 2$ سیکل ساعت = ۲ نانوثانیه

نگاشت مستقیم: direct mapped ← از آدرس هر بلوک در حافظه اصلی به موقعیتی واحد در حافظه نهان بالاتر از سلسله مراتب حافظه، تنها یک نگاشت مستقیم و از قبل مشخص وجود دارد.

الگوی تمام انجمنی: Fully associative ← یک بلوک از حافظه اصلی را می‌توان در هر مکان دلخواه از حافظه نهان قرار داد. در این الگو هر بلوک از حافظه اصلی می‌تواند در هر خانه (entry) از حافظه نهان قرار بگیرد.

برای یافتن یک بلوک خاص در حافظه نهان تمام انجمنی بایستی تمام خانه‌های موجود در آن را جستجو کرد زیرا هر بلوک ممکن است در هر مکانی از حافظه نهان ذخیره شده باشد.

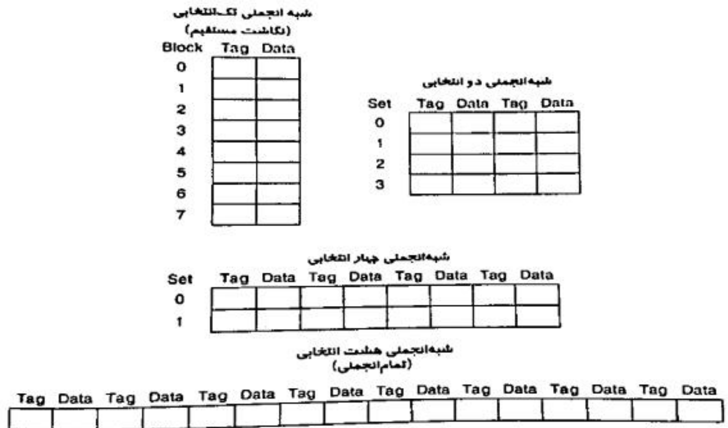
الگوی شبه انجمنی: هر بلوک را می‌توان در تعداد ثابت و مشخص از مکانهای حافظه نهان قرار داد.

حافظه نهان ← شبه انجمنی - n انتخابی ← متشکل از تعدادی مجموعه (Set) و هر مجموعه دارای ظرفیتی معادل n بلوک است.

⑨

در حافظه نهان مبتنی بر نگاشت مستقیم (تعداد بلوک‌ها در حافظه نهان) mod (آدرس بلوک)

در حافظه شبه انجمنی (تعداد مجموعه‌ها در حافظه نهان) mod (آدرس بلوک)



شکل ۵-۱۴ یک حافظهٔ نهان با ۸ بلوک که در چهار الگوی مختلفِ «نگاشت مستقیم»، «شبه‌انجمنی دو انتخابی»، «شبه‌انجمنی چهار انتخابی» و «تمام‌انجمنی» پیکربندی شده است. ظرفیت کلّ حافظهٔ نهان بر حسب تعداد بلوک مساوی است با تعداد مجموعه‌ها ضرب در درجهٔ انجمنی.

مثال: ارتباط بین فقدان و درجهٔ انجمنی در حافظهٔ نهان:

فرض کنید سه حافظهٔ نهان داریم. هر یک مشتمل از ۴ بلوک تک کلمه‌ای هستند. یکی از این حافظه‌های نهان تمام انجمنی دیگری شبه انجمنی دو انتخابی و سومی نگاشت مستقیم طراحی شده است. با فرض آنکه دنباله بلوک‌ها ۰، ۸، ۰، ۶، ۸ درخواست شده، بگید تعداد فقدان‌ها را برای هر کدام از معماری‌های یاد شده حساب کنید.

ساده‌ترین الگو نگاشت مستقیم است. ابتدا تعیین کنیم که هر یک آدرس‌های بلوک در حافظه اصلی به کدام بلوک در حافظه نهان نگاشت پیدا می‌کند

Block Address	Cache block
0	(0 Mod 4) = 0
6	(6 Mod 4) = 2
8	(8 Mod 4) = 0

Address of memory block accessed	Hit or Miss	Contents of cache block after reference			
		0	1	2	3
0	miss	Memory[0]			
8	miss	[8]			
0	miss	[0]			
6	miss	[0]		Memory[6]	
8	miss	[8]		〃[6]	

با این اوصاف حافظه نهان با معماری نگاشت مستقیم جمعاً به ازای پنج بار دستیابی به داده، پنج بار با پدیده فقدان مواجه می‌شود.

در حافظه نهان با معماری شبه انجمنی جمعاً دو مجموعه (با اندیس‌های صفر و یک) داریم که هر مجموعه مشتمل بر دو عنصر و جزء داده است. در اینجا باید ابتدا تعیین کنیم هر یک از آدرس‌های بلوک به کدام یک از مجموعه‌ها نگاشت می‌شود.

۱۵

نگاشته می‌شود:

Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

به دلیل آنکه در هر بار فقدان داده در حافظهٔ نهان، این آزادی انتخاب را داریم که کدامین درایه از حافظهٔ نهان را بیرون انداخته و آن را جایگزین کنیم بنابراین به قاعده‌ای در خصوص نحوهٔ جایگزینی نیازمندیم. در «حافظهٔ نهان شبه‌انجمنی» عموماً رسم بر این است که بلوکی که اخیراً کمترین مراجعه به آن صورت گرفته از یک مجموعه حذف و بلوک جدید به جای آن می‌نشیند؛ یعنی بلوکی که در گذشته‌های دورتر مورد استفاده قرار گرفته برای رونویسی انتخاب خواهد شد (به زودی با قواعد جایگزینی دیگر نیز آشنا خواهید شد). با تکیه بر این قاعدهٔ جایگزینی، محتوای «حافظهٔ نهان شبه‌انجمنی» پس از هر مراجعه به صورت زیر تغییر خواهد کرد:

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

توجه داشته باشید که وقتی به بلوک ۶ مراجعه می‌شود این بلوک جایگزین بلوک شمارهٔ ۸ خواهد شد چرا که بلوک ۸ در مقایسه با بلوک شمارهٔ ۰ دیرتر مورد مراجعه قرار گرفته است. با این اوصاف یک حافظهٔ نهان «شبه‌انجمنی دو انتخابی» (2-way set associative) جمعاً چهار بار با فقدان داده مواجه خواهد شد، یعنی یکی کمتر از حافظهٔ نهان با معماری «نگاشت مستقیم»!

در «حافظهٔ نهان تمام‌انجمنی» کلّ حافظه مشتمل بر چهار بلوک (یعنی یک مجموعهٔ واحد با چهار انتخاب برای جایگزینی) است و طبعاً هر بلوک حافظهٔ اصلی می‌تواند در هر بلوک دلخواه از حافظهٔ نهان نشانده شود. با این اوصاف «حافظهٔ نهان تمام‌انجمنی» تنها با ۳ فقدان بهترین کارآیی را از خود نشان می‌دهد:

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

برای چنین دنباله‌ای از مراجعات به حافظه، سه بار فقدان بهترین حالتی است که می‌توانیم به آن برسیم چرا که در این دنبالهٔ آدرس سه بلوک یکتا وجود دارد[1]. دقت کنید که اگر حافظهٔ نهان به جای ۴ بلوک کنونی هشت بلوک می‌داشت حتی در «حافظهٔ نهان شبه‌انجمنی دو انتخابی» به هیچ جایگزینی بلوک نیاز نداشتیم (این موضوع را خودتان بررسی کنید) و بدین ترتیب کارآیی این حافظهٔ نهان نیز همانند معماری «تمام‌انجمنی» به حداکثر مقدار ممکن می‌رسید. به روالی مشابه اگر حافظهٔ نهان ۱۶ بلوک می‌داشت هر سه الگو تعداد دفعات فقدان یکسان و کارآیی هم‌ترازی می‌داشتند. همین مثال پیش پا افتاده نشان می‌دهد که در تعیین کارآیی حافظهٔ نهان، «ظرفیت حافظهٔ نهان» و «درجهٔ انجمنی» [یا همان آزادی انتخاب در قرار دادن بلوک‌ها در حافظه نهان] مستقل از یکدیگر نیستند.

سؤال اینجاست که با «حافظهٔ نهان انجمنی» به چه میزان می‌توان نرخ فقدان» (miss rate) را کاهش داد؟ شکل ۵-۱۵ نشان می‌دهد که با داشتن حافظهٔ نهان ۶۴ کیلوبایتی و بلوک‌های ۱۶ کلمه‌ای و «درجهٔ انجمنی» متغیر در محدودهٔ نگاشت مستقیم تا «شبه‌انجمنی هشت انتخابی» میزان بهبود نرخ فقدان در چه حدودی است[2]. پیشروی از حالت تک‌انتخابی [نگاشت مستقیم] به سوی دو انتخابی (2-way) نرخ فقدان» را ۱۵% کاهش می‌دهد[3] در حالی که ارتقاء «درجهٔ انجمنی» از ۲ به بعد بهبود چشمگیری را عاید نمی‌کند.

درجهٔ انجمنی	نرخ فقدان داده
۱	۱۰/۳۳%
۲	۸/۶۶%
۴	۸/۳۵%
۸	۸/۱۵%

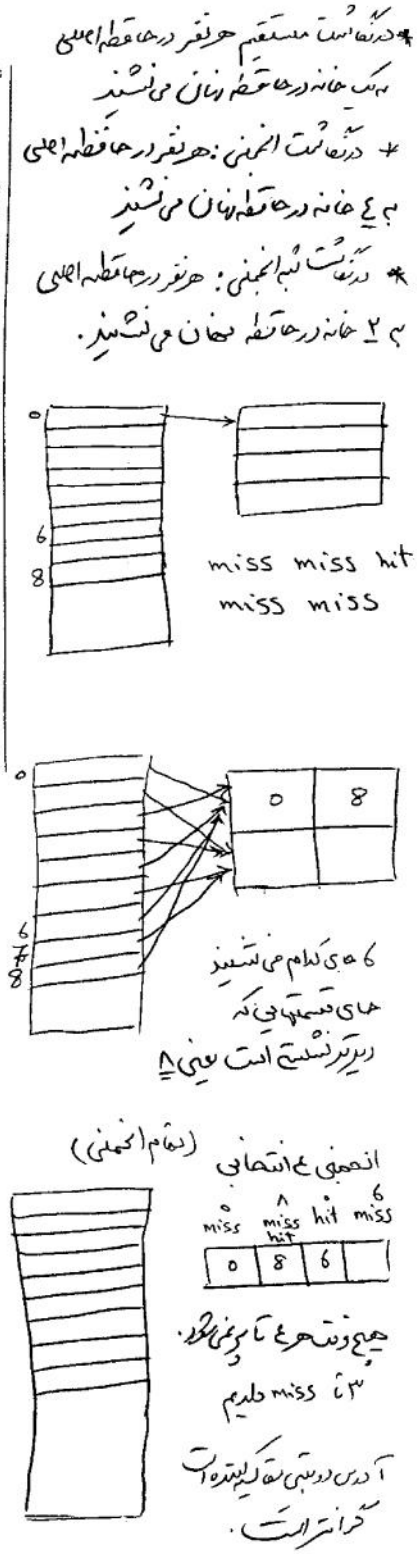
شکل ۵-۱۵ نرخ فقدان در «حافظهٔ نهان داده» (data cache) برای یک سیستم با سازمانی شبیه به حافظهٔ نهان پردازندهٔ Intrinsity FastMATH که در آن درجهٔ انجمنی از تک‌انتخابی (نگاشت مستقیم) تا هشت‌انتخابی متغیر بوده است. مشاهده بر اساس ده برنامهٔ مختلف از محک آزمون SPEC2000 به دست آمده‌اند.

تعیین موقعیت و یافتن یک بلوک در حافظهٔ نهان

اینک بیاید عملیات یافتن یک بلوک در «حافظهٔ نهان شبه‌انجمنی n-انتخابی» (n-way set associative) را بررسی کنیم. دقیقاً همانند حافظهٔ نهان با «نگاشت مستقیم» (direct mapped)، هر بلوک از حافظهٔ نهان شبه‌انجمنی نیز دارای یک «برچسب آدرس» (address tag) است که آدرس بلوک مربوطه را مشخص می‌کند. برای تعیین آنکه آیا آدرس بلوک درخواست شده توسط پردازنده با آدرس یکی از بلوک‌ها در حافظهٔ نهان تطبیق دارد یا خیر، برچسب تمام بلوک‌های موجود در یکی از مجموعه‌های متناسب با آدرس درخواست شده، مقایسه و وارسی می‌شود. شکل ۵-۱۶ نحوهٔ تجزیهٔ یک آدرس را به تصویر کشیده است:

Tag	Index	Block offset

شکل ۵-۱۶ سه بخش از یک آدرس در حافظهٔ نهان با الگوی نگاشت مستقیم یا شبه‌انجمنی چند انتخابی. طبق پس «Index» برای انتخاب یکی از مجموعه‌ها به کار گرفته می‌شود؛ فیلد «برچسب» (Tag) باید با مقدار فیلد برچسب آدرس درایهٔ انتخابی مقایسه شود. فیلد «آفست بلوک» (Block offset) برای انتخاب داده مورد نظر از میان عناصر یک بلوک به کار گرفته می‌شود.



Sets (11)

مثال: اندازه برچسب و درجه انجمنی در حافظه نهان

با فرض در اختیار داشتن یک حافظه نهان با ۶۴ کیلو بلوک و بلوک‌هایی به طول ۴ کلمه و آدرس‌های ۳۲ بیتی تعداد مجموعه‌ها و تعداد کل بیت‌های برچسب را برای حافظه نهان نگاشت مستقیم، انجمنی دو انتخابی، انجمنی ۴ انتخابی و تمام انجمنی محاسبه کنید.

حل: از آنجایی که هر بلوک جمعاً ۱۶ بایت (2^4) ظرفیت دارد از این رو از مجموع ۳۲ بیت آدرس ۲۸ بیت (۳۲-۴) است برای فیلدهای اندیس و برچسب باقی می‌ماند. ۱۲ بایت

حافظه نهان نگاشت مستقیم تعداد مجموعه‌ها با تعداد بلوک‌ها یکسان است. $Log_2(4k)=12$ بنابراین بیت‌های فیلد اندیس

مجموع کل بیت‌های برچسب $(28-12)\times 4K = 16\times 4K$ معادل 64 کیلو بیت خواهد شد.

برای حافظه نهان شبه انجمنی با دو انتخاب (2-way) جمعاً ۲۰۴۸ مجموعه خواهیم داشت. و تعداد بیت‌ها برچسب ۴۸ کیلو بیت.

$(28-11)\times 2\times 2K = 34\times 2K$

حافظه نهان شبه انجمنی با چهار انتخاب (4-way) تعداد کل مجموعه ۱۰۲۴ تاست و تعداد کل بیت‌ها برچسب معادل

$(28-10)\times 4\times 1K = 72\times 1K = 72K$

حافظه نهان تمام انجمنی تنها یک مجموعه با ۴۰۹۶ بلوک وجود دارد و فیلد برچسب ۲۸ بیتی است. بنابراین تعداد کل بیت‌های برچسب معادل $28\times 4\times 1K = 112$ Kbits خواهد بود.

مثال: کارایی حافظه‌های نهان چند سطحی

پردازنده‌ای که همه دستورها در حافظه اول با برخورد موفقیت‌آمیز (hit) به انجام می‌رسد. دارای مقدار مبنای $CPI = 1.0$

نرخ ساعت نیز 4GHZ زمان دسترسی به حافظه اصلی (شامل تمام جریمه‌های ناشی از رسیدگی به فقدان) را 100ns

مجموع نرخ فقدان miss Rate به ازای هر دستورالعمل ۲٪

یک پردازنده یک حافظه نهان سطح دوم L2 cache با زمان دسترسی 5ns

نرخ فقدان ۲ ٪۵ کاهش یابد ← آنگاه پردازنده چه میزان سریع‌تر کار خواهد کرد.

جریمه فقدان $\frac{100ns}{0.25\frac{ns}{clock\ cycle}} = 400$ clock cycle

$N\times T + 0.02N\times 400T = 1NT + 8NT = 9NT$

$T = \frac{1}{4\times 10^9} = \frac{10^{-9}}{4}$

خواندن $Tr = \frac{100\times 10^{-9}}{10^{-9}/4} = 400$

یک سطحی L1 cache → Total CPI = Base + CPI + memory-Stall cycles per instruction

$Total\ CPI = 1.0 + Memory\ Stall --- = 1.0 + 2\% \times 400 = 9$

حافظه نهان دو سطحی → جریمه فقدان → $\frac{5ns}{0.25\frac{ns}{clock\ cycle}} = 20$ clock cycle

Totall CPI = Base CPI + Primary Stall per instruction + Secondary Stall per instruction ⇒ $1 + 2\% \times 20 + 0.5\% \times 400 = 1 + 0.4 + 2 = 3.4$

$9.0 / 3.4 = 2.6$ سریعتر کار خواهد کرد.

$NT + 0.02N\times 20T + 0.005\times 400T$

$\Rightarrow NT + 0.4NT + 2NT = 3.4NT$

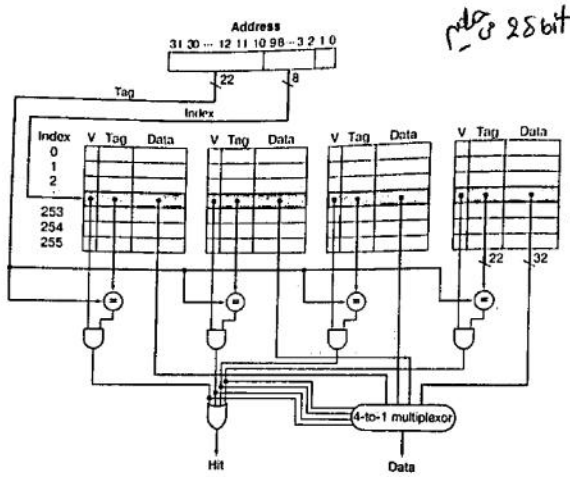
ضریب افزایش سرعت $= \frac{9NT}{3.4NT} = 9/3.4$

(۱۲)

در نگاشت مستقیم یک مقایسه کننده ۱۶ بیتی

در نگاشت تمام انجمنی 4096 مقایسه کننده 28bit می خواهیم

نمونه نگاشت مستقیم



شکل ۵-۱۷ پیاده‌سازی یک حافظهٔ نهان شبه‌انجمنی چهار انتخابی نیازمند ۴ مقایسه‌کننده و یک مالتی‌پلکسر ۴ به ۱ است. مقایسه‌کننده‌ها تعیین می‌کنند که کدام یک از اعضای مجموعهٔ انتخاب‌شده با فیلد برچسب مطابقت دارند (البته در صورت حضور). خروجی مقایسه‌کننده‌ها برای کنترل مالتی‌پلکسر مورد استفاده قرار می‌گیرند تا این مالتی‌پلکسر از بین یکی از ۴ عضو هر بلوک یکی را به خروجی هدایت کند. خروجی Hit تعیین‌کنندهٔ آن است که آیا دادهٔ مورد نظر در حافظهٔ نهان وجود داشته است یا خیر. مقادیر ظاهر شده در خروجی مالتی‌پلکسر در صورت فعال بودن این خط قابل استفاده توسط پردازنده هستند.

آدرس
4K بلوک
16bit
۱۶ بایت فضا داریم
آدرس‌ها ۱۲ رقمی است 4K

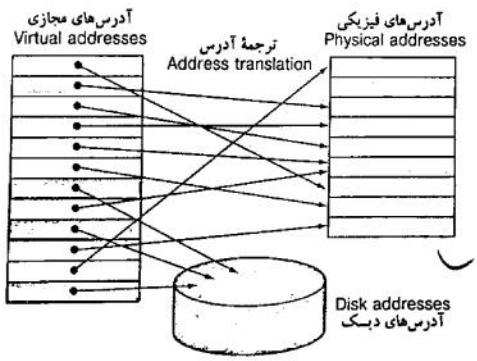
نمونه نگاشت انجمنی ۲ انتخابی

آدرس آدرس
6C4
16bit

انجمنی ۴ انتخابی ← ۴ مقایسه کننده ۱۸ بیتی

حافظه مجازی

حجم حافظه را می‌شناسد اما نوع آن را نمی‌داند.

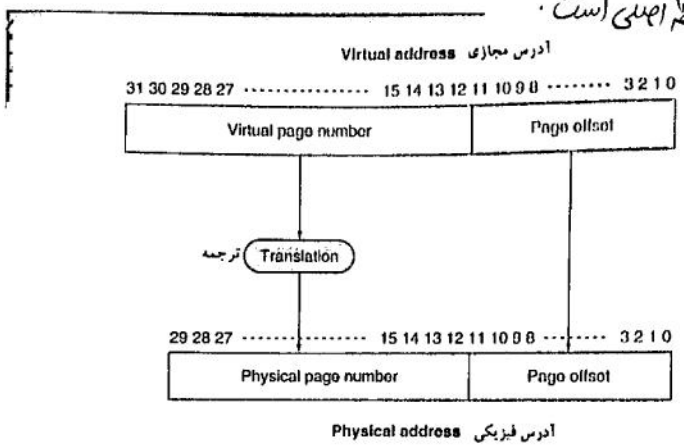


شکل ۵-۱۹ در حافظهٔ مجازی بلوک‌های حافظه (موسوم به صفحات) از مجموعه‌ای از آدرس‌های مجازی به مجموعه‌ای دیگر از آدرس‌های فیزیکی نگاشته می‌شوند. آدرس‌هایی که پردازنده تولید می‌کند مجازی هستند در حالی‌که برای دستیابی به مقادیر واقعی در حافظه به آدرس‌های فیزیکی نیاز است. چه حافظهٔ مجازی و چه حافظهٔ فیزیکی به قطعاتی موسوم به صفحه (page) تکه‌تکه می‌شوند و هر صفحهٔ مجازی به یک صفحهٔ فیزیکی نگاشته خواهد شد. البته این امکان وجود دارد که یک صفحهٔ مجازی در حافظهٔ اصلی موجود نباشد. بدین ترتیب به هیچ صفحهٔ فیزیکی نگاشته نشده باشد؛ در این صورت باید بر روی دیسک سخت به دنبال آن گشت. یک صفحهٔ فیزیکی واحد ممکن است با دو یا چند آدرس مجازی متفاوت بین چند برنامه به اشتراک گذاشته شود.

سیستم از حافظه مجازی را صفحه Page می نامند.
هرگاه داده‌ای در حافظه مجازی یافت نشد به فقدان آن اصطلاحاً نقص صفحه Page fault می گویند.
برای رجوع به حافظه مجازی یک آدرس مجازی تولید می‌کند و این آدرس به کمک ترکیبی از سخت افزار و نرم افزار به آدرس فیزیکی ترجمه می‌شود.

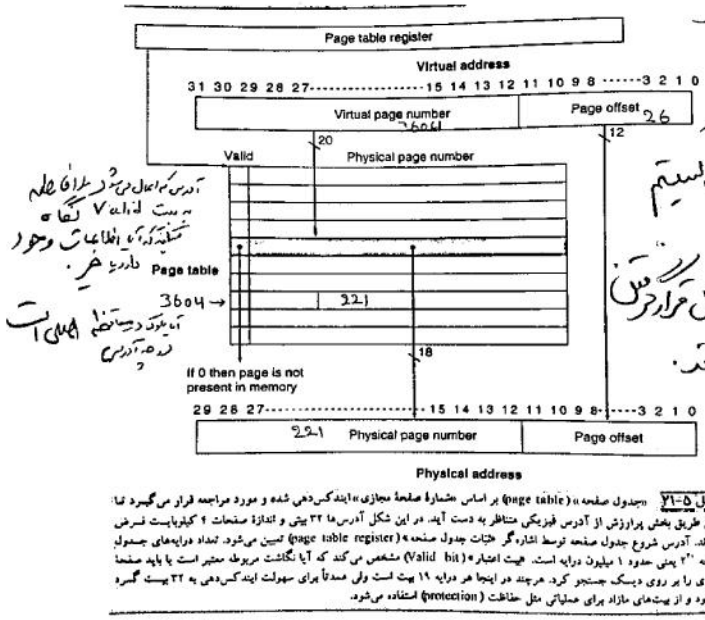
در حافظه مجازی آدرس به دو بخش شماره صفحه مجازی virtual page number و آفست صفحه Page offset تقسیم می‌شود.
شماره صفحه فیزیکی به بخش پردازشی از آدرس فیزیکی را تشکیل می‌دهد.
مقدار آفست صفحه در هیچ تغییری بخش کم اهمیت از آدرس فیزیکی را تشکیل داده است.

(۱۳) در سیستم‌های حافظه مجازی برای تعیین موقعیت و یافتن یک صفحه در حافظه اصلی از جدولی برای اندیس گذاری حافظه استفاده می‌کنیم. چنین ساختمانی اصطلاحاً در جدول صفحات Page table نامیده می‌شود. و محل نگهداری آن در همان حافظه اصلی است.



شکل ۵-۲۰ نگاشت آدرس‌های مجازی به آدرس‌های فیزیکی. اندازهٔ هر صفحه ۲ کیلو بایت (2^{12} بایت) و تعداد صفحات فیزیکی بُجاز در حافظهٔ اصلی 2^{18} صفحه است زیرا از یک آدرس ۳۲ بیتی فقط ۱۸ بیت برای آدرس‌دهی صفحات فیزیکی باقی می‌ماند. با این اوصاف حافظهٔ اصلی می‌تواند حداکثر یک گیگابایت ظرفیت داشته باشد در حالی که فضای آدرس‌دهی مجازی ۴ گیگابایت است.

جدول صفحات اندیس گذاری شده و فیلد شماره صفحه در متن آدرس مجازی به عنوان اندیسی برای مراجعه به این جدول بکار گرفته می‌شود. و درایهٔ متناظر با این اندیس، شماره فیزیکی صفحه متناظر را در حافظهٔ اصلی مشخص می‌کند.



شکل ۵-۲۱ «جدول صفحه» (page table) بر اساس شمارهٔ صفحهٔ مجازی «اندیس‌دهی شده و مورد مراجعه قرار می‌گیرد تا بدین طریق بخش پیرازش از آدرس فیزیکی متناظر به دست آید. در این شکل آدرس‌ها ۳۲ بیتی و اندازهٔ صفحات ۴ کیلوبایت فرض شده‌اند. آدرس شروع جدول صفحه توسط اشاره‌گر ثبات جدول صفحه» (page table register) تعیین می‌شود. تعداد درایه‌های جدول صفحه 2^{20} یعنی حدود ۱ میلیون درایه است. هیت اعتبار» (Valid bit) مشخص می‌کند که آیا نگاشت مربوطه معتبر است یا باید صفحهٔ مجازی را بر روی دیسک جستجو کرد. هرچند در اینجا هر درایه ۱۹ بیت است ولی عمدتاً برای سهولت اندیس‌دهی به ۳۲ بیت گسترش می‌شود و از بیت‌های مازاد برای عملیاتی مثل حفاظت (protection) استفاده می‌شود.

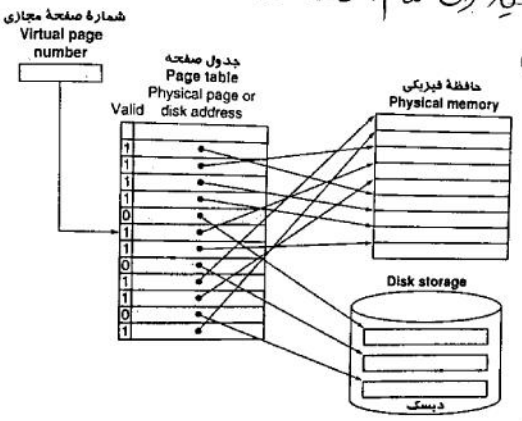
نقطان صفحات: هرگاه دست اعتبار برای یک صفحه مجازی خاموش باشد یعنی آن صفحه در حافظه اصلی قرار ندارد و باید به نقطان صفحه اقدام شود. و در چنین شرایطی مادر کنترل اجرا به سیستم عامل سپرده می‌شود.

آدرس مجازی بخودی خود نمی‌تواند مستقیماً عمل مراجعه به حافظه را انجام دهد بلکه ابتدا باید به آدرس فیزیکی تبدیل شود تا صفحات را بروی دستگاه فیزیکی مشخص کند.

برای مشخص کردن موقعیت جدول صفحات در حافظه اصلی، دست افزار یک ثبات خاص را برای اشاره به نقطه شروع چنین جدولی در اختیار می‌گذارد. چنین ثباتی را ثبات جدول صفحات Page table register می‌گوییم.

(۱۴)

از آنجا که پیشاپیش نمی‌دانیم که یک صفحه در حافظه مجازی جایگزین (replace) خواهد شد، سیستم عامل معمولاً برای هر برنامه‌ای و جداول صفحه‌بندی آن، برای تمام صفحات مورد استفاده، آن برنامه بر روی دیسک ایجاد خواهد کرد.



شکل ۵-۲۲ «جدول صفحه» یکایک صفحات در حافظهٔ مجازی را به صفحات حافظهٔ اصلی یا صفحات ذخیره‌شده بر روی دیسک (به عنوان سطح بعدی در سلسله‌مراتب حافظه) می‌نگارد. بخش «شمارهٔ صفحهٔ مجازی» در یطن هر آدرس به عنوان اندیسی به جدول صفحه اعمال شده و سپس در درایهٔ متناظر، «بیت اعتبار» آزمایش می‌شود؛ اگر این بیت فعال بود درایهٔ متناظر مستقیماً شمارهٔ صفحهٔ فیزیکی را در اختیار می‌گذارد ولیکن در صورت خاموش بودن «بیت اعتبار»، آن صفحه در آدرس مشخصی از دیسک سخت قرار گرفته است. در بسیاری از سیستم‌ها آدرس صفحات فیزیکی حافظه و آدرس صفحات دیسک علیرغم آنکه منطقاً یک جدول واحد هستند ولیکن در دو ساختمان دادهٔ مجزا نگهداری می‌شوند.

برای اینکه به این بخش از دیسک اصطلاحاً فضای مبادله Swap Space گفته می‌شود.

همچنین سیستم عامل در لحظهٔ ایجاد یک برنامه ساختمان داده‌ای را به وجود می‌آورد تا درون آن نشانی دقیق ذخیرهٔ هر صفحهٔ مجازی بر روی دیسک ثبت و ضبط شود.

سیستم عامل همچنین ساختمان داده‌ای را ایجاد می‌کند تا از طریق آن مشخص کند که هریک از برنامه‌ها و هریک از آدرس‌های مجازی آن کدامیک صفحات فیزیکی در حافظهٔ اصلی استفاده کرده‌اند.

با رخ دادن خطای نقص صفحه Page fault اگر تمام صفحات حافظهٔ اصلی در اشغال باشند، سیستم عامل موظف است صفحه‌ای را برای جایگزینی کردن و بیرون انداختن از حافظهٔ اصلی برگزیند. از آنجا که کامل دلیلی تواند رخدادهای نقص صفحه به کمترین ممکن برساند، سیستم عامل می‌کوشد تا صفحه‌ای را برای جایگزینی انتخاب کند که بیشترین مدت آینده تا نزدیک به آن نیازی نخواهد بود.

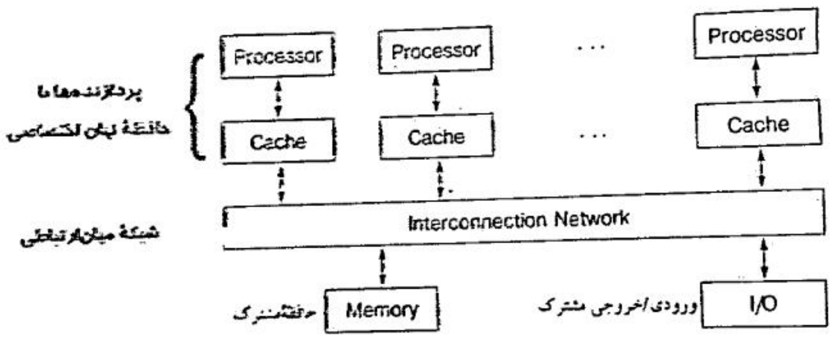
چگونگی نوشتن در حافظهٔ مجازی:

در حافظهٔ مجازی نوشتن در سطح پایه‌ای از سلسله مراتب حافظه (یعنی دیسک سخت) مدت زمانی بسیار بیشتر از سطوح بالاتر طول خواهد کشید. از این رو تعیین کنید برای آنکه سیستم بتواند عمل نوشتن را به صورت تمام عیار انجام دهد کاملاً ممکن است. در مقابل سیستم حافظه باید از روش بازنویسی استفاده کند: write_back تا زمانی که داده‌ها فقط در درون صفحهٔ متناظر در حافظهٔ اصلی نوشته شده و بعداً در زمان جایگزین شدن آن صفحه بر روی دیسک بازنویسی شود.

فصل هفتم

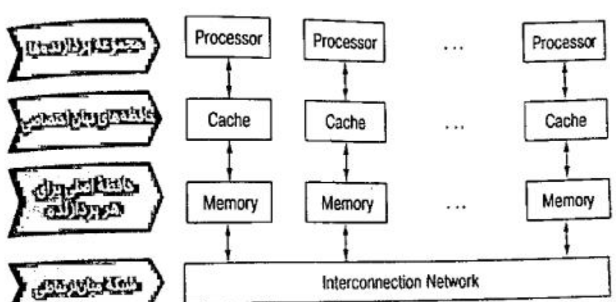
(۱) (فصل هفت)

یک چندپردازنده با حافظه اشتراکی موسوم به Shared memory multiprocessor یا SMP به اشتراک گذاشتن یک فضای آدرس دهی فیزیکی یکتا بین تمام پردازنده‌ها ، از نظر برنامه‌نویسان حافظه‌ای یکپارچه را در اختیار می‌نهد هرچند با چنین تعریفی نامگذاری چندپردازنده با فضای آدرس مشترک گویاتر و دقیق‌تر به نظر می‌رسد.



شکل ۷-۲ سازمان کلاسیک «چندپردازنده» با فضای حافظهٔ مشترک.

چندپردازنده‌ها با فضای آدرس دهی یکتا به دو شیوه طراحی می‌شوند . ۱- زمان دسترسی به حافظه اصلی فارغ از آنکه کدام پردازنده تقاضای دسترسی داشته و کدامین کلمه از حافظه را درخواست کرده است ثابت و یکسان باقی می‌ماند. به چنین ماشینهایی عموماً چندپردازنده‌ی UMA گفته می‌شود.
۲ - برخی از دسترسی‌ها و مراجعات به حافظه بسته به آنکه کدام پردازنده، کدامین کلمه حافظه را درخواست داده است بسیار سریعتر از بقیه صورت می‌گیرد . چنین ماشین‌های چندپردازنده‌ی NUMA نام گرفته است.



شکل ۷-٤ سازمانی کلاسیک از یک چندپردازنده که در آن هر یک از پردازنده‌ها از فضای آدرس اختصاصی خودشان بهره می‌گیرند و از طریق یک شبکهٔ میان‌ارتباطی (interconnection network) با یکدیگر به «مبادله پیام» می‌پردازند. این ساختار به طور سنتی «چندپردازندهٔ مبتنی بر مبادلهٔ پیام» نامیده می‌شود. دقت کنید که برخلاف ساختار SMP در شکل ۷-۲ شبکهٔ میان‌ارتباطی بین حافظهٔ نهان و حافظهٔ اصلی قرار ندارد بلکه بین گروه‌های مشتمل بر «پردازنده حافظه» ایفای نقش می‌کند.

مثال بهره‌وری حافظه : یک پردازنده اشتراکی SMP را که مجهز به ۲۰ گیگابایت از حافظه اصلی است در مقایسه با پنج کامپیوتر خوشه‌ای clustered که هرکدام ٤ گیگابایت حافظه دارند و سیستم عاملی به یک گیگابایت نفضای همان حافظه را اشغال می‌کند . در نظر بگیرید: در مقایسه بهره‌ی اندازه حافظه تجربی در حقیقت برای هر کدام

$$\frac{۲۰-۱}{۵ \times (٤-۱)} = \frac{۱۹}{۱۵} = ۱,۲۵$$

Kazem.faridi@iaubaft.ac.ir