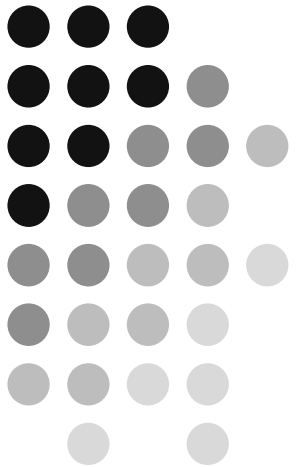
معماری کامپیوتر

ساختار CPU



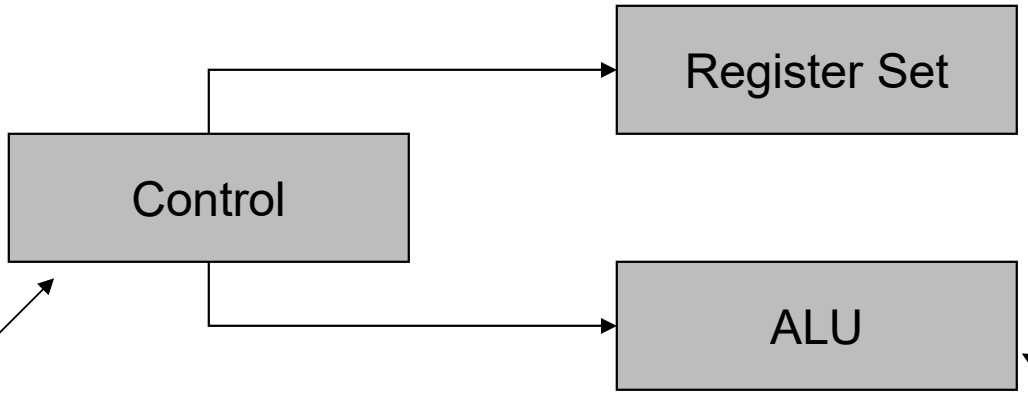
فصل هشتم کتاب موریس مانو

محمدعلی شفیعیان

http://shafieian-education.ir

بهار ۹۸

اجزای تشکیل دهنده CPU



داده ها را ذخیره میکند

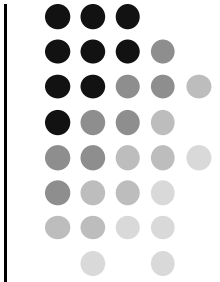
مشخص میکند که چه عملی در چه زمانی انجام گیرد.
بصورت سخت افزاری و یا میکروپروگرام سیگنال لازم را به بخشهای مختلف میفرستد.

عملیات محاسباتی و منطقی را انجام میدهد

CPUاز دید استفاده کننده

- از دید یک برنامه نویس که به زبان سطح پایین برنامه نویسی می کند یک CPU دارای مشخصه های زیر خواهد بود
  - Instruction format
  - The instruction Set
  - Addressing modes
  - Register organization

معماری های مختلف CPU



- Accumulator machine
- Register machine :PowerPC
- Stack machine :the Java virtual machine

مثال: ماشین های X86 دارای یک معماری با مجموعه دستورات پیچیده ای است که تمامی جنبه های معماری های فوق را در بر می گیرد.

اهمیت معماری CPU چقدر است؟

- استفاده کننده نهائی:
  هیچ!

- برنامه نویس سطح بالا
  خیلی کم. تا حدی که بتواند کامپایلر مناسب را انتخاب نموده و عملکرد برنامه را بهینه کند

- برنامه نویس سطح پائین/ طراح OS
  این افراد باید اطلاعات کافی در مورد رجیسترها، ساختار حافظه، انواع داده های موجود و عملکرد دستورات داشته باشند.

ساختار رجیسترها

- یکی از مهمترین ویژگی های تعیین کننده برا ی یک CPUساختار رجیسترهای داخلی آن است. این رجیستر ها به دو دسته تقسیم بندی می شوند:
- رجیسترهائی که استفاده کننده آنها را می بیند! و می تواند از طریق برنامه نویسی به آنها دسترسی داشته باشد
  - Data registers
  - Address registers
    - index register
    - segment pointer
    - stack pointer
  - Condition codes (flags)

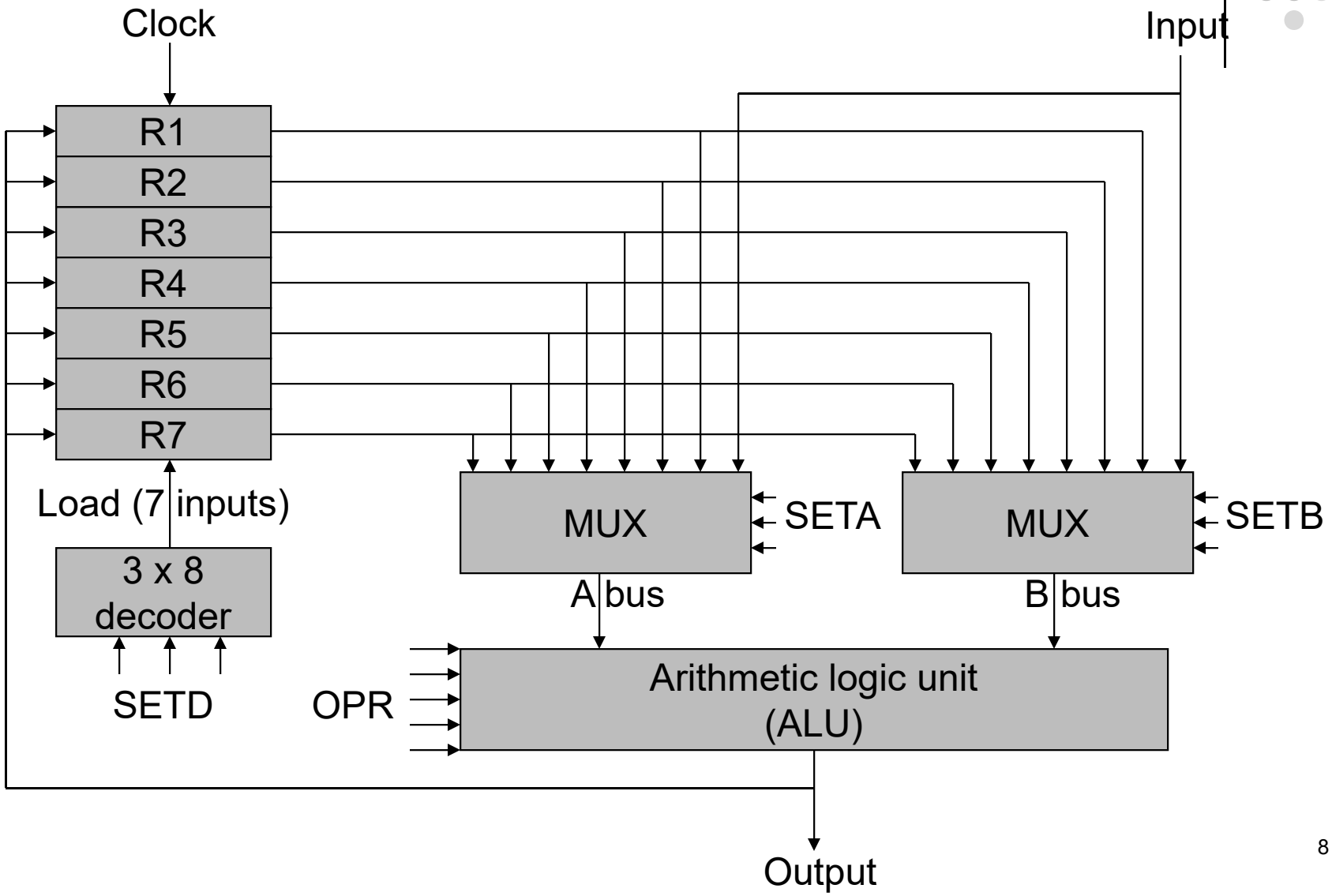
- رجیسترهائی که برای کنترل و نگهداری وضعیت CPUبکار می روند. این رجیسترها توسط واحد کنترل برای اجرای دستورات مورد استفاده واقع می شوند

  - Program counter
  - Instruction register

رجیسترهای عمومی

- محلی برای ذخیره داده ها هستند که در داخل CPU تعبیه شده اند تا با فراهم آوردن دسترسی سریع و آسان عملکرد CPUرا افزایش دهند.
- این رجیستر ها توسط یک باس مشترک مخصوص به هم وصل می شوند
- استفاده کننده می تواند از این رجیسترها برای کاربردهای مختلف محاسباتی، منطقی و شیفت استفاده نماید.

مجموعه رجیسترهای عمومی و ALUمشترک



مجموعه رجیسترهای عمومی و ALU مشترک

در شکل فوق:

- خروجی هر رجیستر به دو MUX متصل شده است. این کار باعث می شود تا هر یک از آنان را بتوان آزادانه بعنوان مبدا عملیات ALU انتخاب نمود.

- برای اینکه بتوان خروجی ALU را به هر یک از رجیستر ها منتقل نمود این خروجی به ورودی تمام رجیسترها متصل شده و علاوه بر آن با استفاده از یک دیکدر مقصد عملیات را مشخص می کنیم.

- یک ALU ممکن است که قادر به انجام عملیات مختلفی باشد، برای انتخاب یک عمل مورد نیاز از خطوط کنترلی OPR استفاده می شود.

مثالی از عملیات ALU

- برای مثال فرض کنید که می خواهیم micro-operation زیر را انجام دهیم:

$$R1 \leftarrow R2 + R3$$

- برای انجام این عمل واحد کنترل باید سیگنالهای لازم را برای انتخاب ورودیهای متناسب دیکدر MUXA, MUXB, وALU انتخاب نماید:

1. تعیین مقدار مناسب برای ورودی MUXA یعنی SELA طوری که محتوی رجیستر R2 در روی باس A قرار گیرد.
2. تعیین مقدار مناسب برای ورودی MUXB یعنی SELB طوری که محتوی رجیستر R3 در روی باس B قرار گیرد.
3. تعیین مقدار لازم برای ورودی OPR که ALU را وادار به انجام عمل جمع A+B نماید.
4. در نهیت انتخاب مقدار مناسب برای دیکدر SELD به نحوی که خروجی ALU را به رجیستر R1 منتقل نماید.

سیگناهای کنترلی

- هر ۴ دسته سیگنال فوق باید بطور همزمان توسط واحد کنترل ایجاد شده و در شروع یک سیکل کلاک آماده باشند.

- در طول یک سیکل کلاک داده رجیسترها از طریق مالتی پلکسرها و ALU به باس خروجی رسیده و آماده می شود تا با آمدن لبه پالس ساعت بعدی وارد رجیستر مقصد گردد.

- برای اینکه بتوانیم CPUسریعی داشته باشیم باید ALUبا مداراتی ساخته شود که بتوانند به سرعت عمل مورد نظر را انجام دهند

کلمه کنترل

SELA	SELB	SELD	OPR

Binary code	*SELA*	*SELB*	*SELD*
000	***Input***	***Input***	***None***
001	***R1***	***R1***	***R1***
010	***R2***	***R2***	***R2***
011	***R3***	***R3***	***R3***
100	***R4***	***R4***	***R4***
101	***R5***	***R5***	***R5***
110	***R6***	***R6***	***R6***
111	***R7***	***R7***	***R7***

- در مثال نشان داده شده تعداد ۱۴ متغیر باینری وجود دارند که مقدار آنها باید توسط واحد کنترل تعیین گردد. هر ترکیب این متغیرها یک کلمه کنترل نامیده میشود. این کلمه به ۴ فیلد تقسیم می شود

عملیات ALU

- در CPU انجام عملیات محاسباتی و منطقی بر عهده ALU است. عمل شیفت را می توان توسط یک Shifter که قبل و یا بعد از ALU قرار می گیرد انجام داد. در مواردی هم ممکن است عمل شیفت توسط خود ALUانجام شود. در فصل ۴ طراحی چنین ALU را دیدیم که عملیات آن در جدول مقابل ذکر شده است

OPR Select	*Operation*	*Symbol*
00000	***Transfer A***	***TSFA***
00001	***Increment A***	***INCA***
00010	***Add A + B***	***ADD***
00101	***Subtract A – B***	***SUB***
00110	***Decrement A***	***DECA***
01000	***And A and B***	***AND***
01010	***OR A and B***	***OR***
01100	***XOR A and B***	***XOR***
01110	***Complement A***	***COMA***
10000	***Shift right A***	***SHRA***
11000	***Shift left A***	***SHLA***

مثال

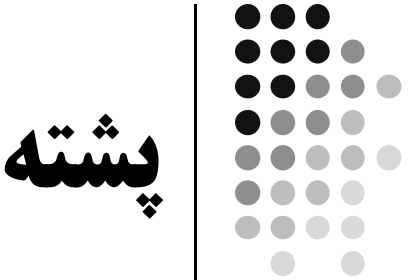
- برای انجام ریزعمل زیر

$R1 \leftarrow R2 - R3$

می بایست کلمه کنترلی بصورت زیر انتخاب شود:

Field:	SELA	SELB	SELD	OPR
Symbol	R2	R3	R1	SUB
Control word	010	011	001	00101

همانطور که قبلا دیدیم یک راه پیاده سازی واحد کنترل استفاده از میکروپروگرامینگ است که در آن هر کلمه کنترلی در یک محل از حافظه ROM ذخیره خواهد شد.



پشته (Stack)

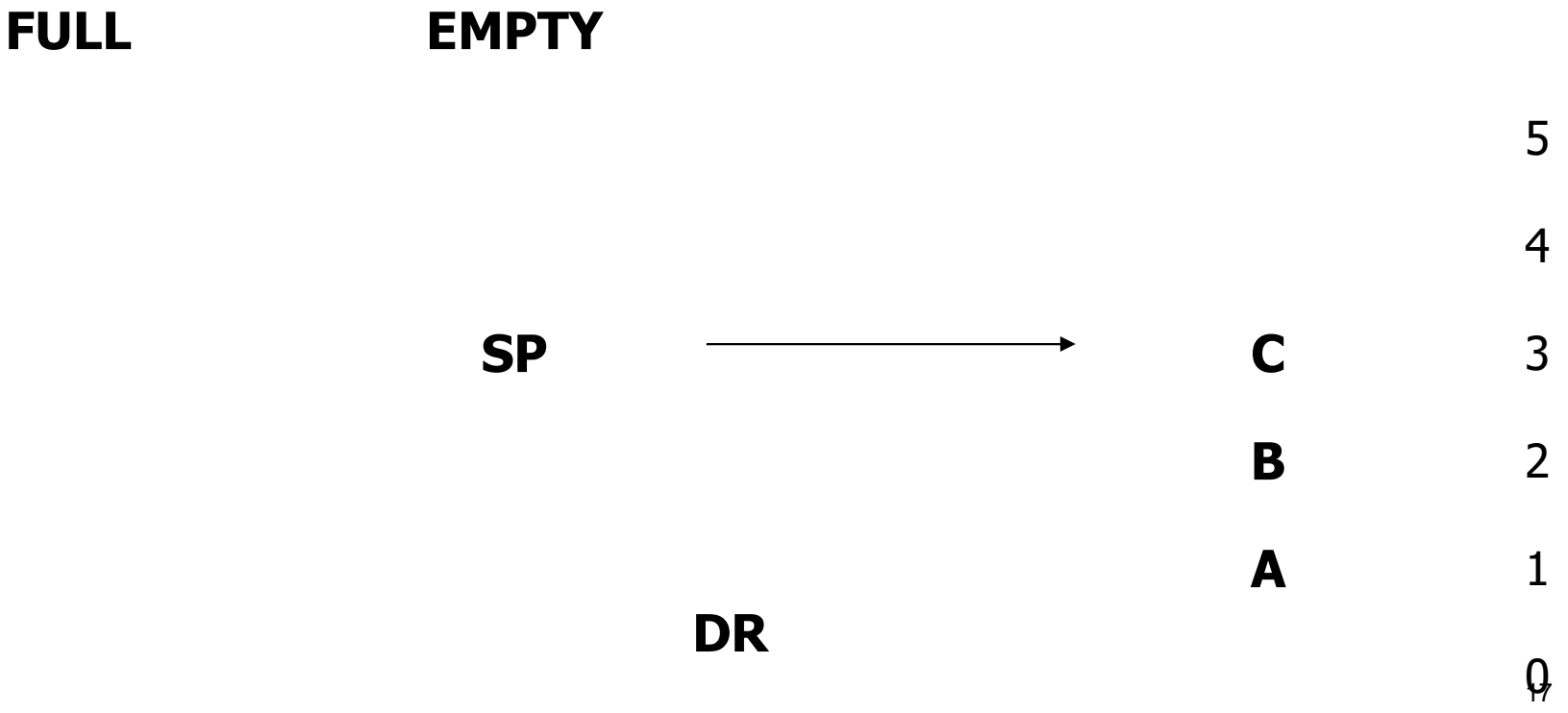
- موارد استفاده از پشته
  - ذخیره متغیرهای یک برنامه
  - ذخیره محتوی سایر رجیسترها وقتی که یک برنامه فرعی صدا زده می شود
  - کمک به ترجمه عملیات محاسباتی با روش RPN

- رجیستر SPمحل آخرین داده ذخیره شده در پشته را مشخص می کند

پیاده سازی پشته

1. پیاده سازی پشته بوسیله رجیسترها
2. پیاده سازی پشته در قسمتی از حافظه RAM

پیاده سازی پشته بوسیله رجیسترها

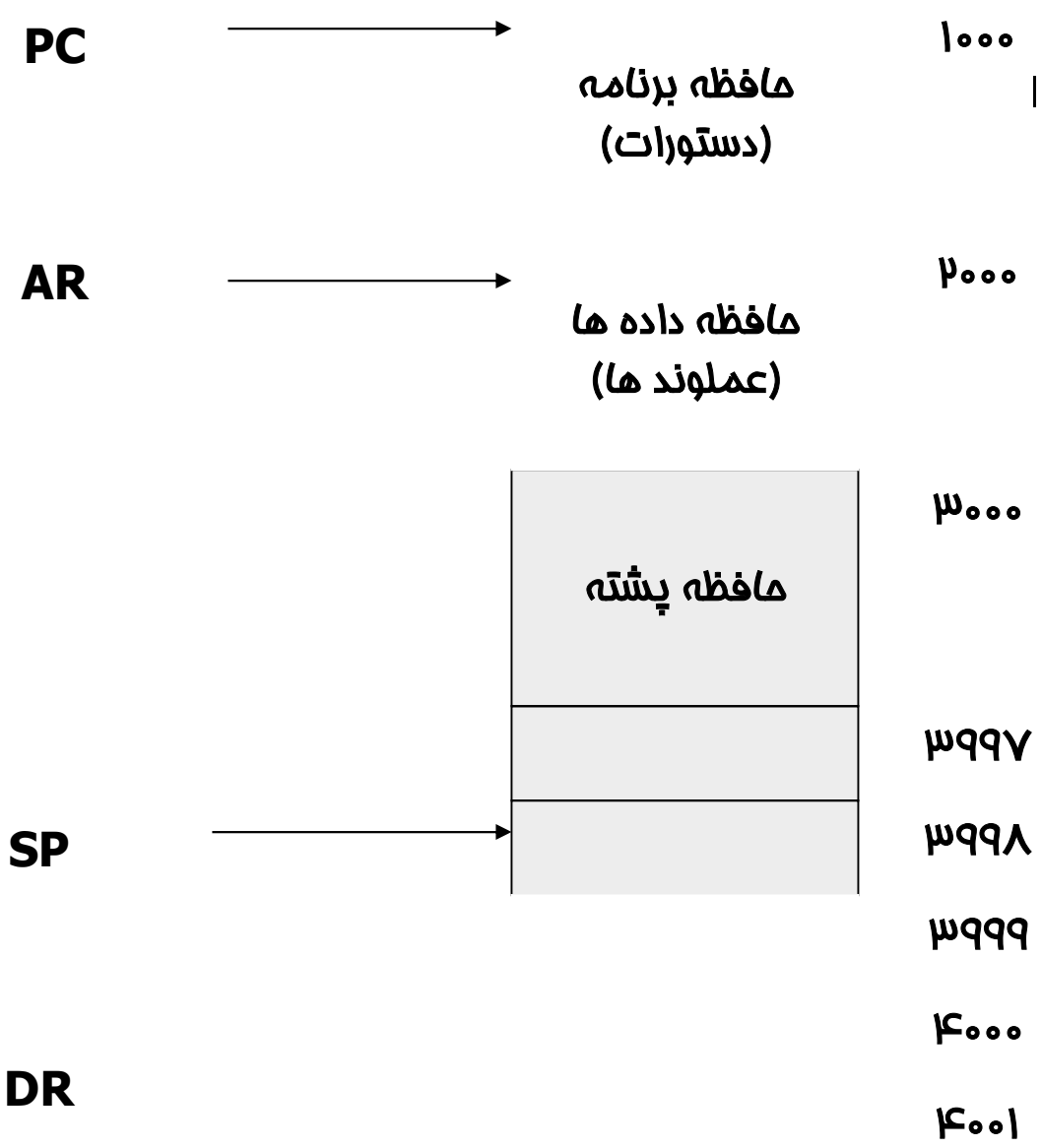


عملیات روی پشته

- Push
  - SP←SP+1
  - M[SP]←DR ,EMTY←0
  - if(SP=0) Then (FULL←1)

- POP
  - DR←M[SP] , FULL←0
  - SP←SP-1
  - if(SP=0) Then (EMTY←1)

پیاده سازی پشته در قسمتی از حافظه RAM



عملیات روی پشته

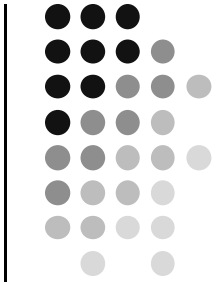
- Push
  - SP←SP-1
  - M[SP]←DR
- POP
  - DR←M[SP]
  - SP←SP+1

محدودیت های پشته

هنگام استفاده از پشته:

باید مواظب بود که بزرگ شدن پشته باعث دست اندازی آن به داده های سایر برنامه ها در RAM نشود .همچنین برنامه های دیگر پشته را خراب نکنند.

نمایش لهستانی معکوس
Reverse Polish Notation (RPN)



- روش معمولی نمایش عبارات ریاضی:

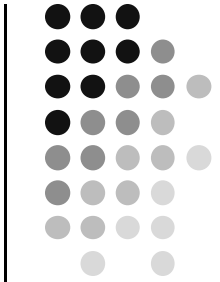
A*B+C*D

- روش PN برای نمایش عبارات ریاضی :عملگرها قبل از عملوندها قرارمی گیرند.

+*AB*CD

- روش RPN برای نمایش عبارات ریاضی :عملگرها بعد از عملوندها قرا رمی گیرند

AB*CD*+



استفاده از پشته برای پیاده سازی RPN

در برخی ماشینهای حساب و کامپیوترها از ترکیب پشته و RPN برای محاسبه عبارات ریاضی استفاده می کنند.

1. ابتدا عبارت بصورت RPN نوشته می شود (معمولا این کار توسط کامپایلر انجام می شود)
2. در هنگام محاسبه
   - با برخورد به عملوندها آنها را در پشته PUSH می کنیم
   - با برخورد با عملگرها ، دو داده موجود در بالای پشتهPOP شده و عمل مورد نظر بر روی آنها انجام و حاصل درپشته PUSH می شود.

مثال

برای محاسبه عبارت زیر یک کامپایلر ممکن است کد زیر را تولید نماید.

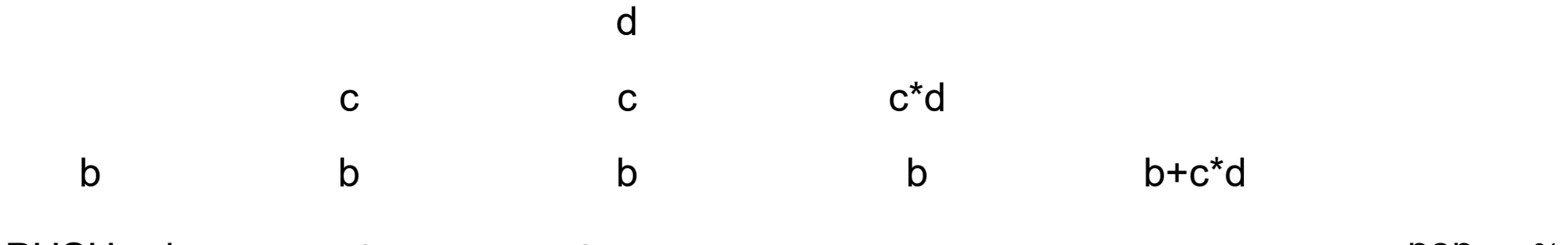
a = b + c * d;

PUSH	b
PUSH	**c**
PUSH	**d**
MUL	
ADD	
POP	**a**

عبارت معادل RPN بصورت زیر خواهد بود:

bcd*+

در اینصورت محتوی پشته بصورت زیر خواهد بود:



فرمت دستورالعمل

- معمول‌ترین فرمت یک دستورالعمل

1. قسمت Opcode، مشخص‌کننده نوع عملیات دستور
2. قسمت آدرس، مشخص‌کننده آدرس یک خانه حافظه یا ثبات پروسسور
3. قسمت حالت آدرس‌دهی، تعیین‌کننده عملوند یا آدرس مؤثر

Opcode	Mode	Add/reg

دستورات سه آدرسی

در کامپیوترهای سه‌آدرسی، هر قسمت آدرس، برای مشخص نمودن یک ثبات پردازنده و یا آدرس یک عملوند در حافظه تخصیص داده می‌شود.

ADD R1, A, B	R1 ← M[A] + M[B]
ADD R2, C, D	R2 ← M[C] + M[D]
MUL X, R1, R2	M[X] ← R1 × R2

دستورات دوآدرسی

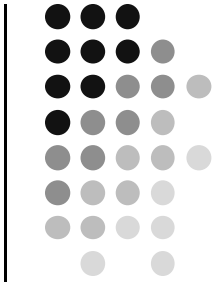
دستورات دوآدرسی معمول‌ترین فرمت دستور در کامپیوترها هستند. قسمت آدرس می‌تواند یک ثبات پردازنده یا یک خانه حافظه را مشخص نماید.

MOV R1, A	R1 ← M[A]
ADD R1, B	R1 ← R1 + M[B]
ADD R2, D	R2 ← R2 + M [D]
MUL R1, R2	R1 ← R1 × R2
MOV X, R1	M[A] ← R1

دستورات یک‌آدرسی

دستورات یک‌آدرسی، برای تمام عملیات بر روی داده‌ها، ثبات AC را به‌کار می‌برد.

LOAD A	AC ← M[A]
ADD B	AC ← AC + M[B]
STORE T	M[T] ← AC
LOAD C	AC ← M[C]
ADD D	AC ← AC + M[D]
MUL T	AC ← AC × M[T]
STORE X	M[X] ← AC



دستورات صفر آدرسی

دستورات صفر آدرسی، برای تمام عملیات بر روی داده‌ها، از پشته استفاده می کند.

```
Push  A
Push  B
ADD
Push C
Push    D
ADD
MUL
POP   X
```

حالات آدرس‌دهی

دستورات یک کامپیوتر عملی را بر روی داده ذخیره شده در حافظه و یا رجیسترهای CPU انجام می دهند.روش مشخص کردن عملوند یک دستورالعمل حالات آدرس دهی و یا addressing mode نامیده می شود.

اصولا حالات مختلف آدرس‌دهی عملوند دستور، تسهیلات زیر را در سیستم فراهم می‌آورد:

1. قابلیت ایجاد شمارنده برای برنامه حلقه، و شاخص‌بندی در داده‌ها و هم‌چنین ایجاد اشاره‌گر حافظه و جابجایی برای کاربر فراهم می‌شود
2. امکان تقلیل تعداد بیت‌های قسمت آدرس دستور، فراهم می‌شود.

انواع حالت های آدرس دهی

- Implied Addressing Mode
- Immediate Addressing Mode
- Register Addressing
- Register Indirect Addressing
- Autoincrement or Autodecrement
- Direct Addressing Mode
- Indirect Addressing Mode
- Relative Addressing Mode
- Index Addressing Mode
- Base Register Addressing Mode

- آدرس دهی ضمنی
- آدرس دهی بلادرنگ
- آدرس دهی ثبات
- آدرس دهی غیر مستقیم بکمک ثبات
- آدرس دهی افزایش و یا کاهشی خودکار
- آدرس دهی مستقیم
- آدرس دهی غیر مستقیم
- آدرس دهی نسبی
- آدرس دهی شاخص
- آدرس دهی با ثبات پایه

آدرس دهی ضمنی

- Implied Addressing Mode

در این روش اپراندها بصورت ضمنی در داخل دستورالعمل مشخص می شوند.

- مثل دستور CMA که محتوی آکومولاتور را متمم می کند.
- دستورات صفر آدرسی مورد استفاده در کامپیوترهای stack machine نیز از آدرس دهی ضمنی استفاده می کنند زیرا عملوندها بطور ضمنی در بالای پشته در نظر گرفته می شوند.

آدرس دهی بلادرنگ

- Immediate Addressing Mode

در این روش مقدار عملوند در داخل خود دستورالعمل داده می شود.

این مد آدرس دهی برای مقدار دهی رجیسترها بکار می رود.

- مثل دستور زیر در پردازنده x86

MOV CX, 1024

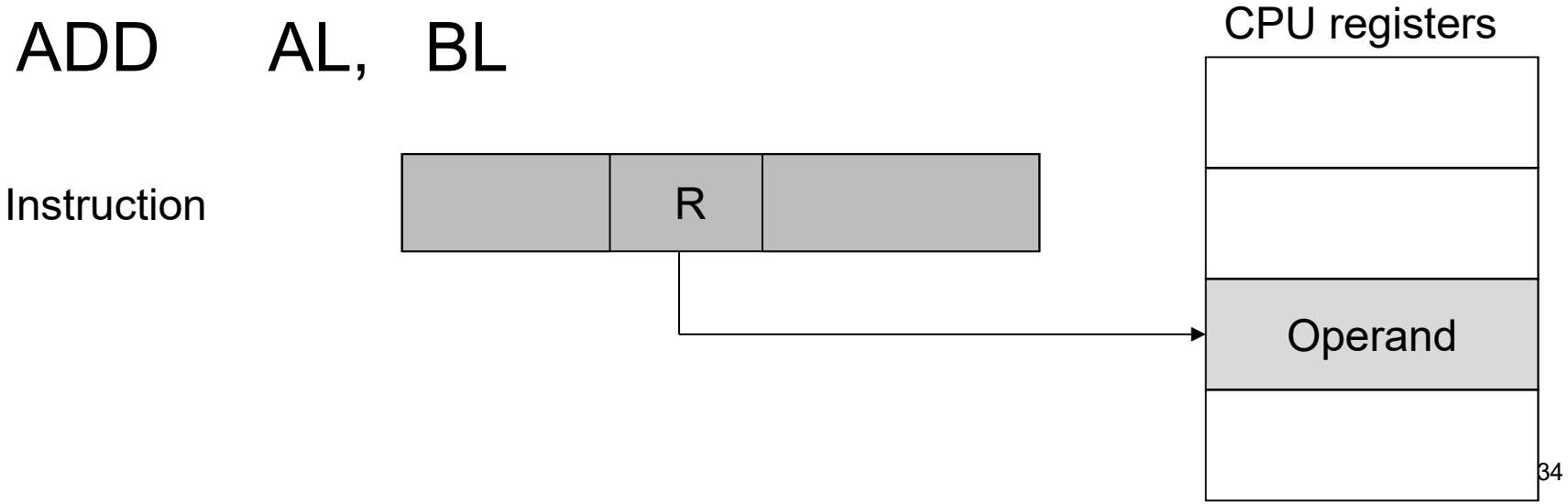
آدرس دهی ثبات

- Register Addressing

در این روش عملوندها در داخل رجیسترهای پردازنده قرار دارند. با استفاده از K بیت می توان تعداد 2^k رجیستر را مشخص نمود.

- مثل دستور زیر در پردازنده x86

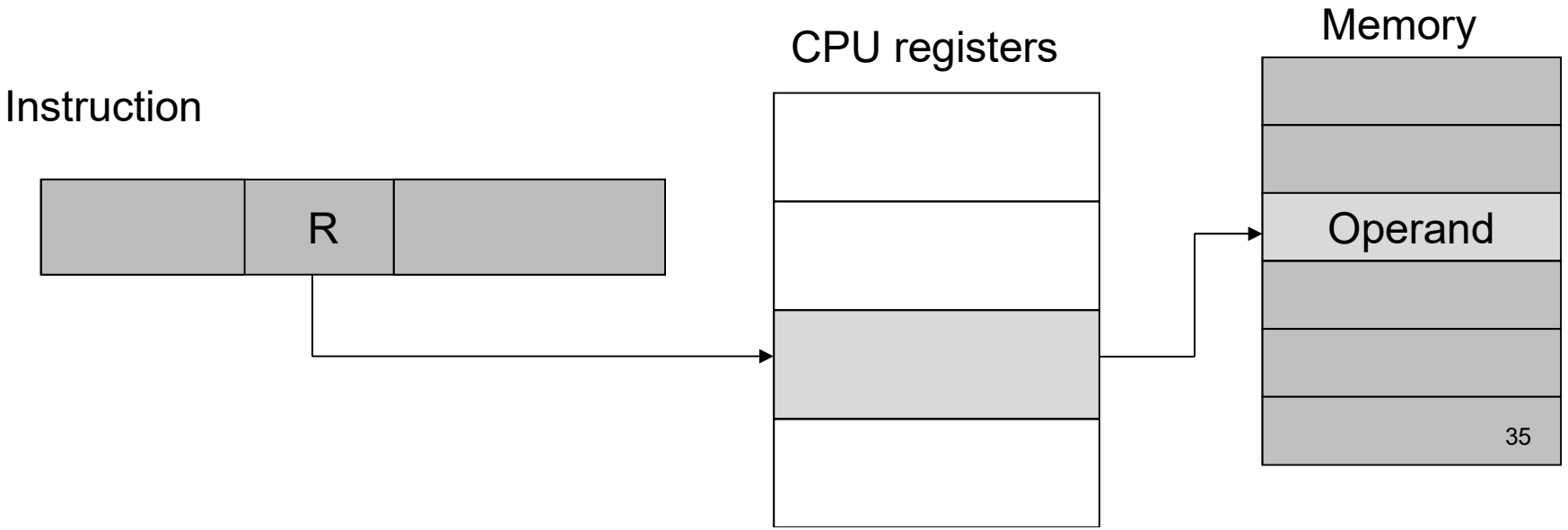
ADD AL, BL



آدرس دهی غیر مستقیم به کمک ثبات

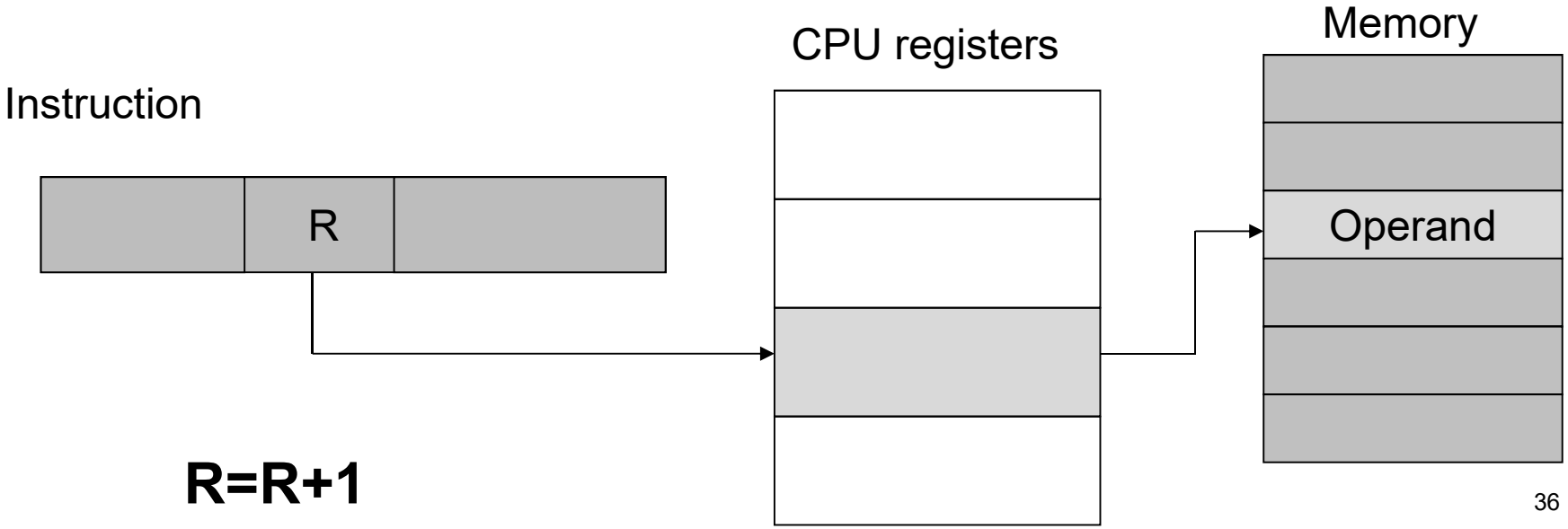
- Register Indirect Addressing
- در این روش دستورالعمل رجیستری را مشخص می کند که محتوی آن آدرس عملوند در حافظه را مشخص خواهد نمود.
- مثل دستور زیر در پردازنده x86

MOV BX,[SI]



آدرس دهی افزایش و یا کاهشی خودکار

- Autoincrement or Autodecrement addressing
- این روش مشابه آدرس دهی غیر مستقیم به کمک ثبات است با این تفاوت که مقدار رجیستر بعد از استفاده برای محاسبه آدرس موثر افزایش و یا کاهش می یابد.



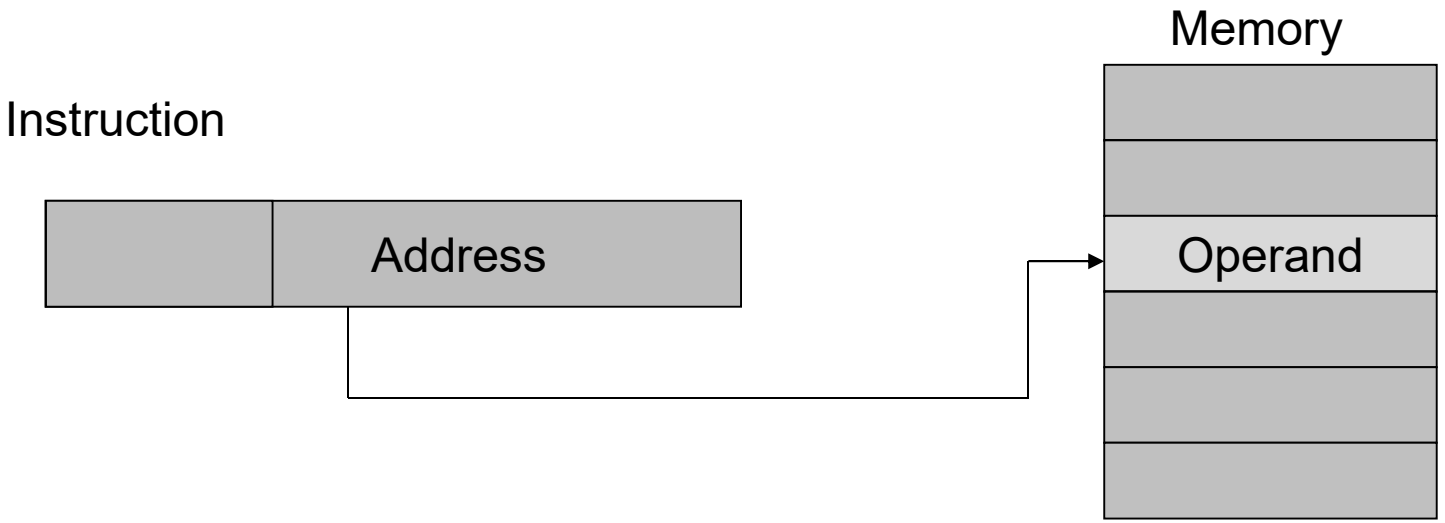
آدرس دهی مستقیم

- Direct Addressing Mode

در این روش آدرس عملوند در داخل دستورالعمل ذکر می شود.

- مثل دستور زیر در پردازنده x86

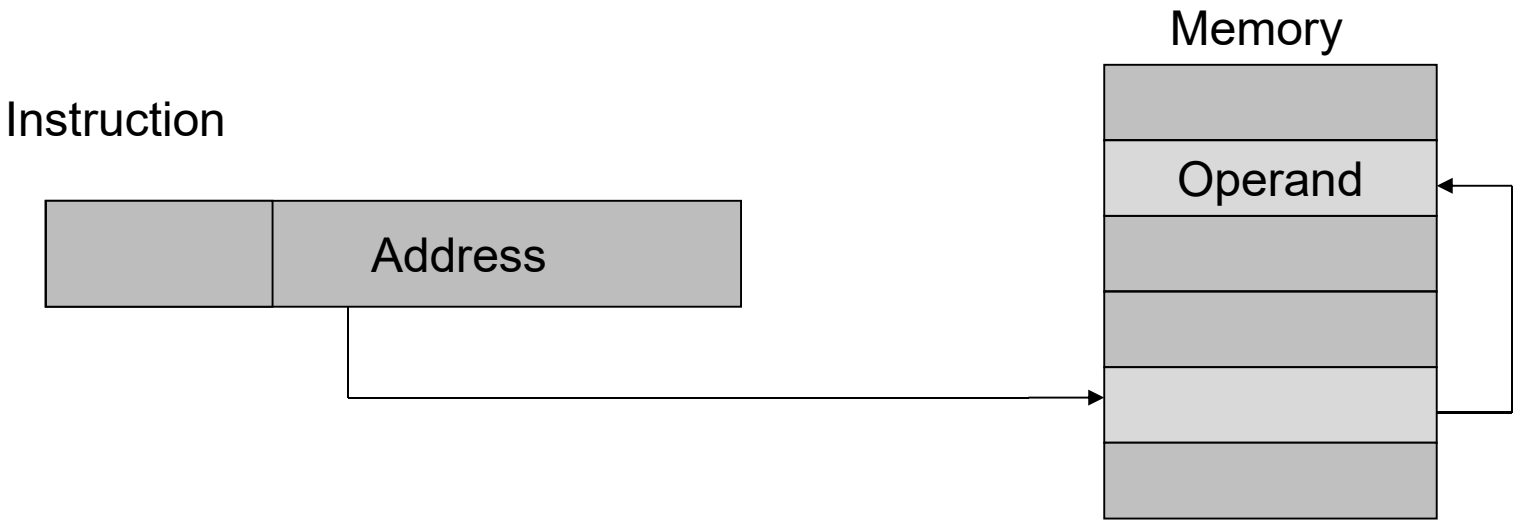
MOV AX,[3000]



آدرس دهی غیرمستقیم

- Indirect Addressing Mode

در این روش آدرس موجود در دستورالعمل محلی از حافظه را مشخص می کند که آدرس عملوند درآنجا قرار دارد .در این حالت برای دسترسی به عملوند دوبار رجوع به حافظه مورد نیاز است: یکبار برای پیدا کردن آدرس آن و بار دیگر برای خواندن مقدار آن .

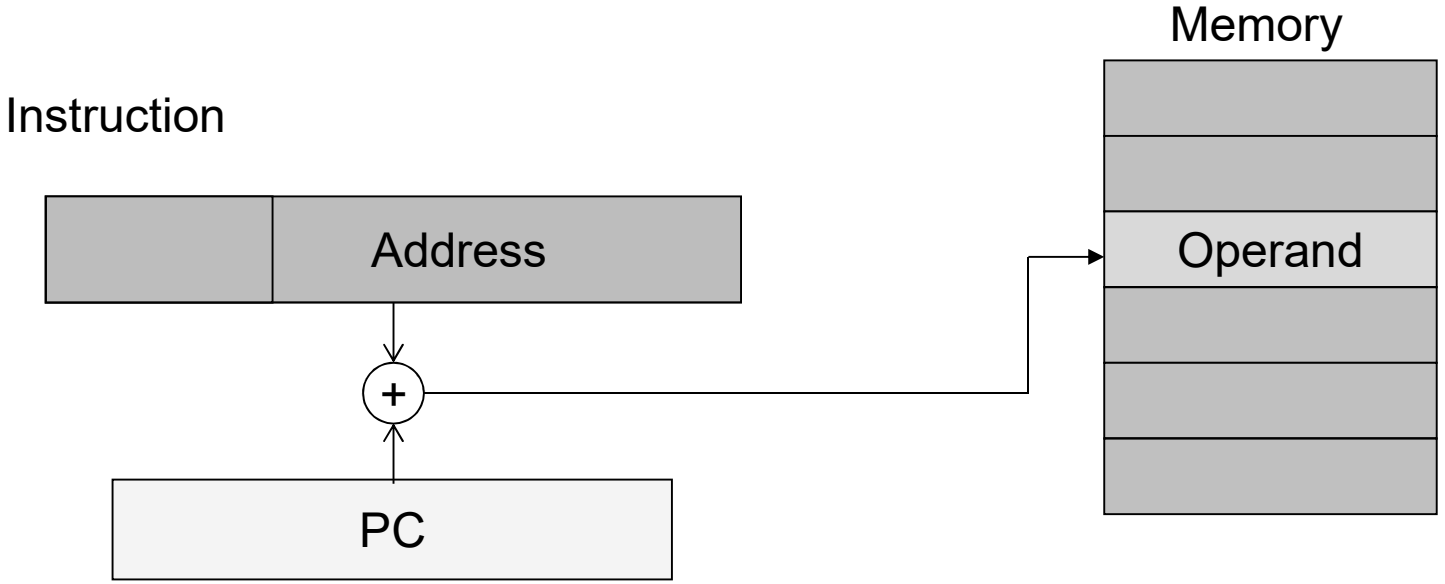


آدرس دهی نسبی

- Relative Addressing Mode
- در این روش آدرس موثر از جمع آدرس مشخص شده در داخل دستورالعمل و محتوای PC حاصل می شود:

Effective Address= address part of instruction + content of PC

- آدرس دهی نسبی در دستورالعمل های انشعابی که آدرس پرش در نزدیکی دستور قراردارد

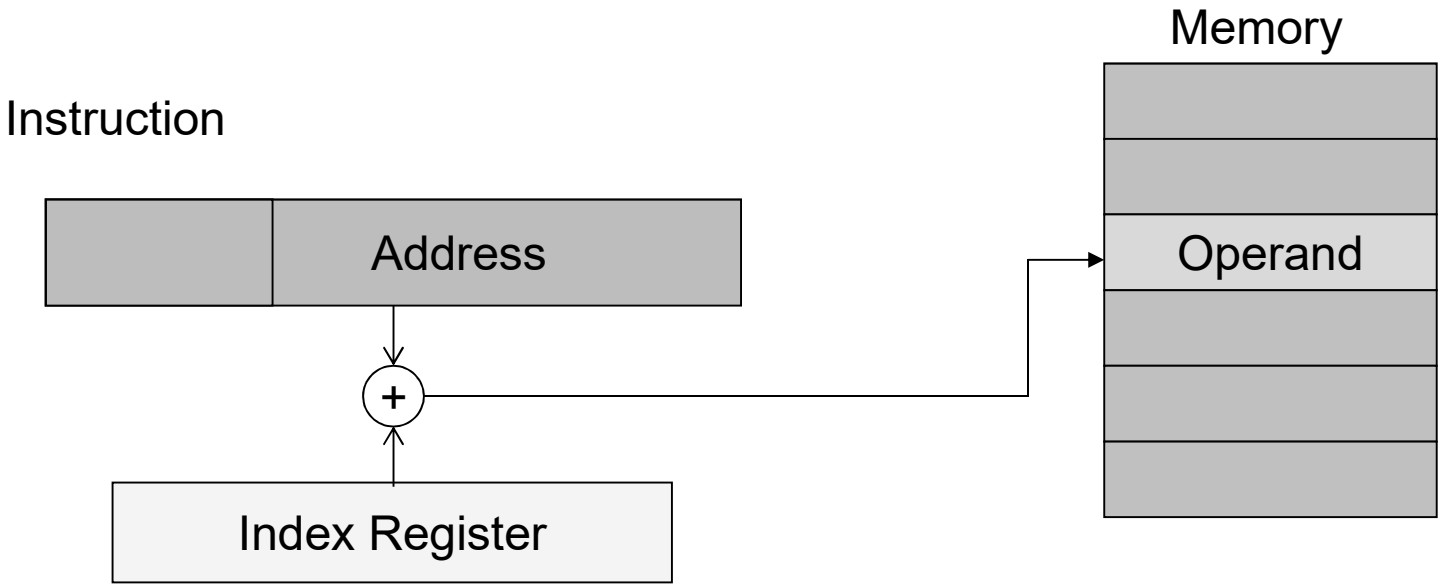


آدرس دهی شاخص

- Index Addressing Mode
- در این روش آدرس موثر از جمع آدرس مشخص شده در داخل دستورالعمل و محتوی یک رجیستر مخصوص که رجیستر Index نامیده میشود حاصل می شود:

Effective Add.= address part of instruction + content of index register

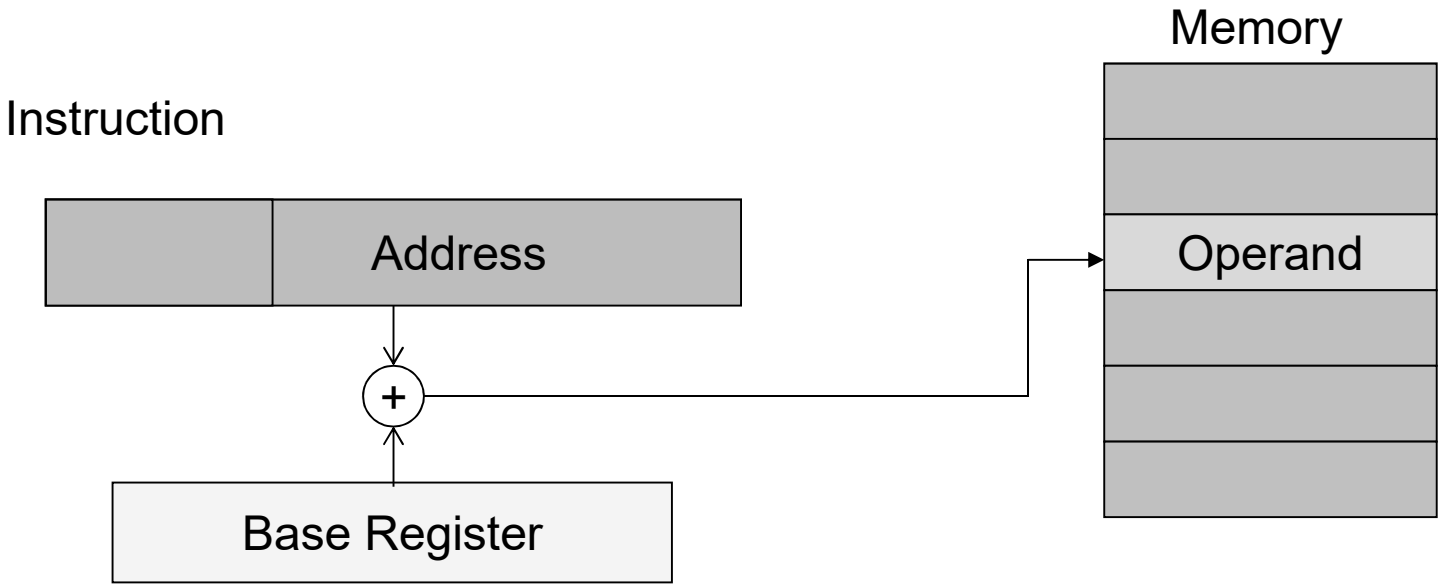
- معمولا از این روش برای دسترسی به داده های یک آرایه استفاده می شود که محل شروع داده ها در حافظه در دستورالعمل مشخص می شود و فاصله داده مورد نظر تا محل شروع توسط رجیستر Index تعیین می گردد.



آدرس دهی با ثبات پایه

- Base Register Addressing Mode
- این روش مشابه آدرس دهی با ثبات شاخص است با این تفاوت که به جای ثبات شاخص از ثبات پایه استفاده می شود. تفاوت این دو روش در نحوه استفاده از رجیسترها است.

Effective Add.= address part of instruction + content of Base register



حالتهای آدرس دهی x86

Mode	Address
Immediate	Operand = A
Register	EA=R
Operand	EA=(SR)+A
Register Indirect	EA=(SR)+(B)
Based	EA=(SR)+(B) +A
Index	EA=(SR)+(I) +A
Scaled Index	EA=(SR)+(I)*S +A
Based Index	EA=(SR)+(B) +(I)
Based Scaled Index	EA=(SR)+(I)*S + (B)
Based Index with Displacement	EA=(SR)+(B) +(I) +A
Based Scaled Index with Displacement	EA=(SR)+(B) +(I)*s +A

EA = Effective Address
(X) = contents of X
SR = Segment register
A = Contents of an address field in the instruction
R =register
B =baseregister
I = index register
S = scale factorB=base

مثال عددی

مقدار آکومولاتور در صورت اجرای دستور موجود در آدرس 200 برای حالتهای مختلف آدرس دهی چیست؟

PC = 200

R1 = 400

XR = 100

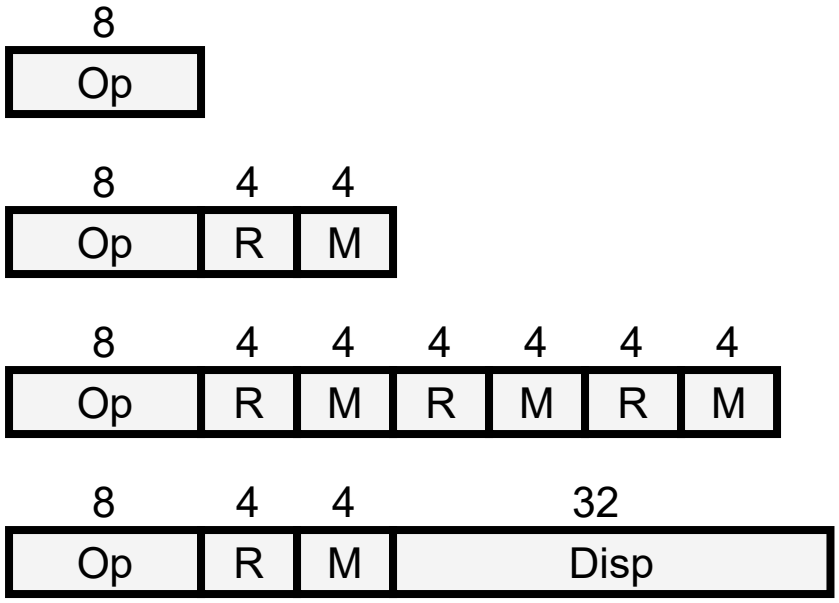
AC

آدرس	حافظه	حالت
200	بارکردن AC	
201	۵۰۰ = آدرس	
202	دستور بعدی	
399	۴۵۰	
400	۷۰۰	
500	۸۰۰	
600	۹۰۰	
702	۳۲۵	
800	۳۰۰	

مثال عددی

حالات آدرس‌دهی	آدرس مؤثر	محتوای اکومولیتور
مستقیم	۵۰۰	۸۰۰
بلافصل	۲۰۱	۵۰۰
غیرمستقیم	۸۰۰	۳۰۰
نسبی	۷۰۲	۳۲۵
شاخص‌دار	۶۰۰	۹۰۰
ثبات	–	۴۰۰
غیرمستقیم از طریق ثبات	۴۰۰	۷۰۰
افزایش خودکار	۴۰۰	۷۰۰
کاهش خودکار	۳۹۹	۴۵۰

دستورات با طول متغیر



VAX instrs: 1-53 bytes!

- در اغلب پردازنده های تجاری نظیر8086 طول دستورات متفاوت می باشد.
- استفاده از دستورات با طول متفاوت باعث کدینگ موثر دستورات می گردد زیرا بیت های بلا استفاده را کاهش می دهد .
- اما در عین حال اینکار می تواند پیاده سازی سریع دستورات را مشکل نماید زیرا دسترسی به عملوندها بصورت متوالی انجام می شود.

مجموعه دستورات کامپیوتر

- دستورات مورد استفاده در کامپیوترهای مختلف از لحاظ تعداد، عملکرد، نشانه های مورد استفاده برای اسمبلی، و کد باینری بسیار متفاوت هستند با این وجود تمامی آنها دارای دستوراتی از گروههای زیر میباشند:
  - دستورات انتقال داده
  - دستورات محاسباتی، منطقی و جابجائی
  - دستورات کنترل برنامه

دستورات معمول انتقال اطلاعات

نام	فرم سمبولیک	
Load	LD	انتقال داده بین حافظه و رجیسترها
Store	ST	
Move	MOV	انتقال داده بین رجیسترها انتقال داده بین حافظه و رجیسترها
Exchange	XCH	تعویض داده های دو رجیستر و یا یک رجیستر و حافظه
Input	IN	انتقال داده بین ورودی/خروجی و رجیسترها
Output	OUT	
Push	PUSH	انتقال داده بین پشته و رجیسترها
Pop	POP	

هشت حالت آدرس‌دهی برای دستور بارکردن

حالت آدرس‌دهی	زبان اسمبلی	زبان انتقال ثبات‌ها
آدرس‌دهی مستقیم	LD ADR	AC ← M[ADR]
آدرس‌دهی غیرمستقیم	LD @ADR	AC ← M [M[ADR]]
آدرس‌دهی نسبی	LD $ADR	AC ← M [PC + ADR]
آدرس‌دهی بلافصل	LD #NBR	AC ← NBR
آدرس‌دهی شاخص	LD ADR (X)	AC ← M [ADR + XR]
آدرس‌دهی ثبات	LD RI	AC ← RI
آدرس‌دهی طریق ثبات غیرمستقیم	LD (RI)	AC ← M [RI]
آدرس‌دهی افزایش خودکار	LD (RI)+	AC ← M [RI], RI ← RI + 1

دستورات معمولی ریاضی

نماد دستور	نام دستور	
INC	Increment	افزایش‌دهنده یک
DEC	Decrement	کاهش‌دهنده یک
ADD	Add	جمع
SUB	Subtract	تفریق
MUL	Multiply	ضرب
DIV	Divide	تقسیم
ADDC	Add with carry	جمع با بیت نقلی
NEG	Negate (2's complement)	مکمل ۲

دستورات منطقی و عملیات روی بیت

نماد دستور	نام دستور	
CLR	Clear	صفر کردن
COM	Complement	مکمل کردن
AND	AND	AND
OR	OR	OR
XOR	XOR	XOR
CLRC	Clear carry	صفر کردن بیت نقلی
SETC	Set carry	یک کردن بیت نقلی
COMC	Complement carry	مکمل کردن بیت نقلی
EI	Enable Interrupt	فعال کردن وقفه
DI	Disable Interrupt	غیرفعال کردن وقفه

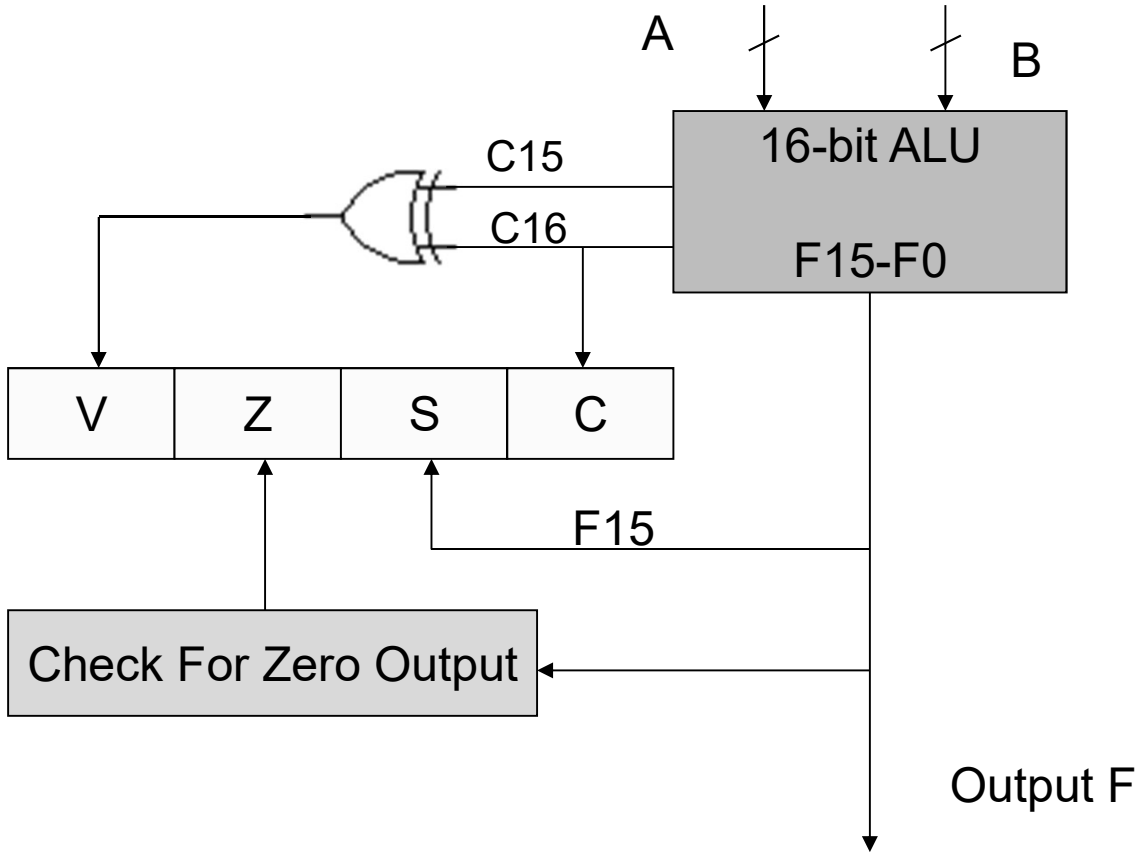
دستورات معمولی شیفت

فرم نمادین	نام دستور	
SHR	Logical Shift Right	شیفت به راست
SHL	Logical Shift Left	شیفت به چپ
SHRA	Arithmetic Shift Right	شیفت ریاضی به راست
SHLA	Arithmetic Shift Left	شیفت ریاضی به چپ
ROR	Rotate Right	چرخش به راست
ROL	Rotate Left	چرخش به چپ
RORC	Rotate Right through carry	چرخش به راست از طریق بیت نقلی
ROLC	Rotate Left through carry	چرخش به چپ از طریق بیت نقلی

دستورات متداول کنترل برنامه

فرم نمادین	نام دستور	
BR	Branch	انشعاب
JMP	Jump	پرش
SKP	SKIP	رد کردن
CALL	Call	صدا زدن
RET	Return	بازگشت
CMP	Compare(By subtraction)	مقایسه
TST	Test(By ANDing)	تست

بیت های وضعیت



وقفه

- منظور از وقفه انتقال کنترل برنامه از یک برنامه در حال اجرا به یک برنامه سرویس دهی خاص در نتیجه یک درخواست داخلی یا خارجی است که پس از اجرای روال سرویس دهی، کنترل به برنامه اصلی باز می گردد.

وقفه

- روال وقفه مشابه فراخوانی زیرروال است با این تفاوت که :
- برنامه فرعی (زیرروال) در اثر اجرای یک دستور شروع می شود، اما روال وقفه به علت اعمال یک سیگنال داخلی یا خارجی است.

- آدرس برنامه فرعی در بخش آدرس دستور واقع است، اما در وقفه آدرس سرویس دهی وقفه توسط سخت افزار مشخص می شود.

- در زمان اجرای برنامه فرعی، تنها محتوای PC ذخیره می شود؛ در صورتی که به هنگام اجرای یک سرویس وقفه علاوه بر محتوای PC، وضعیت کلی CPU نیز ذخیره می شود. به این وضعیت کلی CPU کلمه حالت (Program Status Word = PSW) گفته می شود.

انواع مختلف وقفه در کامپیوتر

- وقفه های خارجی
  - این وقفه ها سخت افزاری بوده وسیگنال وقفه توسط یک دستگاه I/O اعمال می شود.

- وقفه های داخلی
  - در اثر استفاده غلط و نابجای یک دستور حاصل می شود .مثل رخ دادن Overflowانجام یک تقسیم بر صفر، سرریز شدن پشته که متناسب با هر کدام از آنها یک سرویس دهی وقفه اجرا می شود که معمولا پیغامی است در جهت تصحیح داده یا دستورالعمل .این وقفه گاهی trap نامیده می شود.

- وقفه های نرم افزاری
  - نوع خاصی از صدا زدن برنامه فرعی هستند . و در اثر یک دستورالعمل اجرا می شود. برای انجام عملیات خاصی نظیر تغییر مد برنامه از حالت user به supervisor بکار می روند. تفاوت آن با اجرای زیرروال آن است که علاوه بر ذخیره محتوای PC وضعیت کلی CPU را نیز ذخیره می کند.

کامپیوترهای با مجموعه دستورات کاهش یافته

RISC = Reduced Instruction Set Computer

- دستورات ساده و کمی دارند
- هدف این است که بتوان دستورات را به سرعت اجرا کرد. اغلب دستورات RISC در یک سیکل اجرا می شوند (بعد از واکشی و رمزگشایی)
- چون دستورات در زمان مشابهی اجرا می شوند عمل pipeline دارای بازده بالایی خواهد بود.
- کم بودن دستورات باعث ساده شدن واحد کنترل و مسیرهای ارتباطی می شود که منجر به کم شدن تعداد قطعات مورد نیاز برای ساخت پردازنده می شود. این کار سرعت پردازنده را نیز افزایش می دهد.

ویژگی های کامپیوترهای RISC

- تعداد دستورالعمل ها محدود می باشد.
- غالب دستورات دارای طول ثابت و فرمت یکسان می باشند. این کار باعث می شود تا خواندن دستورات و رمزگشائی آنها سریع باشد .زیرا لازم نیست تا معلوم شدن طول دستور برای رمزگشایی آن صبر کرد. یکسانی فرمت باعث سهولت رمزگشایی می گردد زیرا کد و آدرس همه دستورات در محل یکسانی قرار دارند.
- کنترل از نوع سخت افزاری است.

ویژگی های RISC

- پردازش داده ها در CPU انجام می شود.
- تعداد رجیسترهای زیادی دارند.
- کم شدن دستورات باعث کوچک شدن واحد کنترل و آزاد شدن فضا برای گنجاندن تعداد بیشتری رجیستر می شود.
- متغیرهای محلی، نتایج میانی و پارامترهای توابع در داخل رجیسترها ذخیره شده و تعداد رجوع به حافظه کاهش پیدا می کند.
- استفاده از تکنیک همپوشانی register windows که باعث افزایش سرعت صدا زدن تابع و بازگشت از آن می شود.

ویژگی های RISC

- مراجعه به حافظه محدود به دستورات Load (LD) و Store (ST) است.
  - باقی دستورات بر روی محتوی رجیسترها عمل می کنند.
- تعداد مد های آدرس دهی آنها کم است
  - معمولا فقط از مدهای آدرس دهی زیر استفاده می شود:
    - register addressing
    - direct addressing
    - register indirect addressing
    - displacement addressing

مثالی از دستورات RISC

برنامه محاسبه $X = (A + B) \times (C + D)$

LOAD	R1, A	R1 ← M[A]
LOAD	R2, B	R2 ← M[B]
LOAD	R3, C	R3 ← M[C]
LOAD	R4, D	R4 ← M[D]
ADD	R1, R1, R2	R1 ← R1 + R2
ADD	R3, R3, R4	R3 ← R3 + R4
MUL	R1, R1, R3	R1 ← R1 × R3
STORE	X, R1	M[X] ← R1

کامپیوترهای با مجموعه دستورات پیچیده CISC

- معماری یک کامپیوتر متاثر از مجموعه دستورات انتخاب شده برای آن است.

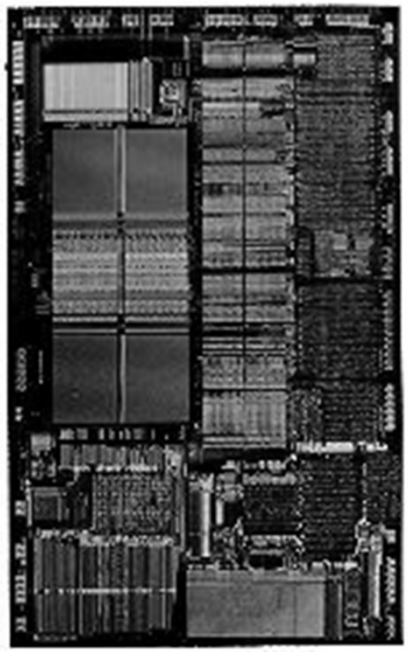
- در کامپیوتر های اولیه سعی بر این بود تا تعداد دستورات کم و ساده باشد تا پیاده سازی سخت افزاری آن ممکن باشد.

- با پیشرفت در زمینه سخت افزار و ارزان شدن آن تعداد دستورات کامپیوتر ها افزایش یافته و بر پیچیدگی آنها افزوده گردید .هدف این بود تا هر چه بیشتر نیاز های کاربران را در سخت افزار گنجانده و با کاهش فاصله بین زبانهای سطح بالا و دستورات کامپیوتر کار ترجمه دستورات سطح بالا را ساده تر کنند. این نوع کامپیوترها گاها تا 200 دستور و تعداد بسیار زیادی مد آدرس دهی داشتند. این کامپیوتر ها را Complex Instruction Set Computer (CISC) می نامند.

ویژگی های کامپیوترهای CISC

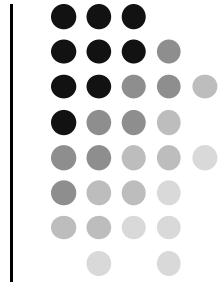
- تعداد زیادی دستورالعمل دارند (۱۰۰ تا ۲۰۰ عدد)
- دستوراتی برای انجام کارهای ویژه دارند که معمولا به ندرت مورد استفاده قرار می گیرند
- تعداد زیادی مد آدرس دهی دارند.
- طول دستورالعمل ها متغیر است پس کدگشایی (Decoding) آن پیچیده است.
- دستوراتی دارند که عملیاتی را بر روی اپراندهای موجود در حافظه انجام می دهند و مستقیما داده های حافظه را دستکاری می کنند.
- برای اجرای دستورات به چندین کلاک نیاز هست.

Intel 486™ DX CPU

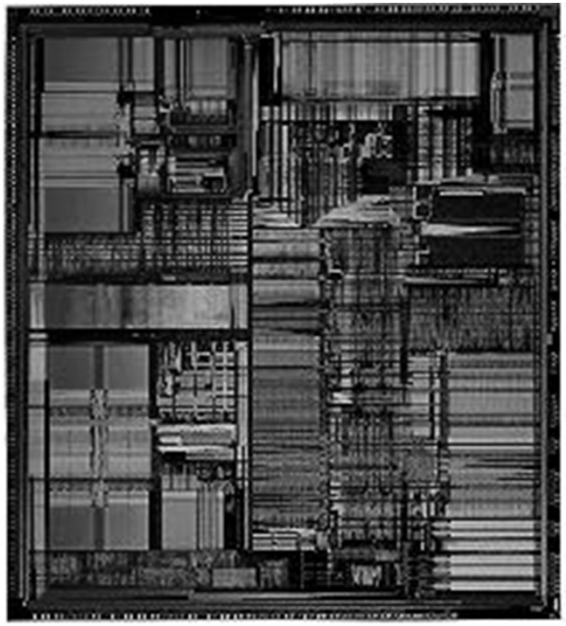


- Design 1986 – 1989
- 25 MHz, 33 MHz
- 1.2 M transistors
- 1.0 micron
- 5 stage pipeline
- Unified 8 KByte code/data cache (write-through)
- First IA-32 processor capable of executing 1 instruction per clock cycle

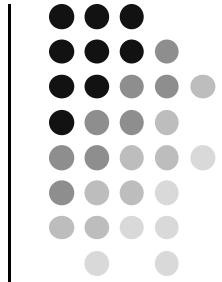
Pentium® Processor



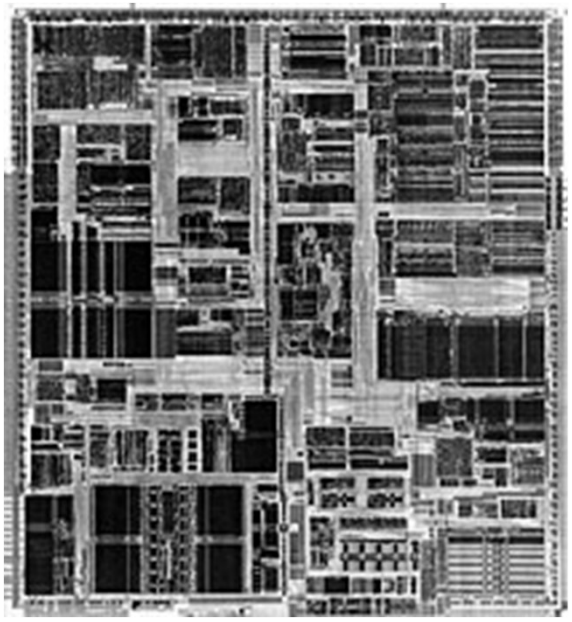
- Design 1989 – 1993
- 60 MHz, 66 MHz
- 3.1 M transistors
- 0.8 micron
- 5 stage pipeline
- 8 KByte instruction and 8 KByte data caches (writeback)
- Branch predictor
- Pipelined floating point
- First superscalar IA-32: capable of executing 2 instructions per clock



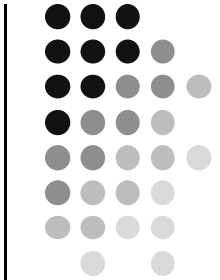
Pentium® II Processor



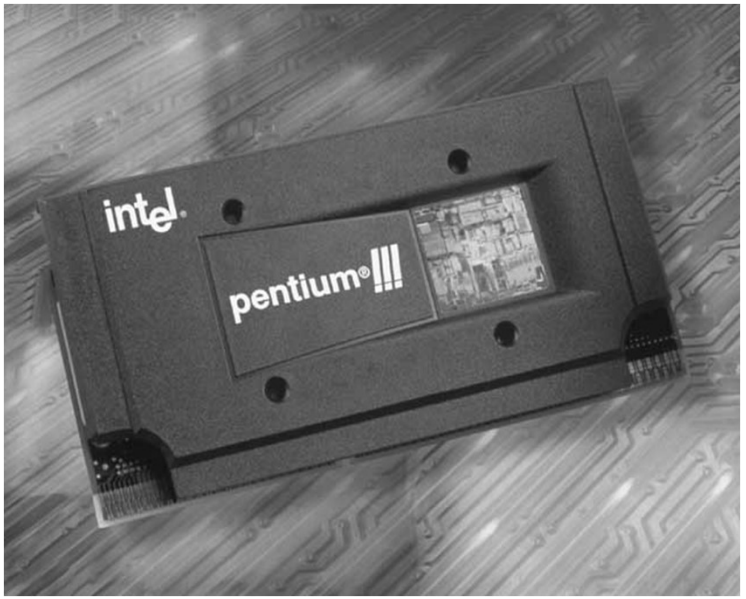
- Design 1995 – 1997
- 233 MHz, 266 MHz, 300 MHz
- 7.5 M transistors
- 0.35 micron
- 16 KByte L1I, 16 KByte L1D, 512 KByte off-die L2
- First compaction of P6 microarchitecture



Pentium® III Processor (Katmai)

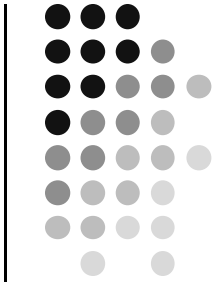


- Introduced: 1999
- 450 MHz, 500 MHz, 533 MHz, 600MHz
- 9.5 M transistors
- 0.25 micron
- 16 KByte L1I, 16 KByte L1D, 512 KByte off-chip L2
- Addition of SSE instructions.



SSE: Intel Streaming SIMD Extensions to the x86 ISA

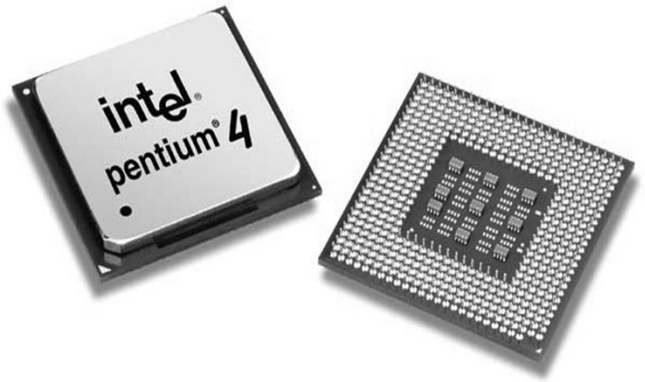
Pentium® III Processor (Coppermine)



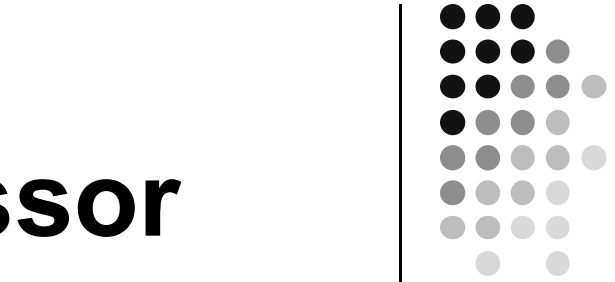
- Introduced: 1999
- 500MHz … 1133MHz
- 28 M transistors
- 0.18 micron
- 16 KByte L1I, 16 KByte L1D, 256KByte on-chip L2
- Integrate L2 cache on chip, It topped out at 1GHz.

Pentium® IV Processor

- Introduced: 2000
- 1.3GHz … 2GHz … 3.4GHz
- 42M … 55M … 125 M transistors
- 0.18 … 0.13 … 0.09 micron
- Latest one: 16 KByte L1I, 16 KByte L1D, 1M on-chip L2
- Very high clock speed and SSE performance



Intel® Itanium® Processor



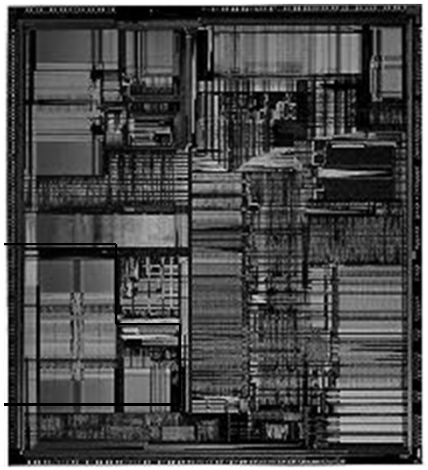
- Design 1993 – 2000
- 733 MHz, 800 MHz
- 25 M transistors
- 0.18 micron
- 3 levels of cache
  - 16 KByte L1I, 16 KByte L1D
  - 96 KByte L2
  - 4 MByte off-die L3
- Superscalar degree 6, in-order machine
- First implementation of 64-bit Itanium architecture

Intel® Itanium 2® Processor

- Introduced: 2002
- 1GHz
- 221 M transistors
- 0.18 micron
- 3 levels of cache
  - 32 KByte I&D L1
  - 256 KByte L2
  - integrated 1.5MByte L3
- Based on EPIC architecture
- Enhanced Machine Check Architecture (MCA) with extensive Error Correcting Code (ECC)

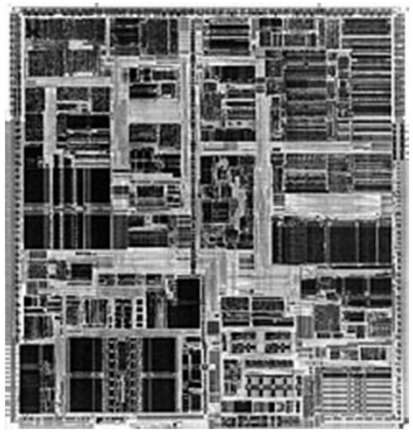
Cache Size Becoming Larger and Larger

1993: Pentium



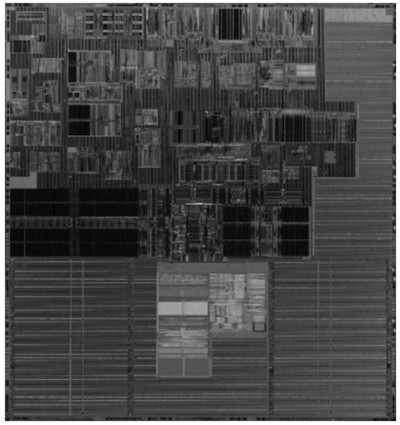
8 KByte I-cache and 8 KByte D-cache

1997: Pentium-II

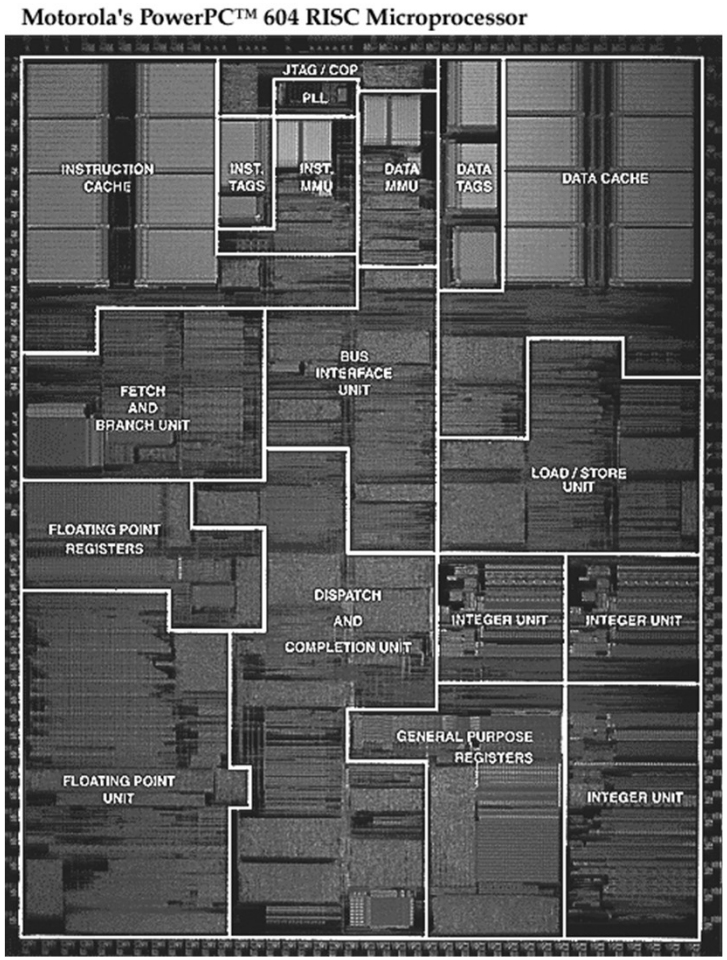


- 16 KByte L1I, 16 KByte L1D
- 512 KByte off-die L2

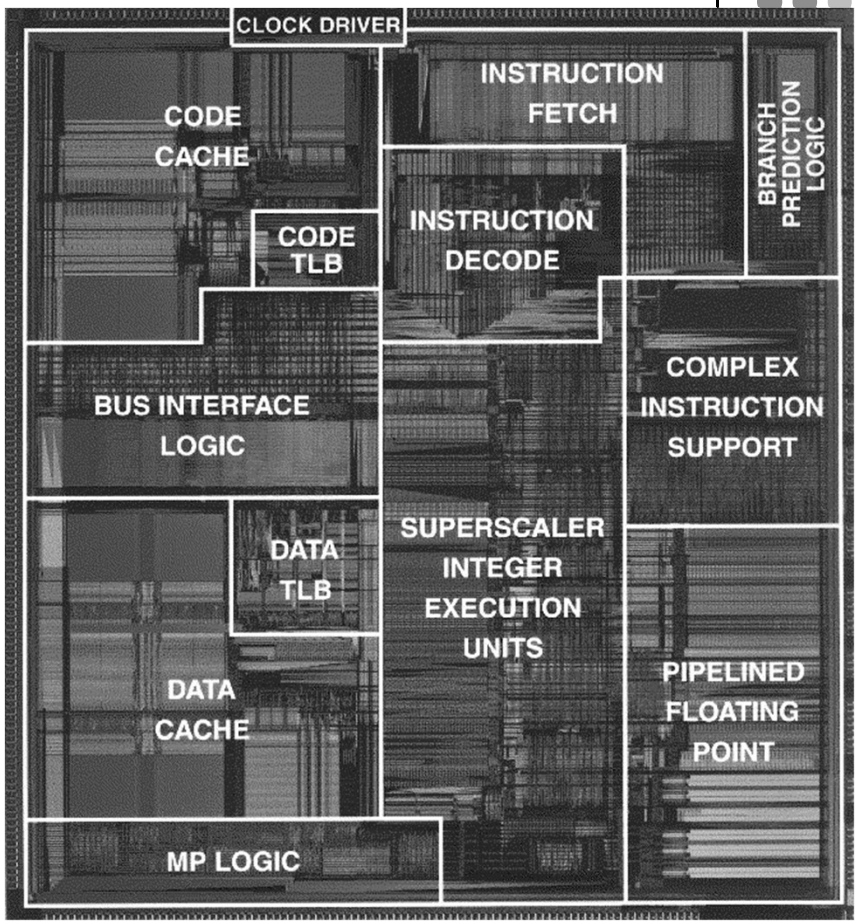
2002: Itanium-2



- Level 1: 16K KByte I-cache, 16 KByte D-cache
- Level 2: 256 KB
- Level 3: integrated 3 MB or 1.5 MB



Motorola's PowerPC 604



Pentium

Intel Pentium 4 Northwood

Buffer Allocation & Register Rename

Instruction Queue (for less critical fields of the uOps)

General Instruction Address Queue & Memory Instruction Address Queue (queues register entries and latency fields of the uOps for scheduling)

Floating Point, MMX, SSE2 Renamed Register File 128 entries of 128 bit.

uOp Schedulers

FP Move Scheduler: (8x8 dependency matrix)

Parallel (Matrix) Scheduler for the two double pumped ALU's

General Floating Point and Slow Integer Scheduler: (8x8 dependency matrix)

Load / Store uOp Scheduler: (8x8 dependency matrix)

Load / Store Linear Address Collision History Table

Integer Execution Core

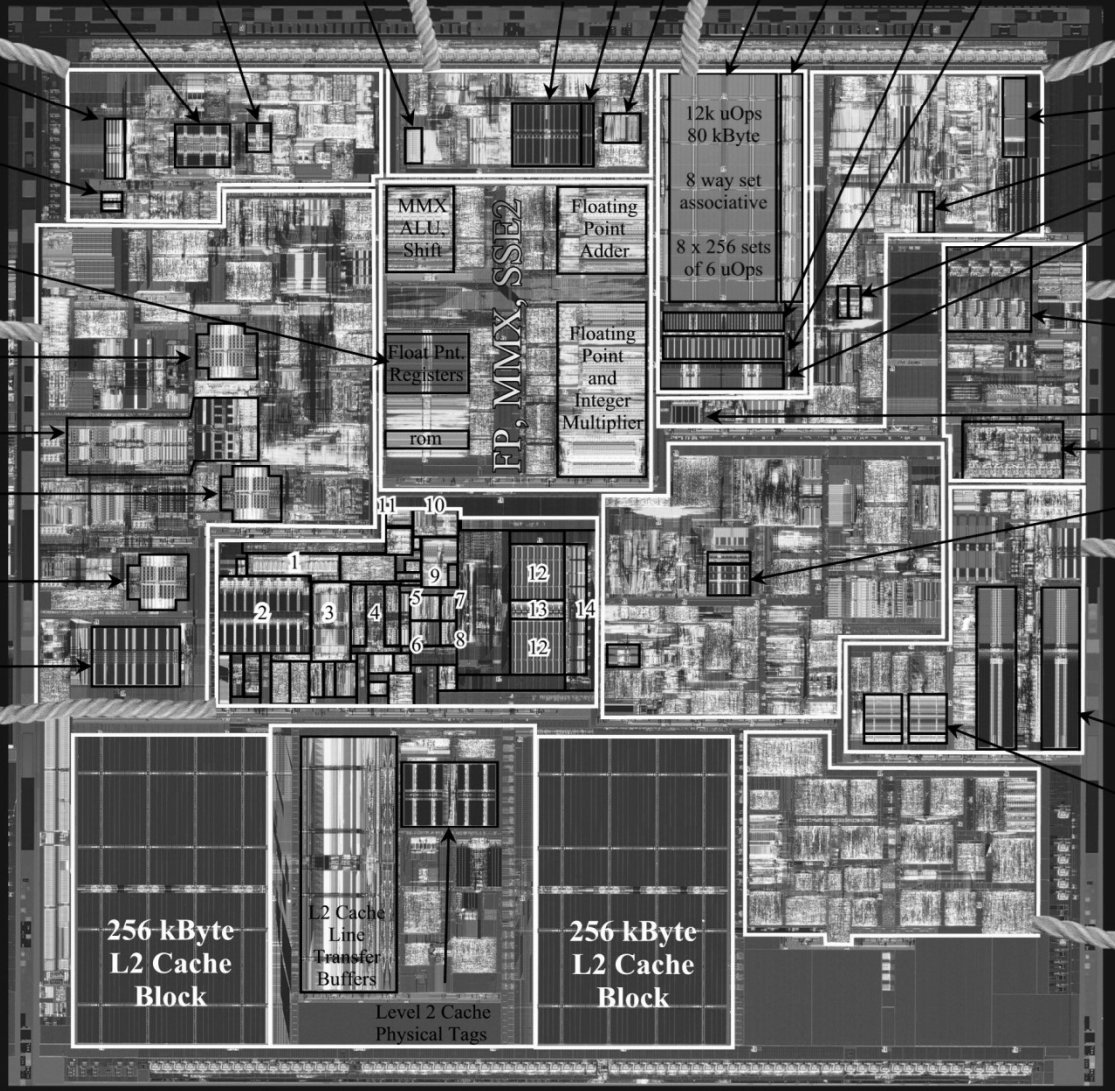
(1) uOp Dispatch unit & Replay Buffer Dispatches up to 6 uOps / cycle
(2) Integer Renamed Register File 128 entries of 32 bit + 6 status flags 12 read ports and six write ports
(3) Databus switch & Bypasses to and from the Integer Register File.
(4) Flags, Write Back
(5) Double Pumped ALU 0
(6) Double Pumped ALU 1
(7) Load Address Generator Unit
(8) Store Address Generator Unit
(9) Load Buffer (48 entries)
(10) Store Buffer (24 entries)

Execution Pipeline Start

Register Alias History Tables (2x126)
Register Alias Tables
uOp Queue

Instruction Trace Cache

Micro code Sequencer
Micro code ROM & Flash
Trace Cache Fill Buffers
Distributed Tag comparators 24 bit virtual Tags



Trace Cache Access, next Address Predict

Trace Cache Branch Prediction Table (BTB), 512 entries.

Return Stacks (2x16 entries)

Trace Cache next IP's (2x)

Miscellaneous Tag Data

Instruction Decoder

Up to 4 decoded uOps/cycle out. (from max. one x86 instr/cycle) Instructions with more than four are handled by Micro Sequencer

Trace Cache LRU bits

Raw Instruction Bytes in Data TLB, 64 entry fully associative, between threads dual ported (for loads and stores)

Instruction Fetch from L2 cache and Branch Prediction

Front End Branch Prediction Tables (BTB), shared, 4096 entries in total

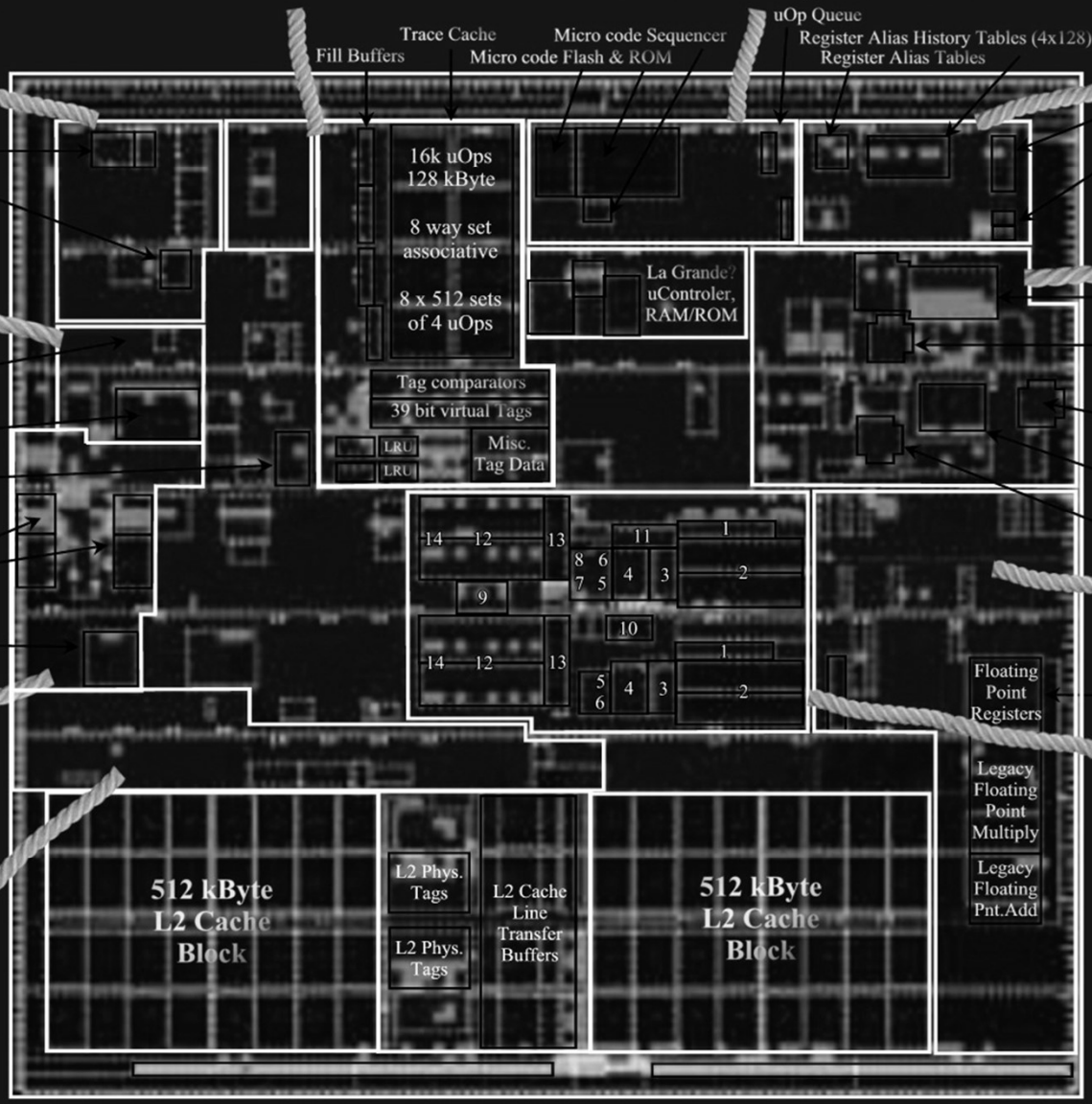
Instruction TLB's 2x64 entry, fully associative for 4k and 4M pages. In: Virtual address [31:12] Out: Physical address [35:12] + 2 page level bits

Front Side Bus Interface, 400..800 MHz

(11) ROB Reorder Buffer 3x42 entries
(12) 8 kByte Level 1 Data cache four way set associative. 1R/1W
(13) Summed Address Index decode and Way Predict
(14) Cache Line Read / Write Transferbuffers and 256 bit wide bus to and from L2 cache

April 19, 2003 www.chip-architect.com

Intel Pentium 5 Prescott



Trace Cache Access, next Address Predict

Trace Cache Branch Prediction Table (BTB), 1024 entries.

Return Stacks (4 x16 entries)

Trace Cache next IP's (4x)

Instruction Trace Cache

Execution Pipeline Start

Buffer Allocation & Register Rename

Instruction Queue (for less critical fields of the uOps)

General Instruction Address Queue & Memory Instruction Address Queue (queues register entries and latency fields of the uOps for scheduling)

Instruction Decoder

Up to 4 decoded uOps/cycle out. (from max. one x86 instr/cycle) Instructions with more than four are handled by Micro Sequencer

Raw Instruction Bytes in

Data TLB, 64 entry fully associative, between threads dual ported (for loads and stores)

Front End Branch Prediction Tables (BTB), shared, 4096 entries in total

Instruction TLB's 128 entry, fully associative for 4k and 4M pages. In: Virtual address [47:12] Out: Physical address [39:12] + 2 page level bits

uOp Schedulers

Parallel (Matrix) Scheduler for the two double pumped ALU's

General Floating Point and Slow Integer Scheduler: (8x8 dependency matrix)

FP Move Scheduler: (8x8 dependency matrix)

Load / Store Linear Address Collision History Table

Load / Store uOp Scheduler: (8x8 dependency matrix)

FP, MMX, SSE1..3

Floating Point, MMX, SSE1..3 Renamed Register File 256 entries of 128 bit.

Instruction Fetch from L2 cache and Branch Prediction

Integer Execution Core

(1) uOp Dispatch unit & Replay Buffer Dispatches up to 6 uOps / cycle

(2) Integer Renamed Register File 256 entries of 32 bit (+ 6 status flags) 12 read ports and six write ports

(3) Databus switch & Bypasses to and from the Integer Register File.

(4) Flags, Write Back

(5) Double Pumped ALU 0

(6) Double Pumped ALU 1

(7) Load Address Generator Unit

(8) Store Address Generator Unit

(9) Load Buffer (96 entries)

(10) Store Buffer (48 entries)

Front Side Bus Interface, 533..800 MHz

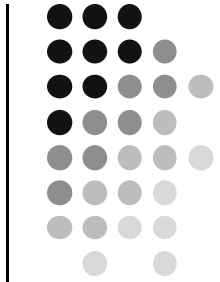
(13) Databus multiplexing

(14) Cache Line Read / Write Transferbuffers and 256 bit wide bus to and from L2 cache

(11) ROB Reorder Buffer 4x64 entries

(12) 16 kByte Level 1 Data cache four way set associative. 1R/1W

April 19, 2003 www.chip-architect.com



There was a Door to which I found no Key

There was a Veil past which I could not see

Some little Talk awhile of ME and THEE

There seemed – and then no more of THEE and ME

Ommar Khayam
The Sage

برای کسب اطلاعات بیشتر در مورد این درس می‌توانید به وب سایت آموزشی در لینک زیر مراجعه نمایید

http://shafieian-education.ir