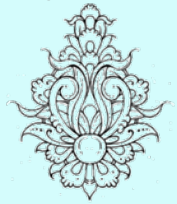
معماری کامپیوتر (۱۳۹۱-۱۱-۱۳)

جلسه‌ی هفدهم

دانشگاه شهید بهشتی
دانشکده‌ی مهندسی برق و کامپیوتر
بهار ۱۳۹۱
احمد محمودی ازناوه

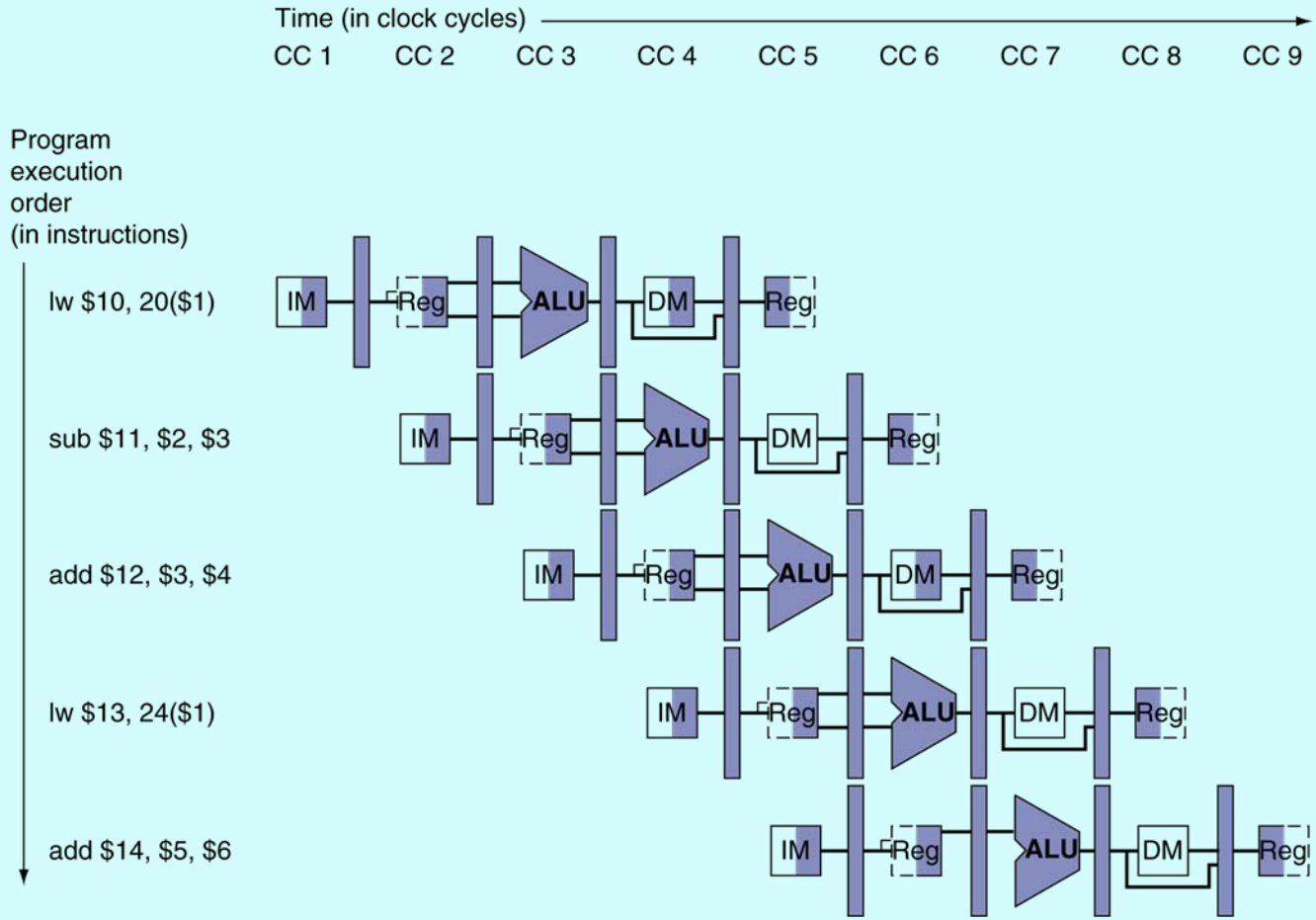
فهرست مطالب

- واحد کنترل در خط لوله
  - پیش‌فرستادن (هدایت رو به جلو)
  - واحد تشخیص مخاطره



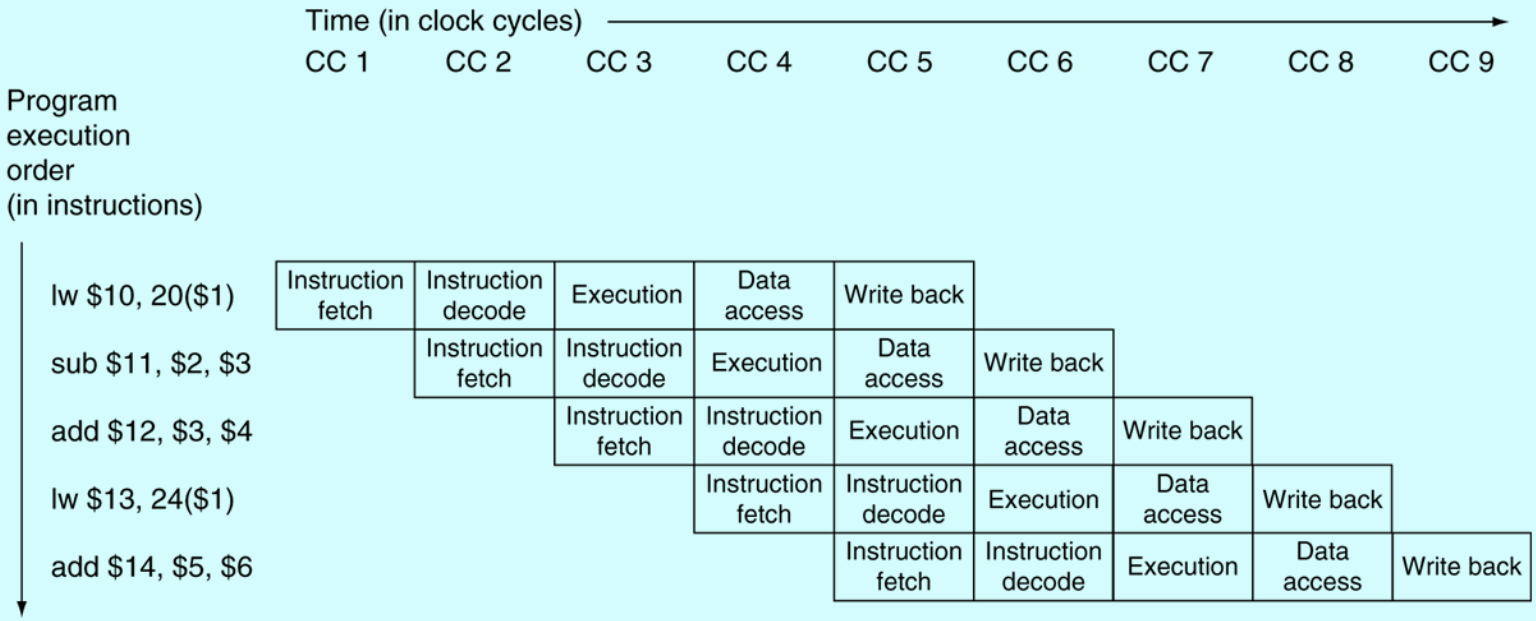
نمودار خط لوله به صورت چندسیکلی

- در این شیوه به کارگیری منابع نشان داده شده است



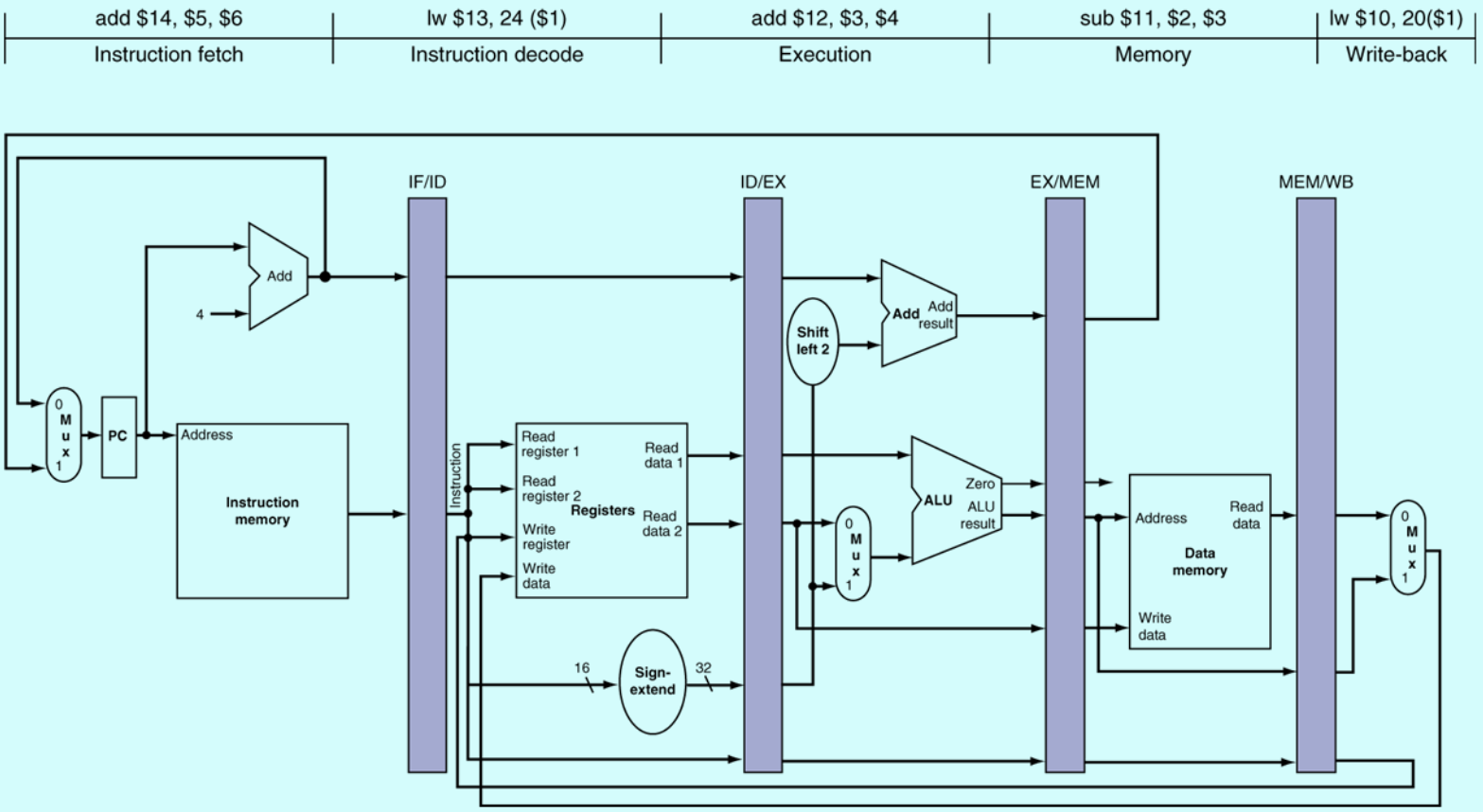
نمودار خط لوله به صورت چندسیکلی (ادامه...)

- **شیوه‌ی متعارف**

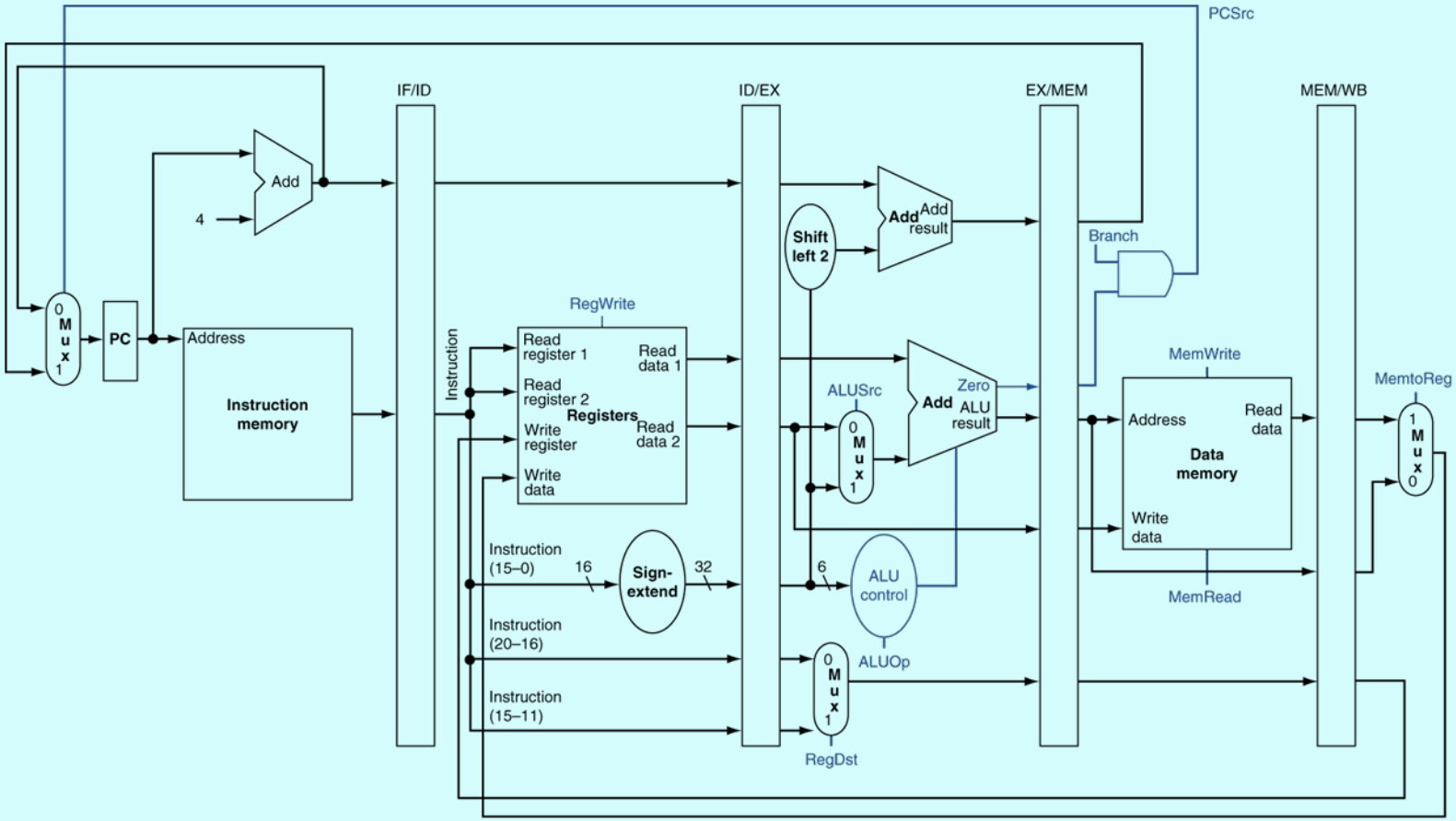


نمودار خط لوله به صورت تک‌سیکلی

- **حالت خط لوله در یک سیکل**

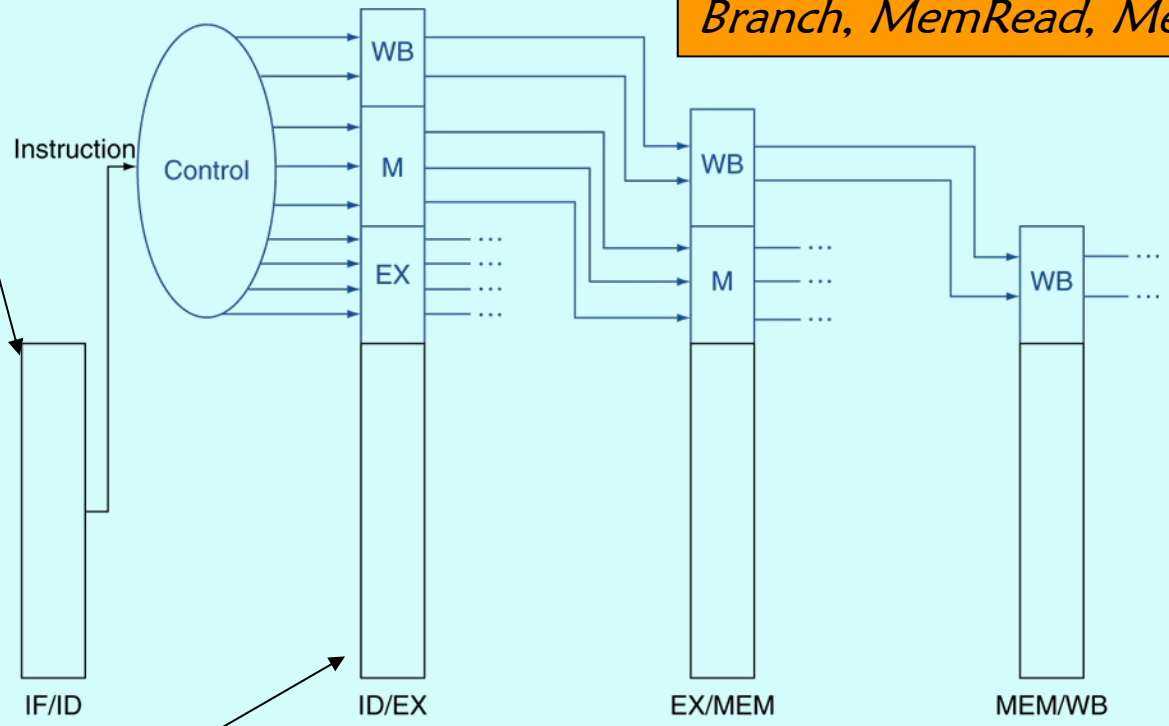


واحد کنترل خط لوله



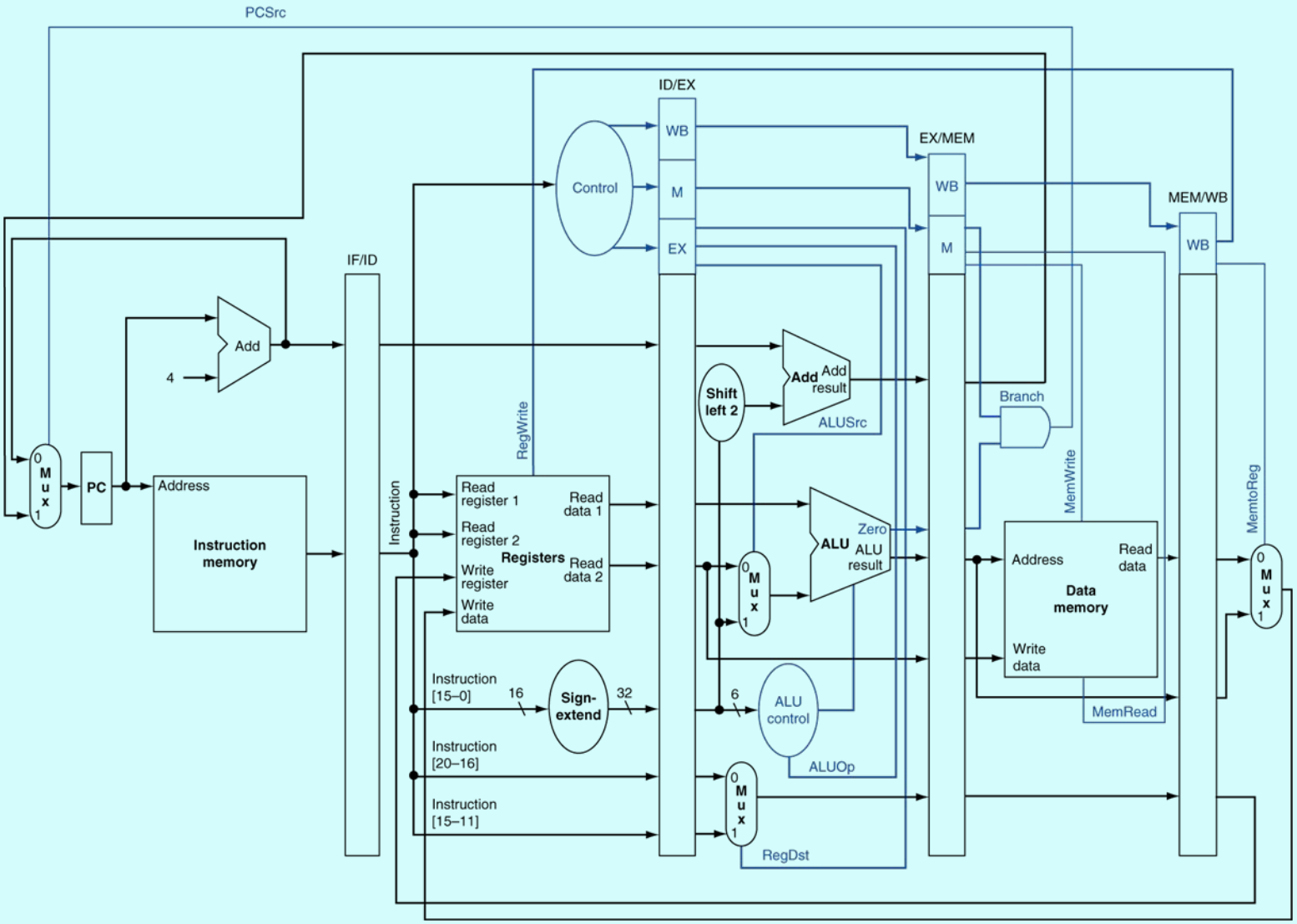
واحد کنترل خط لوله (ادامه...)

در این بخش می‌باید دستورالعمل بعدی واکشی شود، کاری که برای همه‌ی دستورها به یک شیوه خواهد بود



همانند گام پیشین، نیاز به سیگنال کنترلی خاصی نیست

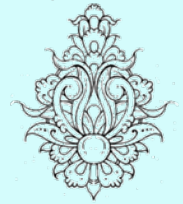
واحد کنترل خط لوله (ادامه...)



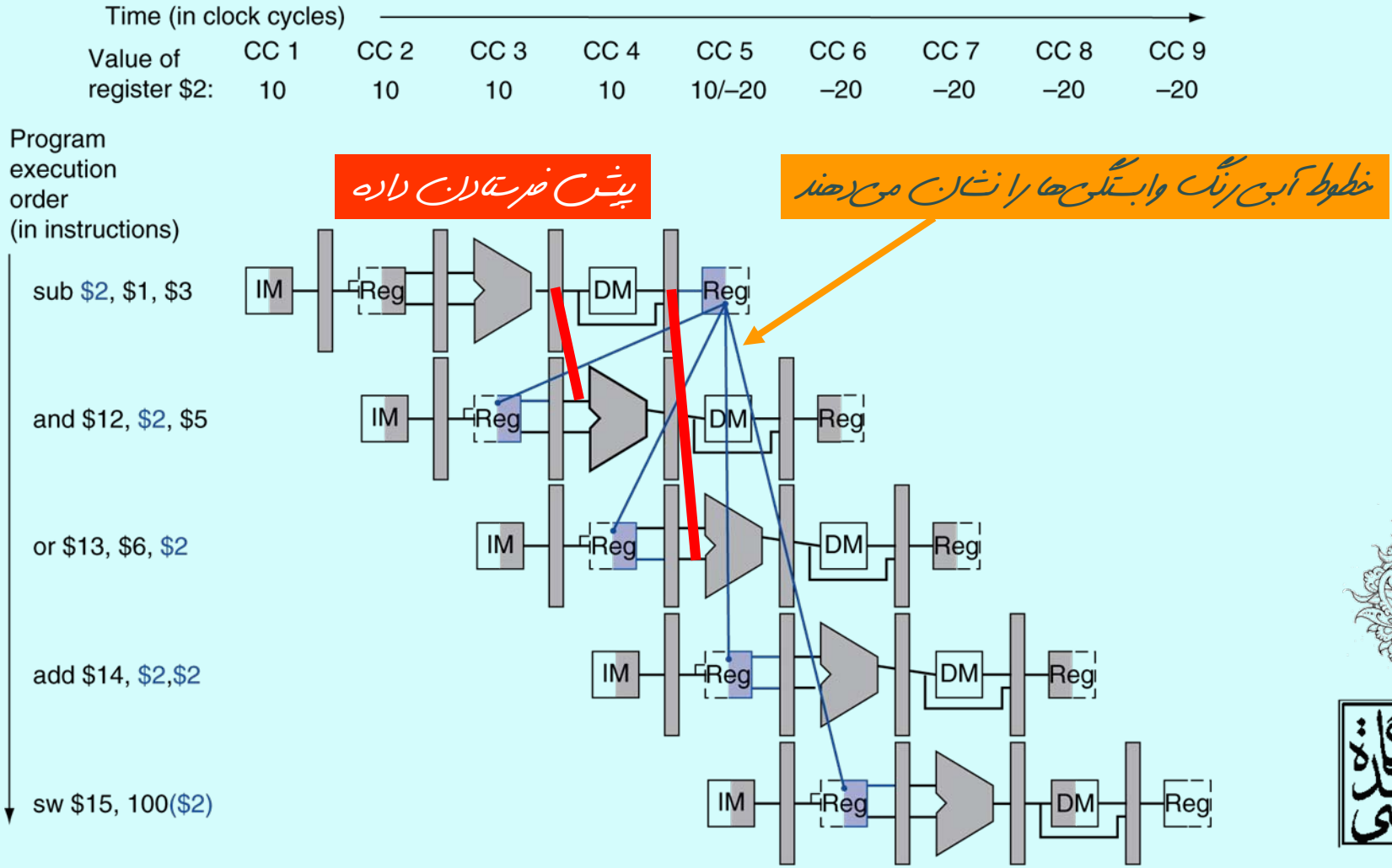
مخاطرات داده‌ای- مثال

```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

- در این قطعه برنامه، چهار دستورالعمل آخر به مقدار $2 وابسته هستند.
- چگونه می‌توان با پیش‌فرستادن مشکل وابستگی را حل کرد؟



مخاطرات داده‌ای (ادامه...)



پیش فرستادن

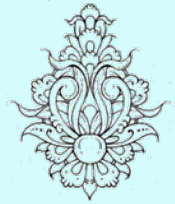
- چنان‌چه دیده شد، برای رهایی از مخاطرات داده، پیش‌فرستادن داده، راهکاری متداول است.
  - در ادامه خواهیم دید پیش‌فرستادن چگونه انجام می‌شود. برای سادگی تنها حالتی را بررسی خواهیم کرد، که داده در مرحله‌ی EX تولید می‌شود.

ID/EX.RegisterRs

شماره‌ی ثباتی را نشان می‌دهد که مقدار آن در ثبات ID/EX خط لوله قرار دارد.

- عملوندهای ALU در کدام ثبات قرار دارند؟

ID/EX.RegisterRs, ID/EX.RegisterRt



پیش فرستادن (ادامه...)

- **مخاطره‌ی داده در موارد زیر روی می‌دهد:**

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

Fwd from EX/MEM pipeline reg

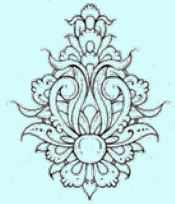
Fwd from MEM/WB pipeline reg

نوع یک

نوع دو

بدون مخاطره

```
sub $2,  $1,$3
and $12, $2, $5
or  $13, $6, $2
add $14, $2, $2
sw  $15, 100($2)
```



تشخیص نیاز به پیش فرستادن

- بنابراین، می‌توان با مقایسه‌ی محتوای ثبات‌ها، مداری برای کنترل پیش‌فرستادن داده طراحی کرد.

```
sub $2, $1,$3
and $12,$2,$5
```

EX/MEM.RegisterRd = ID/EX.RegisterRs=$2

ولی

در همه‌ی دستورالعمل‌های مقدار خروجی ALU، در ثبات نوشته نمی‌شود بدین ترتیب این راهکار در همه‌ی موارد درست نخواهد بود.

برای پیش‌گیری از این مسأله می‌توان از سیگنال‌های کنترلی مربوط به WB ذخیره شده در ثبات‌های خط لوله استفاده نمود.

EX/MEM.RegWrite, MEM/WB.RegWrite

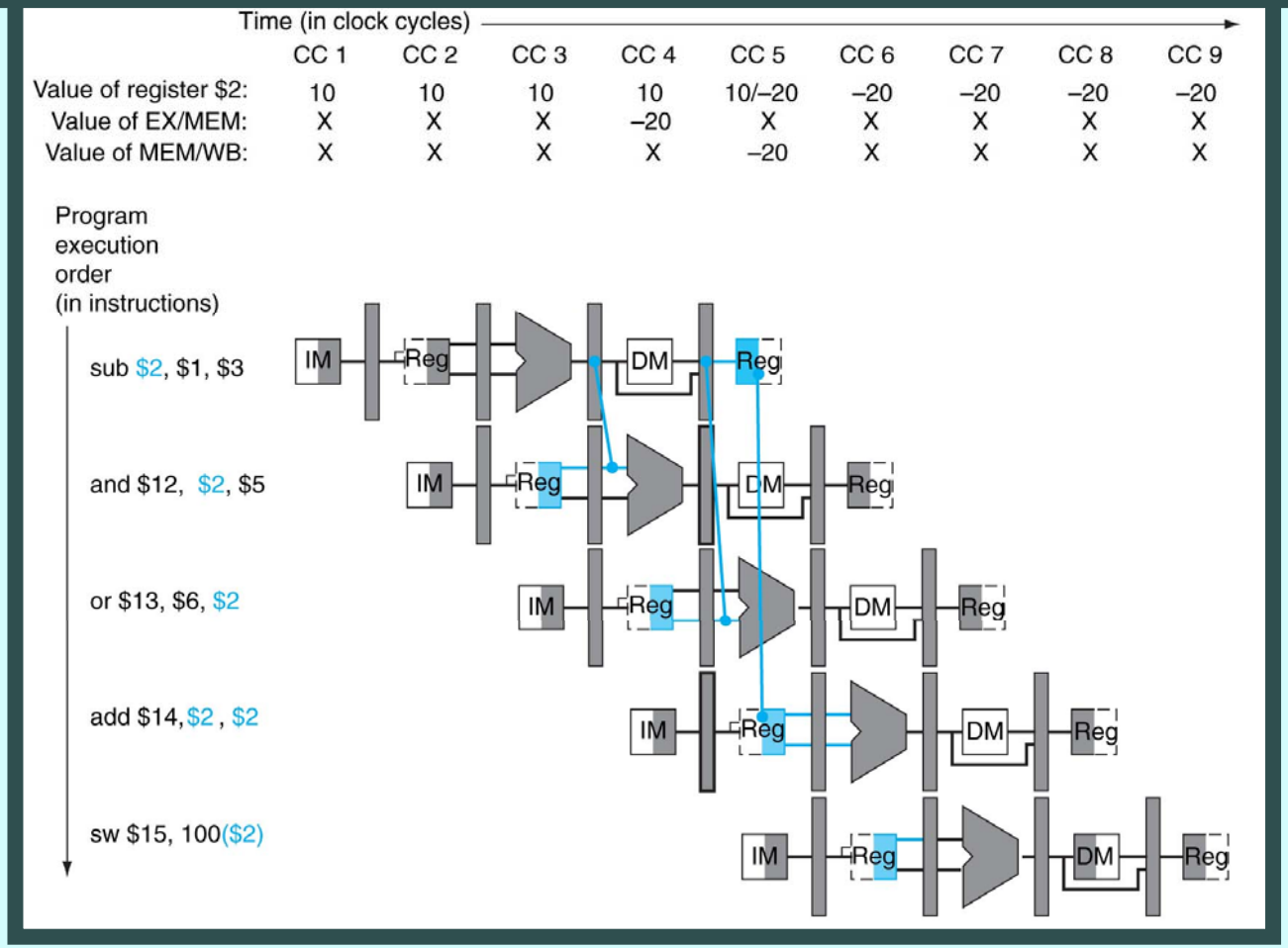
همچنین در صورتی که ثبات شماره‌ی صفر به عنوان مقصد یک دستور استفاده شده باشد، باید از پیش فرستادن جلوگیری کرد.

EX/MEM.RegisterRd ≠ 0,
MEM/WB.RegisterRd ≠ 0

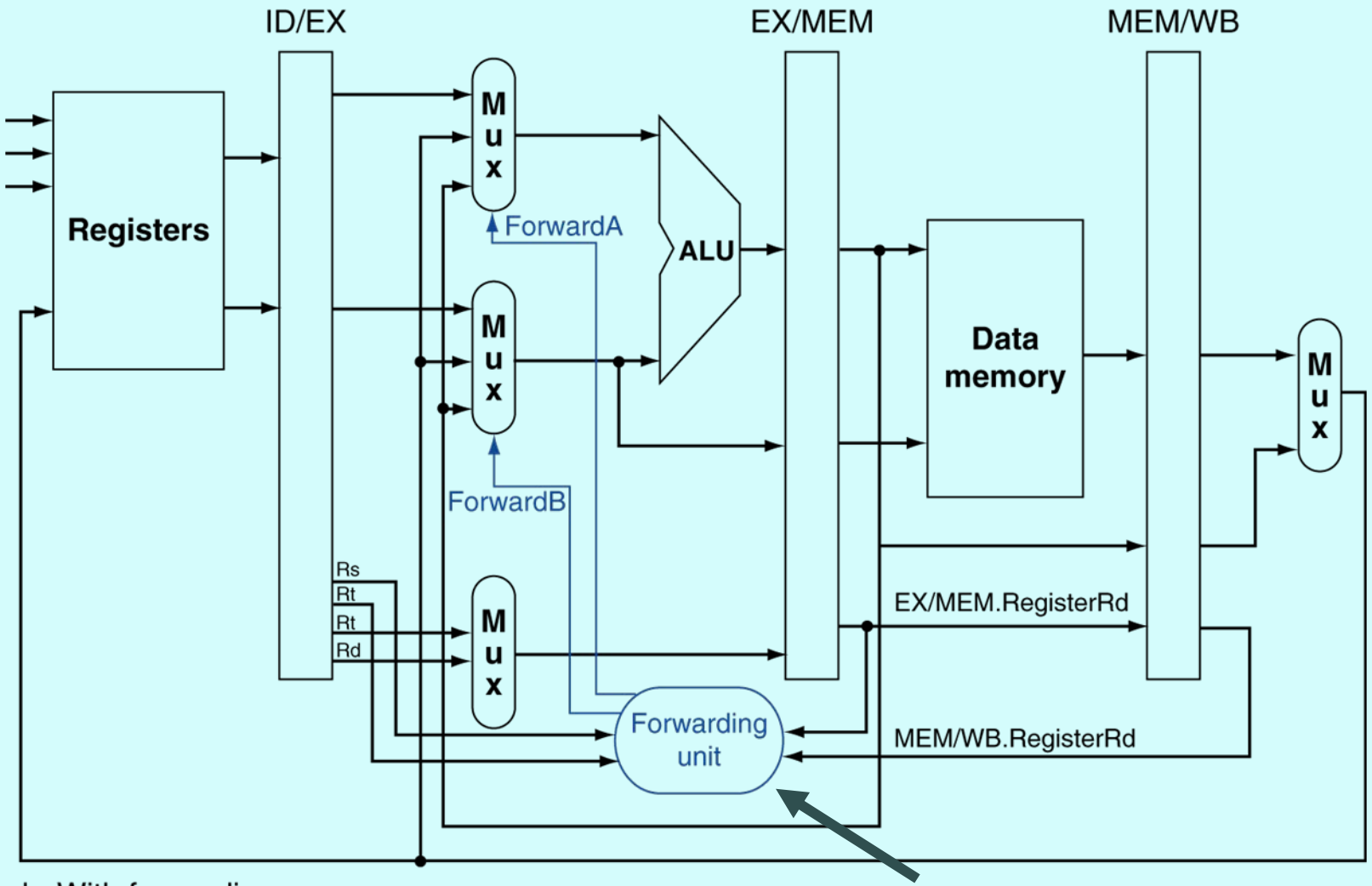
مسیر پیش فرستادن داده

با تشخیص زمان‌هایی که پیش‌فرستادن لازم است، نیمی از مشکلات حل شد،
هنوز نیمه‌ی دیگر باقی مانده است

:D

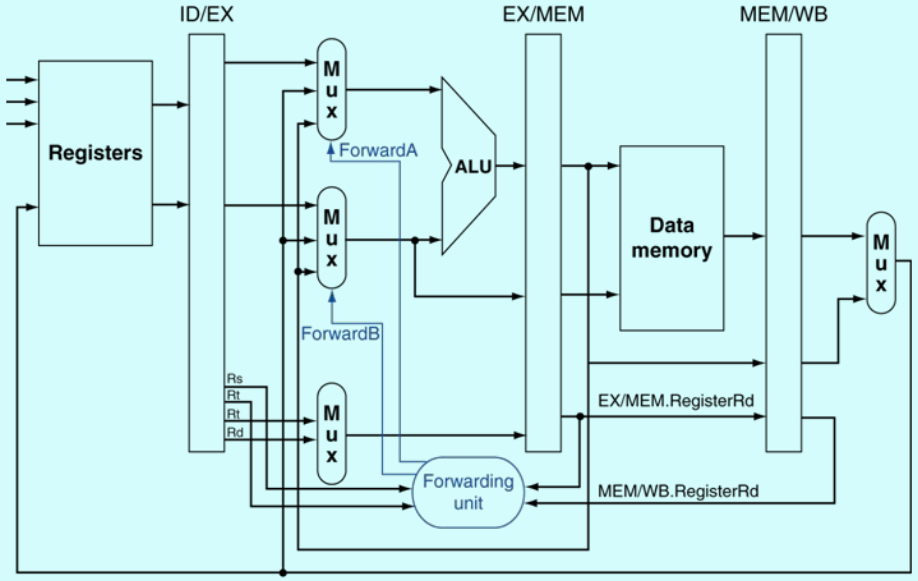


مسیر پیش فرستادن داده (ادامه...)



کنترل پیش فرستادن در مرحله‌ی EX قرار دارد، چرا که
خانه‌ی بافر پیش فرستادن در این مرحله قرار دارد.

مسیر پیش فرستادن داده (ادامه...)



b. With forwarding

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

شرایط پیش‌فرستادن

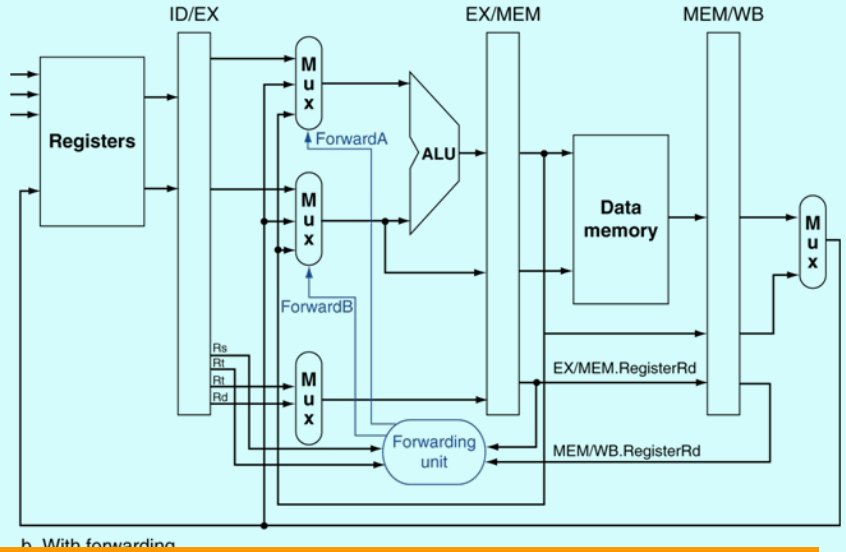
- EX hazard
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 10
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 10
- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01

اگر هر دوی این شرایط با هم رخ داد، چه اشکالی ایجاد می‌شود؟

شرایط پیش‌فرستادن (ادامه...)

- در این قطعه برنامه هر دو نوع مخاطره رخ می‌دهد.

```
add $1,$1,$2
add $1,$1,$3
add $1,$1,$4
```

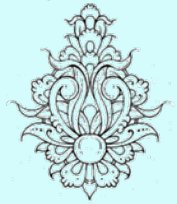


در این حالت آخرین نتیجه باید فرستاده شود،
در نتیجه داده‌ی موجود در مرحله‌ی MEM فرستاده می‌شود.

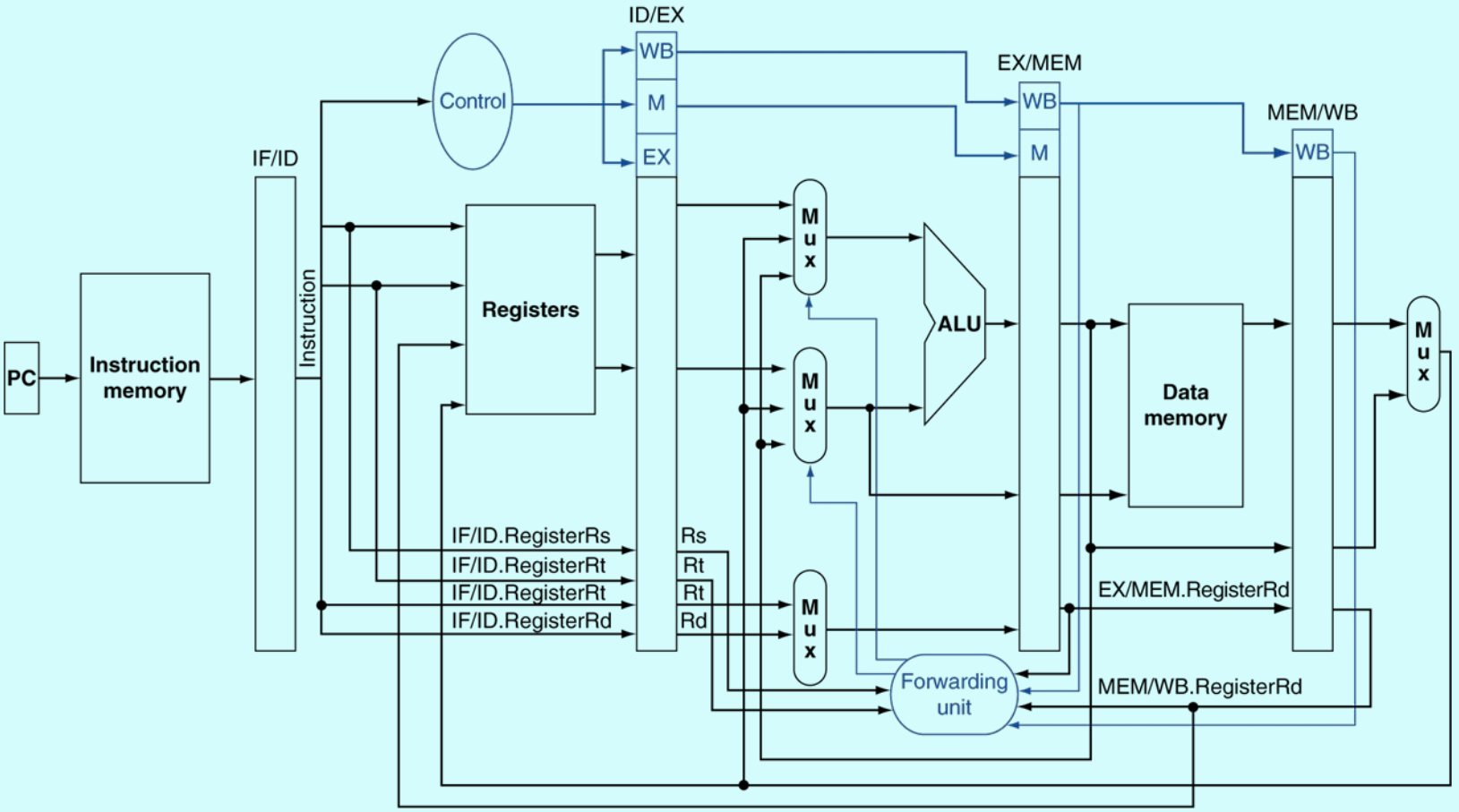
- بنابراین باید تغییراتی در مخاطره‌ی MEM بدهیم

شرایط پیش‌فرستادن (ادامه...)

- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01

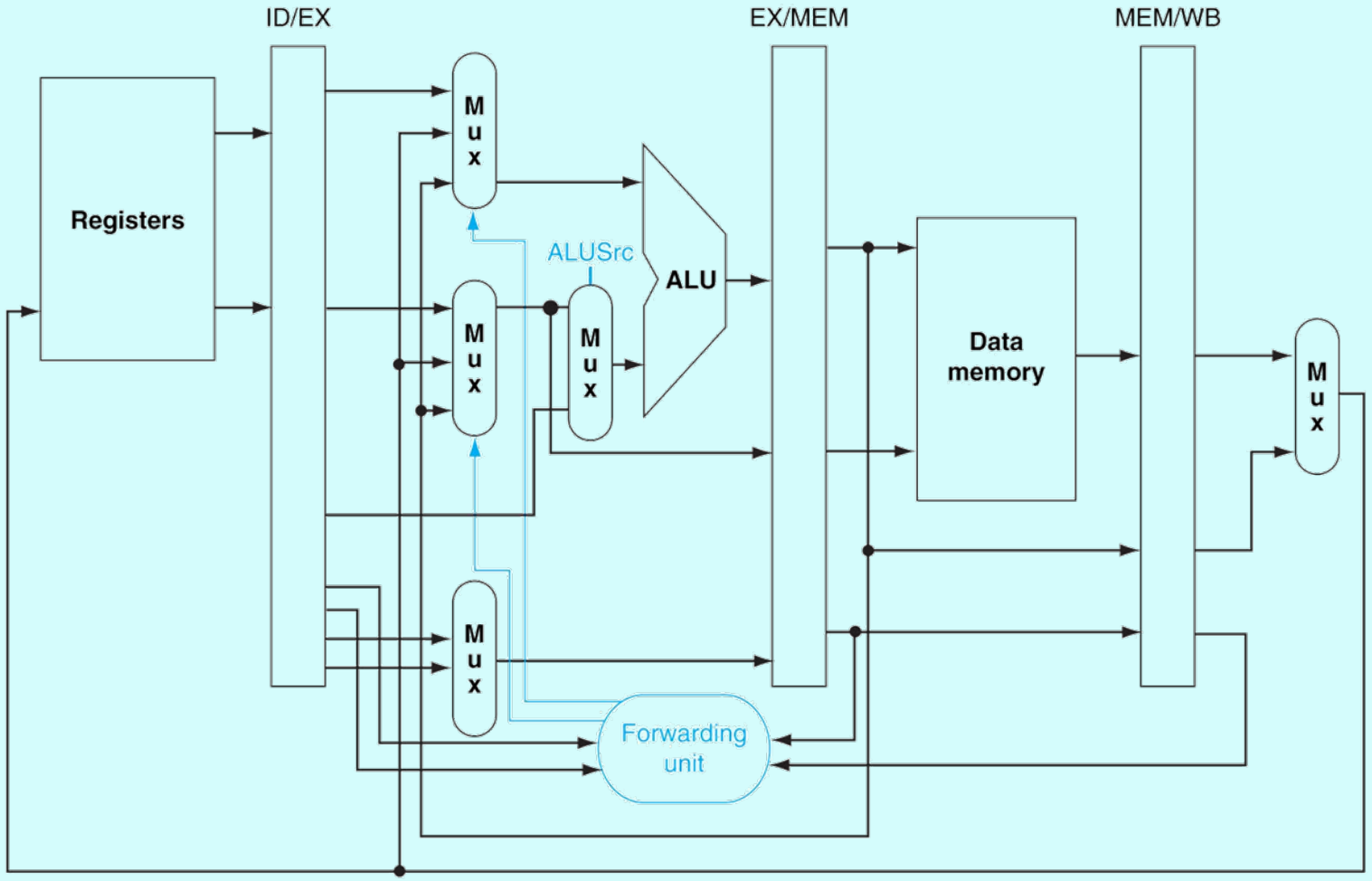


شرایط پیش‌فرستادن (ادامه...)

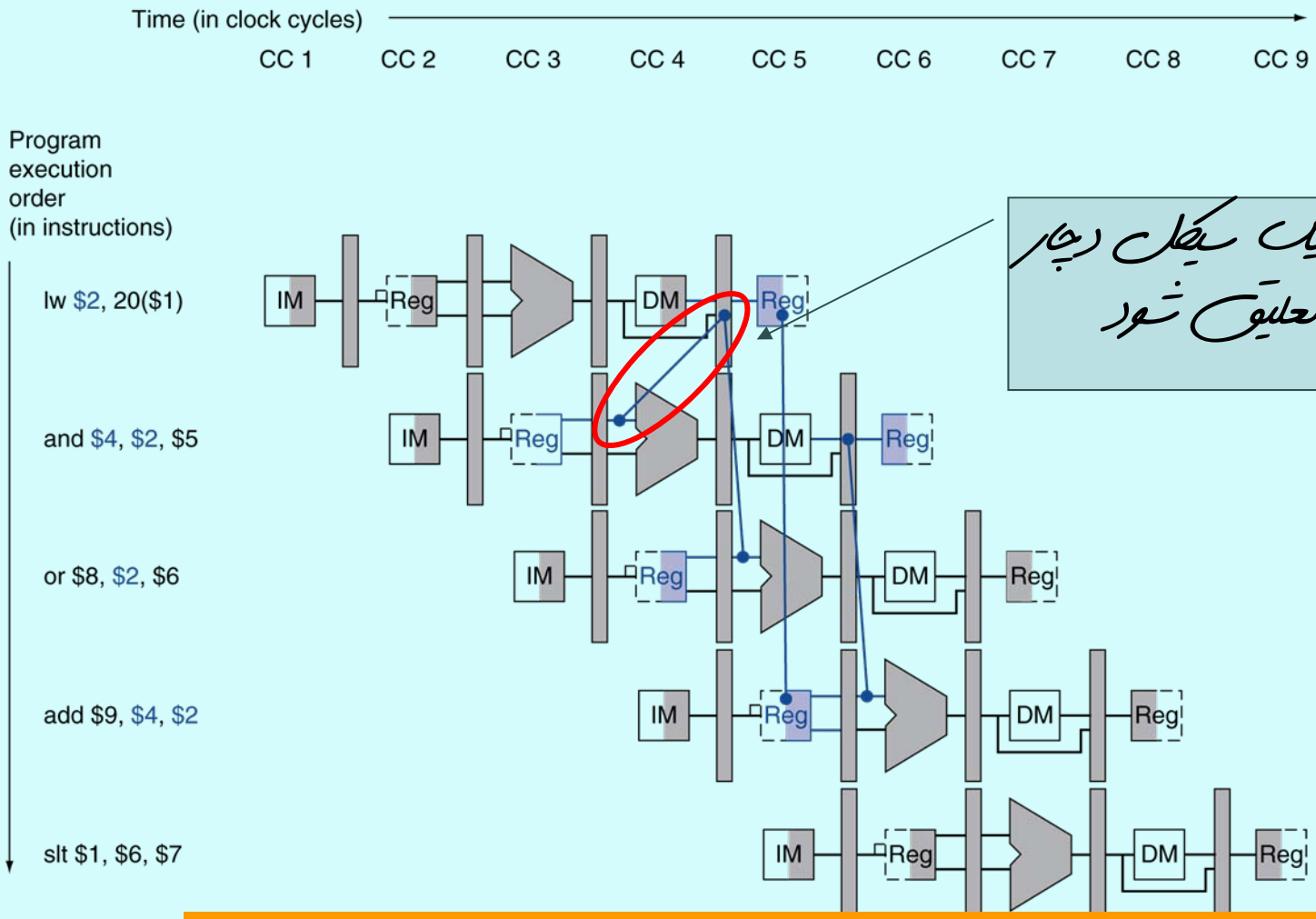


در این شکل بخشی که برای ارسال داده‌ی ثابت به ALU بود، حذف شده است

شرایط پیش‌فرستادن (ادامه...)



پیش فرستادن و تعلیق



بنابراین افزون بر واحد پیش‌فرستادن به واحد تشخیص مخاطره نیز نیاز داریم

واحد تشخیص مخاطره

به نظر شما این واحد در کدام مرحله قرار دارد؟

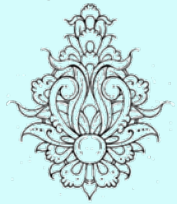
- در ID هنگامی که دستورالعمل کدگشایی می‌شود، وقوع مخاطره بررسی می‌شود.
- به عنوان مثال در استفاده از داده‌ی در حال بارگذاری

– IF/ID.RegisterRs, IF/ID.RegisterRt حاوی شماره‌ی ثبات‌های عملوند ALU می‌باشد.

اگر

ID/EX.MemRead and
((ID/EX.RegisterRt = IF/ID.RegisterRs) or
(ID/EX.RegisterRt = IF/ID.RegisterRt))

تعلیق: یک حباب وارد کن



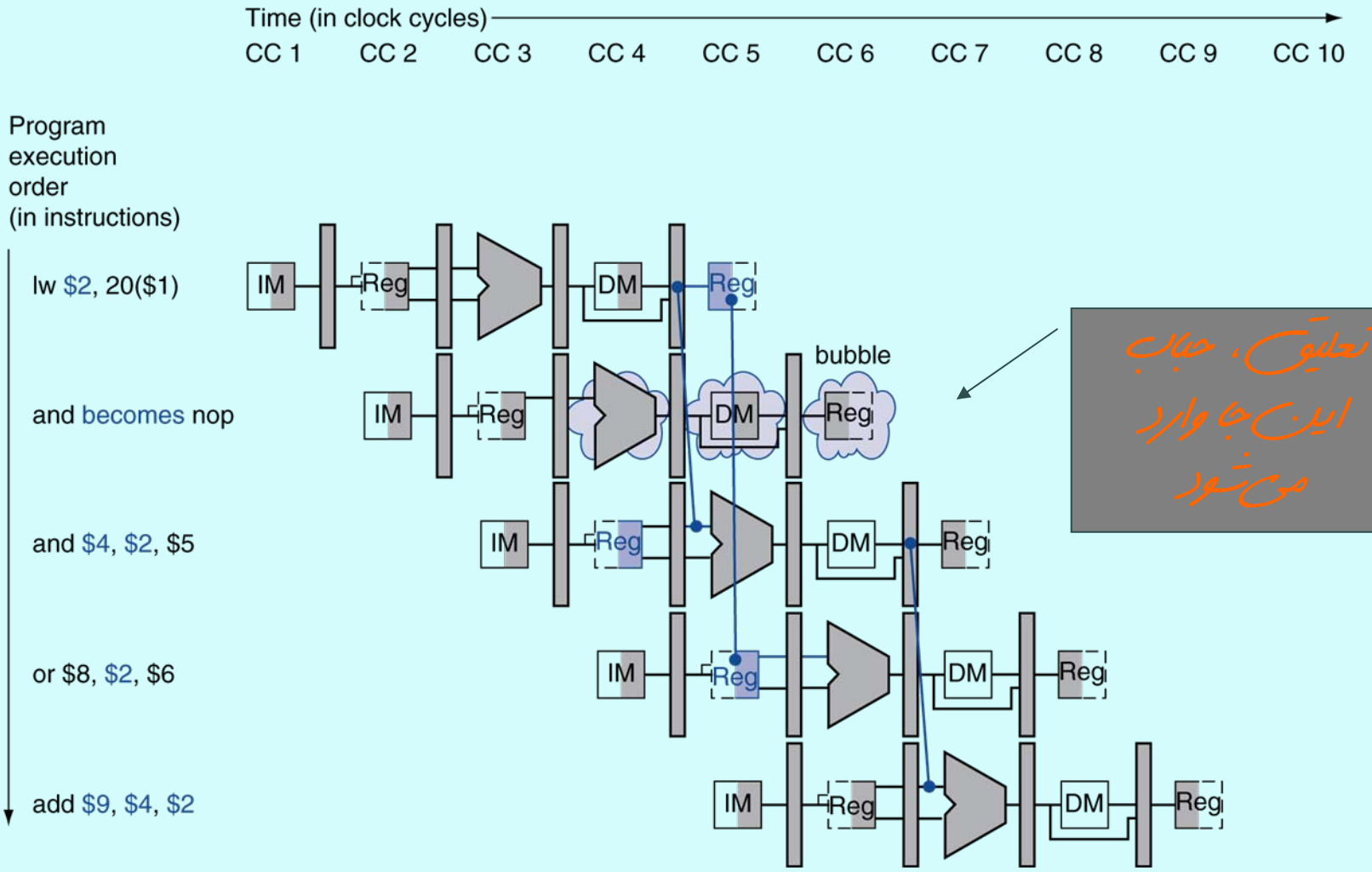
وارد کردن حباب

- در این حال دستور تهی (nop) وارد خط لوله می‌شود.
  - تمام خطوط کنترلی غیر فعال (برابر با ’0‘) می‌شود.
    - به جز سیگنال‌های نوشتن در حافظه، مقدار باقی سیگنال‌ها اهمیتی ندارد.
  - مقدار PC افزایش نمی‌یابد.
  - دستورالعمل دوباره واکشی می‌شود.
  - و دوباره کدگشایی می‌شود.

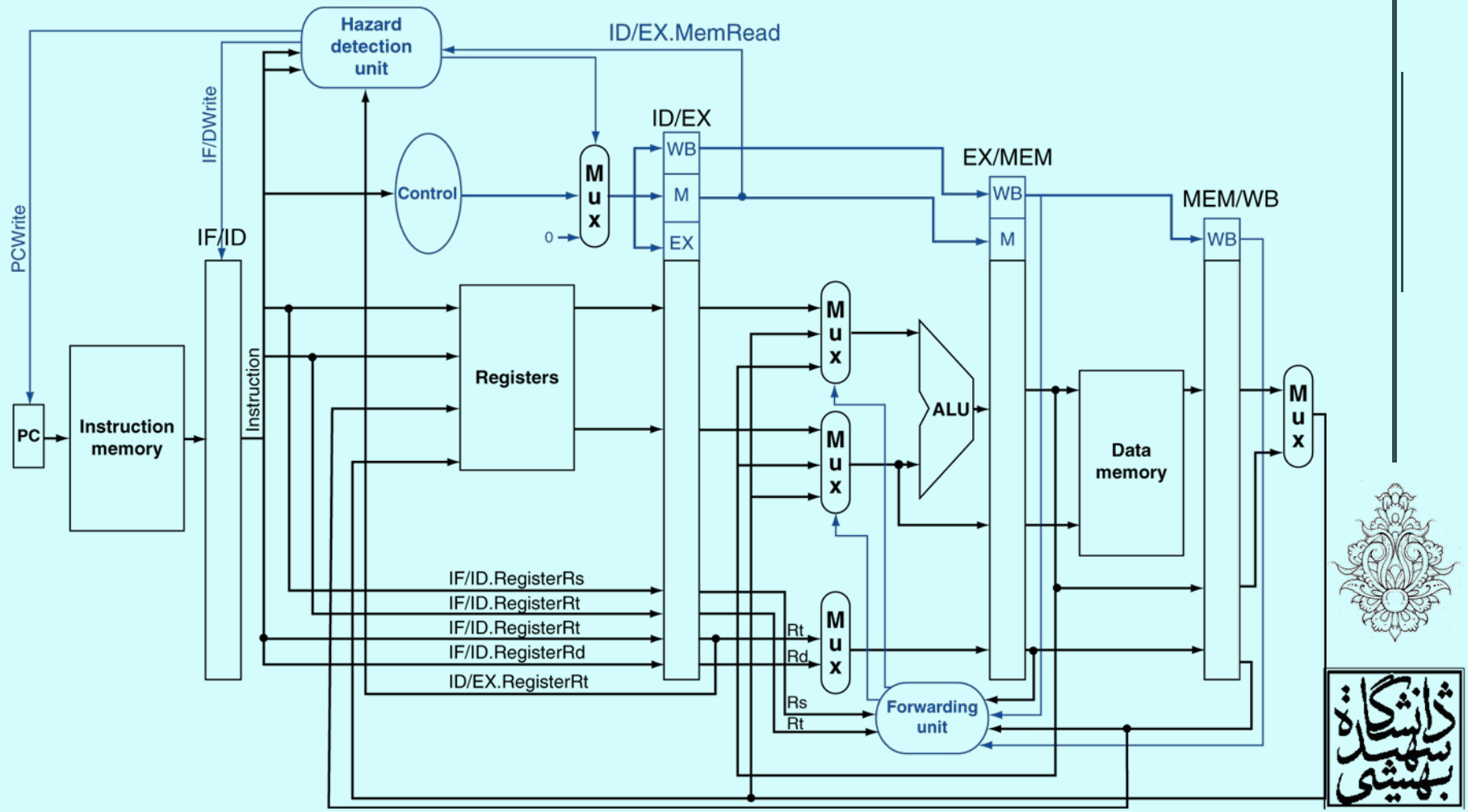
The BIG Picture

هر چند در تشخیص مخاطرات، سخت‌افزار نقش اصلی را ایفا می‌کند لازم است کامپایلر بر نحوه‌ی کار خط لوله مسلط باشد

وارد کردن حباب (ادامه...)

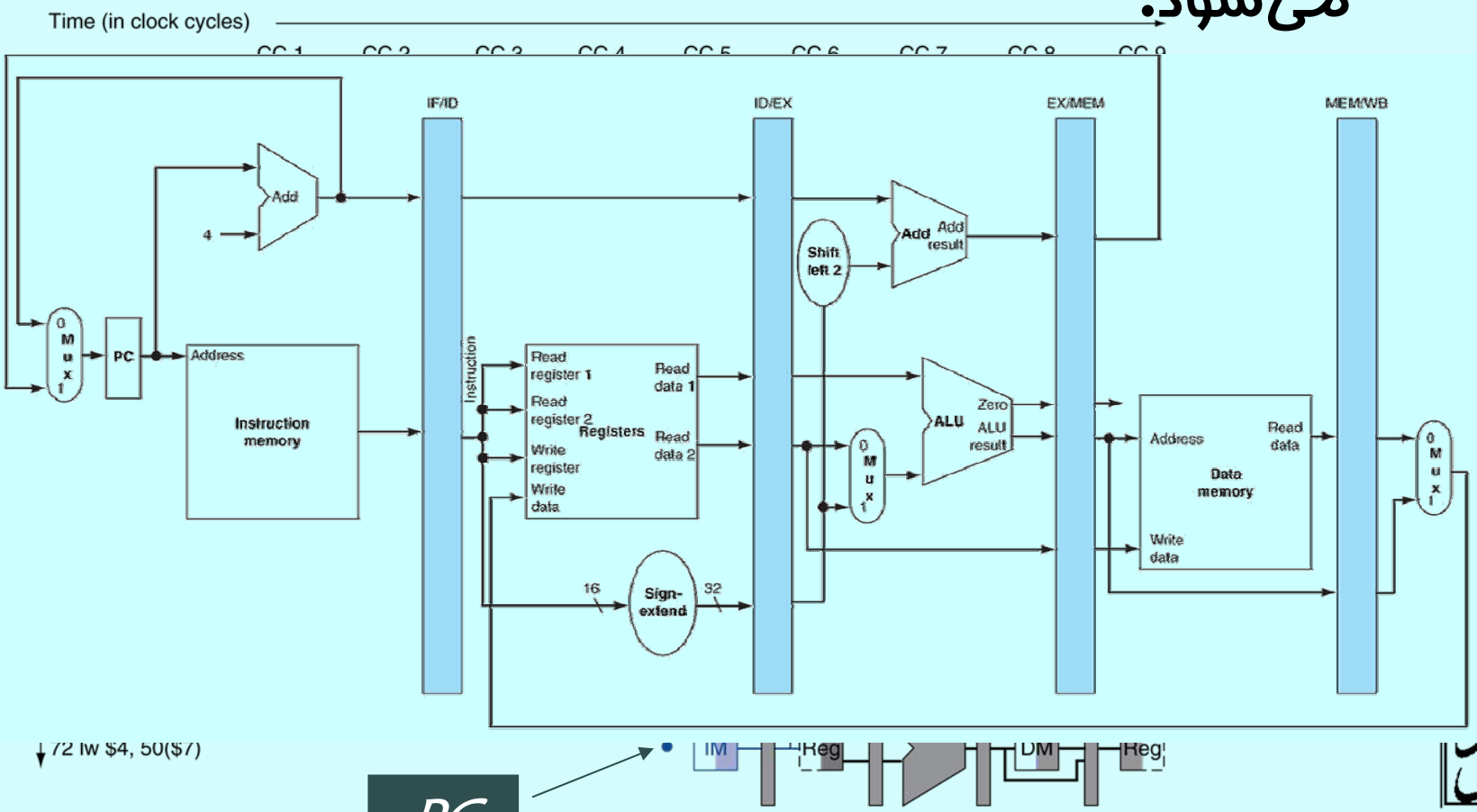


داده‌گذر همراه با مدار تشخیص مخاطره



مخاطرات کنترل

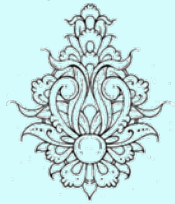
- نتیجه‌ی دستور پرش در مرحله‌ی MEM مشخص می‌شود.



مخاطرات کنترلی (ادامه...)

- ایجاد تعلیق، موجب کندی می‌شود.
- یک راه حل، این است که فرض کنیم هیچ پرشی انجام نمی‌شود.
  - در صورت تحقق، اجرای دستورات واکشی شده، ملغی می‌گردد.
  - برای این کار سیگنال‌های کنترلی غیرفعال می‌شوند.
  - دستورالعمل‌ها از ثبات خط لوله پاک می‌شوند.

flush



مخاطرات کنترلی (ادامه...)

- راه دیگر، کوتاه‌کردن مسیر انجام دستورالعمل‌های پرش شرطیست، (در مرحله‌ی ID) که شامل دو کار است.

Target address adder

  - محاسبه‌ی سریع آدرس محل پرش

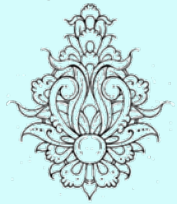
Register comparator

  - محاسبه‌ی سریع شرط

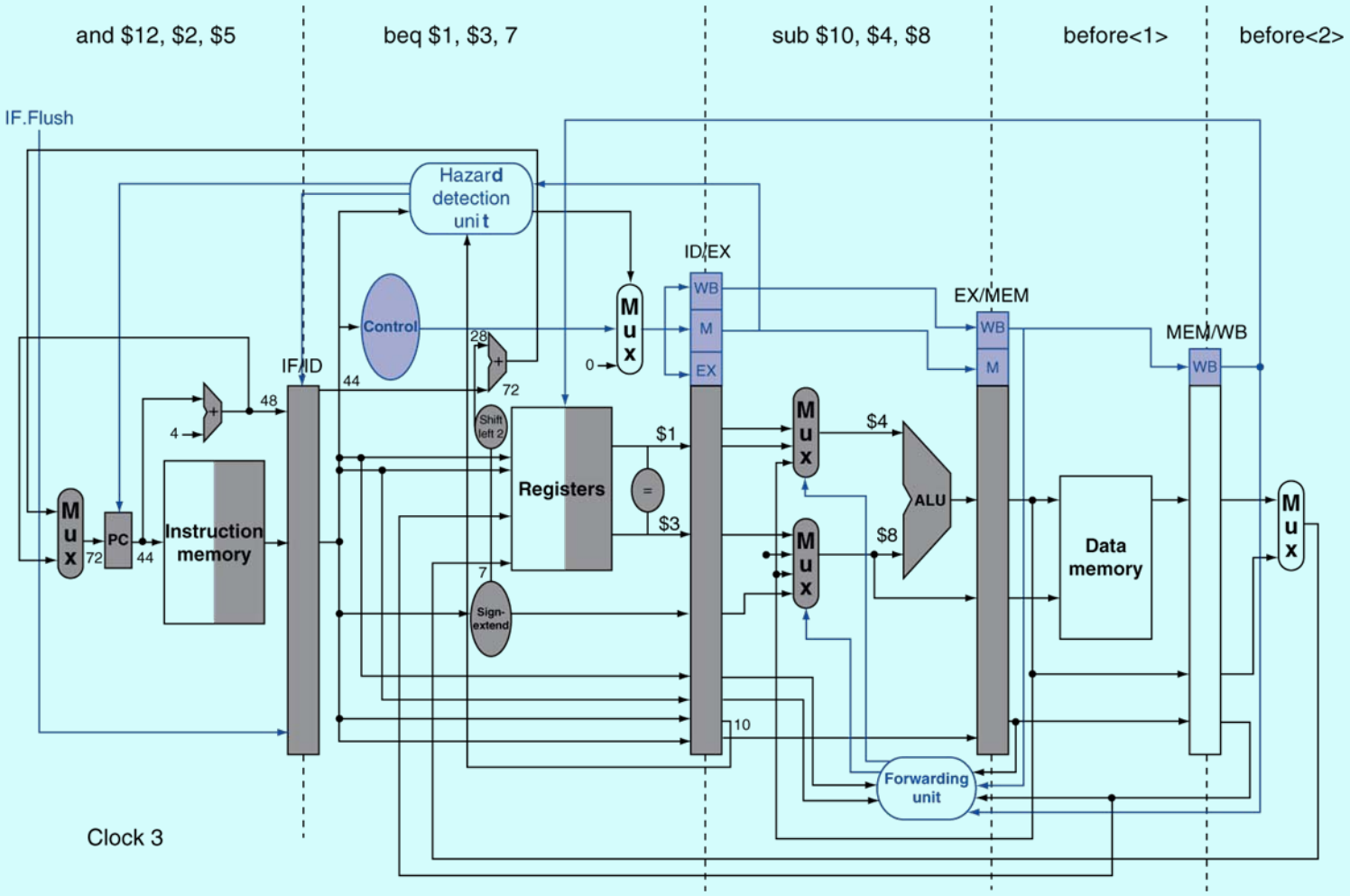
- در صورت انجام چنین کاری می‌باید تغییراتی در مدار تشخیص مخاطره و مدار ایجاد حباب به وجود آورد.

مثال

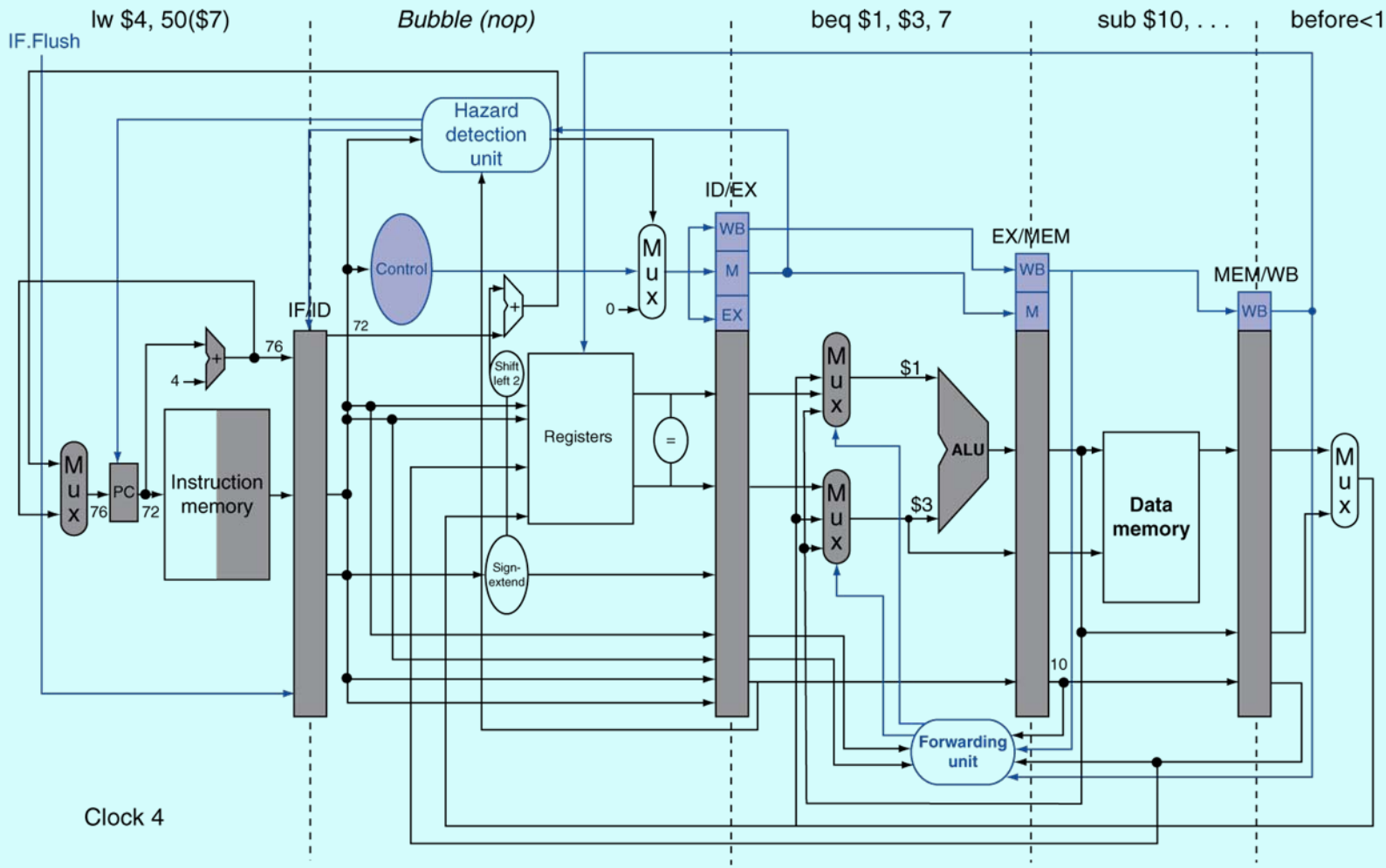
```
36:   sub   $10, $4, $8
40:   beq   $1,  $3, 7
44:   and   $12, $2, $5
48:   or    $13, $2, $6
52:   add   $14, $4, $2
56:   slt   $15, $6, $7
      ...
72:   lw    $4, 50($7)
```



در صورت تحقق شرط

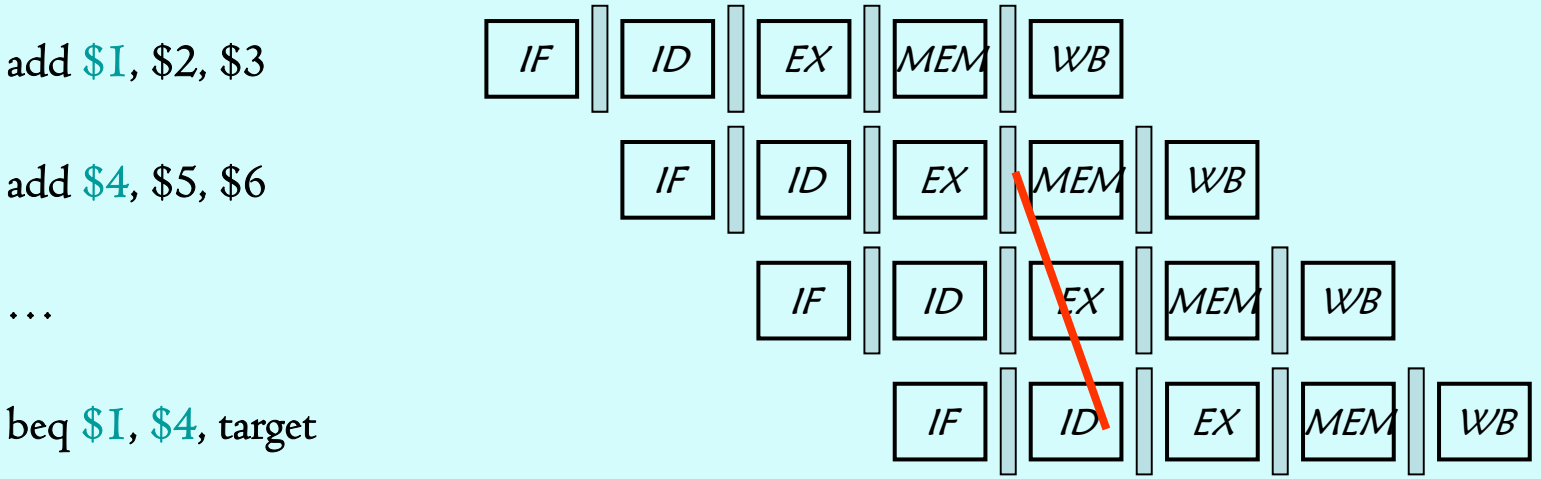


در صورت عدم تحقق شرط (ادامه...)



مخاطره‌ی داده در پرش شرطی

- در صورتی‌که ثبات مقایسه به داده‌ای احتیاج داشته باشد، که هنوز تکمیل نشده، مخاطره‌ی داده رخ می‌دهد.



با پیش‌گرفتن داده قابل حل می‌باشد.

معماری کامپیوتر (۱۱۳۵-۱۱-۱۳)

جلسه‌ی هجدهم

دانشگاه شهید بهشتی

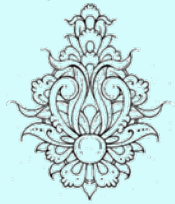
دانشکده‌ی مهندسی برق و کامپیوتر

بهار ۱۳۹۱

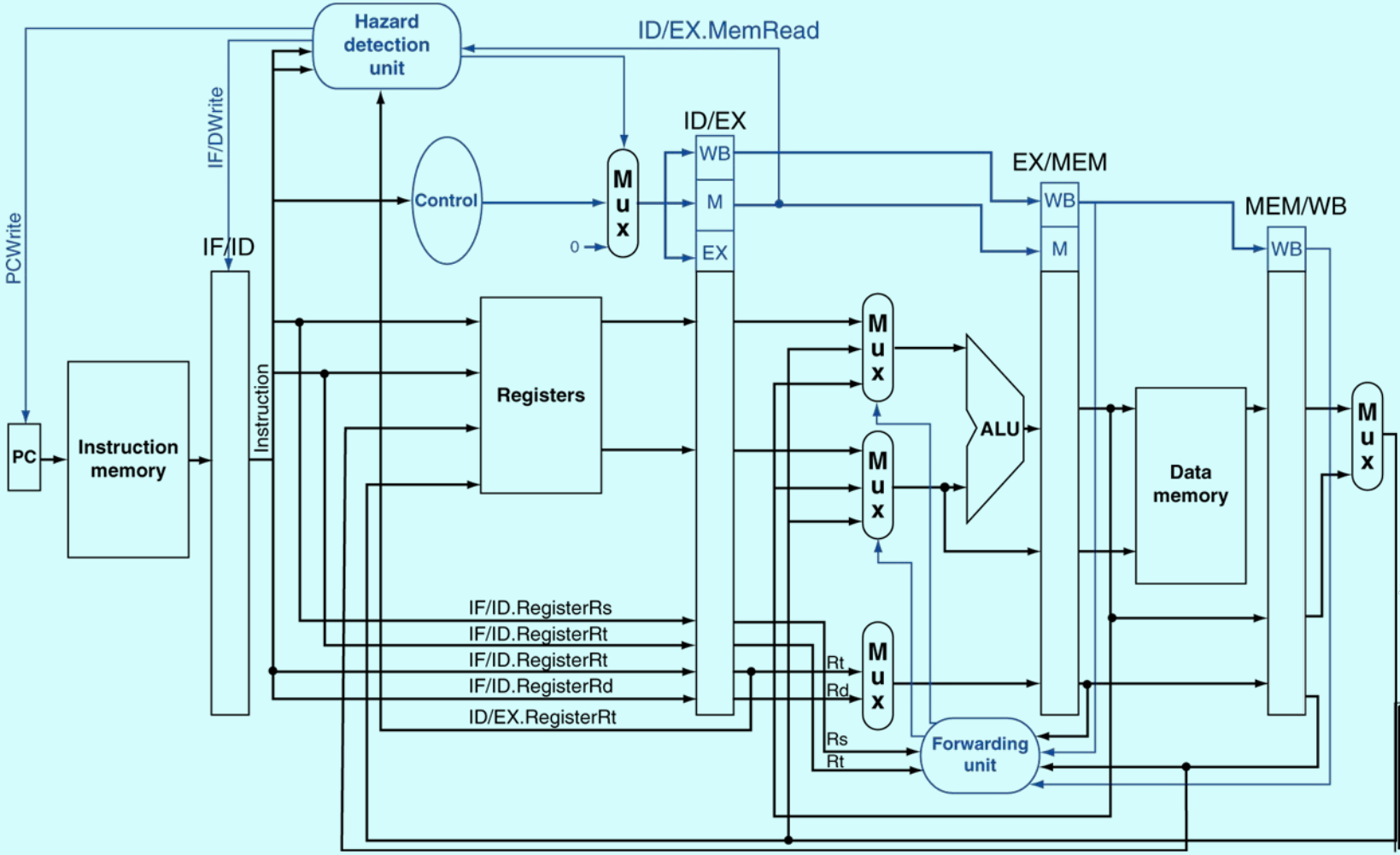
احمد محمودی ازناوه

فهرست مطالب

- مخاطرات کنترلی
  - پیش‌بینی نتیجه‌ی پرش
- استثنائات

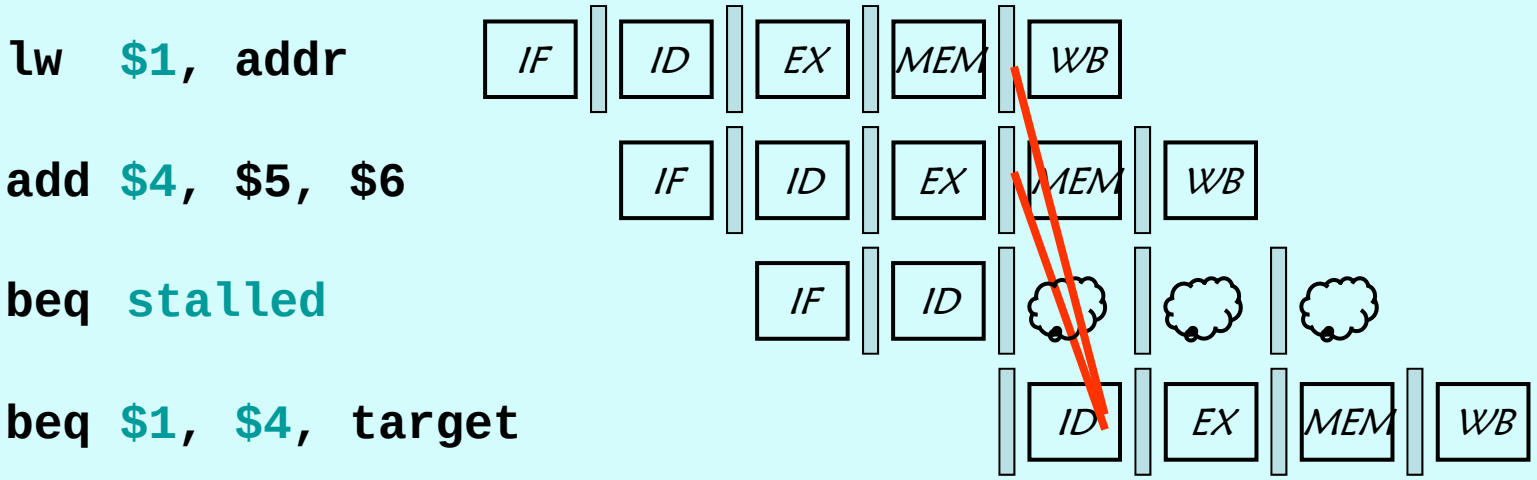


داده‌گذر همراه با مدار تشخیص مخاطره



مخاطره‌ی داده در پرش شرطی (ادامه...)

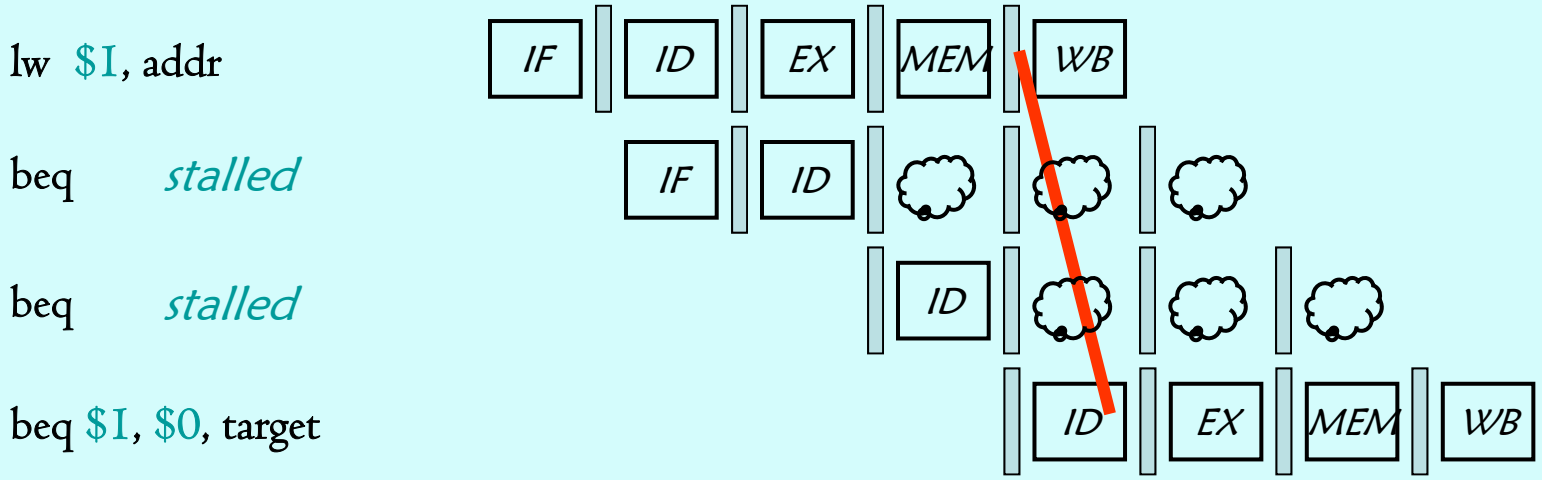
- اگر یکی از ثبات‌های مقایسه در حال بارگذاری از حافظه باشد:



به یک حباب احتیاج خواهیم داشت

مخاطره‌ی داده در پرش شرطی (ادامه...)

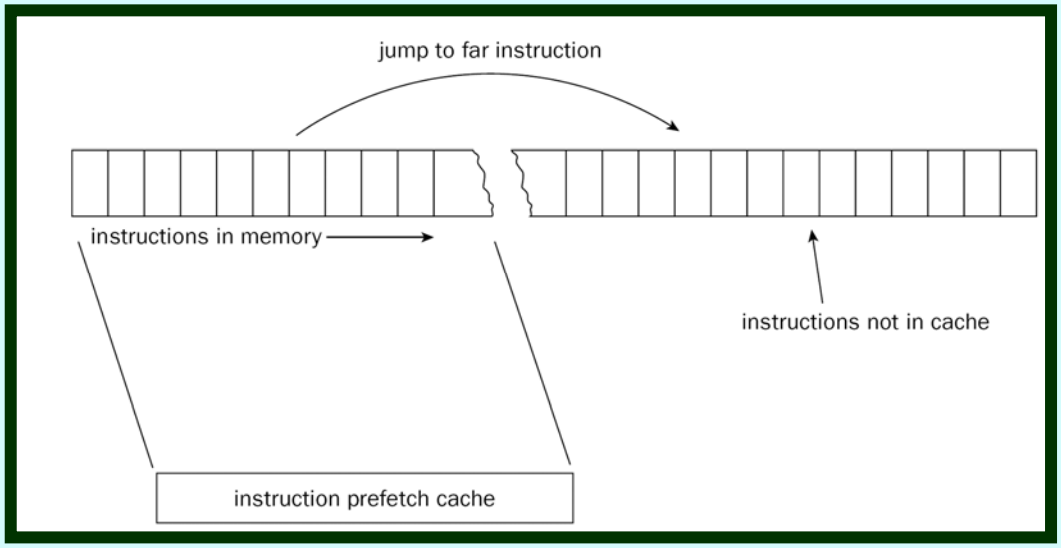
- اگر یکی از ثبات‌های مقایسه در حال بارگذاری از حافظه و دقیقا پیش از دستور پرش باشد:



به دو حباب احتیاج خواهیم داشت

بهینه‌سازی دستورات انشعاب

- دستورات انشعاب، به شدت در کارایی سیستم مؤثر هستند.
- در اکثر پردازنده‌های امروزی برای افزایش کارایی دستورات پیش‌واکشی می‌شود.
- پرش‌های غیرشرطی
  - باعث کاهش کارایی برنامه می‌شوند.



بهینه‌سازی دستورات انشعاب (ادامه...)

- **پرش‌های شرطی**
  - برای تشخیص توالی دستوراتی که هنوز نتیجه‌ی شرط مشخص نیست، از الگوریتم‌های پیش‌بینی استفاده می‌شود.
- **پیش‌بینی نتیجه‌ی شرط به صورت‌های زیر انجام می‌شود:**
  - در پرش رو به عقب، فرض بر آن است که شرط برقرار است.
  - در پرش رو به جلو، فرض بر آنست که شرط برقرار نیست.
  - دستورات پرشی که در اجرای قبلی، انجام شده‌اند، باز هم اجرا خواهند شد.
    - این فرض بر دو فرض قبل غلبه دارد.
    - BTB، برای ره‌گیری آخرین نتیجه مورد استفاده قرار می‌گیرد.

```
movl $100, $ecx
loop1:
addl %cx, %eax
decl %ecx
jns loop1
```

Branch Target Buffer

پیش‌بینی پویا

- در خط لوله‌های عمیق‌تر و در «سوپراسکالرها» زیان ناشی از پرش قابل تحمل نیست.
- در چنین حالاتی از پیش‌بینی پویا استفاده می‌شود.

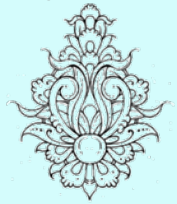
Branch prediction buffer

branch history table

– میان‌گیر پیش‌بینی خطا (جدول تاریخچه)

– نتیجه‌ی آخرین پرش را ذخیره می‌کند.

– در دفعات بعدی مطابق با حالت پیش عمل خواهد کرد.

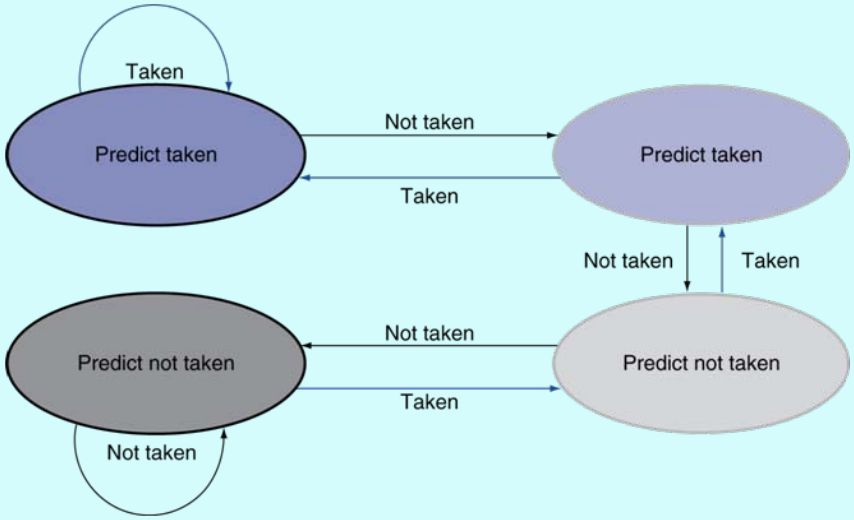


کاستی‌های پیش‌بینی‌کننده‌ی یک بیتی

- **حلقه‌های تودرتو: دو بار نتیجه‌ی پیش‌بینی اشتباه خواهد بود.**

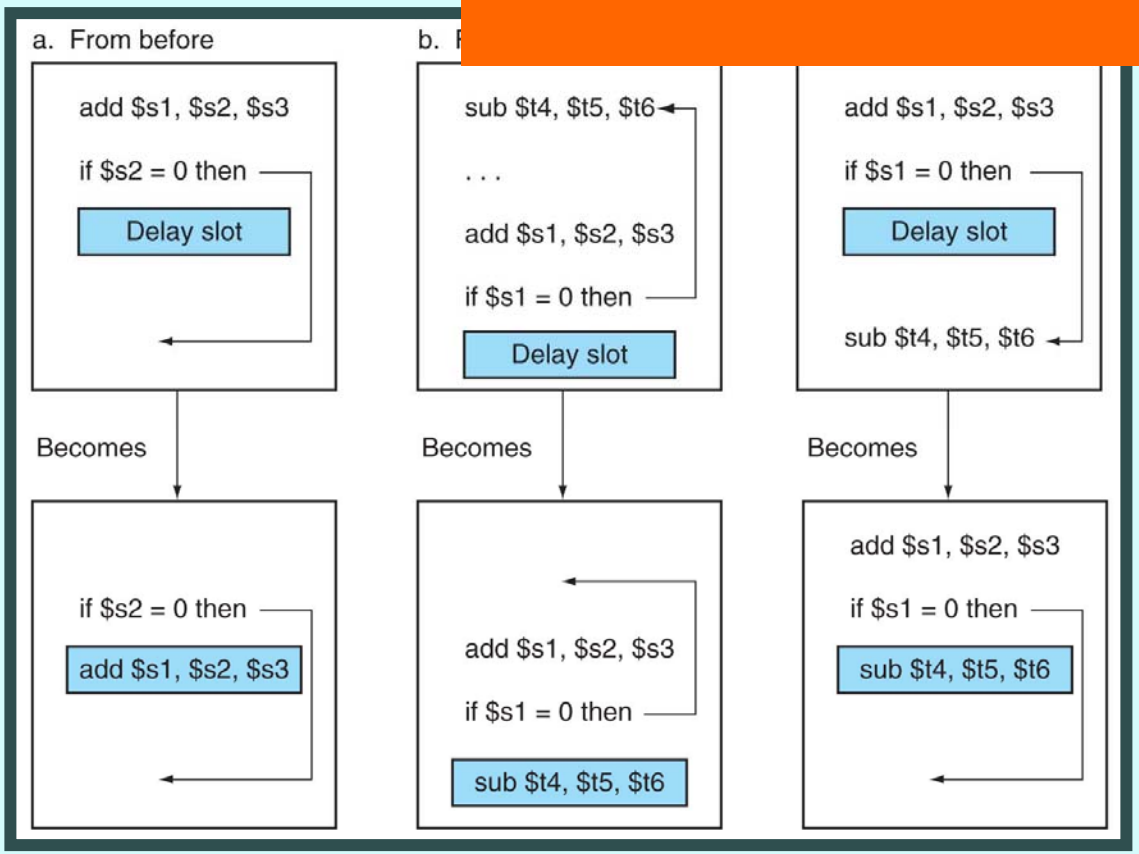
```
outer: …
       …
inner: …
       …
       beq …, …, inner
       …
       beq …, …, outer
```

یکی از راه‌های متداول اختصای دو بیت، برای نگهداری آخرین وضعیت است



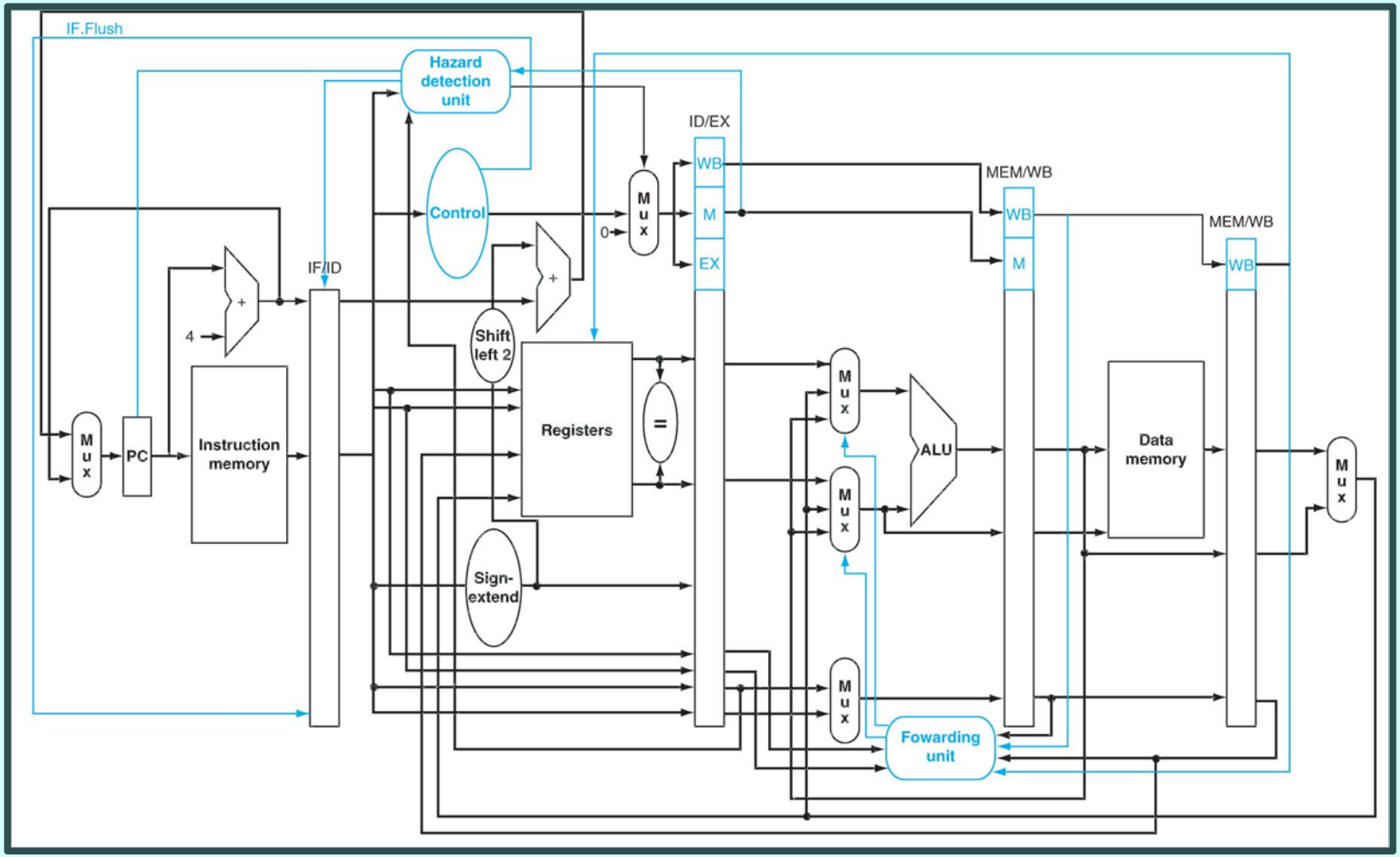
انشعاب تأخیر یافته

راه‌های دیگری نیز برای افزایش کارایی پیشنهاد شده است
یکی از آنها تاخیر پس از پرش شرطی است

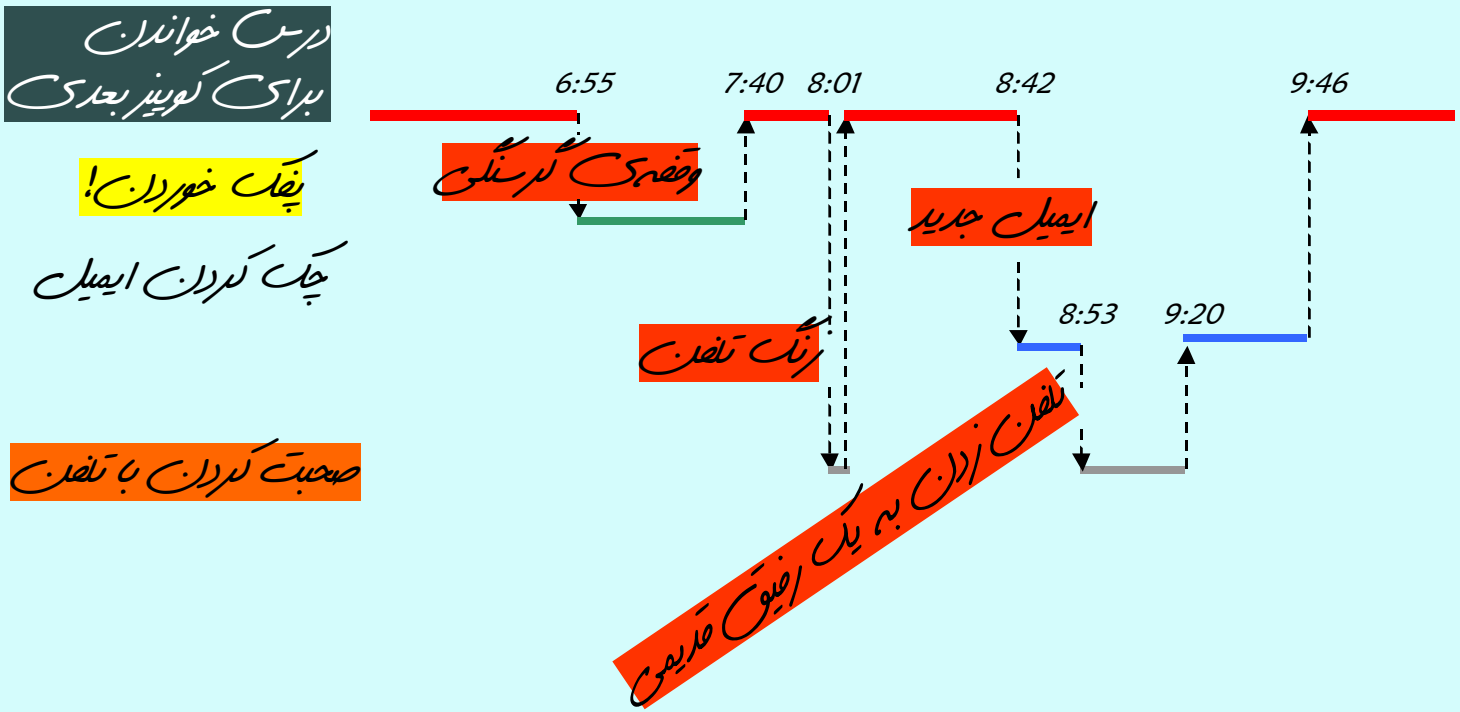


دیگری ذخیره کردن آدرس دستور پرش شرطی است

داده‌گذر نهایی



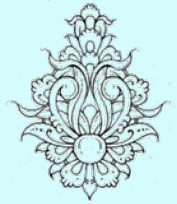
استثنائات


درس خواندن برای کوییز بعدی
پُف خوردن!
چک کردن ایمیل
صحبت کردن با تلفن
6:55
7:40
8:01
8:42
9:46
8:53
9:20
وضعیت گرسنگی
ایمیل جدید
زنگ تلفن
تلفن زدن به یک رفیق قدیمی

استثنائات (ادامه...)

"Unexpected" events

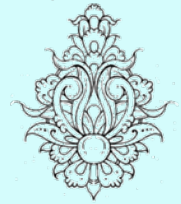
- حوادث پیش‌بینی نشده، می‌تواند روند عادی اجرای برنامه را تغییر دهد.
- در بسیاری از مراجع تمایزی بین وقفه و استثنا قائل نمی‌شوند، در برخی منابع وقفه را حالت کلی‌تر می‌دانند و استثنا را مربوط به عملکرد نادرست .
  - در x86 از واژه‌ی وقفه استفاده شده است.
- در MIPS، از واژه‌ی استثنا (برای هر نوع حادثه با منشأ داخلی و خارجی) استفاده می‌شود. وقفه شامل حوادث با منشأ خارجی می‌باشند.



نمونه‌ای از استثنائات

نوع حادثه	منشأ	واژه‌ی رایج در MIPS
درخواست واحد ورودی/خروجی	خارجی	وقفه
درخواست از سیستم‌عامل از طرف برنامه‌ی کاربر	داخلی	استثنا
رخداد سرریز	داخلی	استثنا
استفاده از دستورالعمل نامشخص	داخلی	استثنا
خرابی سخت‌افزار	هردو	استثنا یا وقفه

برخورد با استثنائات، بدون قربانی کردن کارایی کاری بسیار دشوار است.



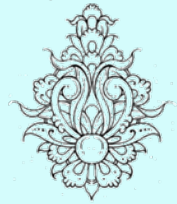
استثنائات (ادامه...)

- عدم توجه کافی به استثنائات در هنگام طراحی واحد کنترل می‌تواند موجب افت کارایی سیستم شود.
- در ادامه به طراحی دو نوع استثناء می‌پردازیم:
  - دستور ناشناخته
  - سرریز

EPC (exception programmer counter)

هنگام بروز استثناء، پردازنده آدرس دستور جاری را در ثبات وقفه ذخیره نموده و کنترل را به بخش خاص از سیستم عامل می‌سپارد

سیستم عامل، در این مواقع واکنشی از پیش تعیین شده انجام خواهد داد. سپس یا اجرای برنامه را خاتمه می‌دهد و یا ادامه‌ی برنامه را اجرا می‌کند.



استثنائات (ادامه...)

- سیستم عامل افزون بر دستوری که موجب رخداد استثنا شده است، می‌باید دلیل آن را نیز بداند.
  - در MIPS از یک ثبات وضعیت (status register) با نام ثبات سبب استفاده می‌شود.

Cause register

0 for undefined opcode, 1 for overflow

- *در صورت بروز وقفه، مقدار PC به 8000 00180 تغییر خواهد کرد. در واقع رویه‌ی رسیدگی‌کننده به وقفه در آن‌جا قرار دارد.*

Exception Handler

- *با بررسی ثبات سبب نوع استثنا و در نتیجه واکنش مناسب تشخیص داده می‌شود.*

استثنائات (ادامه...)

- راه دیگری نیز وجود دارد: **بردار وقفه**
  - که برداری از آدرس‌های رویه‌های رسیدگی‌کننده به وقفه (interrupt handler) می‌باشد. در واقع در این شیوه مشخصا رویه‌ی مورد نظر فراخوانی می‌شود.

Exception type	Exception vector address (in hex)
Undefined instruction	$8000\ 0000_{hex}$
Arithmetic overflow	$8000\ 0180_{hex}$

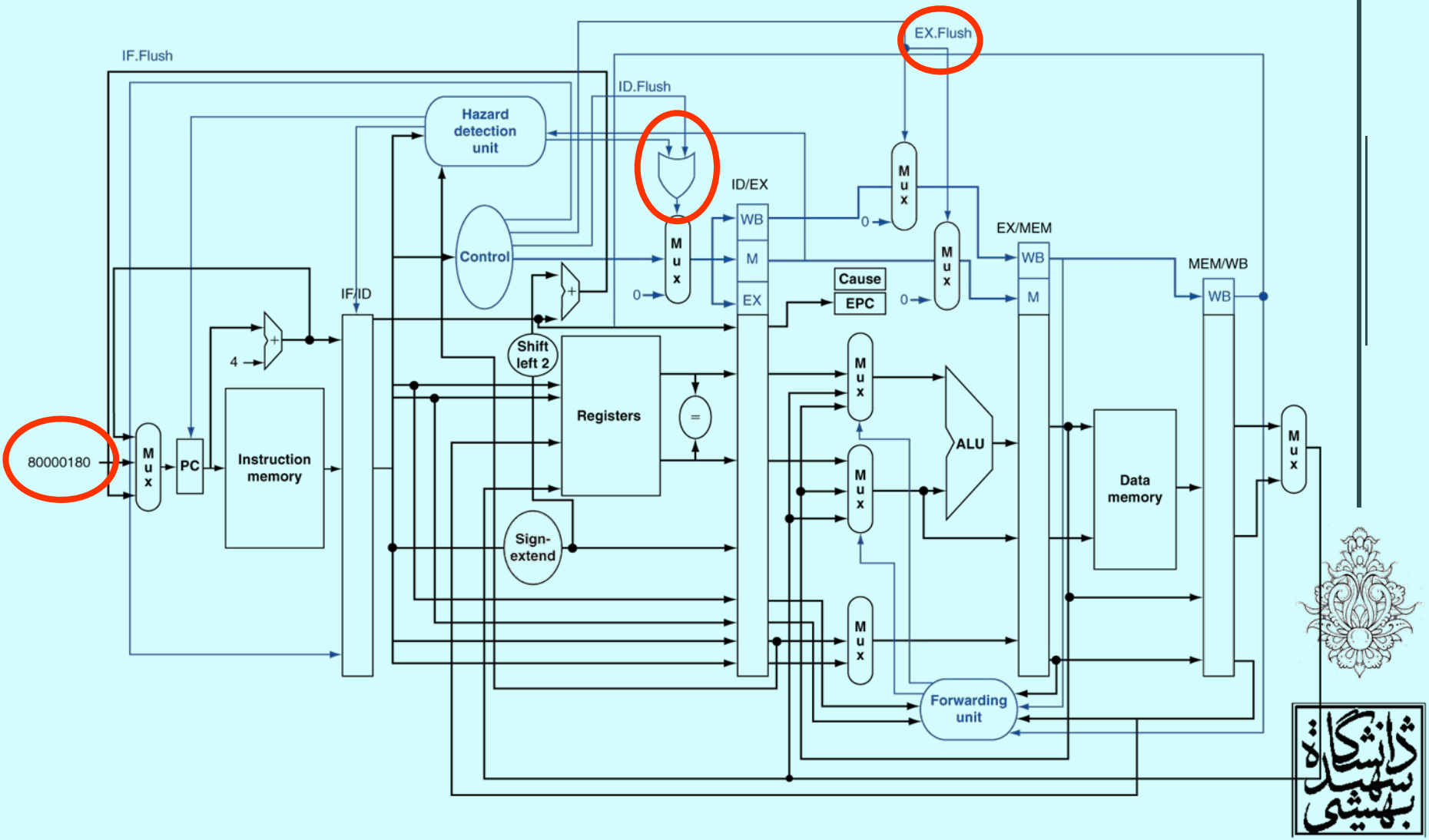
بروز استثنا در پردازنده‌ی مجهز به خط لوله

- در یک سیستم خط لوله یک استثنا، نوعی مخاطره‌ی کنترلی است.

```
add  $1,  $2,  $1
```

- اجرای دستورات پیش از دستور add می‌باید کامل شود.
- دستور add و دستورهای بعدی از خط لوله تخلیه شوند.
- ثبات‌های سبب و EPC مقدار دهی شوند.
  - کنترل به رسیدگی‌کننده به وقفه سپرده شود.

بروز استثنا در پردازنده‌ی مجهز به خط لوله (ادامه...)



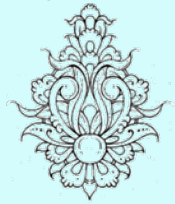
مثال

Exception on add

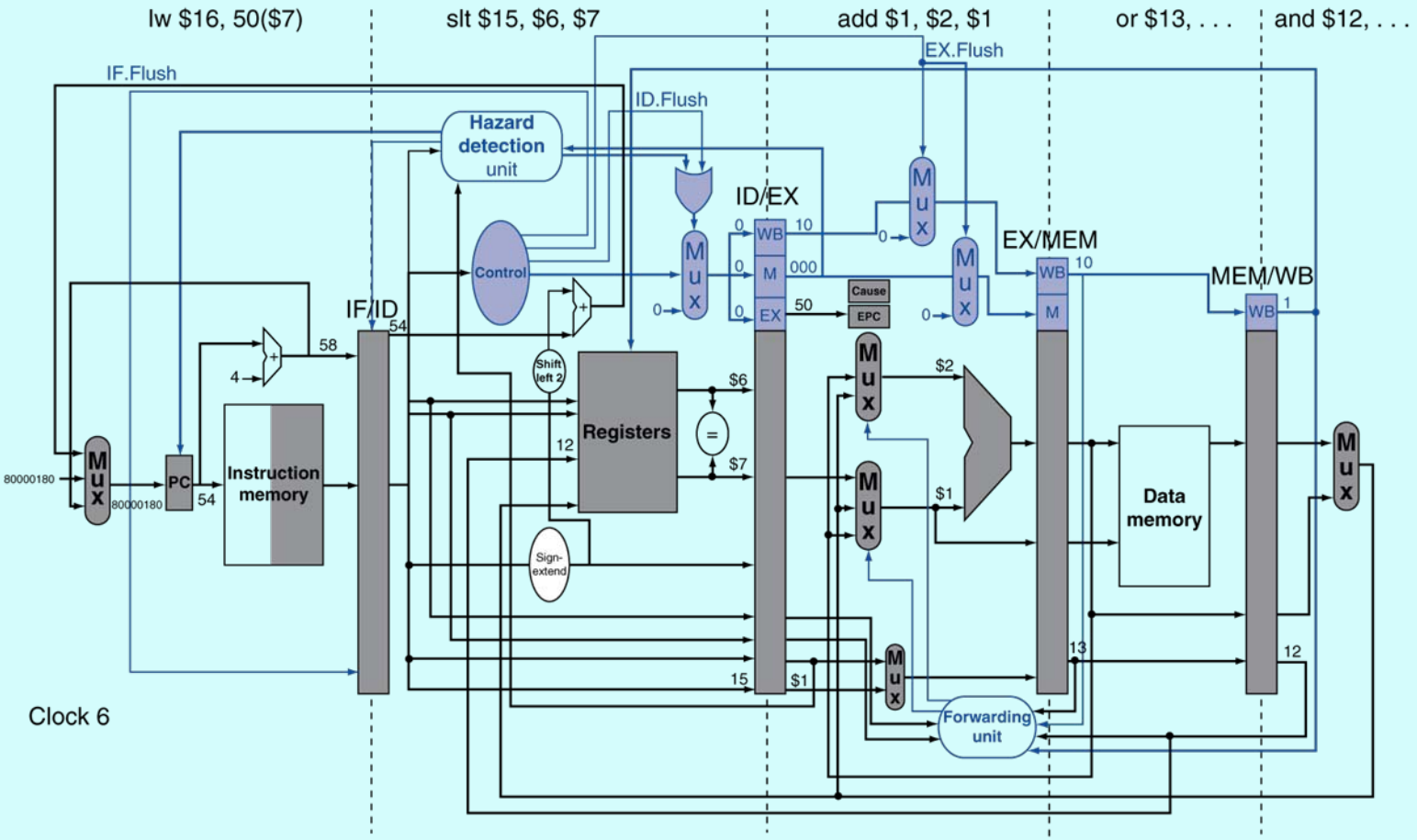
```
40 sub   $11, $2, $4
44 and   $12, $2, $5
48 or    $13, $2, $6
4C add   $1,  $2, $1
50 slt   $15, $6, $7
54 lw    $16, 50($7)
…
```

```
80000180      sw    $25, 1000($0)
80000184      sw    $26, 1004($0)
…
```

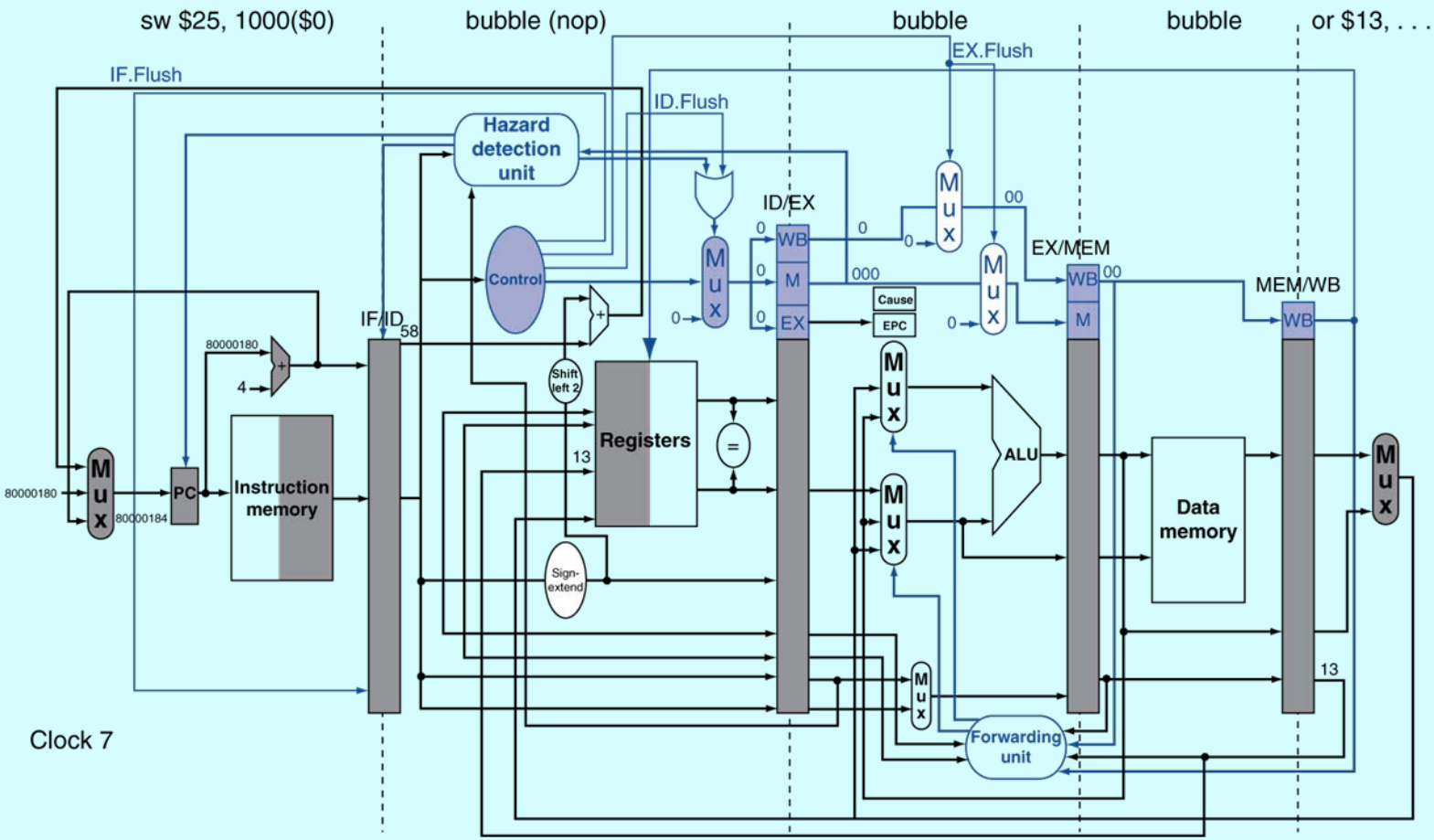
Handler



مثال

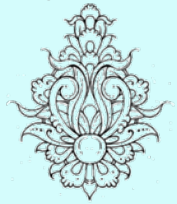

lw $16, 50($7)
slt $15, $6, $7
add $1, $2, $1
or $13, . . .
and $12, . . .
IF.Flush
ID.Flush
EX.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
IF/ID
WB
M
EX
Cause
EPC
Mux
PC
Instruction memory
Registers
Shift left 2
Sign-extend
Data memory
Forwarding unit
80000180
Clock 6

مثال


sw $25, 1000($0)
bubble (nop)
bubble
bubble
or $13, . . .
IF.Flush
ID.Flush
EX.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
IF/ID
WB
M
EX
Cause
EPC
Mux
PC
80000180
80000184
4
Instruction memory
Shift left 2
Registers
Sign-extend
ALU
Data memory
Forwarding unit
13
Clock 7

بروز همزمان چندین استثنا

- **در خط لوله چند دستورالعمل همزمان اجرا می‌شوند.**
  - بنابراین امکان بروز چند استثنا به صورت همزمان وجود دارد.
    - می‌توان به استثنایی که مربوط به دستورالعمل‌های جلوتر هستند، زودتر رسیدگی نمود.



معماری کامپیوتر (۱۳۹۱-۱۱-۱۳)

جلسه‌ی نوردهم

دانشگاه شهید بهشتی

دانشکده‌ی مهندسی برق و کامپیوتر

بهار ۱۳۹۱

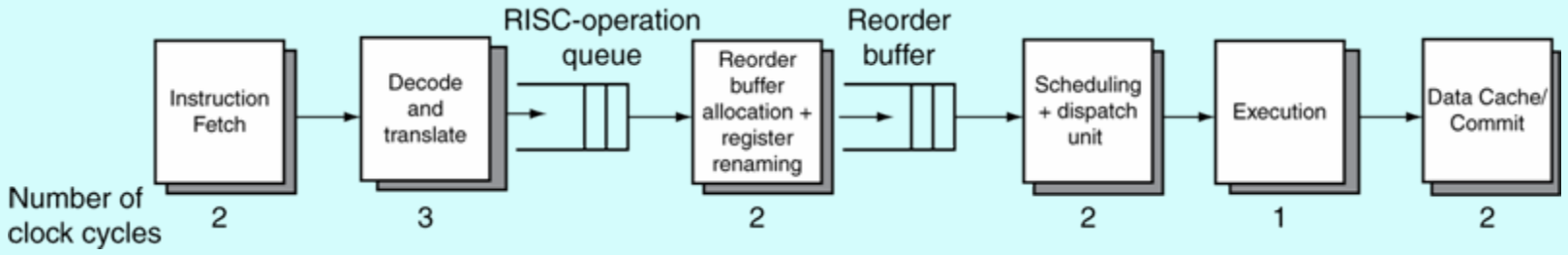
احمد محمودی ازناوه

فهرست مطالب

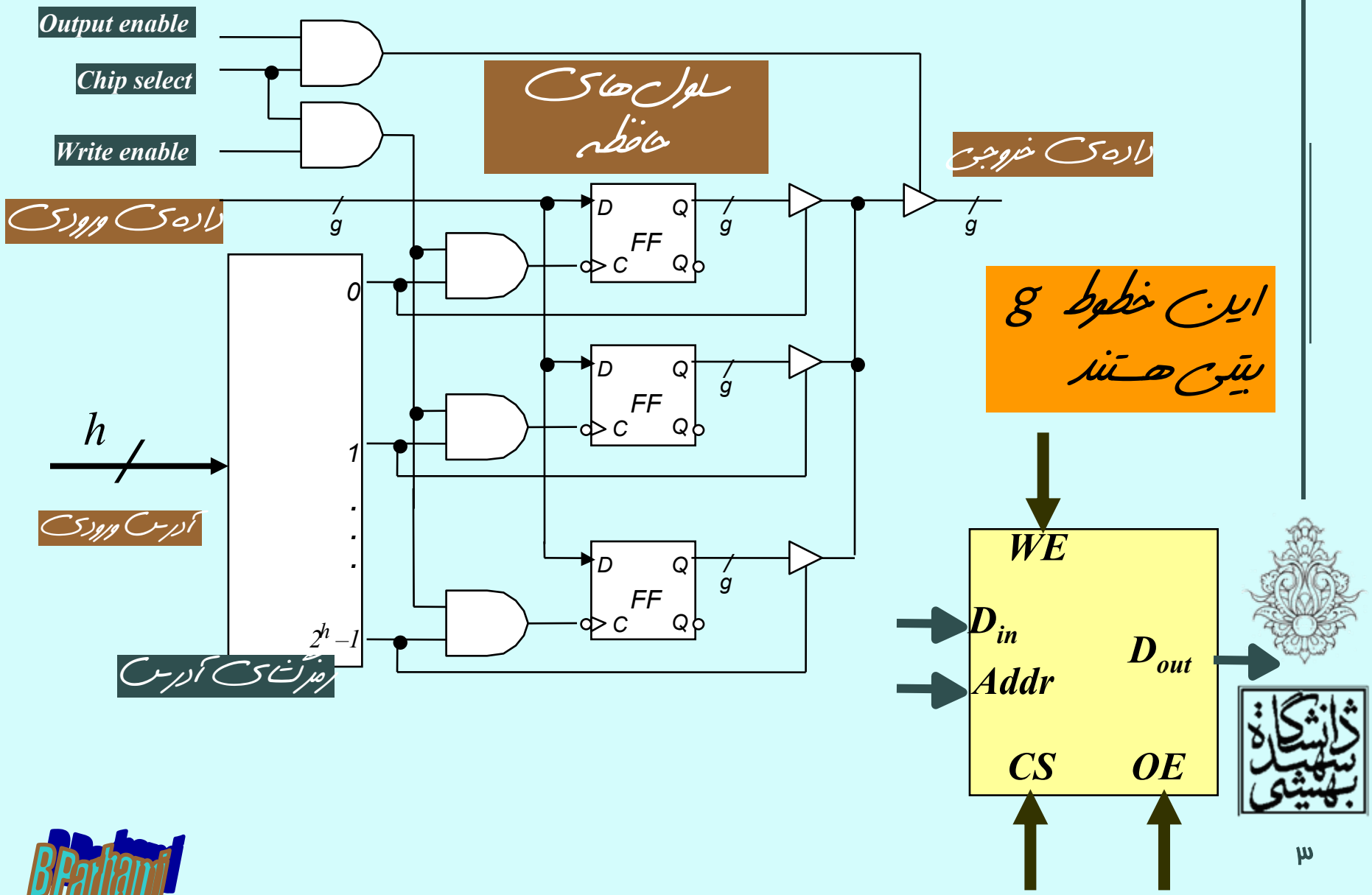
- نگاهی به خط لوله‌ی پردازنده‌های امروزی
- حافظه
  - حافظه‌ی نهان

Opeteron X4

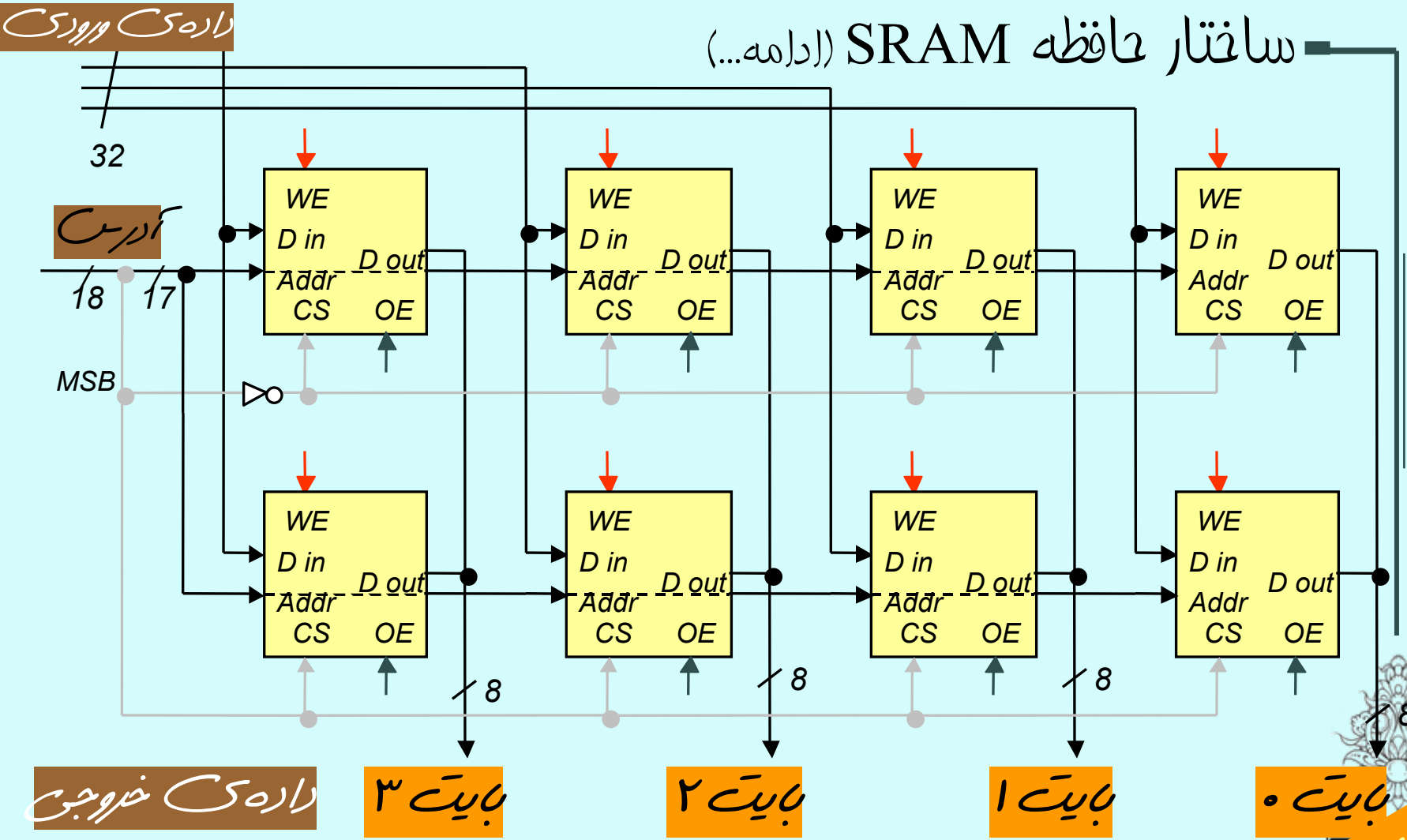
با وجود بحث‌هایی که در مورد خط لوله داشتیم، هنوز مباحث مهمی مطرح نشده است، به همین اندازه بسنده کرده و در پایان نگاهی به ساختار خط لوله در پردازنده‌ی Opeteron X4 خواهیم داشت:



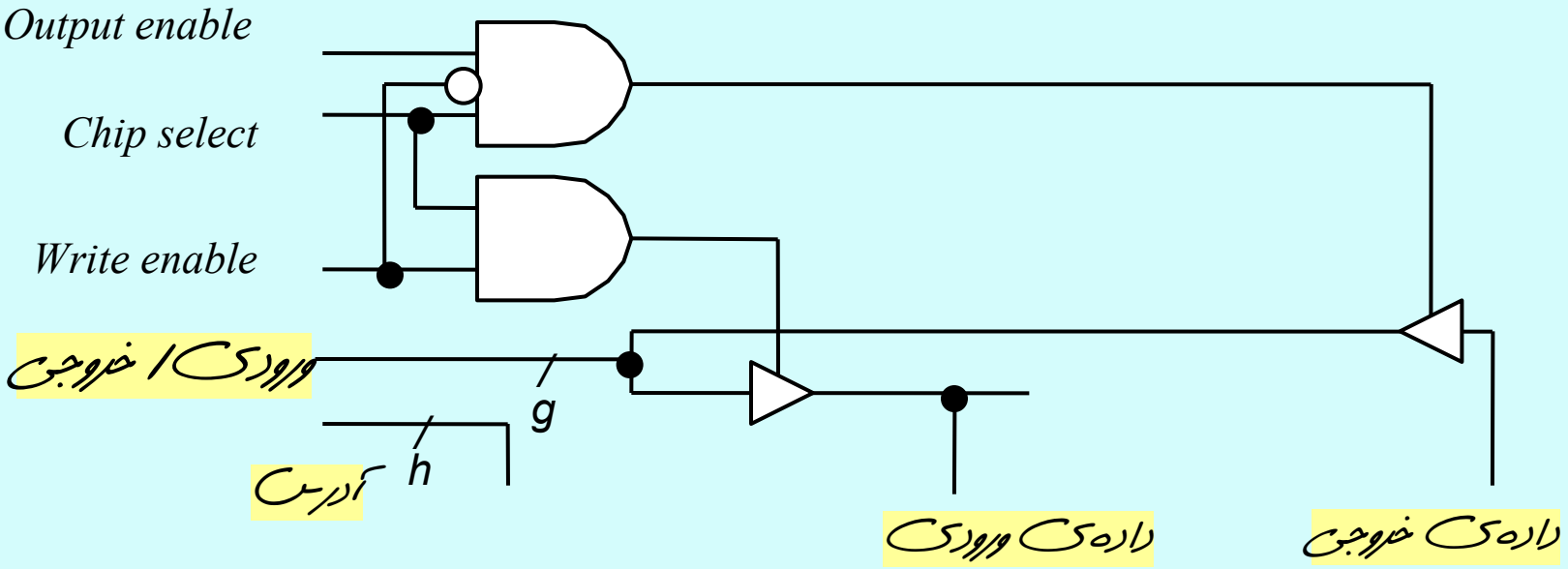
ساختار حافظه SRAM



ساختار حافظه SRAM (ادامه...)

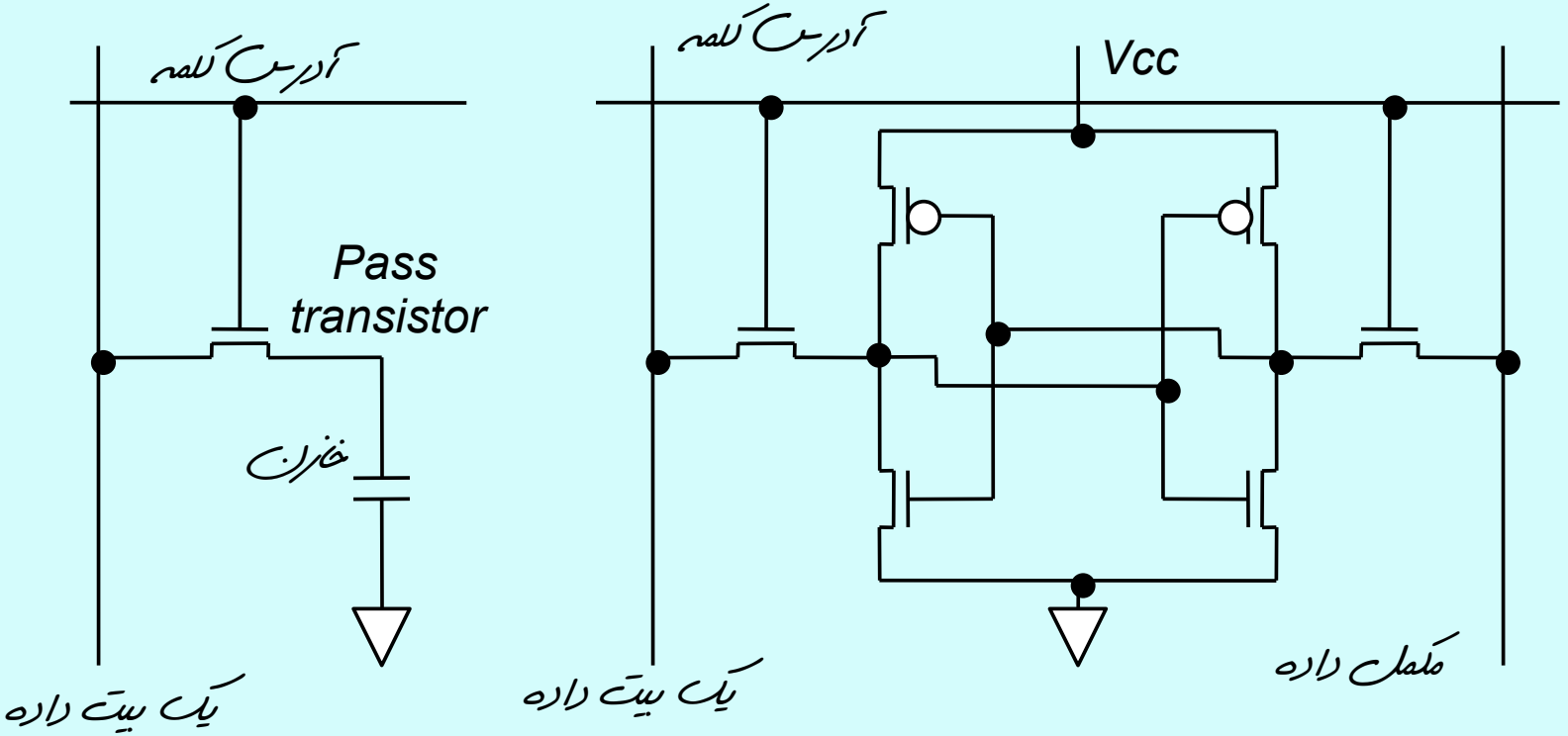


گذرگاه داده دو جهته



هنگامی که داده‌ی ورودی و خروجی به یک گذرگاه مشترک متصل شده باشند،
هنگامی که داده‌ای برای نوشتن در گذرگاه داده قرار می‌گیرد، خروجی سلول‌های حافظه باید غیرفعال شوند

DRAM در برابر SRAM

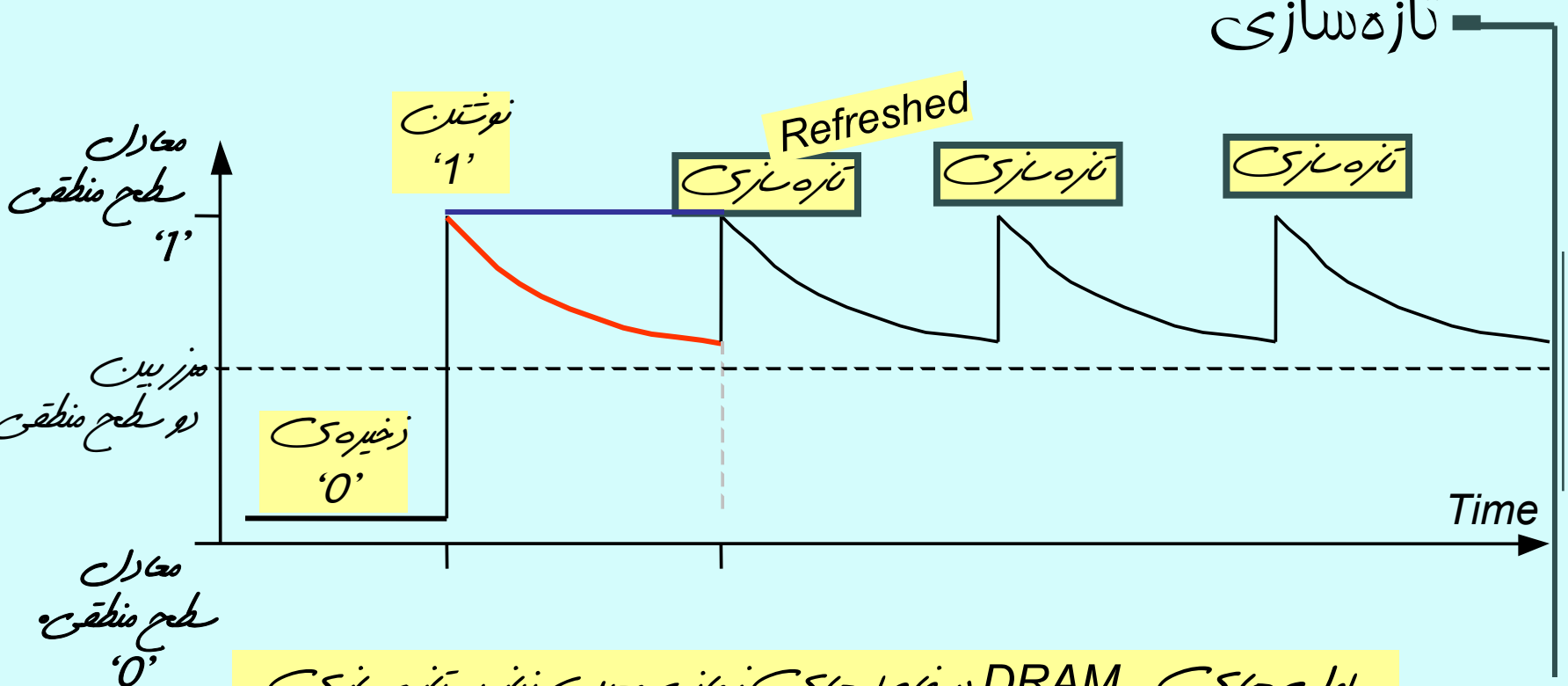


سلول DRAM

سلول SRAM

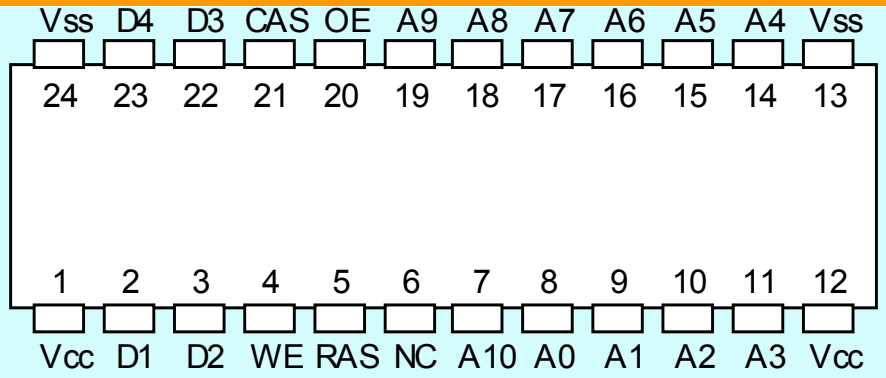
سلول‌های DRAM از سلول‌های SRAM ساده‌تر هستند، در نتیجه با این سلول می‌توان حافظه‌هایی با گنجایش بالاتر و در عین حال ارزان‌تر ساخت

تازه‌سازی



سلول‌های DRAM در فاصله‌های زمانی معین نیاز به تازه‌سازی محتوای خود دارند

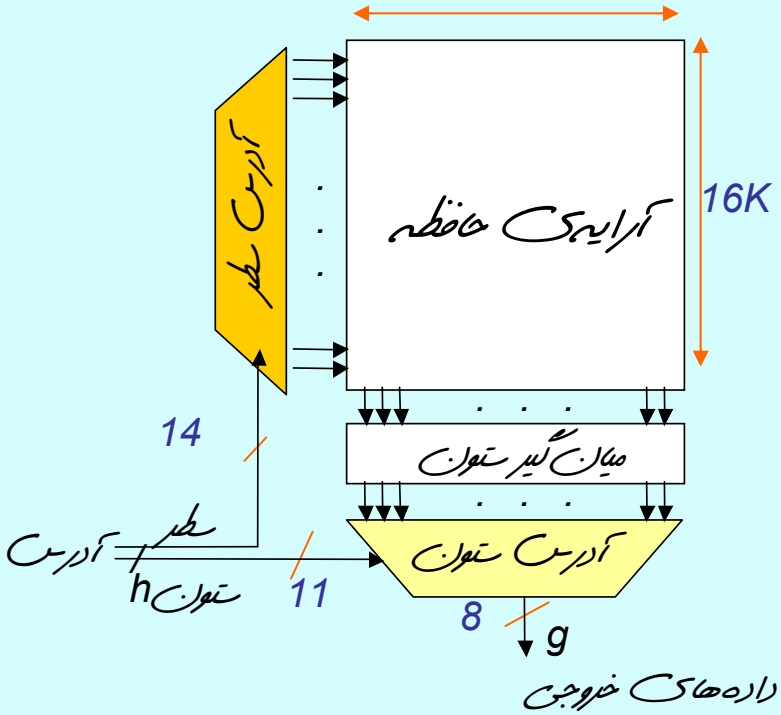
به نظر شما نیاز به تازه‌سازی چه مشکلی ایجاد می‌کند؟



Legend:

A*i*	Address bit *i*
CAS	Column address strobe
D*j*	Data bit *j*
NC	No connection
OE	Output enable
RAS	Row address strobe
WE	Write enable

یک DRAM با گنجایش 256 Mb مفروض است. این حافظه که به صورت $32M \times 8$ قابل دسترسی است، به صورت آرایه‌ای $16K \times 16K$ می‌باشد. هر سطر دست‌کم هر 50 ms **باید** تازه‌سازی شود؛ این کار 100ns زمان می‌برد، چند درصد از پهنای باند هدر می‌رود؟

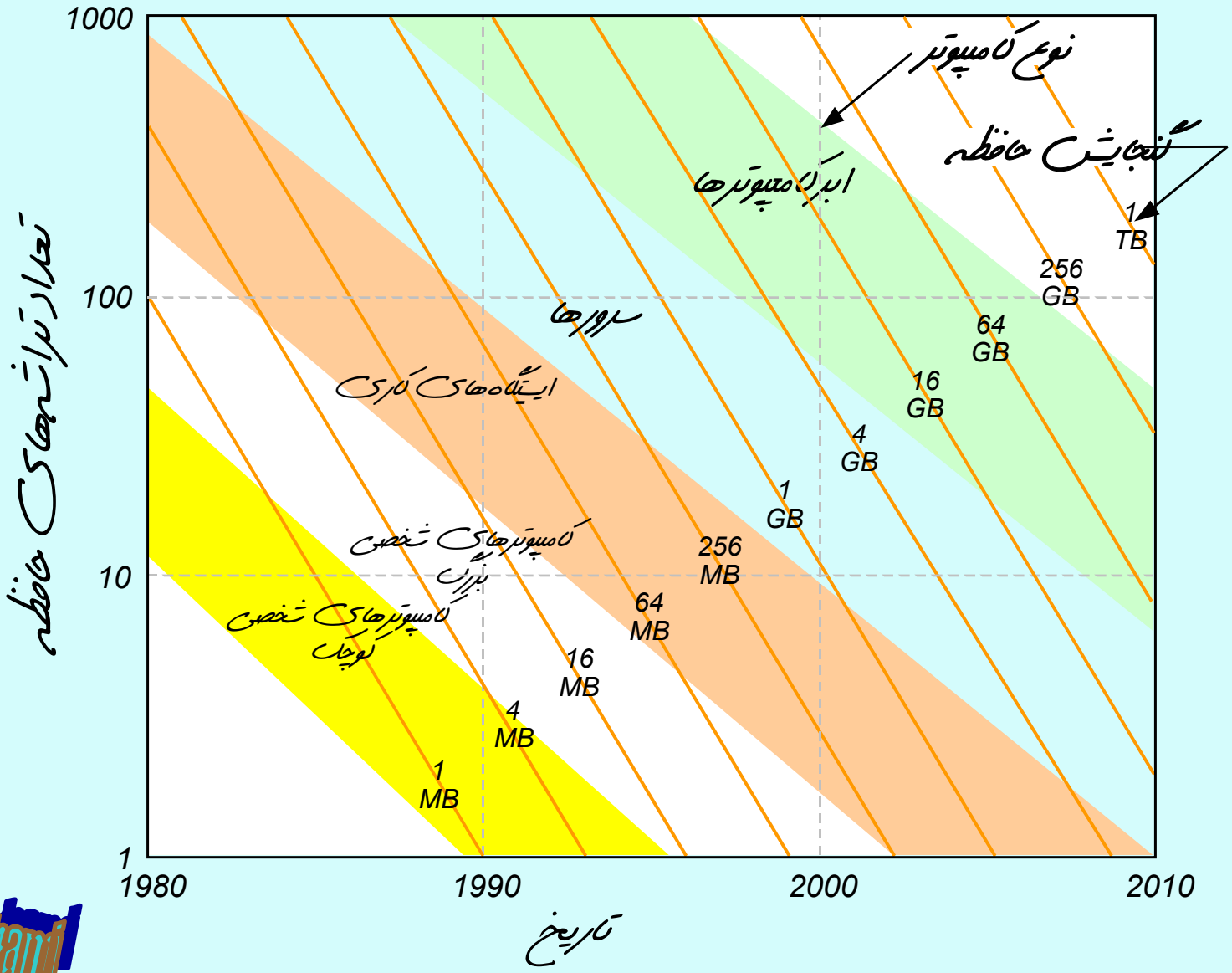


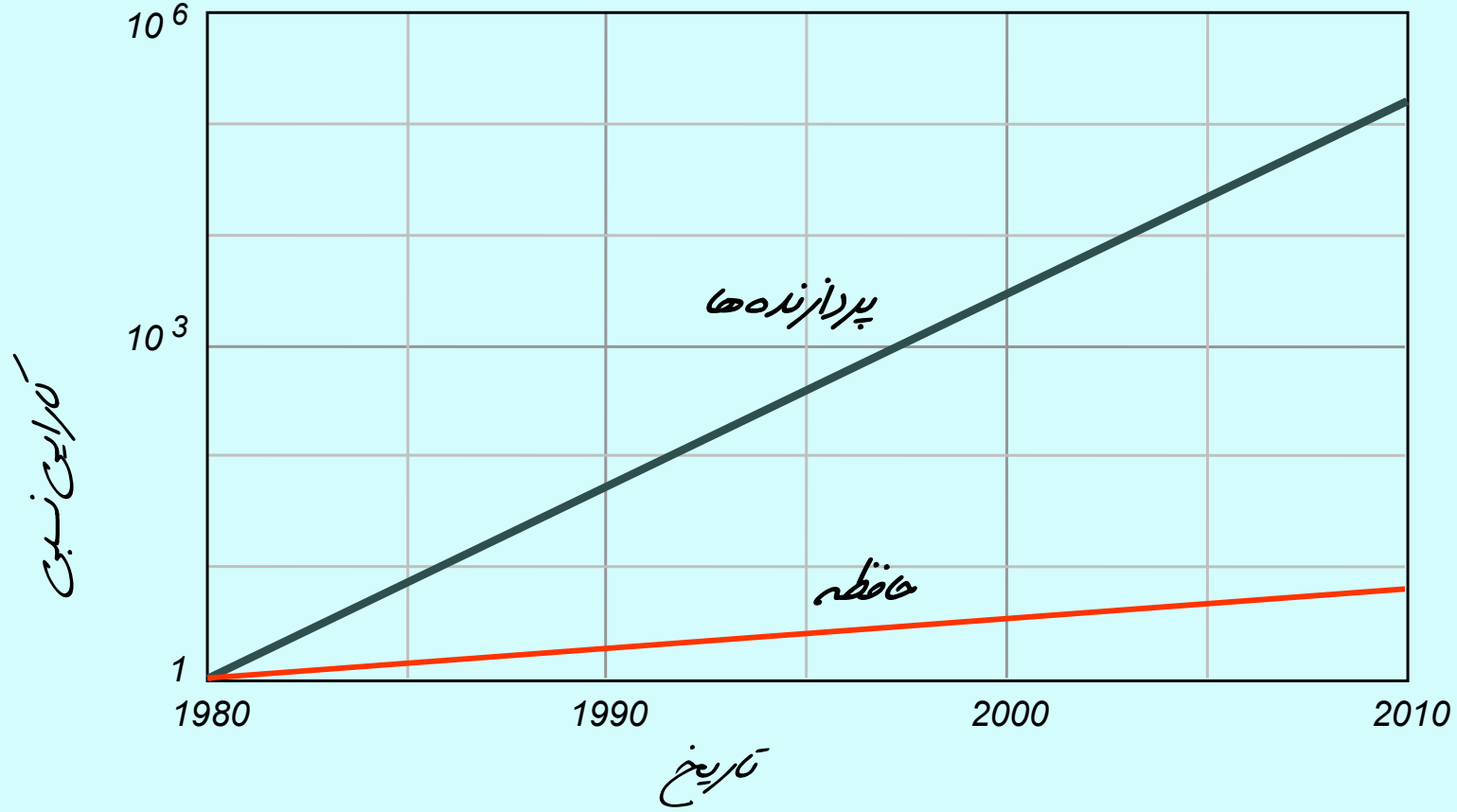
راه حل:

16 $\times 1024 \times 100$ ns = 1.64 ms.

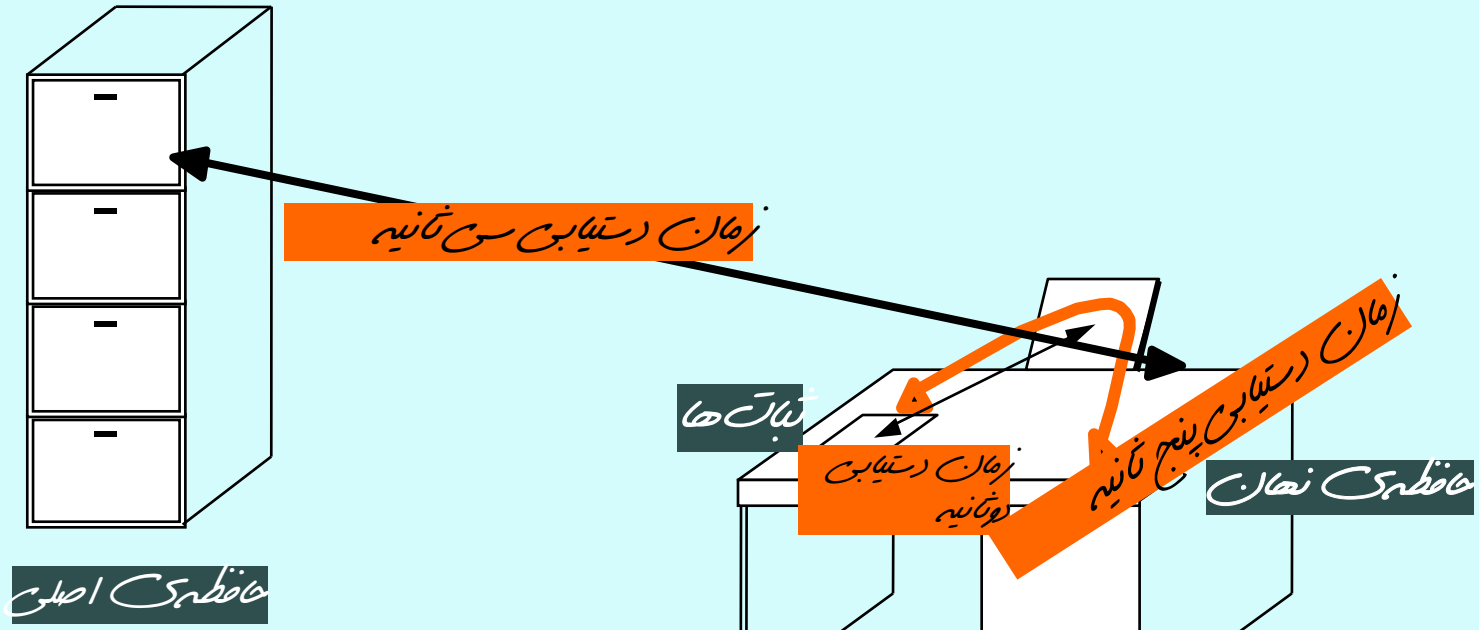
1.64/50 = 3.3%.

از کل پهنای باند بابت تازه‌سازی هدر می‌رود


1000
100
10
1
1980
1990
2000
2010
تعداد ترانزیستورهای حفظه
تاریخ
نوع کامپیوتر
گنجایش حافظه
ابر کامپیوترها
سرورها
ایستگاه‌های کاری
کامپیوترهای شخصی بزرگ
کامپیوترهای شخصی کوچک
1 TB
256 GB
64 GB
16 GB
4 GB
1 GB
256 MB
64 MB
16 MB
4 MB
1 MB


10^6
10^3
1
پردازنده‌ها
حافظه
تراکم نسبی
1980
1990
2000
2010
تاریخ

سلسله‌مراتب حافظه



سرعت، گنجایش و قیمت حافظه

Memory technology	Typical access time	$ per GB in 2008
SRAM	0.5-2.5 ns	$2000-$5000
DRAM	50-70 ns	$20-$75
Magnetic Disk	5,000,000-20,000,000 ns	$0.20-$2

یک حافظه‌ی ایده‌آل، زمان دسترسی پایین (مانند SRAM) و گنجایش بالا (مانند دیسک سخت)

locality

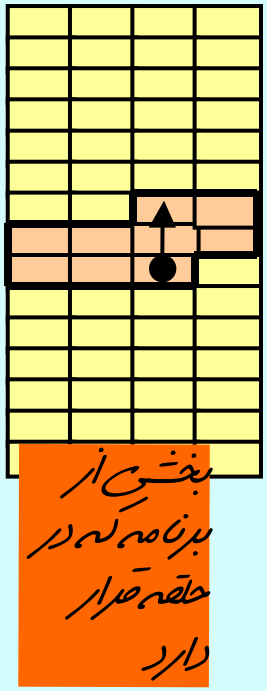
- برای اجرای برنامه‌های ذخیره شده در حافظه، مراجعه به حافظه به بخش کوچکی محدود می‌شود.

همجواری

Temporal locality

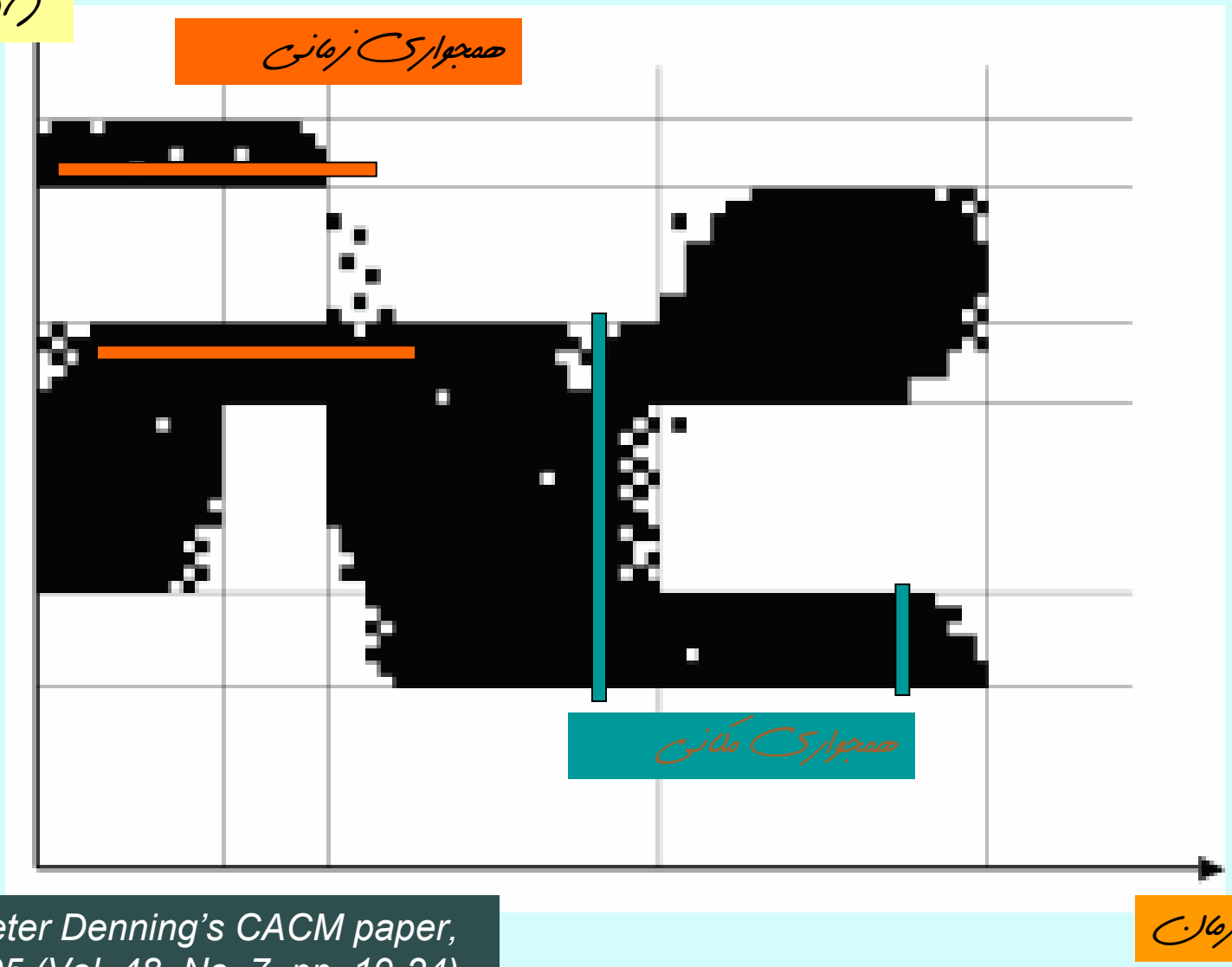
- **همجواری زمانی**: بخشی از حافظه که مورد مراجعه قرار گرفته است، با احتمال بالایی مجدداً مورد استفاده قرار می‌گیرد.
  - متغیرها، و یا دستورهایی که در حلقه قرار گرفته‌اند.

Spatial locality



- **همجواری مکانی**: نواحی مجاور بخشی از حافظه که مورد مراجعه قرار گرفته است، با احتمال بالایی مجدداً مورد استفاده قرار می‌گیرد.
  - آرایه‌ها، و یا توالی دستورالعمل‌های یک برنامه

همجواری (ادامه...)



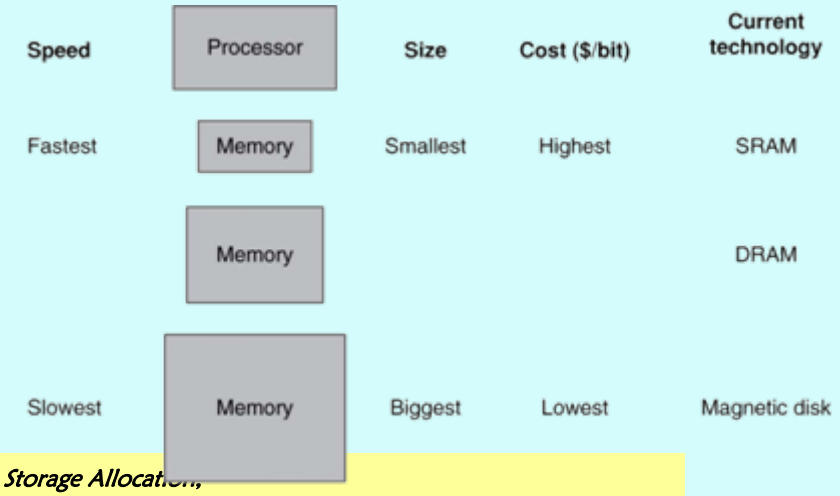
From Peter Denning's CACM paper, July 2005 (Vol. 48, No. 7, pp. 19-24)

همجواری (ادامه...)

- سلسله مراتب در در حافظه:
  - همه چیز را در دیسک سخت ذخیره کن
  - یک نسخه از موارد نیاز را در DRAM بنویس
  - یک نسخه از موارد مورد نیاز که اخیرا به کار گرفته شده‌اند را در SRAM ذخیره کن.

Main memory

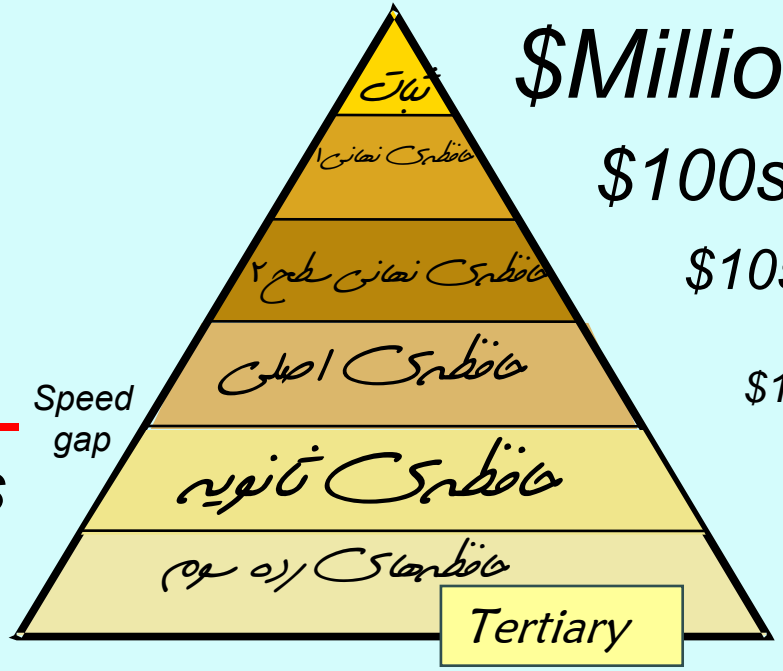
Cache memory attached to CPU



M. V. Wilkes, "Slave Memories and Dynamic Storage Allocation," IEEE Transactions on Electronic Computers, vol. EC-14, no. 2, pp. 270-271, April 1965.

سطوح مختلف حافظه

گنجایش	زمان دستیابی	قیمت هر GB
100s B	ns	$Millions
10s KB	a few ns	$100s Ks
MBs	10s ns	$10s Ks
100s MB	100s ns	$1000s
10s GB	10s ms	$10s
TBs	min+	$1s



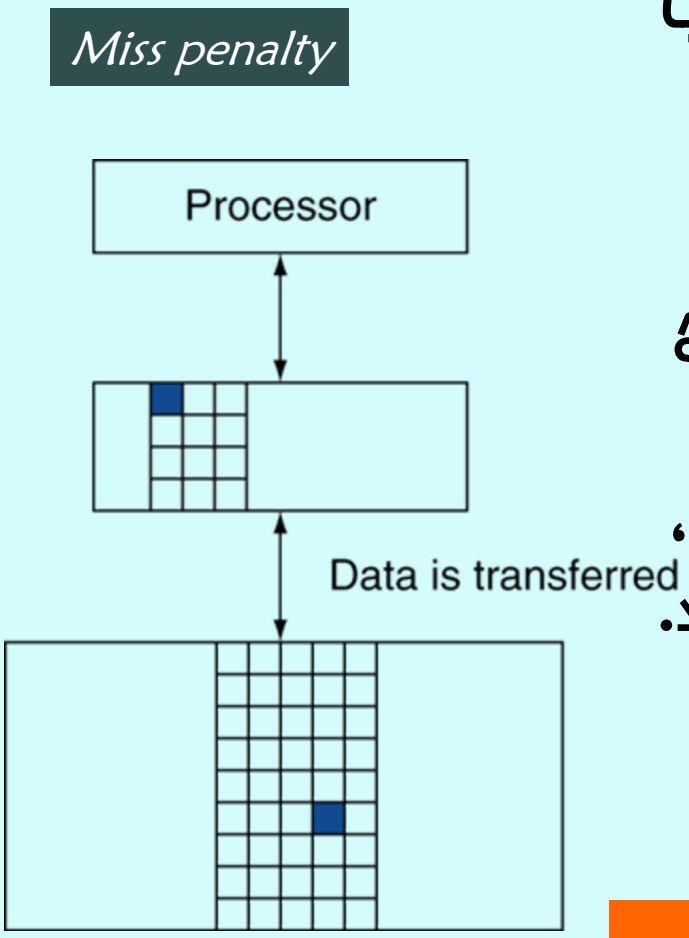
سلسله‌مراتب در حافظه

- کوچک‌ترین واحد حافظه در حافظه‌ی نهان که وجود یا عدم وجود آن قابل بررسی است، سطر یا بلوک نامیده می‌شود.
- در صورتی که داده در سطوح بالا وجود داشت، دستیابی موفق بوده است و hit شده است!

$$\text{Hit ratio} = \frac{\#\text{Hits}}{\#\text{Accesses}}$$

سلسله‌مراتب در حافظه (ادامه...)

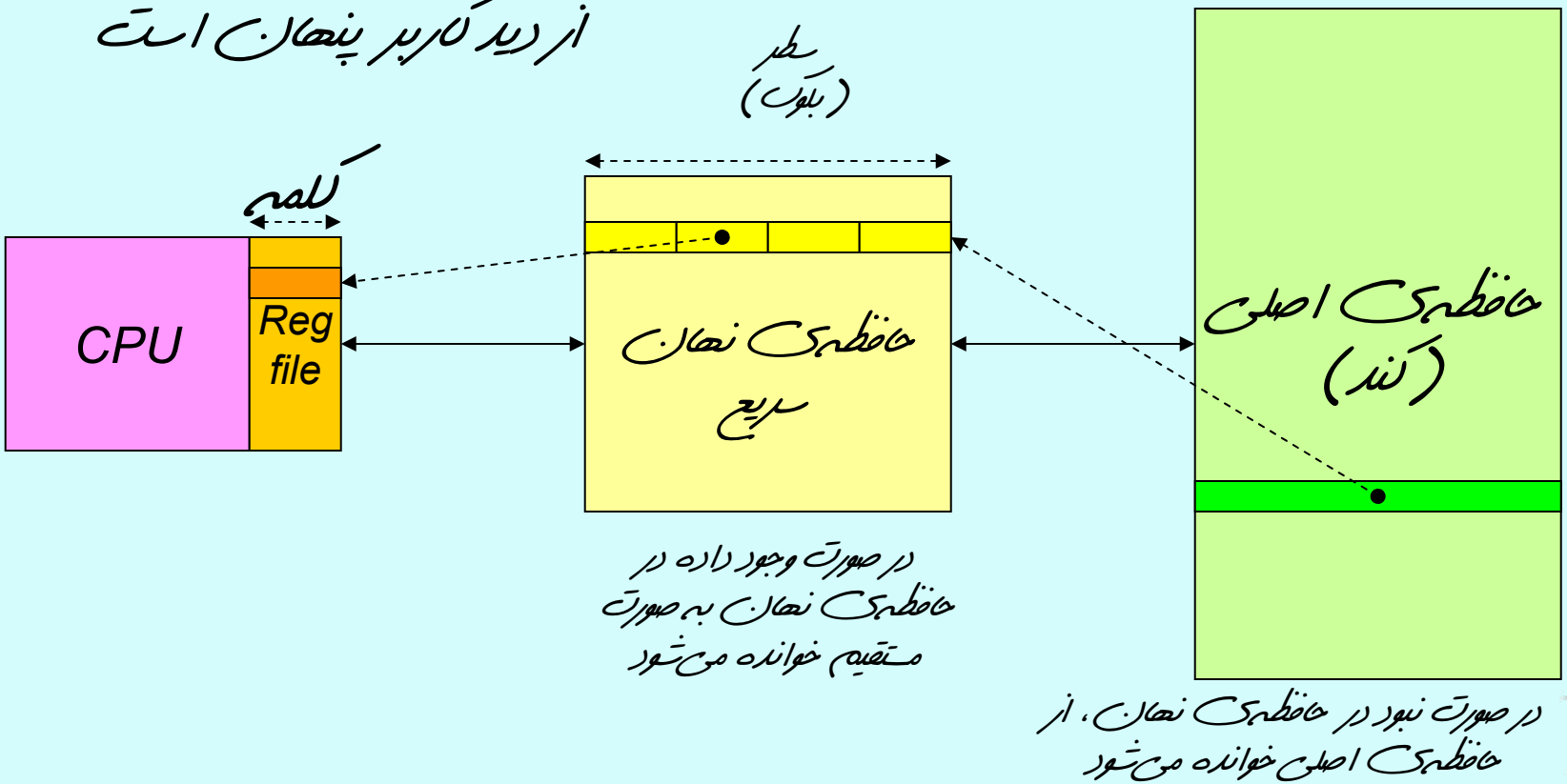
- در غیر این صورت درخواست، با پاسخی موفقیت‌آمیز روبرو نمی‌شود و miss شده است!
  - در این حالت باید داده از سطوح پایین‌تر منتقل شود.
  - زمانی که صرف انتقال می‌شود، جریمه‌ی فقدان نامیده می‌شود.



$$\text{Miss ratio} = \frac{\#\text{Miss}}{\#\text{Accesses}} = 1 - \text{Hit ratio}$$

حافظه‌ی نهان (ادامه...)

دسترسی به حافظه نهان
از دید کاربر پنهان است



در صورت وجود داده در
حافظه‌ی نهان به صورت
مستقیم خوانده می‌شود

در صورت نبود در حافظه‌ی نهان، از
حافظه‌ی اصلی خوانده می‌شود

در صورت در اختیار داشتن یک سطح حافظه‌ی نهان با hit ratio برابر با h

$$Ceff = hCfast + (1 - h)(Cslow + Cfast) = Cfast + (1 - h)Cslow$$

معماری کامپیوتر (۱۳۹۱-۱۱-۱۳)
جلسه‌ی بیستم

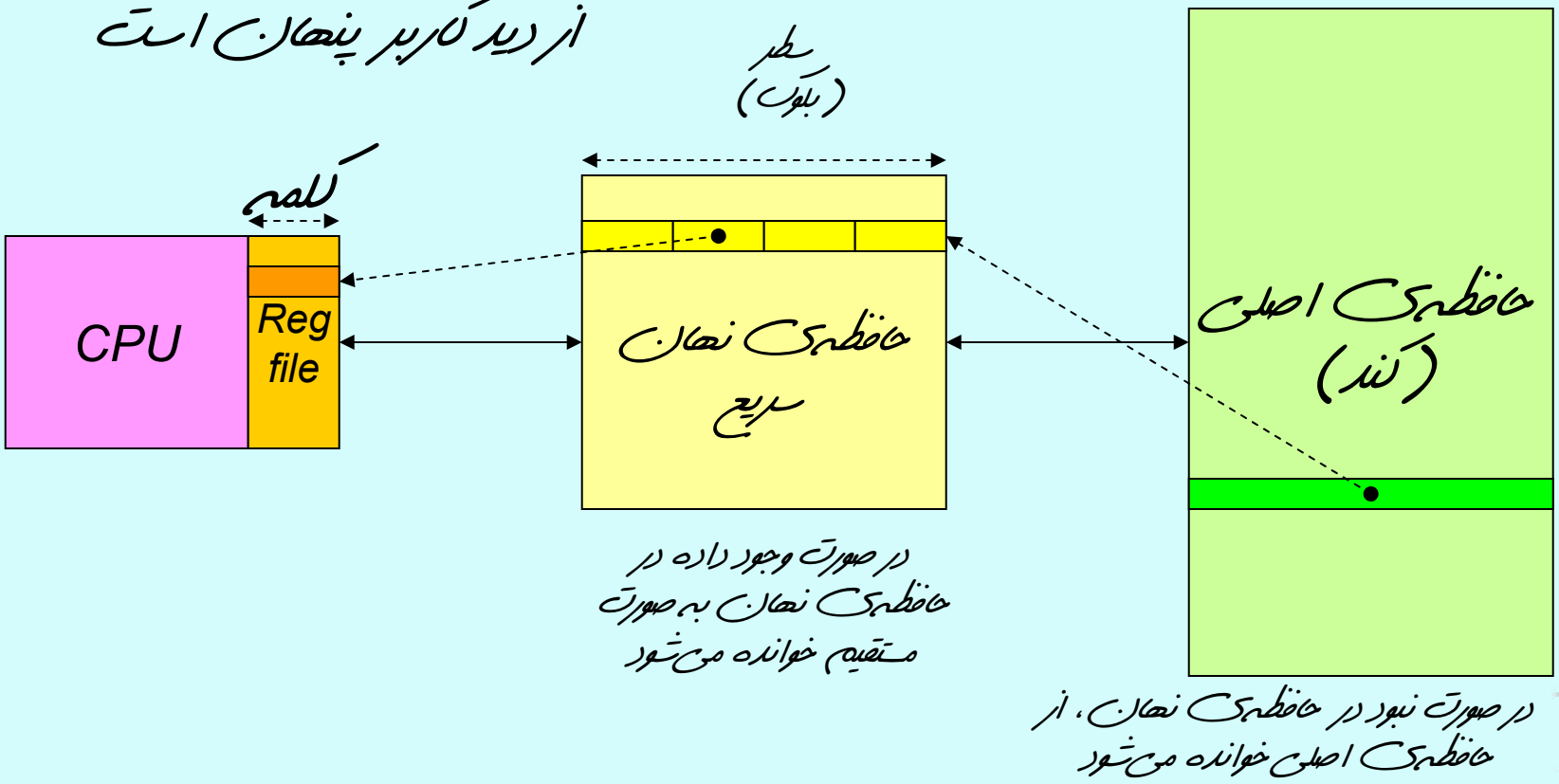
دانشگاه شهید بهشتی
دانشکده‌ی مهندسی برق و کامپیوتر
بهار ۱۳۹۱
احمد محمودی ازناوه

فهرست مطالب

- حافظه‌ی نهان
  - نگاشت مستقیم

حافظه‌ی نهان (ادامه...)

دسترسی به حافظه نهان
از دید کاربر پنهان است



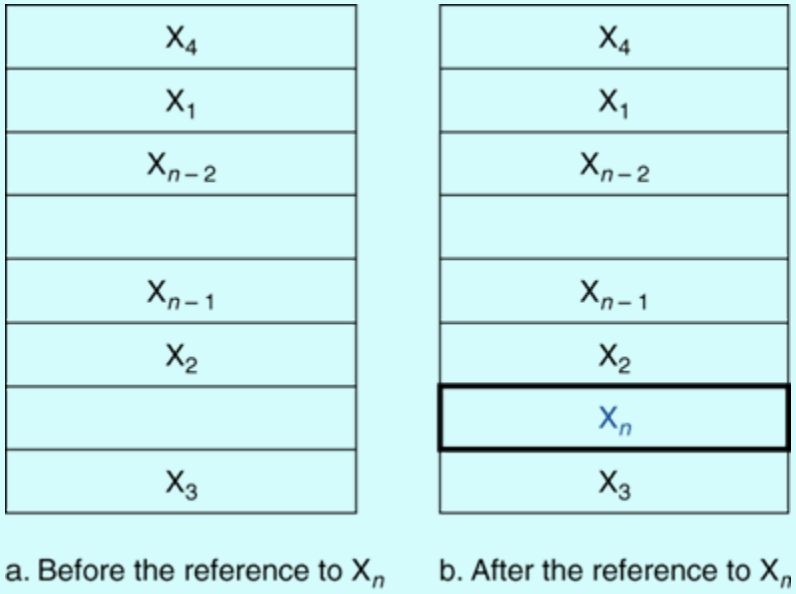
در صورت در اختیار داشتن یک سطح حافظه‌ی نهان با hit ratio برابر با h

$$Ceff = hCfast + (1 - h)(Cslow + Cfast) = Cfast + (1 - h)Cslow$$

حافظه‌ی نهان

- **حافظه‌ی نهان:**
  - در سلسله مراتب حافظه، نزدیک‌ترین واحد به پردازشگر مرکزی است.
  - فرض می‌کنیم به خانه‌های حافظه با آدرس‌های زیر مراجعه می‌شود:

X1, …, Xn–1, Xn

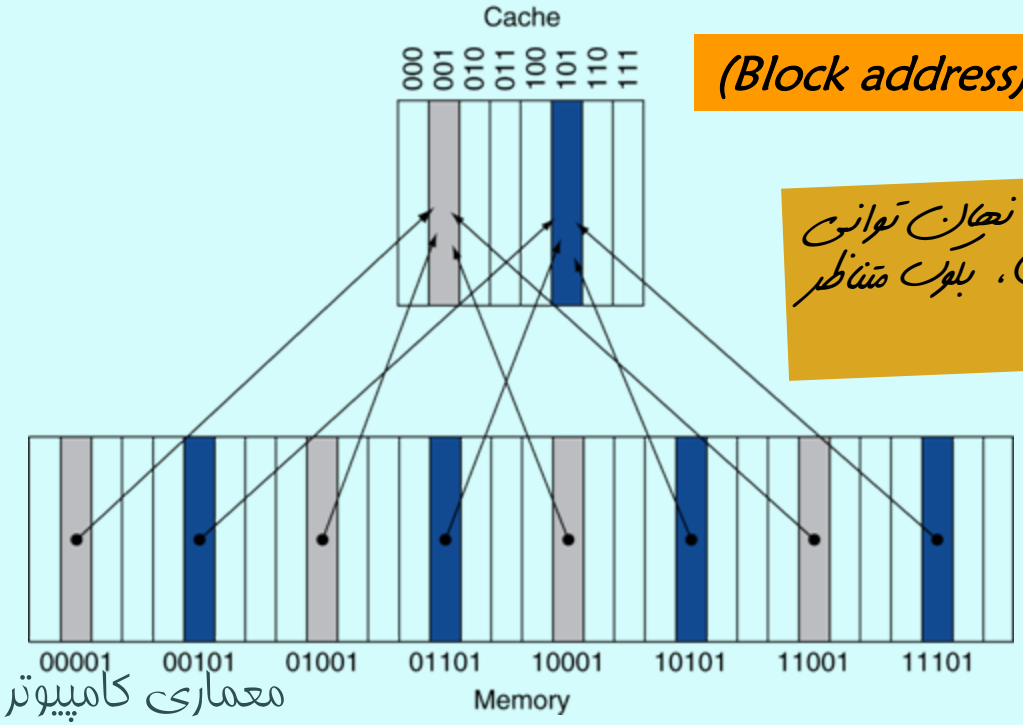


a. Before the reference to X_n b. After the reference to X_n

- از کجا بدانیم که داده در حافظه‌ی نهان وجود دارد؟
- در صورت موجود بودن، چگونه جای آن را بیابیم؟

نگاشت مستقیم

- ساده‌ترین راه، اختصاص یک بلوک از حافظه‌ی نهان برای هر بلوک در حافظه‌ی اصلی است.
- برای محاسبه‌ی آدرس چنین بلوکی از رابطه‌ی زیر استفاده می‌شود.



(Block address) modulo (#Blocks in cache)

در صورتی که تعداد بلوک‌های حافظه‌ی نهان توانی از دو باشد، بیت‌های کم ارزش آدرس، بلوک متناظر با حافظه را نشان خواهد داد

نگاشت مستقیم

- از کجا بدانیم کدام بلوک حافظه در حافظه‌ی نهان قرار گرفته است؟
  - بیت‌های پرارزش بلوک نیز باید ذخیره شوند.
  - این بیت‌ها برچسب (tag) نامیده می‌شود. در واقع با کمک tag مشخص می‌شود کدامین بلوک از بین بلوک‌هایی که مجاز به حضور در یک بلوک حافظه‌ی نهان هستند، در آن‌جا قرار دارند؟
- چگونه می‌توان فهمید که بلوکی از حافظه‌ی نهان خالی است، یا داده‌ی موجود در آن معتبر نمی‌باشد؟
  - به ازای هر بلوک یک بیت به نام بیت اعتبار (valid bit) اختصاص داده می‌شود.

مثالی از حافظه‌ی نهان

- 8-blocks, 1 word/block, direct mapped

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

حالت اولیه

Word addr	Binary addr	Cache block
22	10 110	110

miss

مثالی از حافظه‌ی نهان (ادامه...)

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Word addr	Binary addr	Cache block
26	11 010	010

miss

مثالی از حافظه‌ی نهان (ادامه...)

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Word addr	Binary addr	Cache block	
22	10 110	110	hit
26	11 010	010	hit

مثالی از حافظه‌ی نهان (ادامه...)

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Word addr	Binary addr	Cache block
16	10 000	000
3	00 011	011
16	10 000	000

miss

miss

hit

مثالی از حافظه‌ی نهان (ادامه...)

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Word addr	Binary addr	Cache block
18	10 010	010

miss

مثالی از حافظه‌ی نهان (ادامه...)

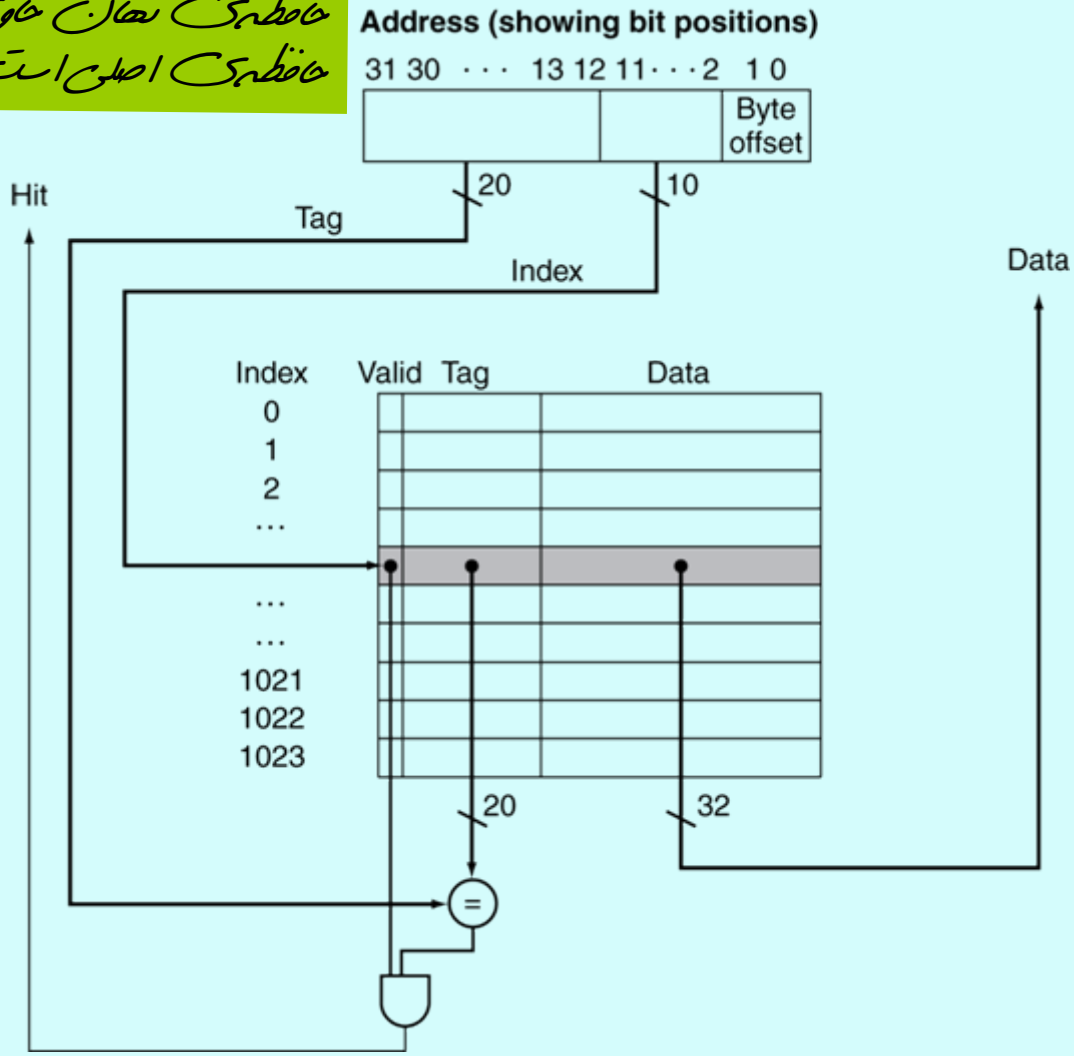
Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

زیر بخش مربوط به آدرس

شاخص (index) : بخش کم ارزش آدرس بلوک که بین حافظه‌ی اصلی و حافظه‌ی نهان مشترک است.

برچسب (tag) : مشخص می‌کند هر بلوک حافظه‌ی نهان حاوی کدامین بلوک از حافظه‌ی اصلی است



مثال

- یک سیستم با مشخصات زیر را در نظر بگیرید:
  - سی‌ودو بیت آدرس
  - حافظه‌ی نهان با نگاشت مستقیم
  - حافظه‌ی نهان با 2^n بلوک؛ شاخص با n بیت
  - اندازه‌ی هر بلوک 2^m کلمه

پرسش: اندازه‌ی بخش برچسب چند بیت خواهد بود؟

پاسخ: $32-(n+m+2)$

پرسش: تعداد کل بیت‌های حافظه‌ی مورد نیاز برای حافظه‌ی نهان را حساب کنید؟

پاسخ:

$$2^n \times (\text{block size} + \text{tag size} + \text{valid field size})$$

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

البته باید توجه داشت، معمولا حجم داده قابل ذخیره سازی به عنوان گنجایش ذکر می‌شود

معماری کامپیوتر (۱۳۹۱-۱۱-۱۳)

جلسه‌ی بیست و یکم

دانشگاه شهید بهشتی
دانشکده‌ی مهندسی برق و کامپیوتر
بهار ۱۳۹۱
احمد محمودی ازناوه

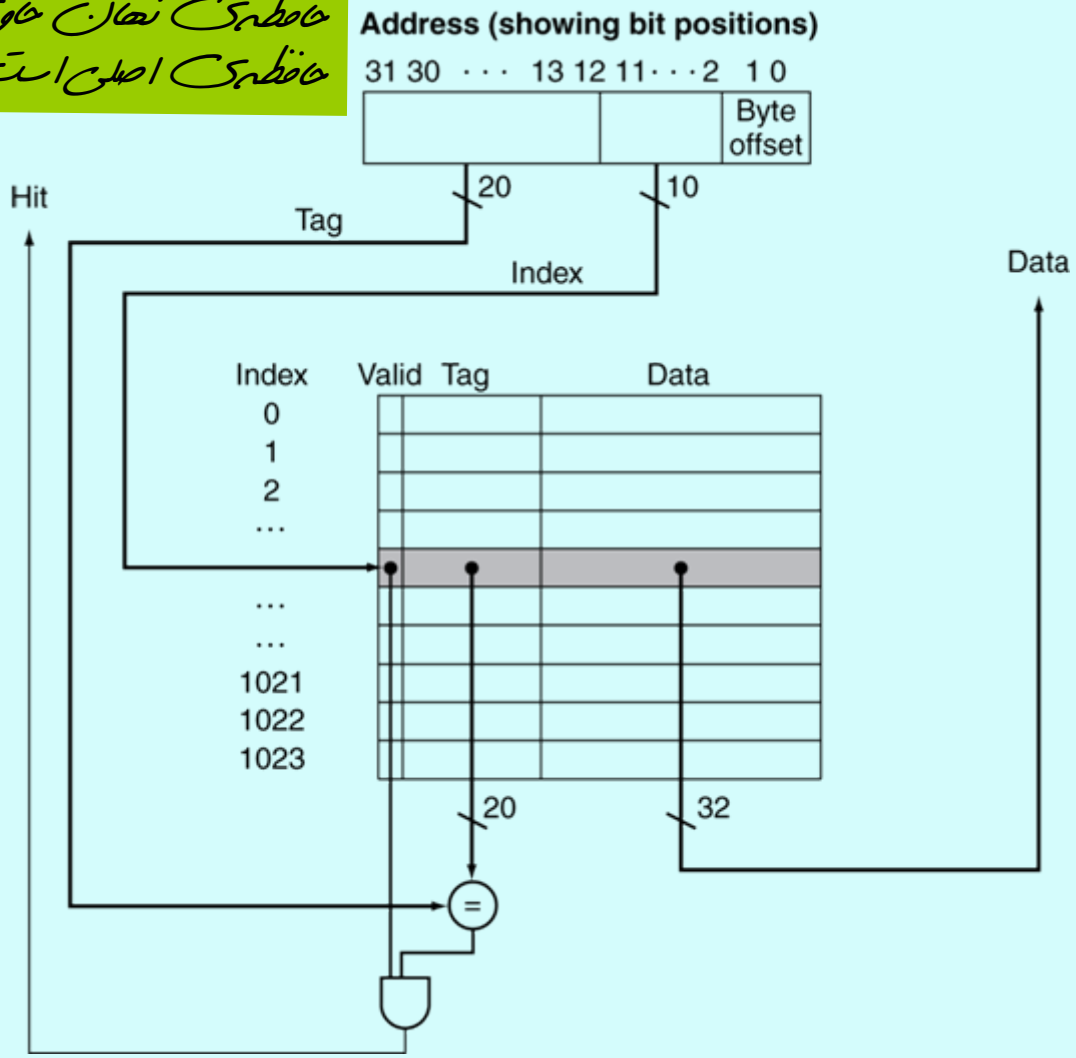
فهرست مطالب

- حافظه‌ی نهان
  - نگاشت مستقیم
- نوشتن در حافظه‌ی نهان
- حافظه‌ی برگ‌برگ شده

زیر بخش مربوط به آدرس

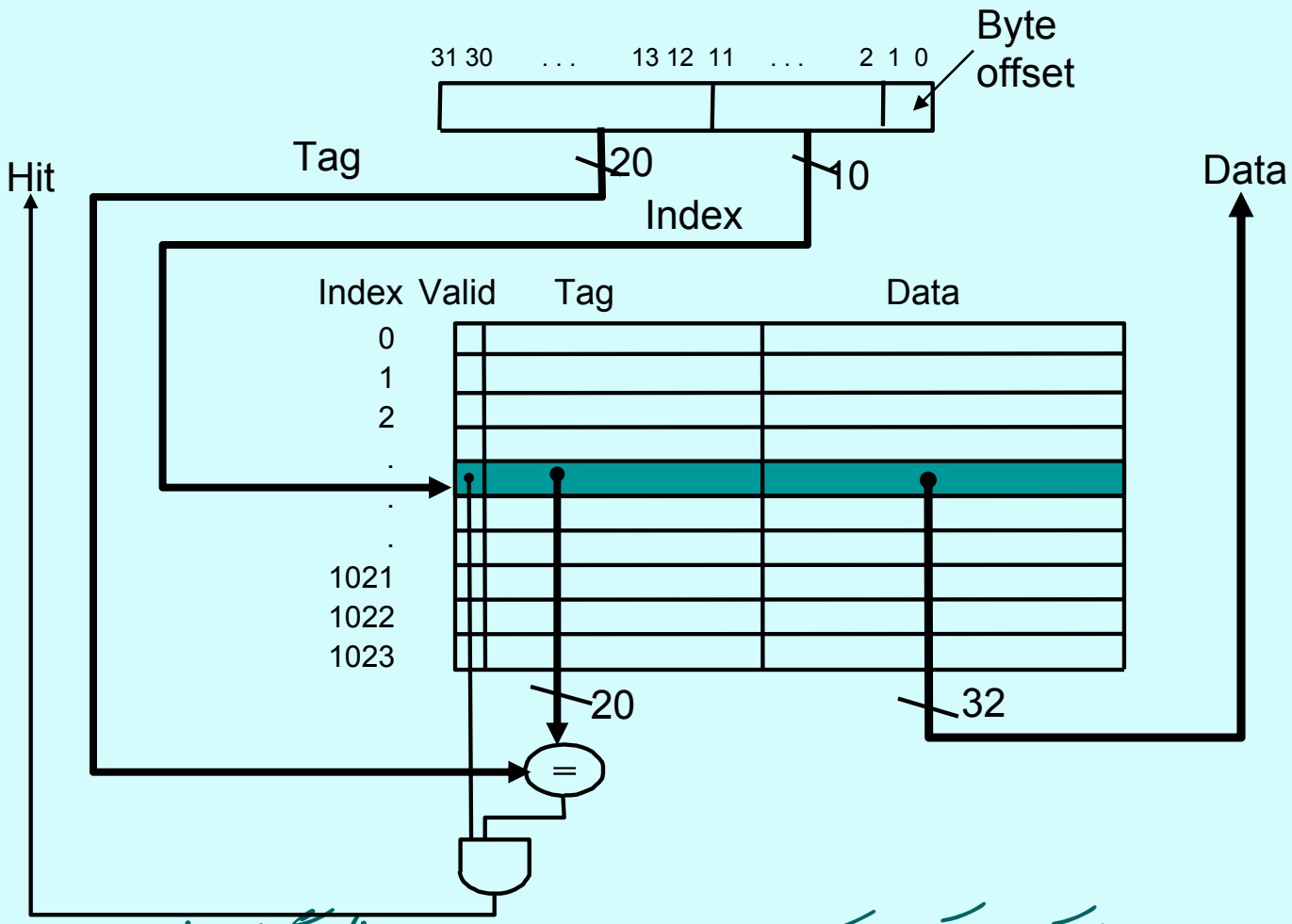
شاخص (index) : بخش کم ارزش آدرس بلوک که بین حافظه‌ی اصلی و حافظه‌ی نهان مشترک است.

برچسب (tag) : مشخص می‌کند هر بلوک حافظه‌ی نهان حاوی کدامین بلوک از حافظه‌ی اصلی است



حافظه‌ی نهان با نگاشت مستقیم

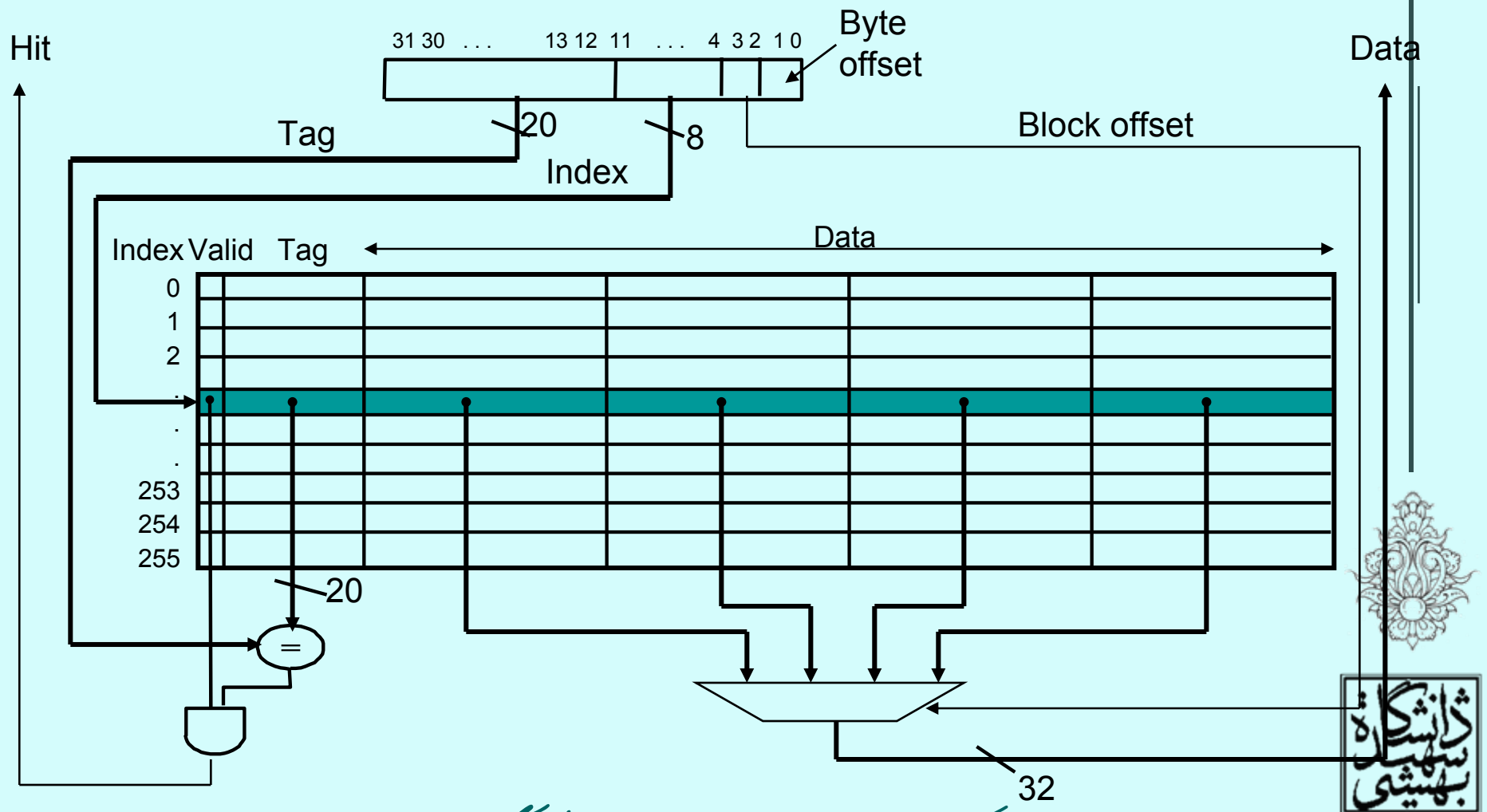
- گنجایش حافظه‌ی نهان 1KW با بلوک‌های یک کلمه‌ای



با استفاده از یک بلوک، کدام نوع همجواری در نظر گرفته شده است؟

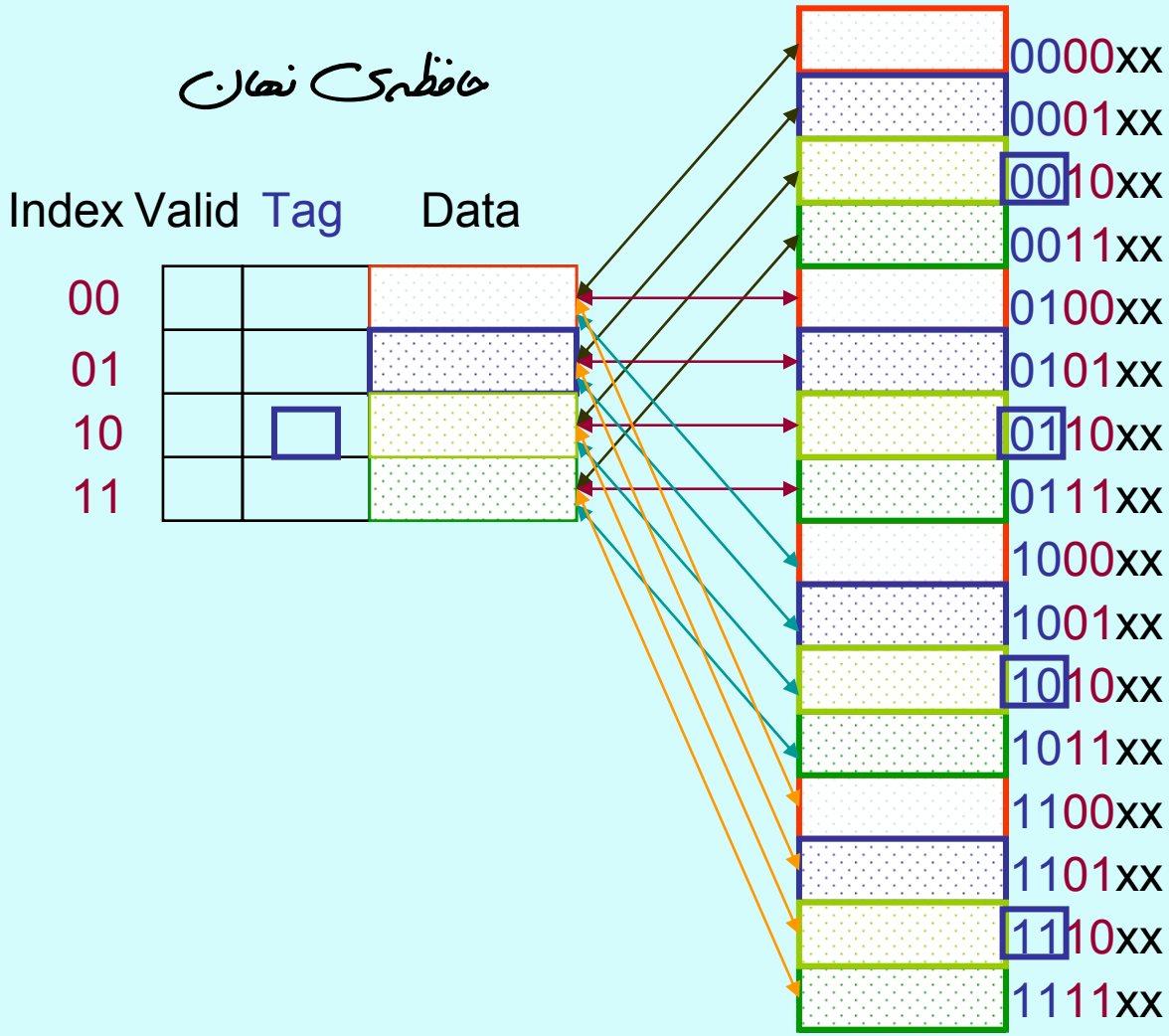
حافظه‌ی نهان با نگاشت مستقیم (ادامه...)

- گنجایش حافظه‌ی نهان 1KW با بلوک‌های چهار کلمه‌ای



در این حالت، کدام نوع همجواری در نظر گرفته شده است؟

مثال


حافظه‌ی نهان
Index Valid Tag Data
00
01
10
11
0000xx
0001xx
0010xx
0011xx
0100xx
0101xx
0110xx
0111xx
1000xx
1001xx
1010xx
1011xx
1100xx
1101xx
1110xx
1111xx

مثال

- یک سیستم با مشخصات زیر را در نظر بگیرید:
  - سی‌ودو بیت آدرس
  - حافظه‌ی نهان با نگاشت مستقیم
  - حافظه‌ی نهان با 2^n بلوک؛ شاخص با n بیت
  - اندازه‌ی هر بلوک 2^m کلمه

پرسش: اندازه‌ی بخش برچسب چند بیت خواهد بود؟

پاسخ: $32-(n+m+2)$

پرسش: تعداد کل بیت‌های حافظه‌ی مورد نیاز برای حافظه‌ی نهان را حساب کنید؟

پاسخ:

$$2^n \times (\text{block size} + \text{tag size} + \text{valid field size})$$

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

البته باید توجه داشت، معمولا حجم داده قابل ذخیره‌سازی به‌عنوان گنجایش ذکر می‌شود

مثالی دیگر

- برای یک حافظه‌ی نهان با شیوه‌ی نگاشت مستقیم با گنجایش 16KB و بلوک‌های چهار کلمه‌ای چند بیت حافظه نیاز داریم؟ (۳۲ بیت خط آدرس وجود دارد)

$16KB \rightarrow 2^{12}\ word$

$16KB \rightarrow 2^{10}\ Block$

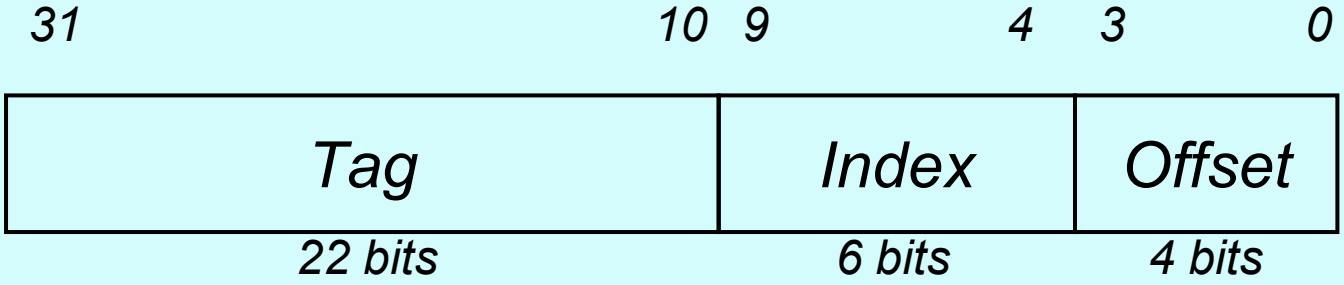
$1\ Block \rightarrow 4\ word\ (2^2)$

$32-(10+2+2)=18\ bit\ for\ tag$ + $1\ bit\ for\ valid\ bit$

$2^{10}\times[4\times32+18+1] = 147Kbits$ پاسخ

باز هم مثال

- یک حافظه‌ی نهان 64بلوک، هر یک به اندازه‌ی ۱۶ بایت مفروض است. آدرس 1200 حافظه‌ی اصلی در کدامین بلوک حافظه‌ی نهان قرار می‌گیرد؟



$$Block\ address = \lfloor 1200/16 \rfloor = 75$$

$$Block\ number = 75\ modulo\ 64 = 11$$

پاسخ

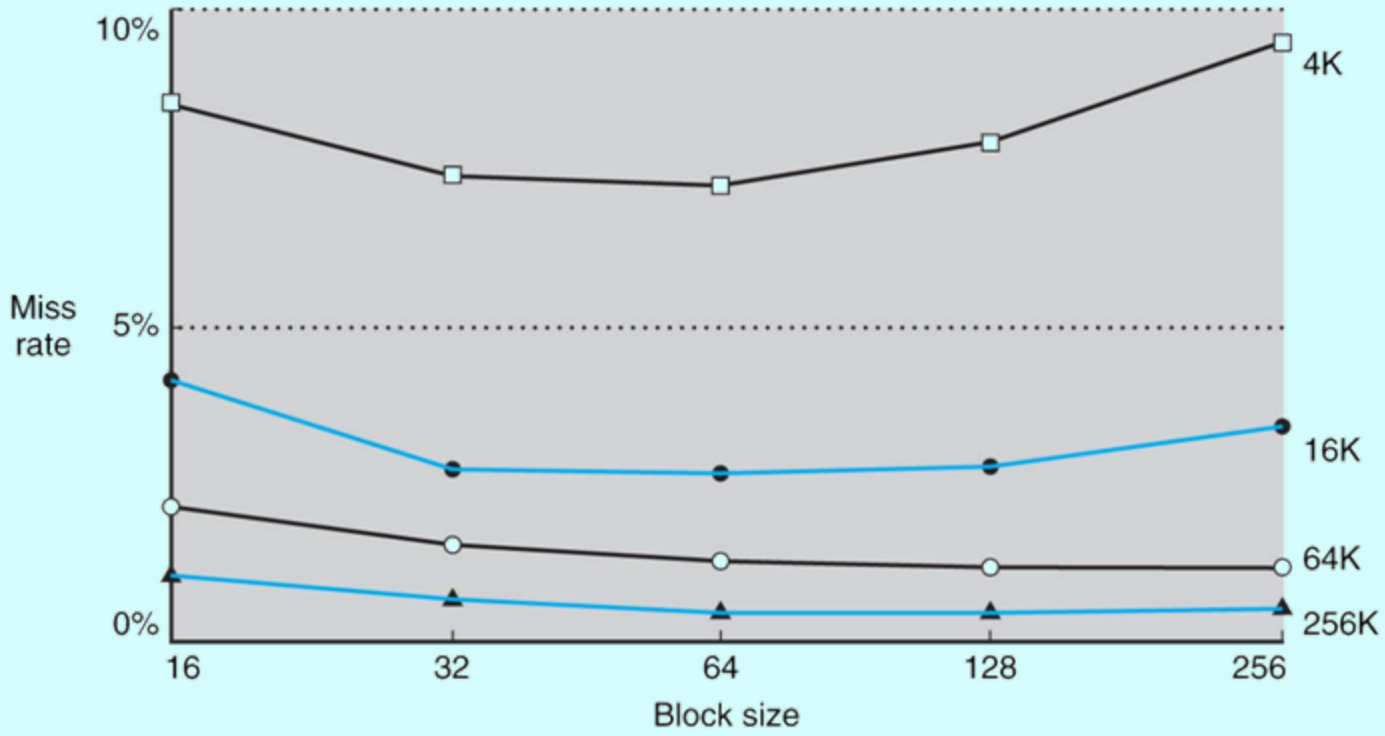
بزرگی بلوک‌ها

- بلوک‌های بزرگ‌تر ← همجواری مکانی بهتر
  - در نتیجه miss rate کاهش می‌یابد.
- با حافظه‌ی نهان با حجم ثابت:
  - افزایش حجم بلوک ← کاهش تعداد بلوک‌ها
    - افزایش رقابت بین بلوک‌ها ← افزایش miss rate
- بلوک‌های بزرگ جریمه‌ی فقدان بالاتری دارند.
  - می‌باید در صورت عدم وجود بلوک در حافظه‌ی نهان، بلوک بزرگ‌تری به حافظه‌ی نهان منتقل شود.
    - با طراحی بهتر حافظه، می‌توان تا حدی بر این مشکل غلبه کرد.

اما

بزرگی بلوک‌ها (ادامه...)

- بدین‌ترتیب مزیت کاهش miss rate تحت‌الشعاع قرار می‌گیرد.



نبود/وجود بلوک در حافظه‌ی نهان

- در صورتی که بلوک مربوط به آدرس مورد نظر در حافظه‌ی نهان وجود داشته باشد:
  - پردازنده به روند عادی خود ادامه می‌دهد.
- در غیر این صورت

freezing the content of temporary register

  - خط لوله دچار تعلیق می‌شود یا وقفه‌ای رخ می‌دهد
  - داده از سطوح پایین‌تر به حافظه‌ی نهان منتقل می‌شود.

in order processr

out-of-order processor

در این شیوه به جای اجرای برنامه بر اساس توالی دستورالعمل‌ها، ترتیب اجرای دستورها براساس فراهم بودن داده‌ها صورت می‌پذیرد

نبود/وجود بلوک در حافظه‌ی نهان (ادامه...)

- در صورتی که دستورالعمل در حافظه‌ی نهان وجود نداشته باشد:
  - آدرس مربوط به آن دستور (PC-4) به حافظه‌ی اصلی فرستاده می‌شود.
  - حافظه‌ی اصلی داده را می‌خواند.
  - داده در حافظه‌ی نهان نوشته می‌شود. بیت‌های برچسب و بیت اعتبار مقداردهی می‌شوند.
  - اجرای دستورالعمل از سر گرفته می‌شود.

نوشتن در حافظه‌ی نهان

- هنگامی نوشتن در حافظه‌ی نهان مطرح می‌شود، اوضاع کمی پیچیده‌تر خواهد شد.
- در چنین حالتی بین حافظه‌ی اصلی و حافظه‌ی نهان نوعی ناهماهنگی (inconsistency) به وجود می‌آید.
- ساده‌ترین راه، write through است، بدین معنا که هر آن چه در حافظه‌ی نهان نوشته می‌شود در حافظه‌ی اصلی نیز نوشته شود.

نوشتن در حافظه‌ی نهان (ادامه...)

- در صورتی که write miss رخ دهد، چه فرآیندی طی می‌شود؟
  - داده از حافظه اصلی به حافظه‌ی نهانی منتقل می‌شود، مقدار مورد نظر نوشته شده و سپس حافظه‌ی اصلی به روز خواهد شد!
  - چنین شیوه‌ای موجب کندی خواهد شد:
    - مثال: در صورتی که CPI=1 (بدون cache miss) و ده درصد دستورات store باشد و برای نوشتن در حافظه‌ی اصلی صد سیکل لازم باشد، effective CPI برنامه‌ی مورد نظر چقدر خواهد بود؟

$$Effective\ CPI = 1 + 0.1 \times 100 = 11$$

میان‌گیر نوشتن

- داده‌های به جای این که مستقیم در حافظه نوشته شوند، در یک بافر (میان‌گیر) نوشته خواهند شد. سپس پردازنده به فعالیت خود ادامه خواهد داد. در این هنگام محتوای بافر به حافظه‌ی اصلی منتقل خواهد شد.
  - در صورتی که بافر پر شود، به ناچار پردازنده دچار تعلیق خواهد شد.
  - در صورتی که نرخ تکمیل نوشتن داده در حافظه کندتر از درخواست‌های پردازنده برای نوشتن باشد

باعث کردن خایده‌ای نخواهد داشت!

write back (copy back)

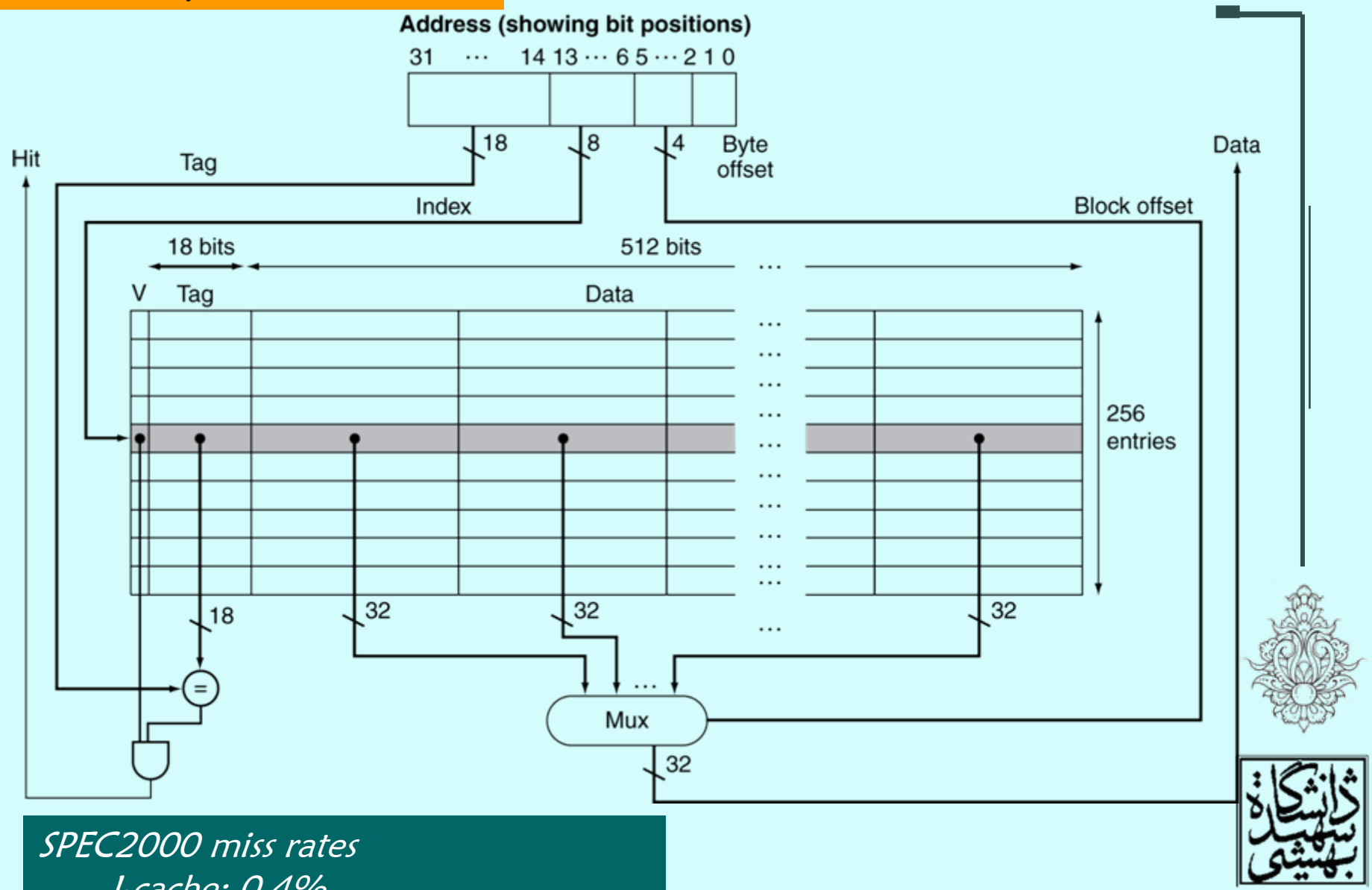
- داده‌ی مورد نظر تنها در حافظه‌ی نهان نوشته می‌شود، و در هنگام جابجایی به حافظه‌ی اصلی انتقال می‌یابد.
- در این صورت باید به نحوی بلوک‌های تغییر یافته (dirty block) را متمایز کنیم.
- در اینجا نیز می‌توان از بافر استفاده نمود.
- طبعاً طراحی سخت‌تر خواهد شد.

مثال: پردازنده‌ی Intrinsity FastMATH

- Intrinsity FastMATH یک پردازنده‌ی سریع درون‌کار است که از معماری MIPS بهره می‌گیرد.
- این پردازنده یک خط لوله‌ی دوازده مرحله‌ای دارد.
- دارای دو حافظه‌ی داده و دستورالعمل می‌باشد.
- برای هر حافظه یک حافظه‌ی نهان با گنجایش 16KB، با بلوک شانزده کلمه‌ای وجود دارد.
- در این حال برای هر حافظه‌ی نهان سیگنال‌های کنترلی مجزا نیاز خواهیم داشت.

Split cache: separate I-cache and D-cache

Intrinsity FastMATH



SPEC2000 miss rates

I-cache: 0.4%

D-cache: 11.4%

Weighted average: 3.2%

نقش حافظه‌ی اصلی

- در صورتی که از DRAM با پنهای یک کلمه برای حافظه‌ی اصلی استفاده کنیم.
- برای دستیابی به محتوای حافظه، زمان‌های دستیابی به قرار زیر می‌باشد:
  - برای ارسال آدرس، یک سیکل گذرگاه
  - برای خواندن داده، پانزده سیکل گذرگاه
  - برای ارسال داده، یک سیکل گذرگاه

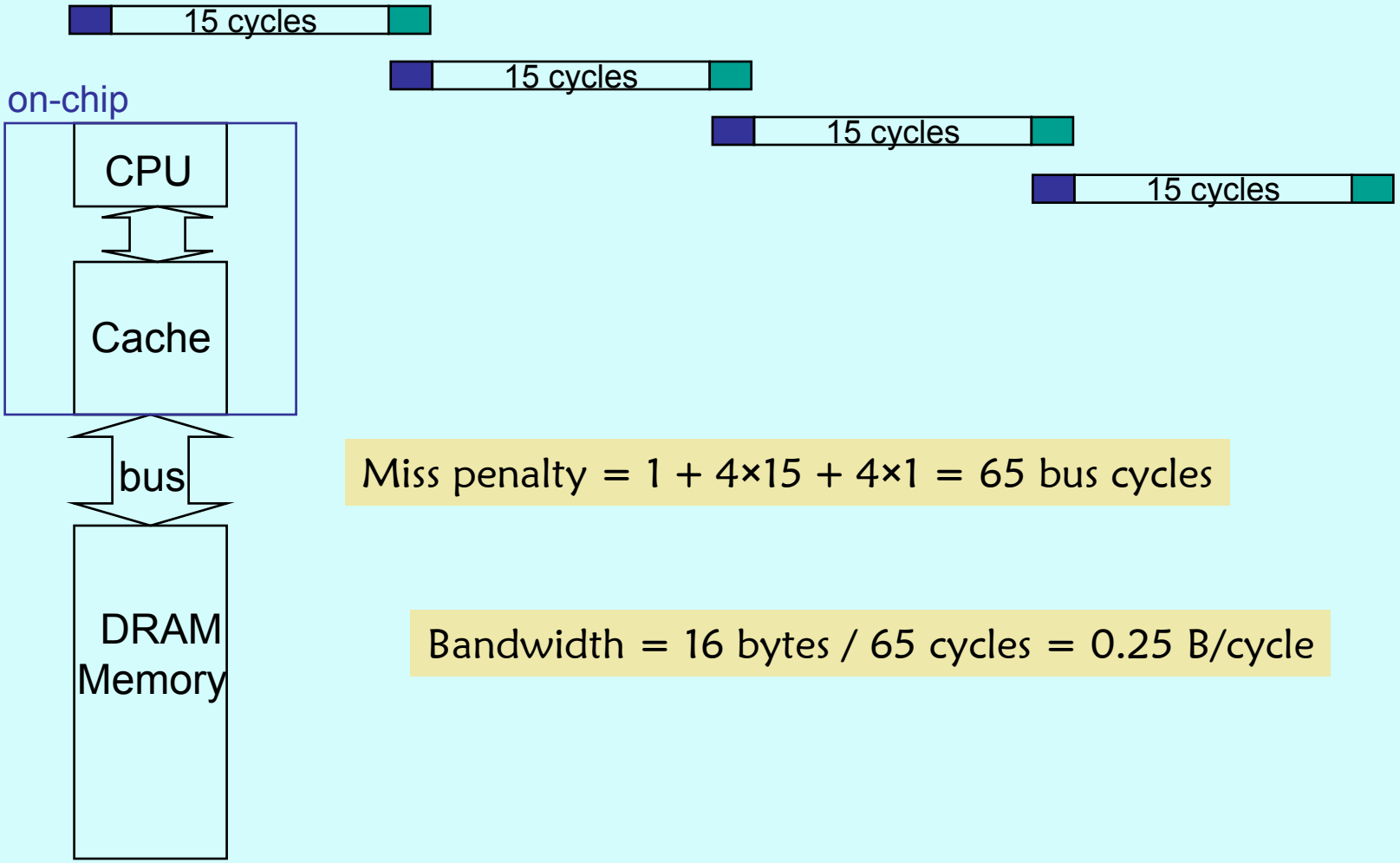
سیکل ساعت گذرگاه از ساعت پردازنده بیشتر است

- جریمه‌ی فقدان را برای خواندن چهار کلمه به دست آورید.

Miss penalty = 1 + 4×15 + 4×1 = 65 bus cycles

Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle

نقش حافظه‌ی اصلی (ادامه...)

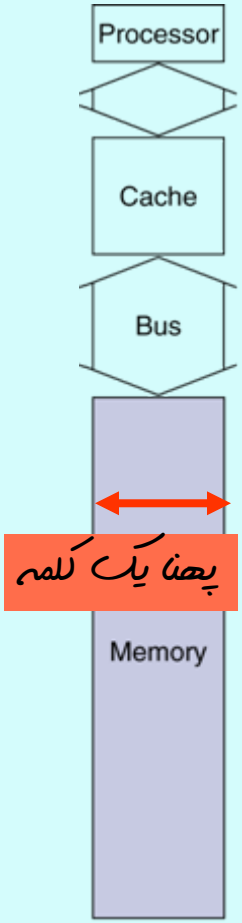


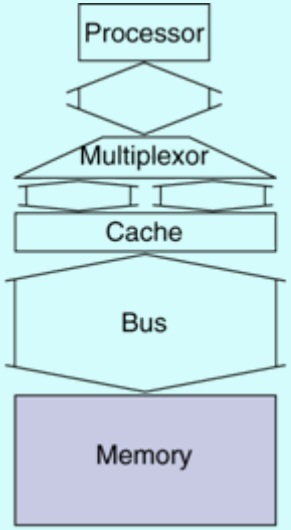
Miss penalty = 1 + 4×15 + 4×1 = 65 bus cycles

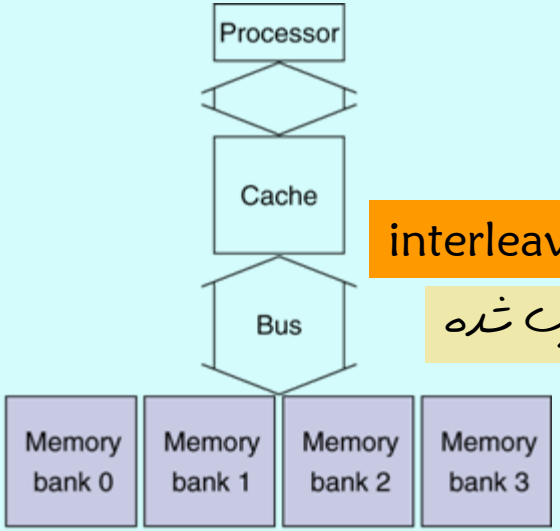
Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle

افزایش پهنای باند حافظه‌ی اصلی

Miss penalty = 1 + 15 + 1 = 17 bus cycles
Bandwidth = 16 bytes / 17 cycles = 0.94 B/cycle



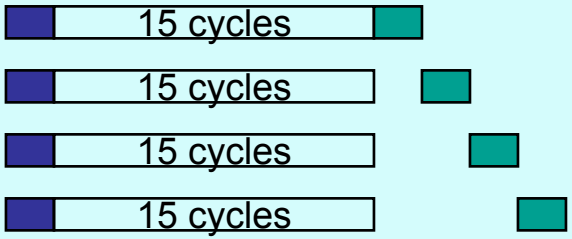




interleaved Memory

حافظه‌ی برگ‌برگ شده

Miss penalty = 1 + 15 + 4×1 = 20 bus cycles
Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle



معماری کامپیوتر (۱۳۹۱-۱۱-۱۳)

جلسه‌ی بیست و دوم

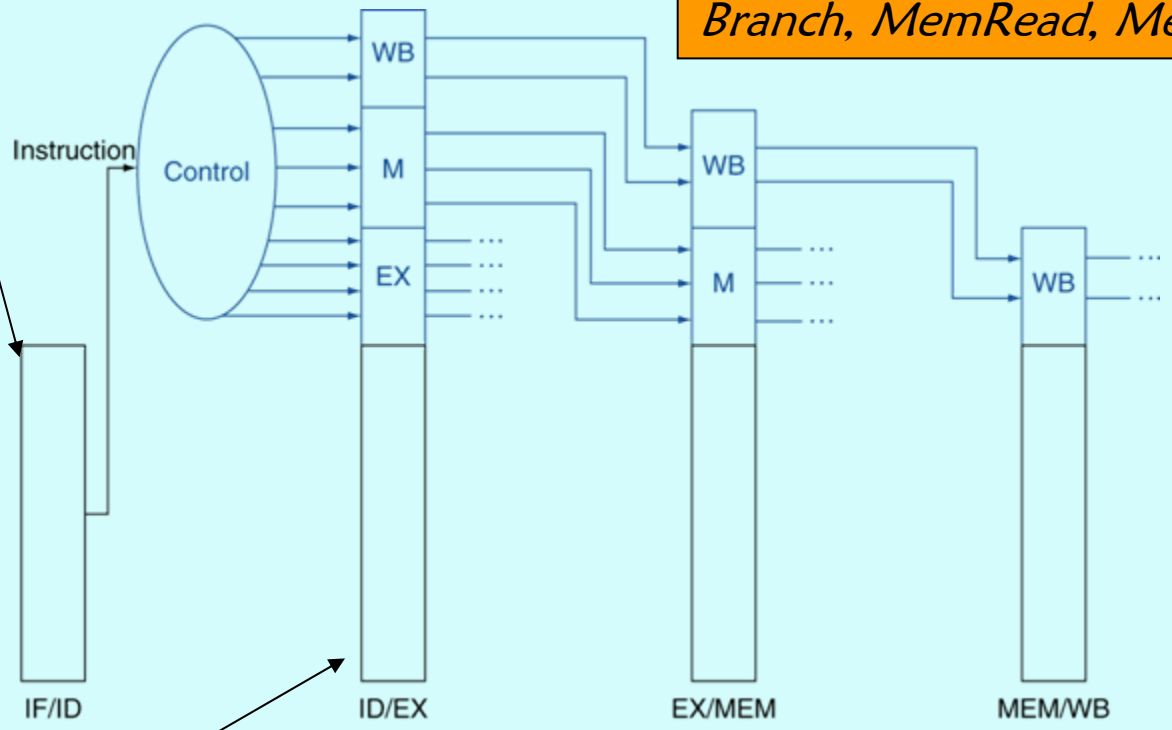
دانشگاه شهید بهشتی
دانشکده‌ی مهندسی برق و کامپیوتر
بهار ۱۳۹۱
احمد محمودی ازناوه

فهرست مطالب

- مروری بر جلسات پیش
- ساختار کنونی حافظه
- حافظه‌ی نهان اشتراکی
  - مجموعه‌های اشتراکی

واحد کنترل خط لوله (ادامه...)

در این بخش می‌باید دستورالعمل بعدی واکشی شود، کاری که برای همه‌ی دستورها به یک شیوه خواهد بود



Branch, MemRead, MemWrite

RegDst, ALUOp, ALUSrc

MemtoReg, RegWrite

همانند گام پیشین، نیاز به سیگنال کنترلی خاصی نیست

تشخیص نیاز به پیش فرستادن

- بنابراین، می‌توان با مقایسه‌ی محتوای ثبات‌ها، مداری برای کنترل پیش‌فرستادن داده طراحی کرد.

```
sub $2, $1,$3
and $12,$2,$5
```

EX/MEM.RegisterRd = ID/EX.RegisterRs=$2

ولی

در همه‌ی دستورالعمل‌های مقدار خروجی ALU، در ثبات نوشته نمی‌شود بدین ترتیب این راهکار در همه‌ی موارد درست نخواهد بود.

برای پیش‌گیری از این مسأله می‌توان از سیگنال‌های کنترلی مربوط به WB ذخیره شده در ثبات‌های خط لوله استفاده نمود.

EX/MEM.RegWrite, MEM/WB.RegWrite

همچنین در صورتی که ثبات شماره‌ی صفر به عنوان مقصد یک دستور استفاده شده باشد، باید از پیش فرستادن جلوگیری کرد.

EX/MEM.RegisterRd ≠ 0,
MEM/WB.RegisterRd ≠ 0

شرایط پیش‌فرستادن

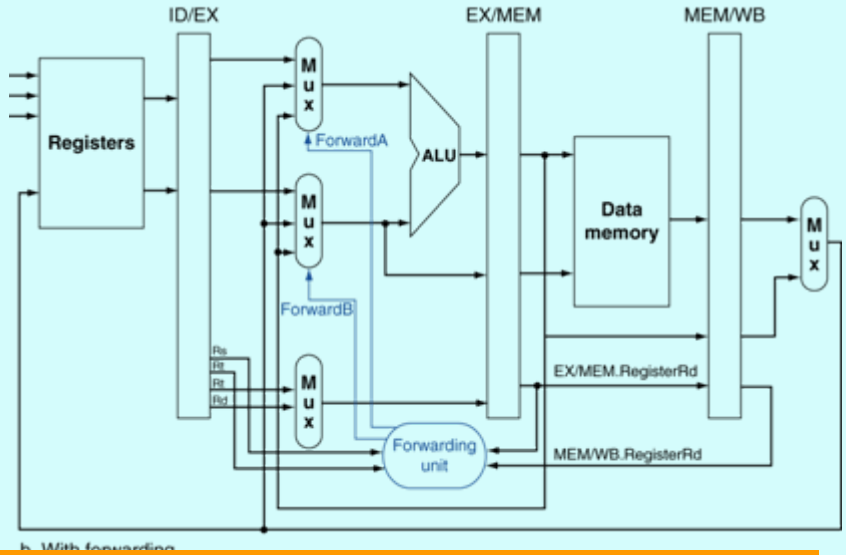
- EX hazard
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 10
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 10
- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01

اگر هر دوی این شرایط با هم رخ داد، چه اشکالی ایجاد می‌شود؟

شرایط پیش‌فرستادن (ادامه...)

- در این قطعه برنامه هر دو نوع مخاطره رخ می‌دهد.

```
add $1,$1,$2
add $1,$1,$3
add $1,$1,$4
```



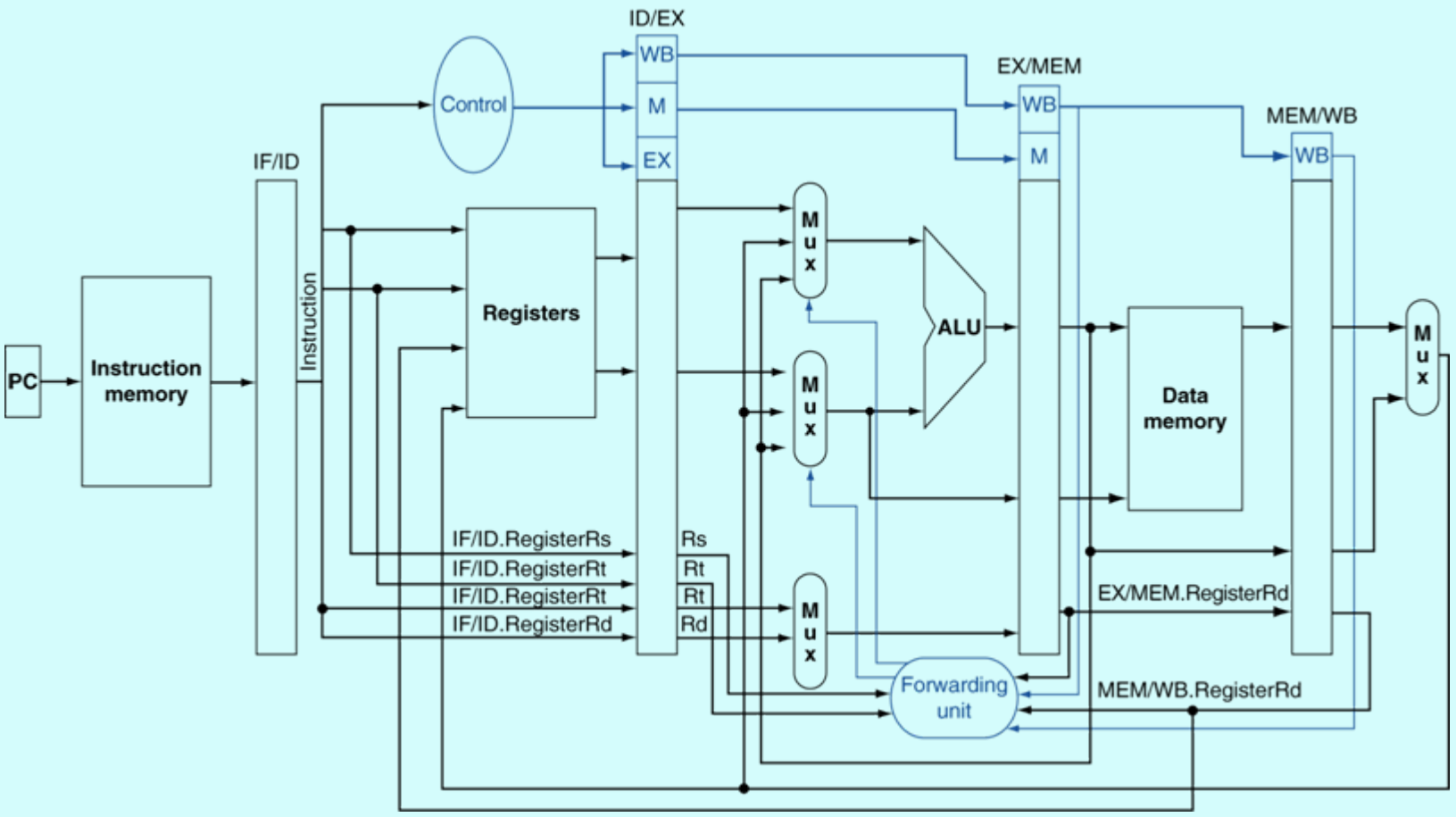
در این حالت آخرین نتیجه باید فرستاده شود،
در نتیجه داده‌ی موجود در مرحله‌ی MEM فرستاده می‌شود.

- بنابراین باید تغییراتی در مخاطره‌ی MEM بدهیم

شرایط پیش‌فرستادن (ادامه...)

- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01

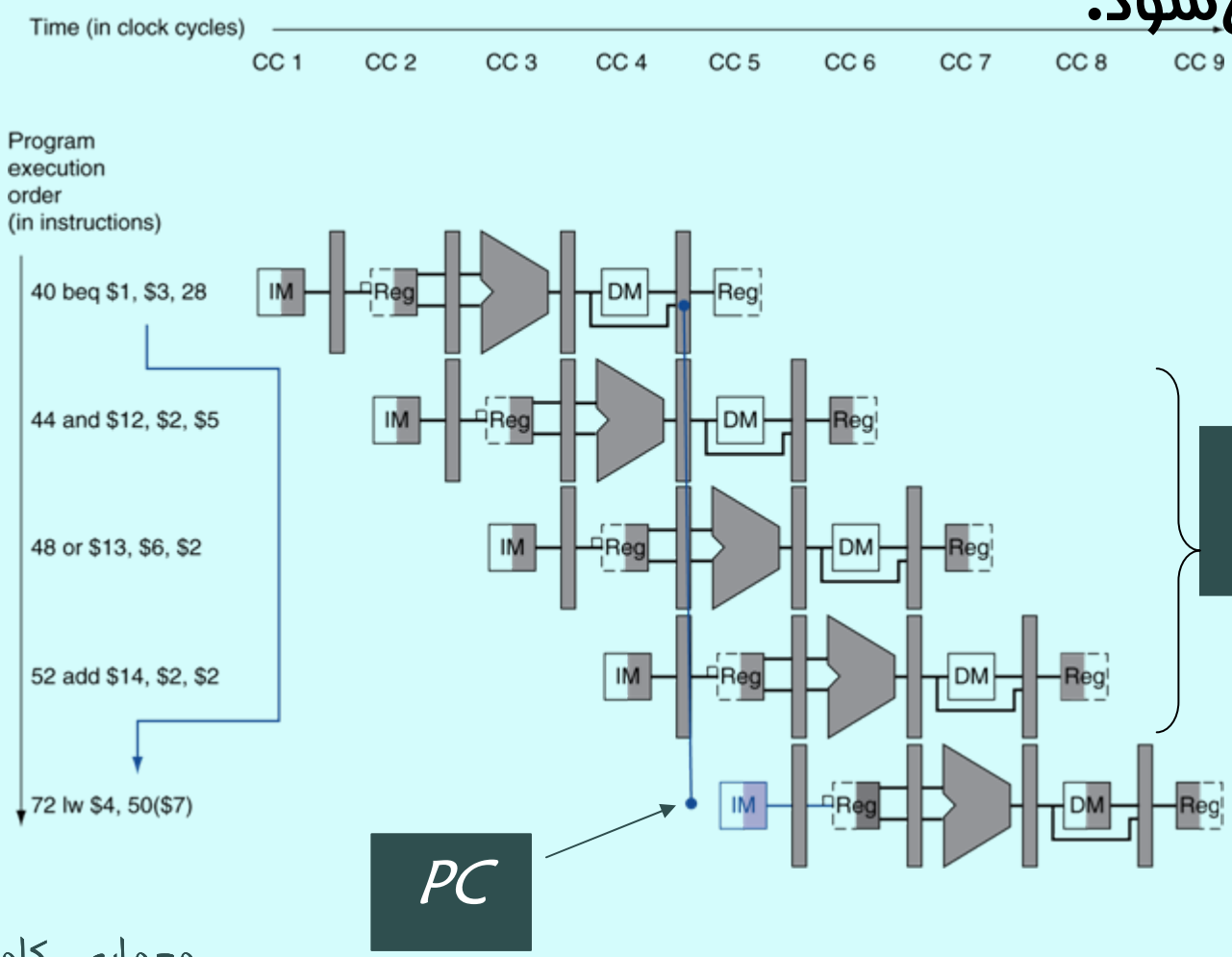
شرایط پیش‌فرستادن (ادامه...)



در این شکل بخشی که برای ارسال داده‌ی ثابت به ALU بود، حذف شده است

مخاطرات کنترل

- نتیجه‌ی دستور پرش در مرحله‌ی MEM مشخص می‌شود.

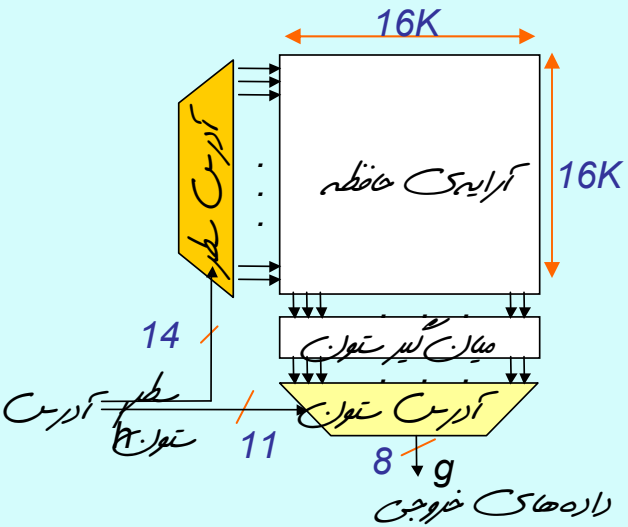


مخاطرات کنترلی (ادامه...)

- ایجاد تعلیق، موجب کندی می‌شود.
- یک راه حل، این است که فرض کنیم هیچ پرشی انجام نمی‌شود.
  - در صورت تحقق، اجرای دستورات واکشی شده، ملغی می‌گردد.
  - برای این کار سیگنال‌های کنترلی غیرفعال می‌شوند.
  - دستورالعمل‌ها از ثبات خط لوله پاک می‌شوند.

flush

ساختار حافظه‌های کنونی



- داده در آرایه مربعی ریخته می‌شود
- داده‌های یک سطر به صورت کامل خوانده می‌شوند.
- بدین‌ترتیب تاخیر (latency) برای خواندن کلمات پی‌درپی کاهش می‌یابد

Double data rate (DDR) DRAM

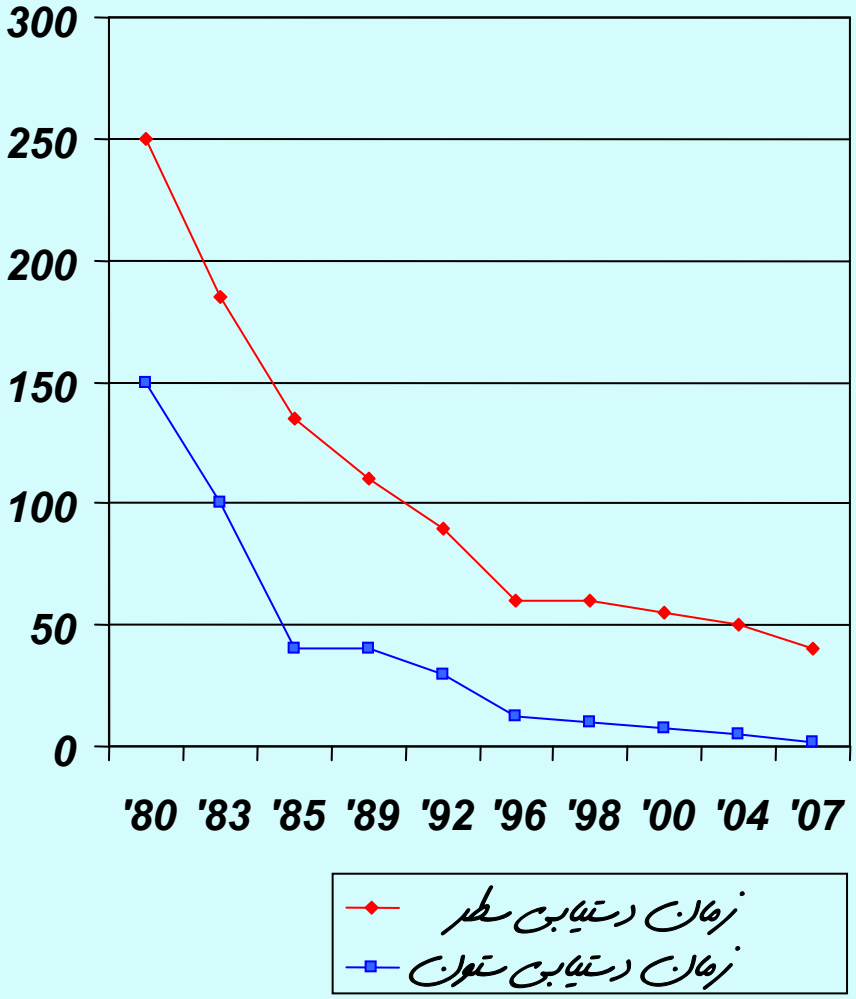
در هر دوره‌ی پالس ساعت داده خوانده می‌شود، دراین شیوه داده به صورت interleaved سامان‌دهی شده است

دارای درگاه‌های ورودی و خروجی مجزاست

Quad data rate (QDR) DRAM

نسل‌های مختلف DRAM

Year	Capacity	$/GB
1980	64Kbit	$1500000
1983	256Kbit	$500000
1985	1Mbit	$200000
1989	4Mbit	$50000
1992	16Mbit	$15000
1996	64Mbit	$10000
1998	128Mbit	$4000
2000	256Mbit	$1000
2004	512Mbit	$250
2007	1Gbit	$50



اندازه‌گیری کارایی حافظه‌ی نهان (نهان‌گاه)

زمان صرف شده توسط پردازنده

زمان اجرای دستورالعمل‌ها

زمان‌هایی که دیگر تعلیق می‌شود

بیشتر به علت cache miss

شامل زمان دسترسی به حافظه‌ی نهان در صورت وجود داده (hit)

$$\text{CPU Time} = (\text{CPU execution clock cycles} + \text{Memory-stall clock cycles}) \times \text{Clock cycle time}$$

$$\text{Memory-stall clock cycles} = \text{Read-stall cycle} + \text{Write stall cycle}$$

$$\text{Read-stall cycle} = \text{Reads} \times \text{Read rate miss rate} \times \text{Read miss penalty}$$

per program

$$\text{Write-stall cycle} = \text{Writes} \times \text{Write rate miss rate} \times \text{Write miss penalty} + \text{Write buffer stall}$$

per program

اندازه‌گیری کارایی حافظه‌ی نهان (ادامه...)

با ساده‌سازی شرایط

Memory stall cycles

$$= \text{Memory accesses} \times \text{Miss rate} \times \text{Miss penalty}$$

per program

$$= \text{Instructions} \times \text{Misses} \times \text{Miss penalty}$$

per program per Instruction

مثال

- سیستمی با شرایط زیر مفروض است:
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- در صورتی که یک حافظه‌ی نهان ایده‌آل جایگزین کینم، افزایش سرعت چه مقدار خواهد بود؟

جریمه‌ی فقدان دستوالعمل

$$0.02 \times I \times 100 = 2 \times I$$

جریمه‌ی فقدان داده

$$0.36 \times I \times 0.04 \times 100 = 1.44 \times I$$

زمان اجرا | زمان تعلیق

$$2 \times I + 2 \times I + 1.44 \times I = 5.44 \times I$$

Actual CPI

$$5.44/2 = 2.72$$

بهبود کرایی

ادامه مثال

در صورتی که سرعت پردازنده را افزایش دهیم، بدون این که در سیستم حافظه تغییری ایجاد نشود، چه اتفاقی خواهد افتاد؟

قانون Amdhal

$$T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$$

- در صورتی که سرعت پردازنده‌ی مثال را دو برابر کنیم، در شرایطی که فرکانس پالس ساعت تغییر نکند، CPI چه تغییری خواهد کرد؟

$1 \times I + 3.44 \times I = 4.44 \times I$

زمان صرف شده برای تعلیق حافظه

حالت اول: $\frac{3.44}{5.44} = 63\%$

حالت دوم: $\frac{3.44}{4.44} = 77\%$

Average Memory Access Time

متوسط زمان دست‌یابی

AMAT

- از سوی دیگر، افزایش کلاک پردازنده نیز منجر به عدم بهره‌وری (در اثر نبود اطلاعات) در حافظه‌ی نهان خواهد شد.
- زمان hit time، نیز می‌تواند زمان کل دستیابی به حافظه را افزایش دهد. این مسأله هنگامی رخ می‌دهد که گنجایش حافظه‌ی نهان افزایش یابد.
- با توجه دشواری‌های مطرح شده، گاهی معیار زیر تعریف می‌شود:

$$AMAT = Hit\ time + Miss\ rate \times Miss\ penalty$$

مثال: یک پردازنده با کلاک 1ns مفروض است، زمان مشخص شدن hit یک سیکل، جریمه‌ی فقدان بیست سیکل است. نرخ فقدان پنج درصد است، متوسط زمان دست‌یابی را حساب کنید.

$$AMAT = 1 + 0.05 \times 20 = 2ns$$

- دستیابی به بلوک‌های زیر در حافظه‌ی اصلی را در نظر بگیرید:

0 4 0 4 0 4 0 4

0 miss

00	Mem(0)

4 miss

01	
00	Mem(0) 4

0 miss

00	
01	Mem(4) 0

4 miss

01	
00	Mem(0) 4

0 miss

00	
01	Mem(4) 0

4 miss

01	
00	Mem(0) 4

0 miss

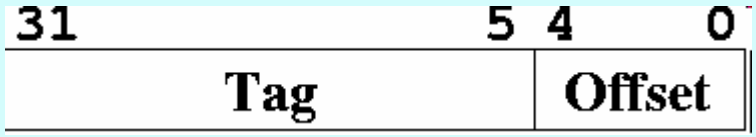
00	
01	Mem(4) 0

4 miss

01	
00	Mem(0) 4

حافظه‌ی نهان (اشتراکی، انجمنی)

- تا کنون از شیوه‌ی نگاشت مستقیم برای جای‌گذاری بلوک‌ها استفاده کردیم.
  - شیوه‌ی دیگر این است که هر بلوک بتواند در هر جای حافظه‌ی نهان قرار بگیرد. این شیوه به نام انجمنی (associative) شهرت دارد.
  - در این صورت بر طول برچسب افزوده می‌شود و تمامی برچسب‌ها می‌باید مورد بررسی قرار گیرند.
  - برای چنین کاری از مقایسه‌کننده‌های موازی استفاده می‌شود که هزینه‌ی سخت‌افزاری بالایی دارد.



حافظه‌ی نهان با مجموعه‌های اشتراکی

- یک راه دیگر برای کاهش هزینه‌ها، استفاده از راهی بین نکاشت مستقیم و حافظه‌ی نهان اشتراکی است:
  - حافظه‌ی نهان با مجموعه‌های اشتراکی
- در این شیوه هر بلوک از حافظه‌ی اصلی در مکان‌های خاصی از حافظه‌ی اصلی می‌توانند قرار گیرند.
- در صورتی که هر بلوک در n محل از حافظه‌ی نهان قابل جای‌گذاری باشد آن را *n*-way set associative می‌نامند.
- در مقابل شیوه‌ی پیشین به اشتراکی کامل (Fully associative) مشهور است.

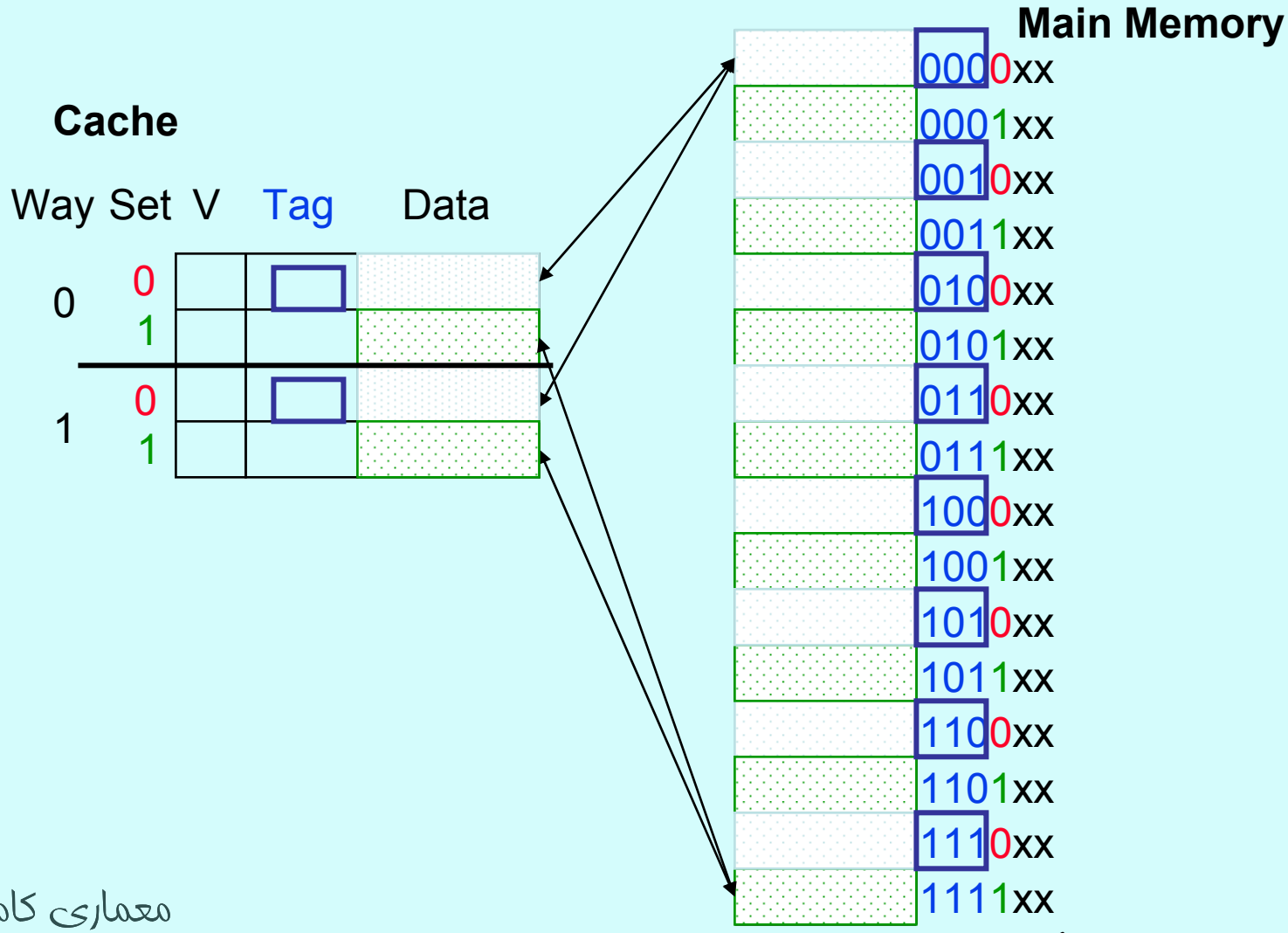
حافظه‌های نهان اشتراکی

(Block address) modulo (#Blocks in cache)

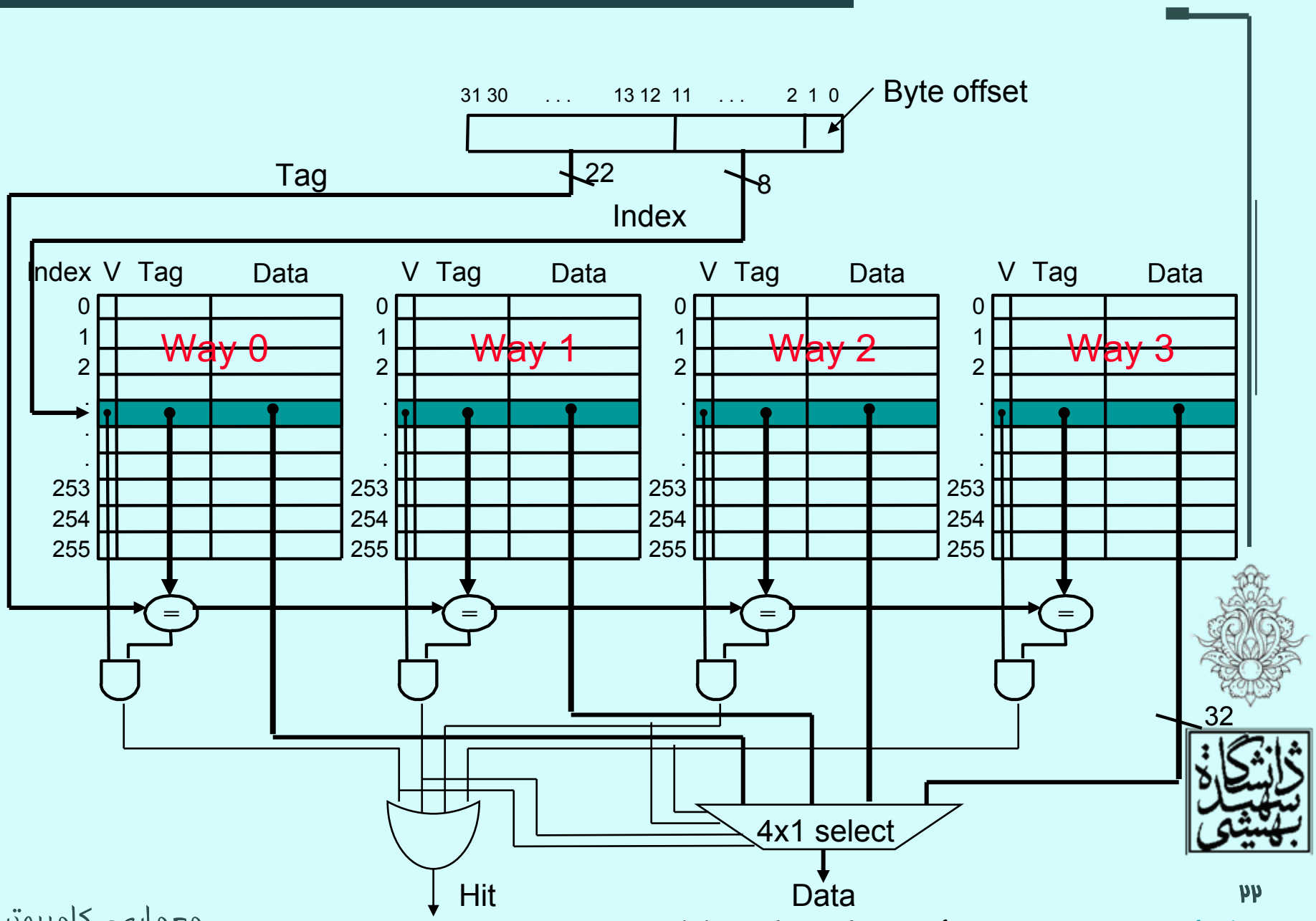
Direct Map

(Block address) modulo (# Sets in cache)

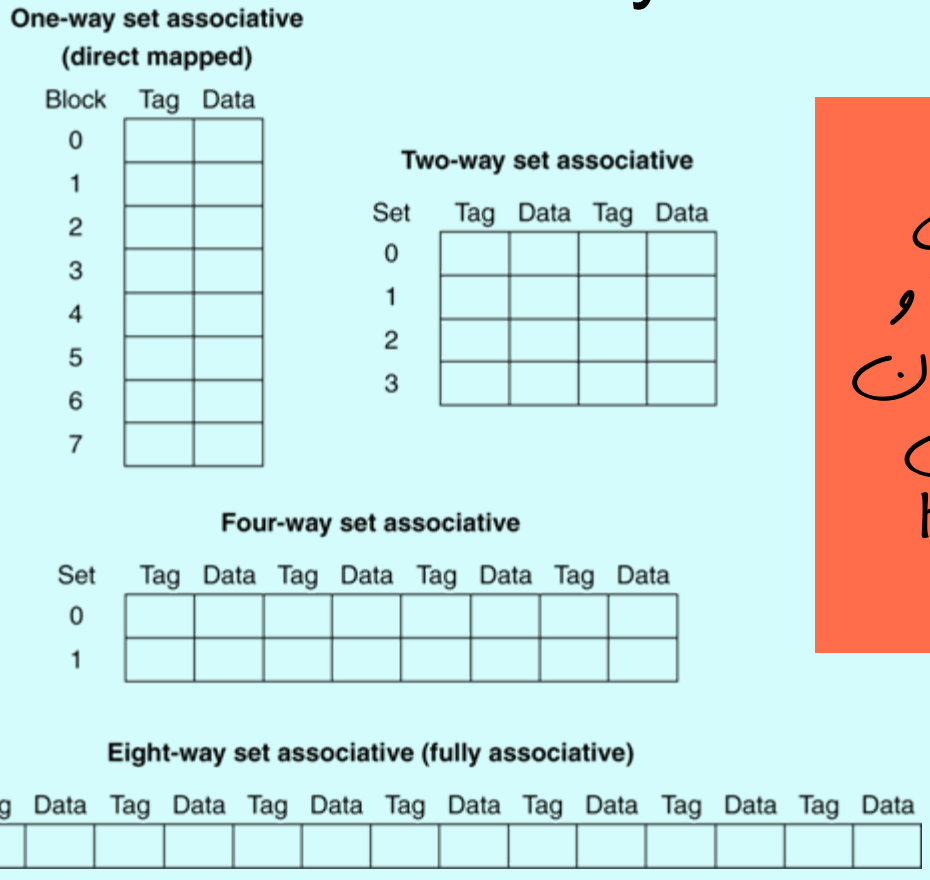
Set Associative


Main Memory
Cache
Way Set V Tag Data
0
1
0
1
0
1
0000xx
0001xx
0010xx
0011xx
0100xx
0101xx
0110xx
0111xx
1000xx
1001xx
1010xx
1011xx
1100xx
1101xx
1110xx
1111xx

Four-Way Set Associative Cache

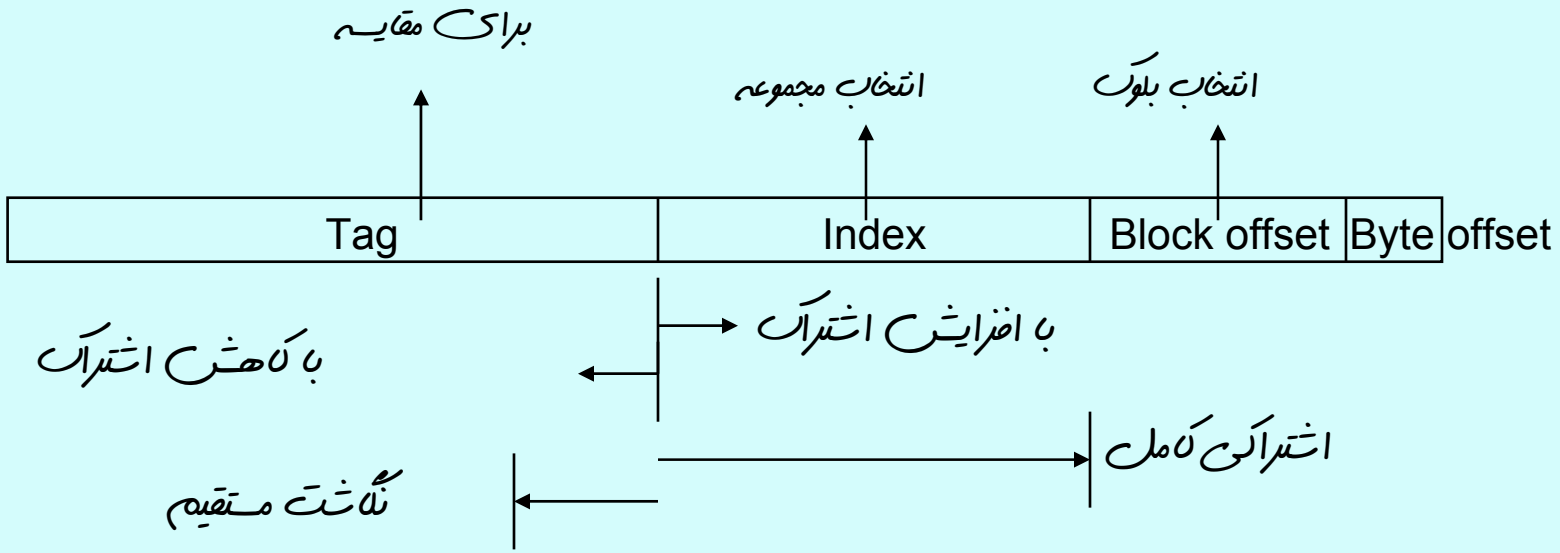


- تمام شیوه‌های جای‌گذری را به نوعی می‌توان n-way set associative دانست.



مهمترین برتری شیوه‌های اشتراکی کاهش miss-rate و بزرگ‌ترین مشکل آن افزایش مشکل آن افزایش hit-time است

طیف اشتراک (ادامه...)



مثال

- یک حافظه‌ی نهان چهار بلوکی در اختیار داریم،
- هدف مقایسه‌ی miss-rate در حالات زیر است:
  - نگاشت مستقیم
  - اشتراکی دو بلوکه
  - اشتراکی کامل
- ترتیب بلوک‌های به صورت زیر است:
  - 0, 8, 0, 6, 8

Direct mapped

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	Mem[0]			
8	0	miss	Mem[8]			
0	0	miss	Mem[0]			
6	2	miss	Mem[0]		Mem[6]	
8	0	miss	Mem[8]		Mem[6]	

مثال

2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	**Mem[0]**			
8	0	miss	Mem[0]	**Mem[8]**		
0	0	hit	**Mem[0]**	Mem[8]		
6	0	miss	Mem[0]	**Mem[6]**		
8	0	miss	**Mem[8]**	Mem[6]		

Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			
8		miss	Mem[0]	**Mem[8]**		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	**Mem[6]**	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	

Fully associative

قابلیت اشتراک تا چه حد؟

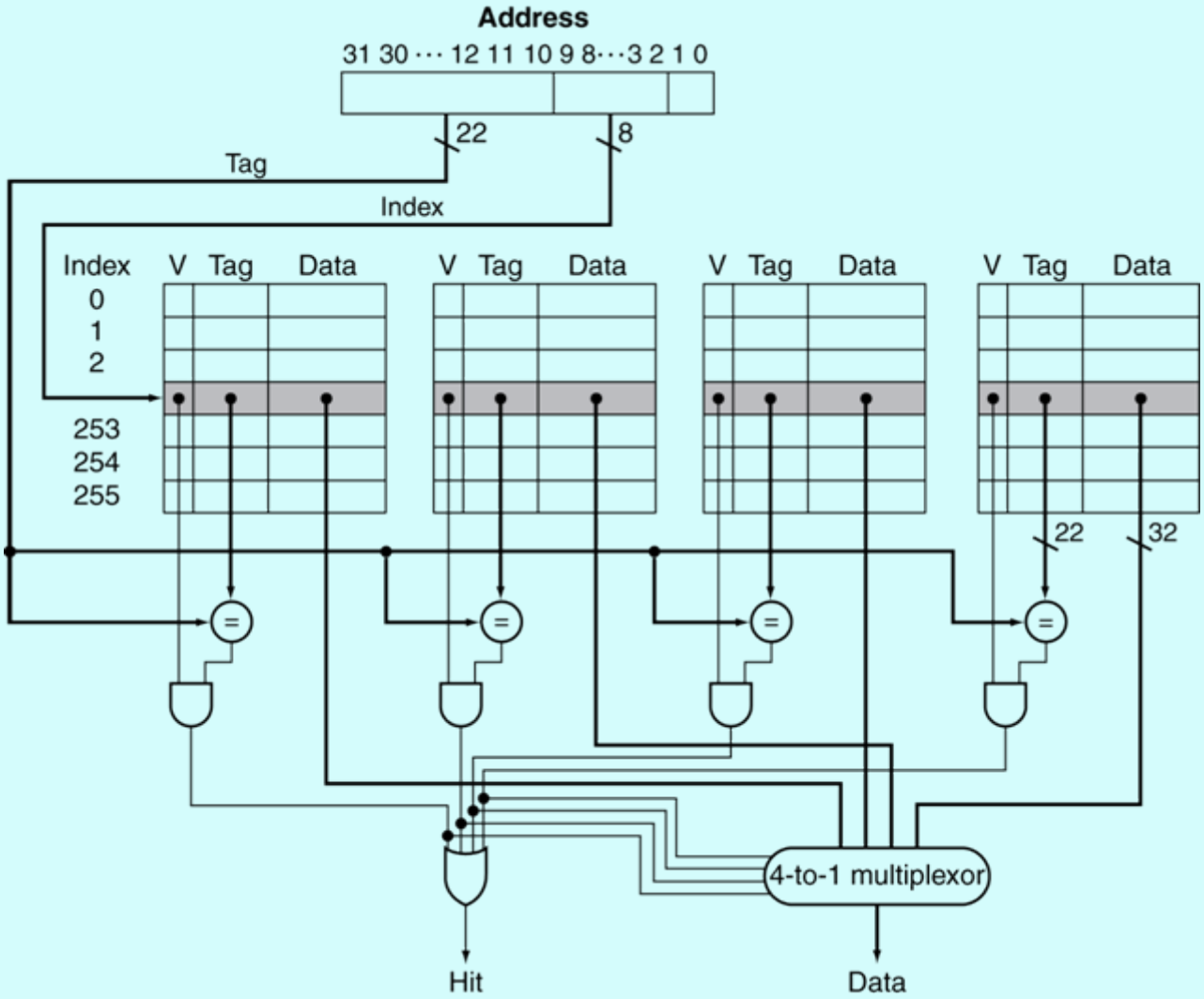
- هر چه قابلیت اشتراک بیشتر باشد، نرخ miss-rate کاهش می‌یابد.
- تا چه حد این قابلیت را افزایش دهیم؟
- نتایج شبیه‌سازی یک سیستم، با 64KB و بلوک‌های شانزده کلمه‌ای که با SPEC2000:

سوال ؟؟

1-way: 10.3%
2-way: 8.6%
4-way: 8.3%
8-way: 8.1%

هر چند miss-rate کاهش می‌یابد، اما روند این کاهش، رفته رفته کم‌تر می‌شود

ساختار حافظه‌ی نهان با مجموعه‌های اشتراکی



در حافظه‌های نهان انجمنی، زمان جستجو اهمیت بسزایی در کارایی دارد

معماری کامپیوتر (۱۳۹۱-۱۱-۱۳)

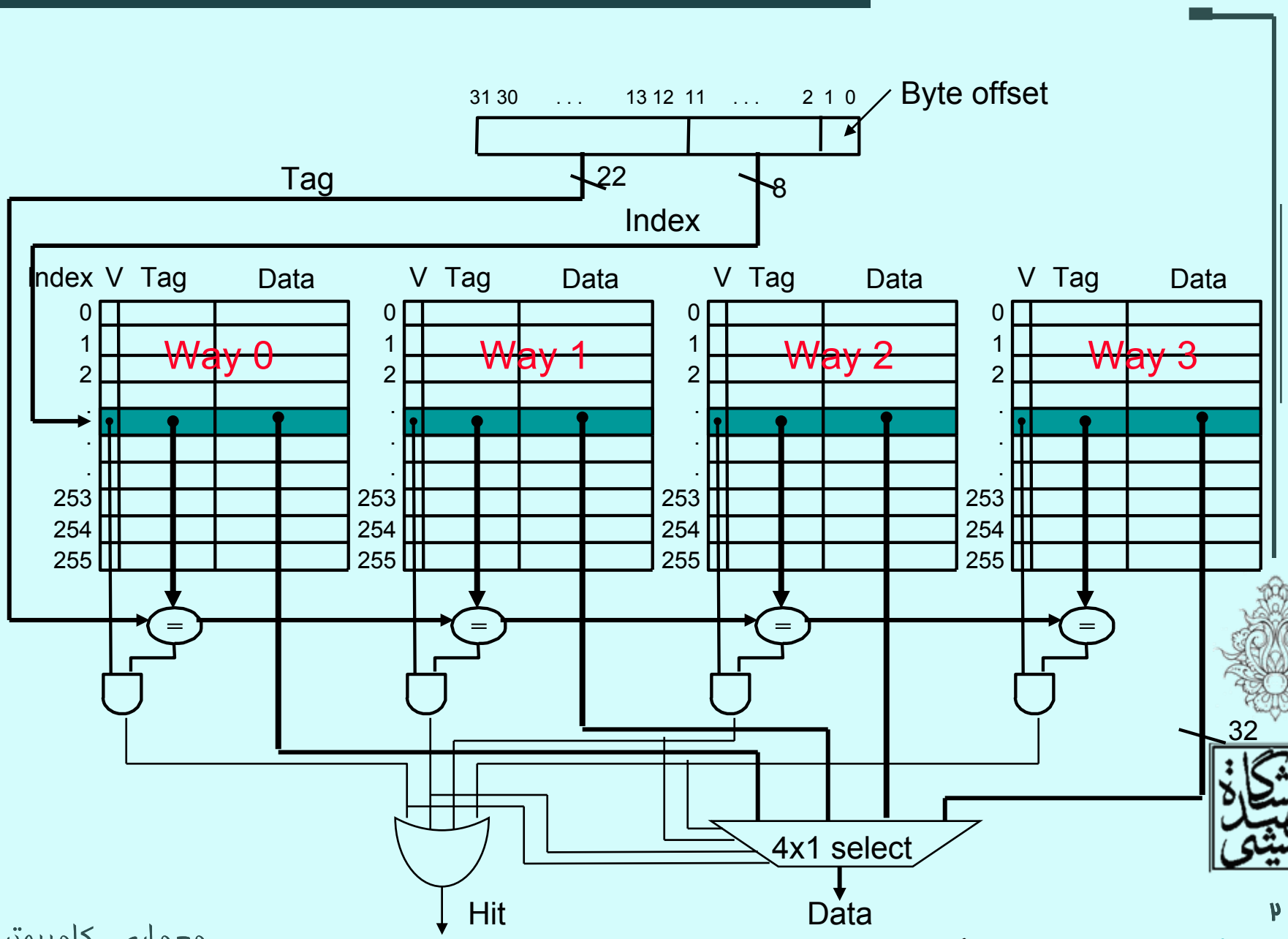
جلسه‌ی بیست و سوم

دانشگاه شهید بهشتی
دانشکده‌ی مهندسی برق و کامپیوتر
بهار ۱۳۹۱
احمد محمودی ازناوه

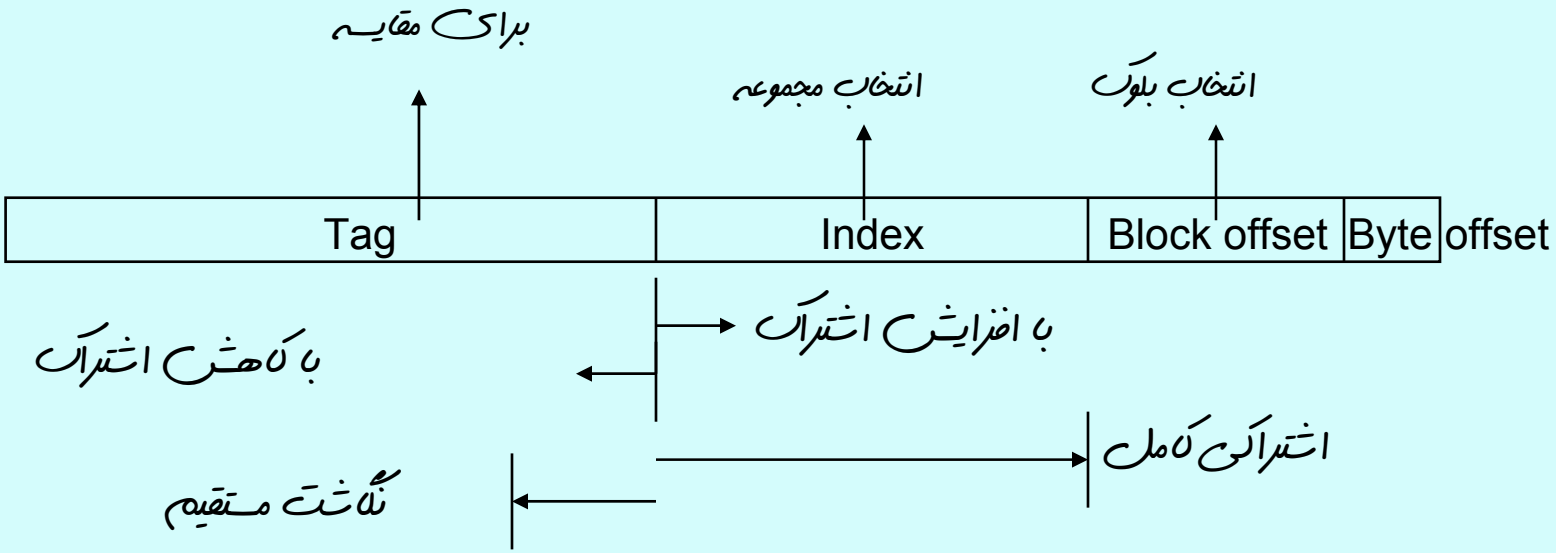
فهرست مطالب

- حافظه‌ی تداعی‌گر
- سیاست‌های جایگزینی
- حافظه‌ی نهان چند سطحی
- حافظه‌ی مجازی

Four-Way Set Associative Cache



طیف اشتراک (ادامه...)



Content Addressable Memory (CAM)

Associative memory

حافظه‌ی انجمنی!!

حافظه‌ی تداعی‌گر

- در بسیاری کاربردها، لازم است یک آیتم در یک جدول (حافظه) جستجو شود، که فرآیندی زمان‌بر است.
- در صورتی که بتوان حافظه‌ای ساخت که با ارائه‌ی داده، آدرس را بیابد، کارایی فرآیند جستجو به صورت مؤثری بهبود خواهد یافت. چنین حافظه‌ای، حافظه‌ی تداعی‌گر خوانده می‌شود.
- این نوع حافظه‌ها، هزینه‌ی بالاتر نسبت به حافظه‌های معمولی دارند، و بدین‌سبب در کاربردهایی که زمان جستجو نقشی حیاتی دارد، به کار می‌روند.

در حافظه‌های نهان انجمنی، علاوه بر داده، (بخشی) از آدرس را نیز ذخیره می‌کند. چنین حافظه‌ای از یک حافظه‌ی معمولی و یک حافظه‌ی تداعی‌گر تشکیل شده است.

Replacement Policy

سیاست جایگزینی

- در نگاشت مستقیم، جایی که بلوک باید در آن قرار گیرد مشخص است.
- در روش‌های اشتراکی بلوک در چند محل متفاوت می‌تواند قرار گیرد.
  - در درجه‌ی اول مکانی انتخاب می‌شود که بیت‌اعتبار آن غیر فعال است.
  - در غیر این صورت از بلوکی که کم‌تر مورد استفاده قرار گرفته است، از حافظه‌ی نهان خارج می‌شود.

هر چه تعداد مجموعه‌های مشترک افزایش یابد، هزینه‌ی سخت‌افزاری LRU افزایش خواهد یافت.

Least-recently used (LRU)

راه دیگر، انتخاب تصادفی است

برای حالت اشتراک با مجموعه‌های بزرگ کارایی یکسانی با LRU دارد

مثال

- یک حافظه‌ی نهان با شرایط زیر مفروض است:
  - 4K blocks, 4-word block size, 32 bit address
- تعداد مجموعه‌ها و طول برچسب را برای حالات زیر حساب کنید؟
  - نگاشت مستقیم، حافظه‌ی نهان اشتراکی دوبلوکی، حافظه‌ی نهان اشتراکی چهار بلوکی و حافظه‌ی اشتراکی کامل به دست آورید.

نگاشت مستقیم

16(=2^4) byte per block

برای آدرس شخص و برچسب 32-4=28

log2(4K)=12

تعداد بلوک‌ها در نگاشت مستقیم طول شاخص را مشخص می‌کنند

28-12=16

تعداد بیت‌های برچسب

مثال (ادامه...)

با افزایش درجه‌ی اشتراک، تعداد بیت‌های شاخص کاهش یافته و بیت‌های برچسب افزایش خواهد یافت.
بنابراین برای اشتراک با دو بلوک 2K مجموعه خواهیم داشت.

28-log2(2K)=17

2-way associative

28-log2(1K)=18

4-way associative

28

fully associative

حافظه‌ی نهان چند سطحی

- حافظه‌های نهان متصل به پردازنده‌ها
  - کوچک، اما بسیار سریع هستند.
- حافظه‌ی نهان سطح ۲ (levlel 2 cache)
  - در صورتی که در حافظه‌ی نهان سطح ۱ داده موجود نباشد، این سطح پاسخگو خواهد بود.
  - بزرگ‌تر، اما کندتر هستند، ولی در هر حال از حافظه‌ی اصلی سریع‌تر هستند.
- حافظه‌ی اصلی پاسخگوی نبود داده در حافظه‌ی نهان سطح ۲ می‌باشد.

مثال

- سیستمی با مشخصات زیر مفروض است:
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- در صورتی که از یک سطح حافظه‌ی نهان استفاده کنیم:

Miss penalty = 100ns/0.25ns = 400 cycles

Effective CPI = 1 + 0.02 × 400 = 9

مثال (ادامه...)

- با افزودن یک سطح دیگر حافظه‌ی نهان با مشخصات زیر:
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%

Penalty = 5ns/0.25ns = 20 cycles

CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4

Performance ratio = 9/3.4 = 2.6

Total CPI= Base CPI+ Primary stalls per instruction

\+

Secondary stalls per instruction

حافظه‌ی نهان چند سطحی

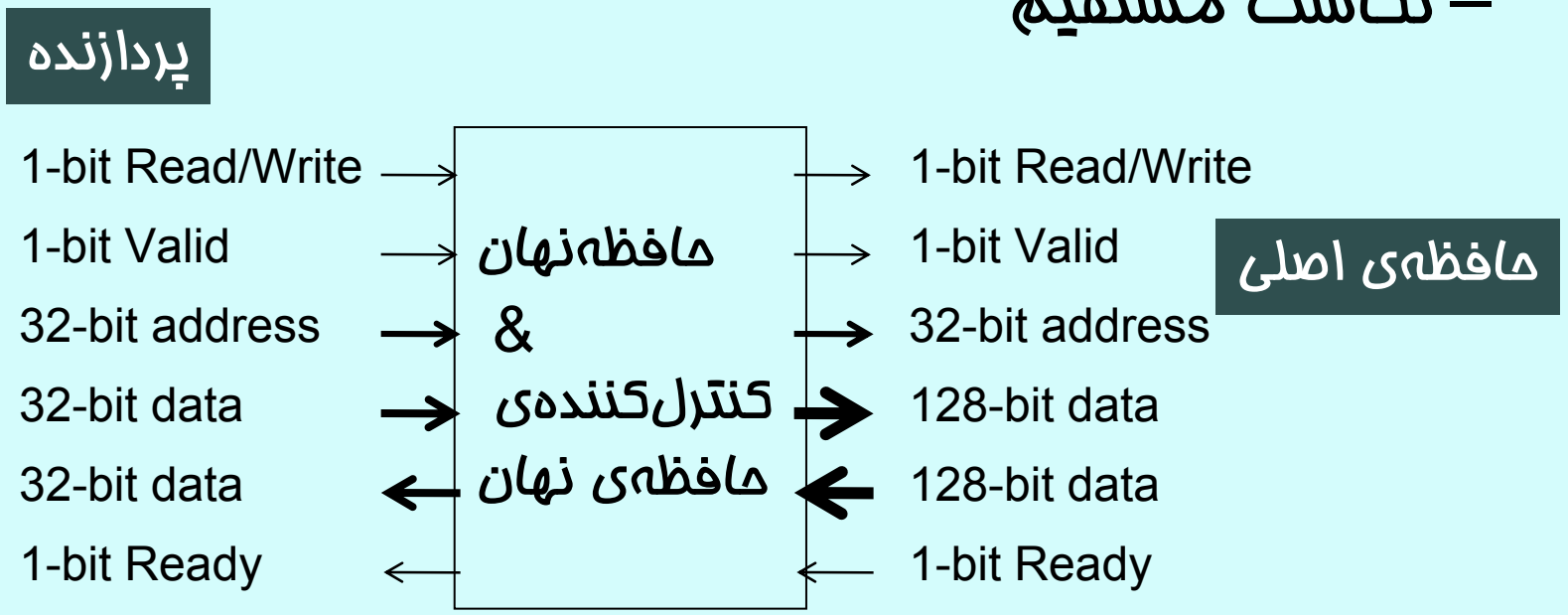
- **حافظه‌ی نهان سطح ۱**
  - تمرکز بر روی hit time
- **حافظه‌ی سطح ۲**
  - تمرکز بر روی کاهش miss rate است

حافظه‌ی نهان در دو پردازنده‌ی واقعی

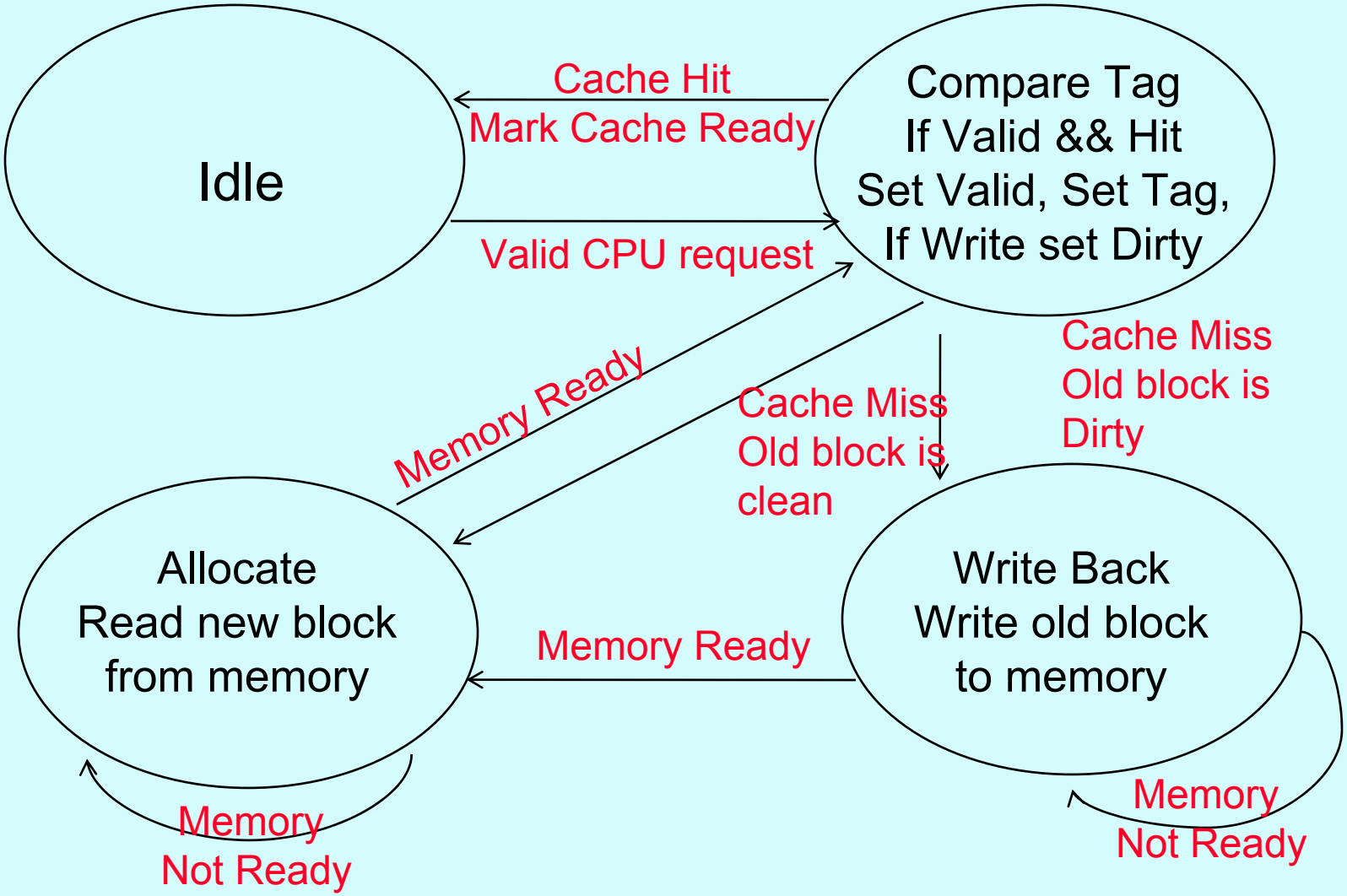
	Intel Nehalem	AMD Barcelona
L1 cache organization & size	Split I$ and D$; 32KB for each per core; 64B blocks	Split I$ and D$; 64KB for each per core; 64B blocks
L1 associativity	4-way (I), 8-way (D) set assoc.; ~LRU replacement	2-way set assoc.; LRU replacement
L1 write policy	write-back, write-allocate	write-back, write-allocate
L2 cache organization & size	Unified; 256MB (0.25MB) per core; 64B blocks	Unified; 512KB (0.5MB) per core; 64B blocks
L2 associativity	8-way set assoc.; ~LRU	16-way set assoc.; ~LRU
L2 write policy	write-back	write-back
L2 write policy	write-back, write-allocate	write-back, write-allocate
L3 cache organization & size	Unified; 8192KB (8MB) shared by cores; 64B blocks	Unified; 2048KB (2MB) shared by cores; 64B blocks
L3 associativity	16-way set assoc.	32-way set assoc.; evict block shared by fewest cores
L3 write policy	write-back, write-allocate	write-back; write-allocate

کنترل حافظه‌ی نهان

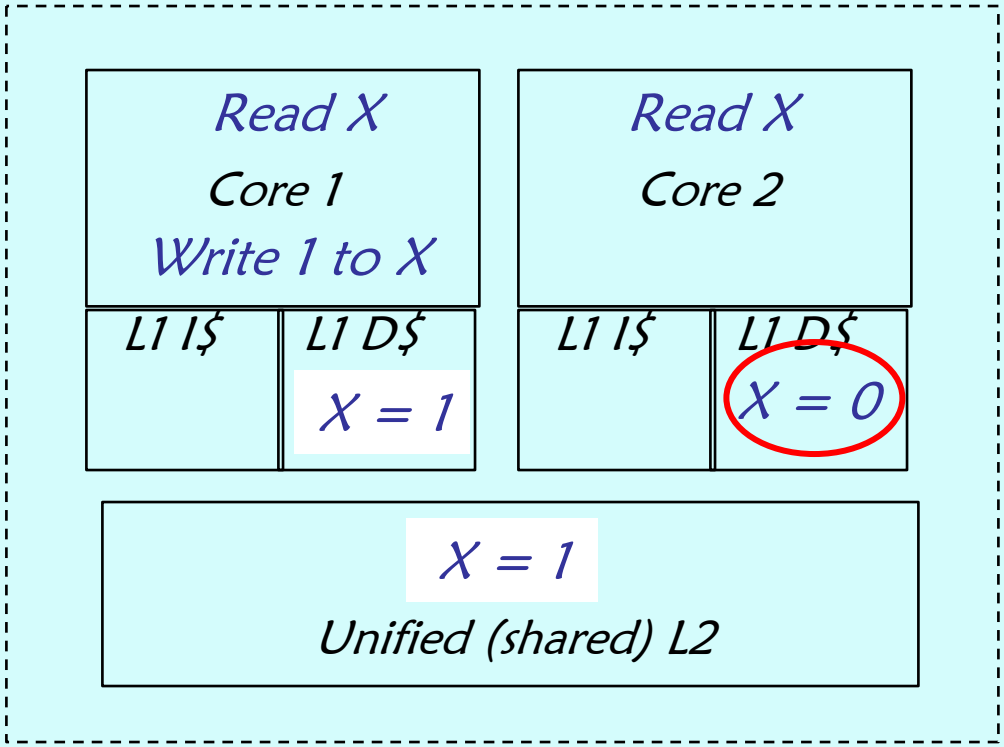
- یک حافظه‌ی نهان با مشخصات زیر مفروض است:
  - Write back
  - اندازه‌ی بلوک‌ها چهار کلمه
  - اندازه‌ی حافظه‌ی نهان 16KB
  - نگاشت مستقیم



نمودار حالت کنترل حافظه‌ی نهان

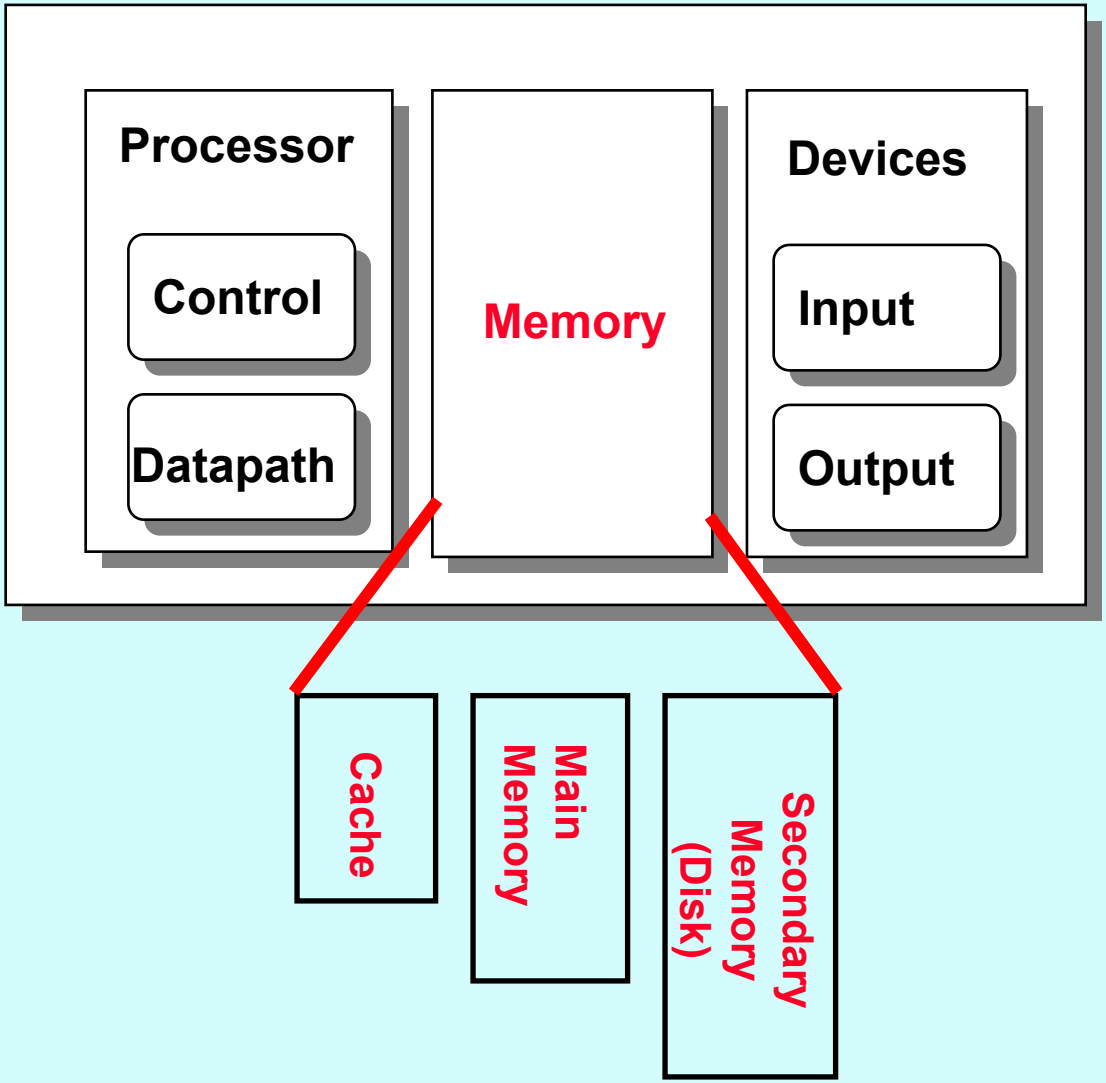


حافظه‌ی نهان در پردازنده‌های چند هسته‌ای



cache coherence problem

ساختار کلی یک کامپیوتر



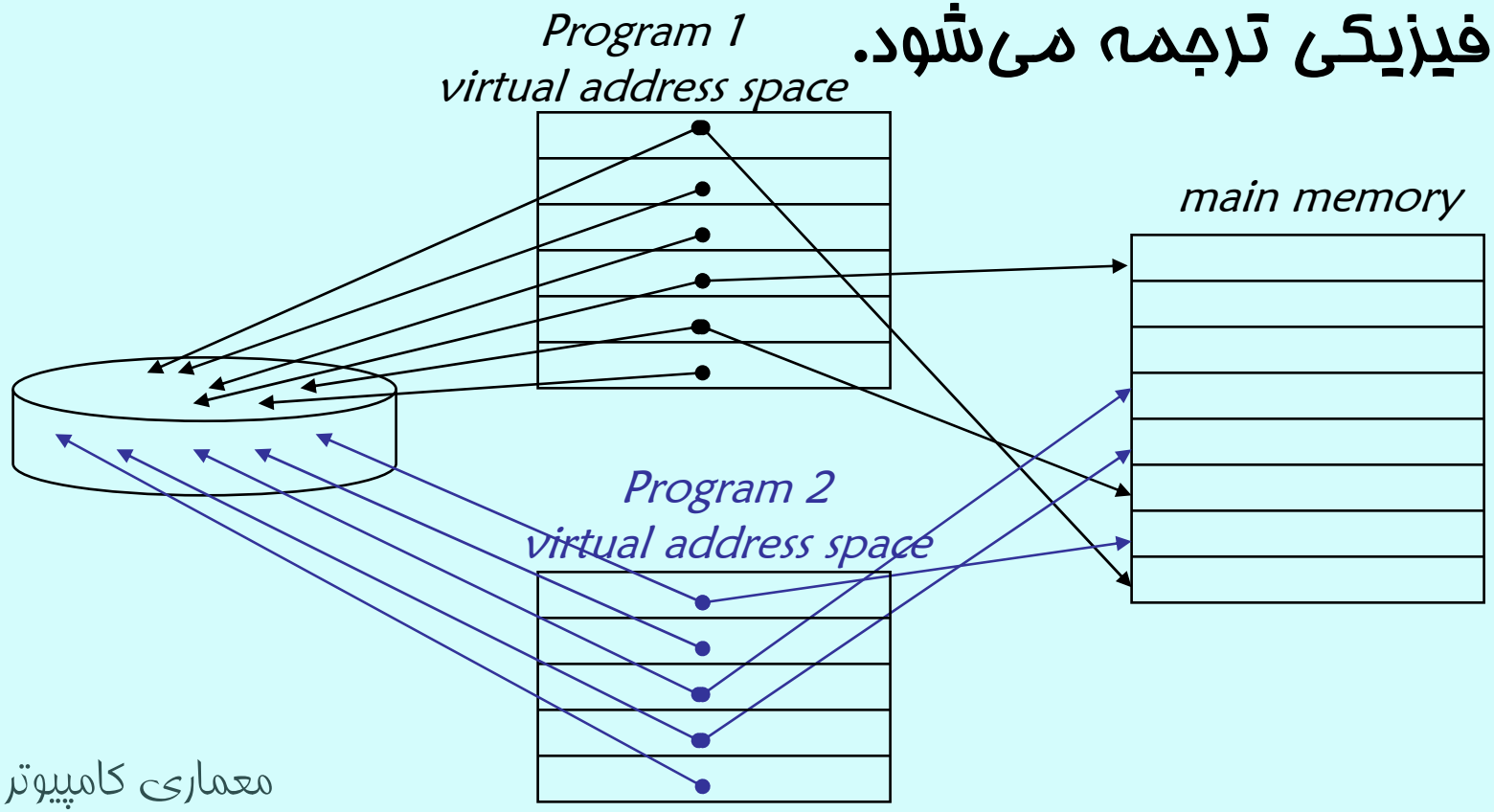
سلسله مراتب در حافظه‌ی اصلی

- ثبات‌ها و حافظه‌ی نهان
  - کامپایلر یا برنامه‌نویس
- حافظه‌ی نهان و حافظه‌ی اصلی
  - کنترل‌کننده‌ی حافظه‌ی نهان
- حافظه‌ی اصلی و حافظه‌ی ثانویه

حافظه‌ی مجازی

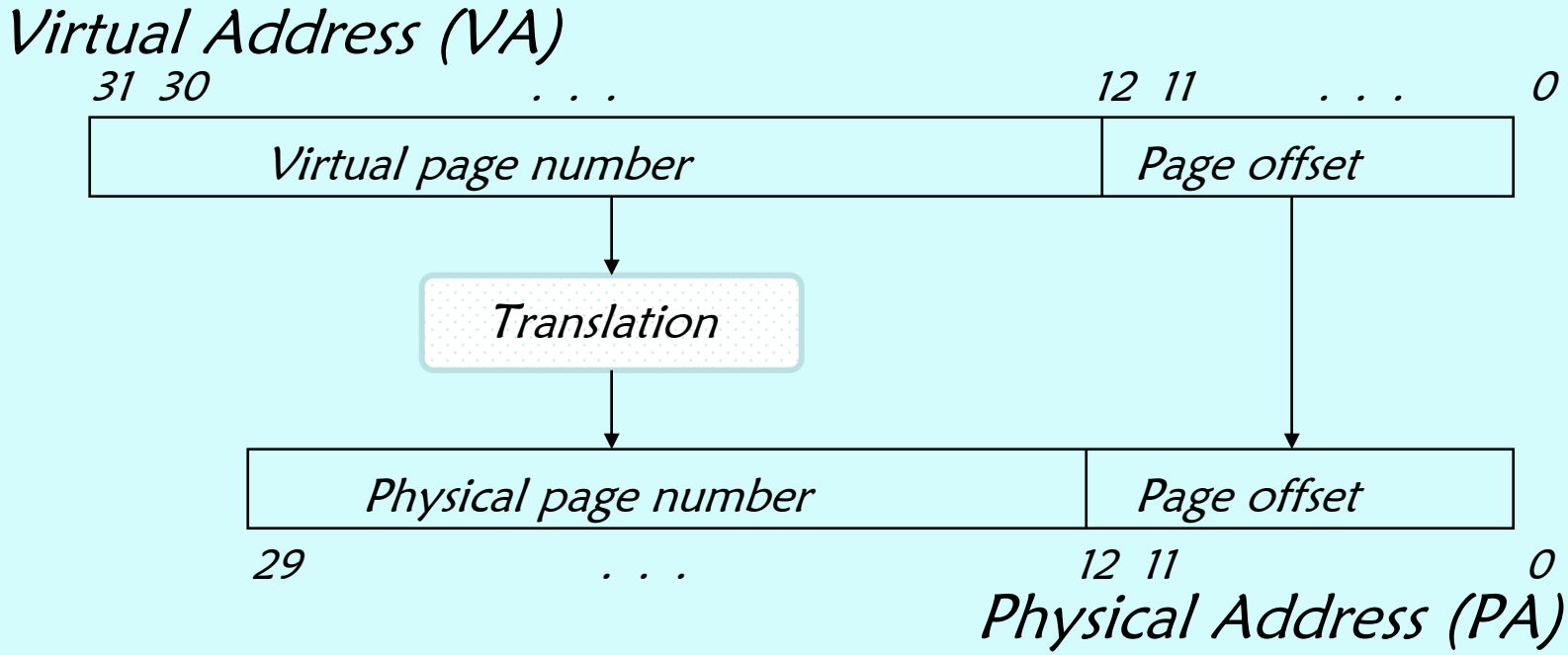
- حافظه‌ی اصلی نقشی مانند حافظه‌ی نهان را برای حافظه‌ی اصلی ایفا می‌کند.
  - مدیریت آن به صورت مشترک توسط پردازنده و سیستم‌عامل صورت می‌پذیرد.
- با کمک آن می‌توان به گونه‌ای کارا و امن حافظه را بین چندین برنامه به اشتراک گذاشت.
- می‌توان به کمک آن برنامه‌هایی را اجرا کرد، که دارای حجمی بیش از حجم حافظه‌ی فیزیکی هستند.
- بارگذاری برنامه در حافظه با سهولت بیش‌تری صورت می‌گیرد.

- در واقع به هر برنامه در زمان کامپایل فضایی اختصاص داده می‌شود.
- در هنگام اجرای برنامه آدرس مجازی به آدرس فیزیکی ترجمه می‌شود.

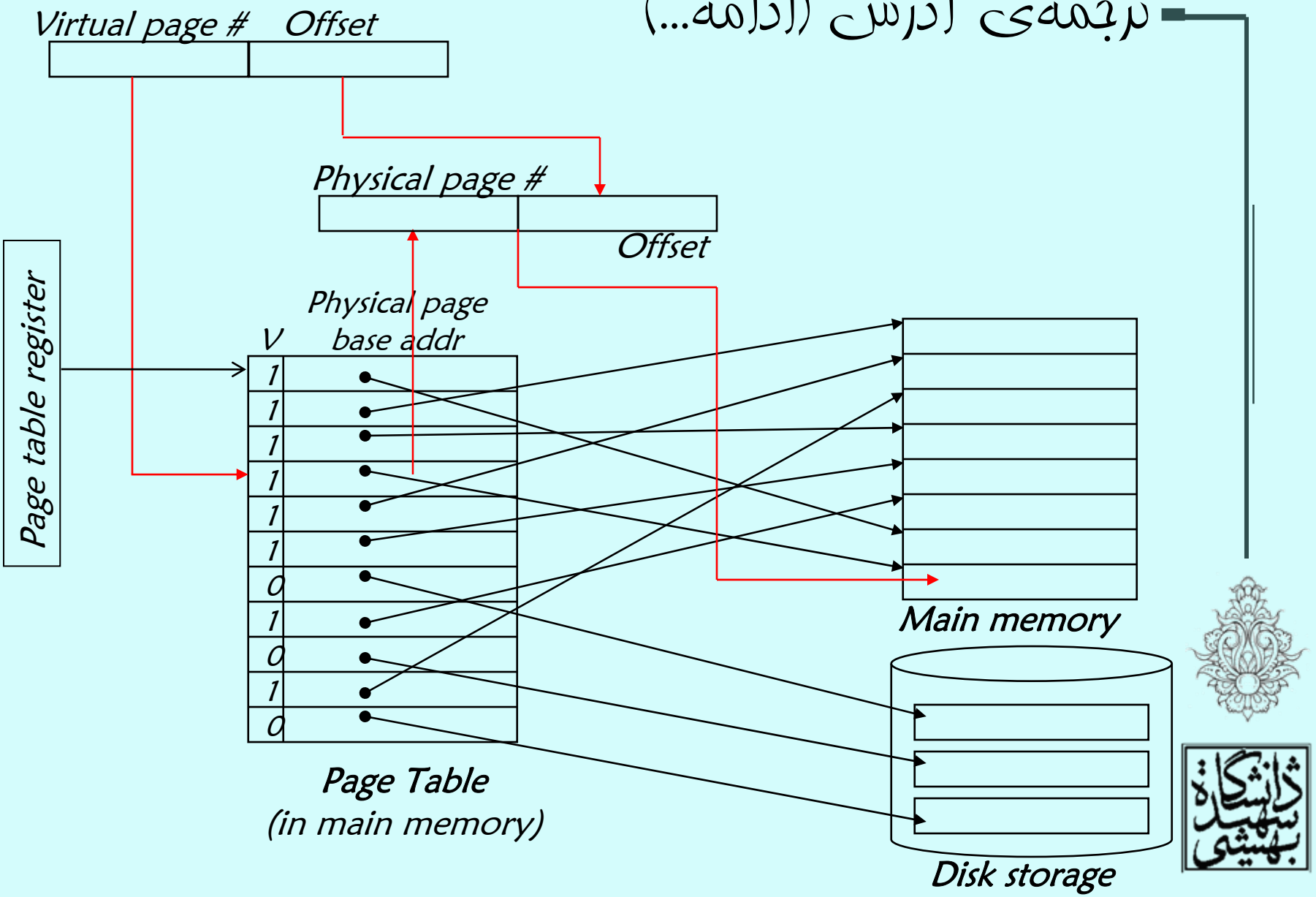


ترجمه‌ی آدرس

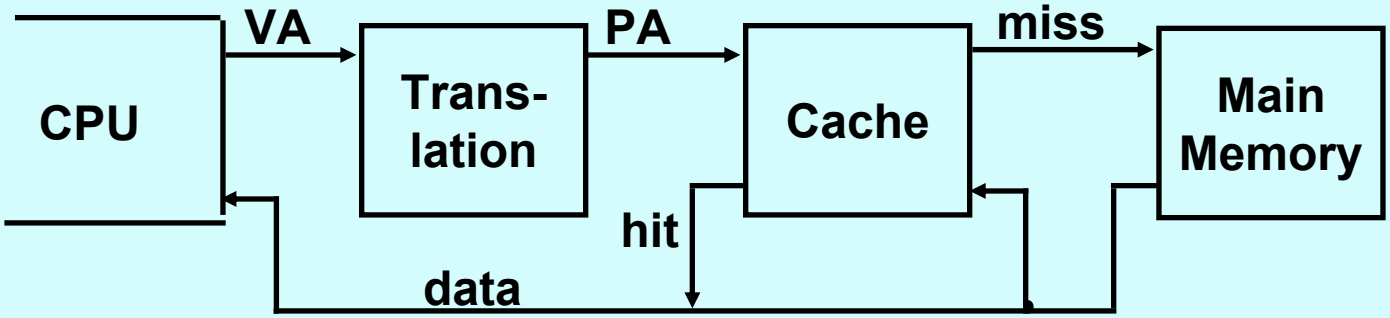
- ترجمه‌ی آدرس با همکاری پردازنده و سیستم‌عامل صورت می‌پذیرد.
- در صورتی که داده در حافظه‌ی اصلی نباشد، «page fault» رخ می‌دهد.



ترجمه‌ی آدرس (ادامه...)



ترجمه‌ی آدرس (ادامه...)



- با این حساب عمل دستیابی به حافظه نهان خیلی زمان‌بر خواهد شد!
- با کمک سخت‌افزار و در نظر گرفتن یک میان‌گیر این مشکل برطرف می‌شود.

Translation Lookaside Buffer

ترجمه‌ی آدرس (ادامه...)