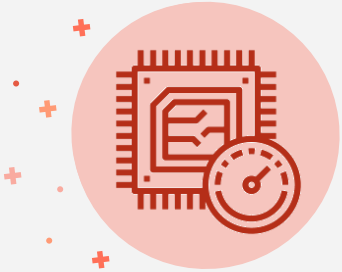


سیستم‌های دیجیتال ۲

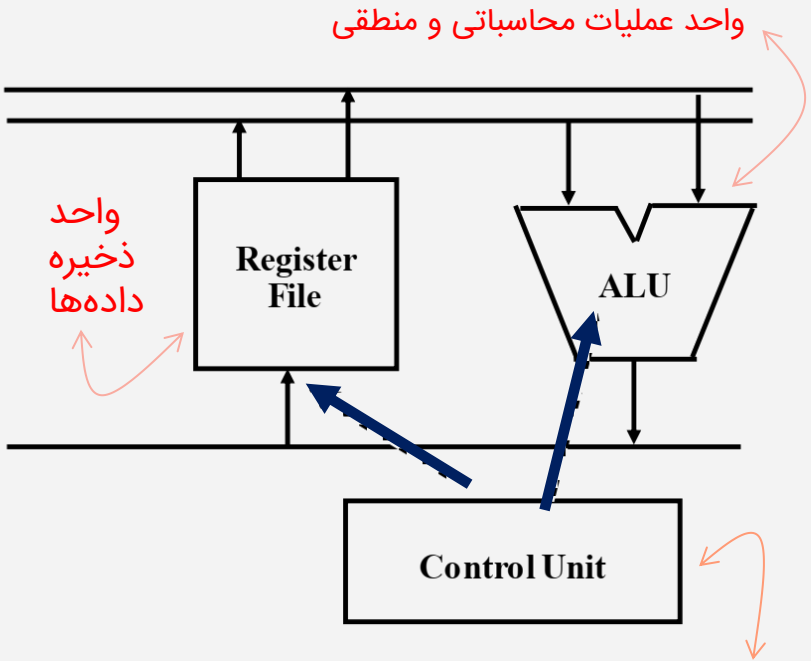
اسلاید ۵

واحد پردازش مرکزی CPU

محمدرضا رازقی‌زاده

اجزای تشکیل دهنده CPU

- Storage Components:
  Registers Flip-flops
- Execution (Processing) Components:
  Arithmetic Logic Unit (ALU): Arithmetic calculations, Logical computations, Shifts/Rotates
- Transfer Components:
  Bus
- Control Components:
  Control Unit



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

CPU از دید استفاده کننده

از دید یك برنامه نویس که به زبان سطح پایین برنامه نویسی می کند CPU دارای مشخصه های زیر خواهد بود:

- ✓ Instruction format
- ✓ The instruction Set
- ✓ Addressing modes
- ✓ Register organization

معماری های مختلف CPU

- ✓ Accumulator machine
- ✓ Register machine
- ✓ Stack machine

مثال: ماشین های X۸۶ دارای یك معماری با مجموعه دستورات پیچیده ای است که تمامی جنبه های معماری های فوق را در بر می گیرد.

ساختار رجیسترها

- یکی از مهمترین ویژگی های تعیین کننده برا ی یك CPUساختار رجیسترهای داخلی آن است. این رجیستر ها به دو دسته تقسیم بندی می شوند:

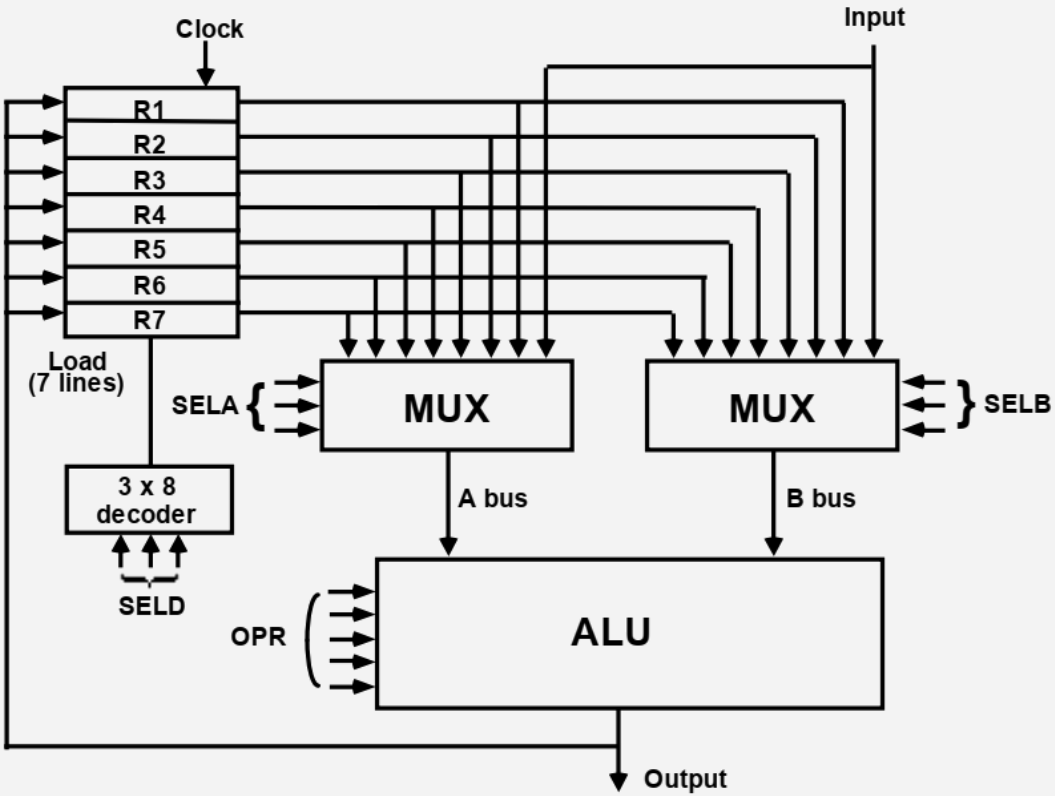
- رجیسترهائی که استفاده کننده آنها را می بیند! و می تواند از طریق برنامه نویسی به آنها دسترسی داشته باشد
  - Data registers
  - Address registers
    - ✓ index register
    - ✓ segment pointer
    - ✓ stack pointer
  - Condition codes (flags)

- رجیسترهائی که برای کنترل و نگهداری وضعیت CPU بکار می روند. این رجیسترها توسط واحد کنترل برای اجرای دستورات مورد استفاده واقع می شوند
  - Program counter
  - Instruction register

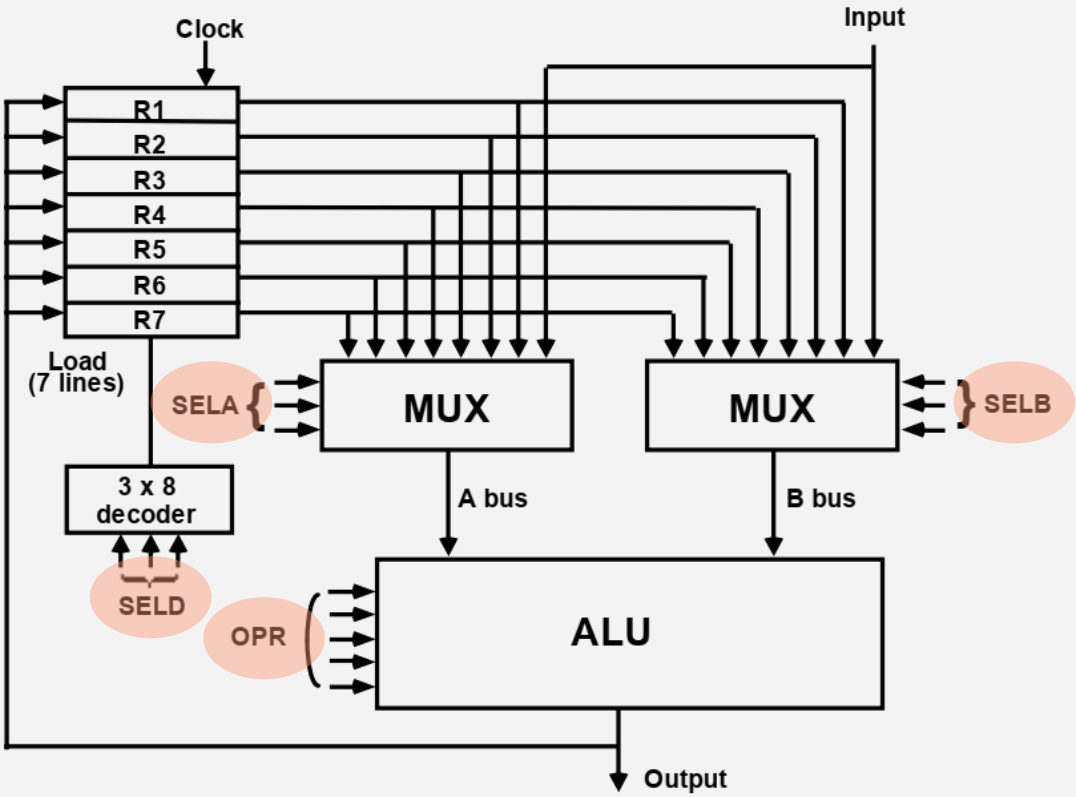
رجیسترهای عمومی

- محلی برای ذخیره داده ها هستند که در داخل CPU تعبیه شده اند تا با فراهم آوردن دسترسی سریع و آسان عملکرد CPU را افزایش دهند.
- این رجیستر ها توسط یك باس مشترك 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 ALU مشترک



مجموعه رجیسترهای عمومی و ALU مشترک



مجموعه رجیسترهای عمومی و ALU مشترک

سازمان باس ترسیم‌شده شامل هفت ثبات در cpu است که عملکرد آن به شرح زیر توضیح می‌باشد:

- خروجی هر ثبات به دو مالتی‌پلکسر متصل است که نقش اساسی در انتقال داده‌ها به ALU دارند.
- دو باس، به نام‌های A و B، برای انتقال داده مورد استفاده قرار می‌گیرند. خطوط انتخاب در هر مالتی‌پلکسر تعیین می‌کنند که داده از یک ثبات انتخاب شود یا از ورودی داده.
- داده‌ها از طریق باس‌های Aو Bبه ALUمنتقل می‌شوند.
- سیگنال OPR نوع عملیاتی را که ALU باید انجام دهد، مشخص می‌کند.
- نتیجه عملیات انجام‌شده توسط ALU می‌تواند به سایر واحدهای سیستم ارسال شود یا در یکی از ثبات‌های پردازنده ذخیره گردد.
- برای انتخاب ثبات مقصد که نتیجه در آن ذخیره می‌شود، از یک دیکدر استفاده می‌شود. این دیکدر یکی از ورودی‌های بارگذاری ثبات را فعال کرده و ثبات مقصد برای ذخیره نتیجه را مشخص می‌کند.

مثالی از عملیات ALU

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

$$R1 \rightarrow R2 + R3$$

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

۱. تعیین مقدار مناسب برای ورودی MUXA یعنی SELA طوری که محتوی رجیستر R۲ در روی باس A قرار گیرد.

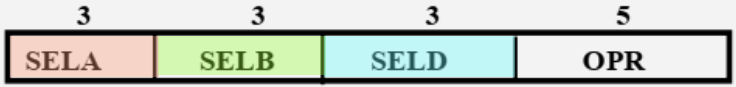
۲. تعیین مقدار مناسب برای ورودی MUXB یعنی SELB طوری که محتوی رجیستر R۳ در روی باس B قرار گیرد.

۳. تعیین مقدار لازم برای ورودی OPR که ALU را وادار به انجام عمل جمع A+B نماید.

۴. در نهایت انتخاب مقدار مناسب برای دیکدر SELD به نحوی که خروجی ALU را به رجیستر R۱ منتقل نماید

سیگناهای کنترلی

- هر ۴ دسته سیگنال فوق باید بطور همزمان توسط واحد کنترل ایجاد شده و در شروع یك سیكل كلاك آماده باشند.
- در طول یك سیكل كلاك 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 CPU سریعی داشته باشیم باید ALU با مداراتی ساخته شود که بتوانند به سرعت عمل مورد نظر را انجام دهند.



Binary Code	SEL A	SEL B	SEL D
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 یك کلمه کنترل نامیده میشود. این کلمه به ۴ فیلد تقسیم می شود

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

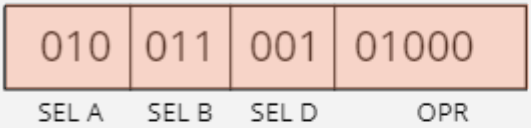
OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	A+B	ADD
00101	A-B	SUB
00110	Decrement A	DECA
01000	A.B	AND
01010	OR A and B	OR
01100	XOR A B	XOR
01110	Complement A	COMA
10000	Shift Right A	SHRA
11000	Shift Left A	SHLA

مثال

برای انجام ریزعمل زیر

$$R1 \rightarrow R2 - R3$$

می بایست کلمه کنترلی بصورت زیر انتخاب شود:



1. SEL A = 010. So, MUX A transfer the content of R2 into bus A.
2. SEL B = 011. So, MUX B transfer the content of R3 into bus B.
3. OPR = 01000. So, ALU performs addition.
4. SEL D = 001. to transfer the result in register R1.

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

پشته (Stack)

- حافظه یک CPU می‌تواند به صورت یک پشته (STACK) سازماندهی شود؛ ساختاری که اطلاعات در آن به صورت آخرین ورودی، اولین خروجی LIFO ذخیره می‌شوند. این بدین معناست که آخرین آیتم ذخیره‌شده اولین آیتمی است که حذف یا از پشته برداشته می‌شود (پاپ می‌شود).
- برای مدیریت آیتم‌ها، پشته از یک رجیستر اشاره‌گر پشته (SP: Stack Pointer) استفاده می‌کند. اشاره‌گر پشته، آدرس آخرین آیتم موجود در پشته را ذخیره می‌کند و به عبارتی به بالاترین عنصر پشته اشاره می‌کند.
- موارد استفاده از پشته
  - ✓ ذخیره متغیرهای یك برنامه
  - ✓ ذخیره محتوی سایر رجیسترها وقتی که یك برنامه فرعی صدا زده می شود
  - ✓ کمك به ترجمه عملیات محاسباتی با روش RPN

پیاده سازی پشته

دو نوع اصلی پشته وجود دارد:

۱. پشته رجیستری (Register Stack):

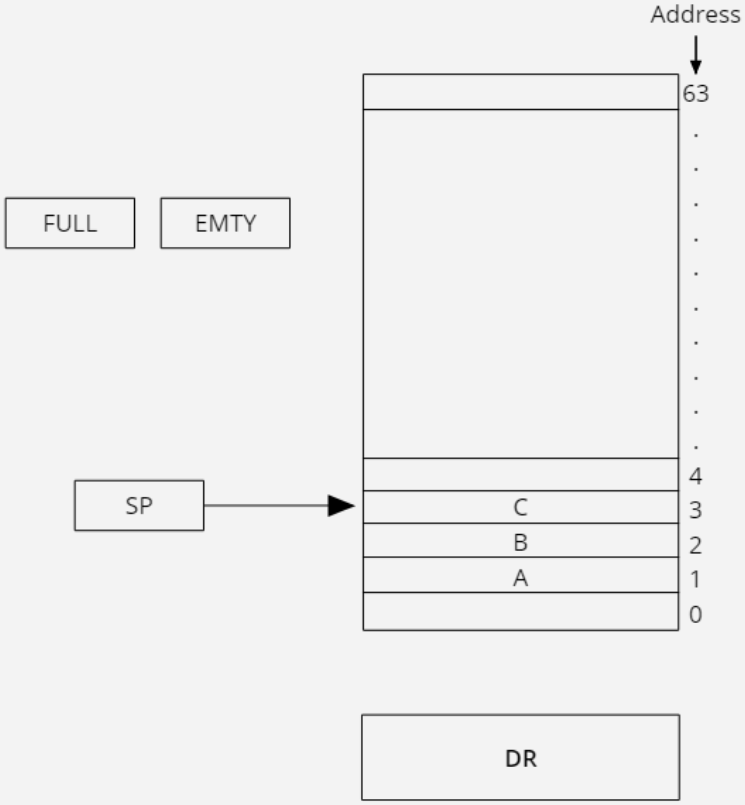
از رجیسترهای پردازنده برای ایجاد ساختار پشته استفاده می‌کند.

این نوع پشته سرعت و کارایی را در برخی عملیات بهبود می‌بخشد.

۱. پشته حافظه‌ای (Memory Stack):

با استفاده از مکان‌های اختصاصی در حافظه، ساختار پشته را پیاده‌سازی می‌کند.

پیاده سازی پشته بوسیله رجیسترها

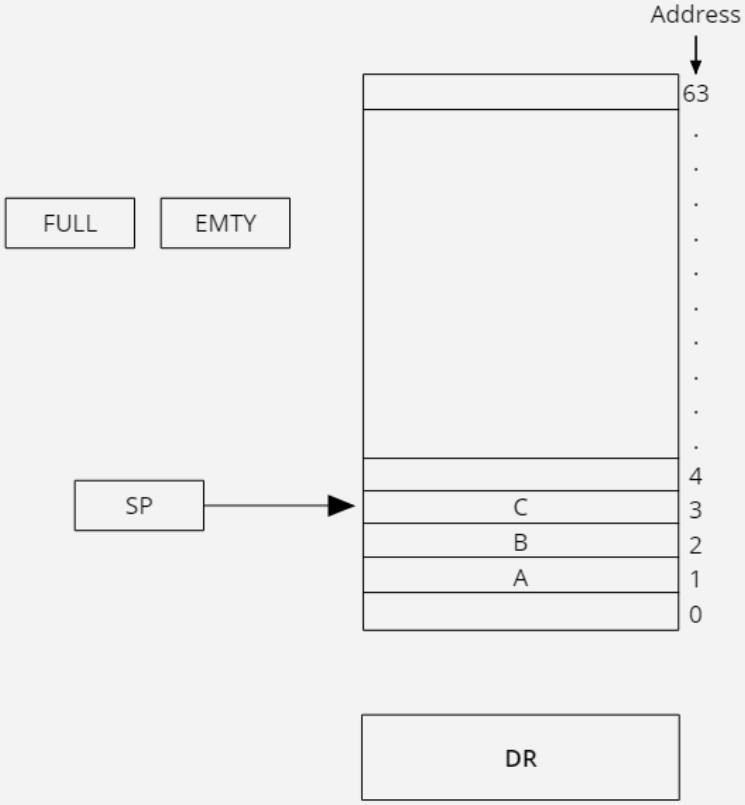


Block diagram of a 64-word stack.

عملیات پشته شامل دو اقدام اصلی است:

- افزودن (Push): زمانی که یک آیتم به پشته اضافه می‌شود، این عمل به عنوان افزودن یا عملیات "پوش" شناخته می‌شود.
- حذف (Pop): زمانی که یک آیتم از پشته حذف می‌شود، این عمل به عنوان حذف یا عملیات "پاپ" شناخته می‌شود.

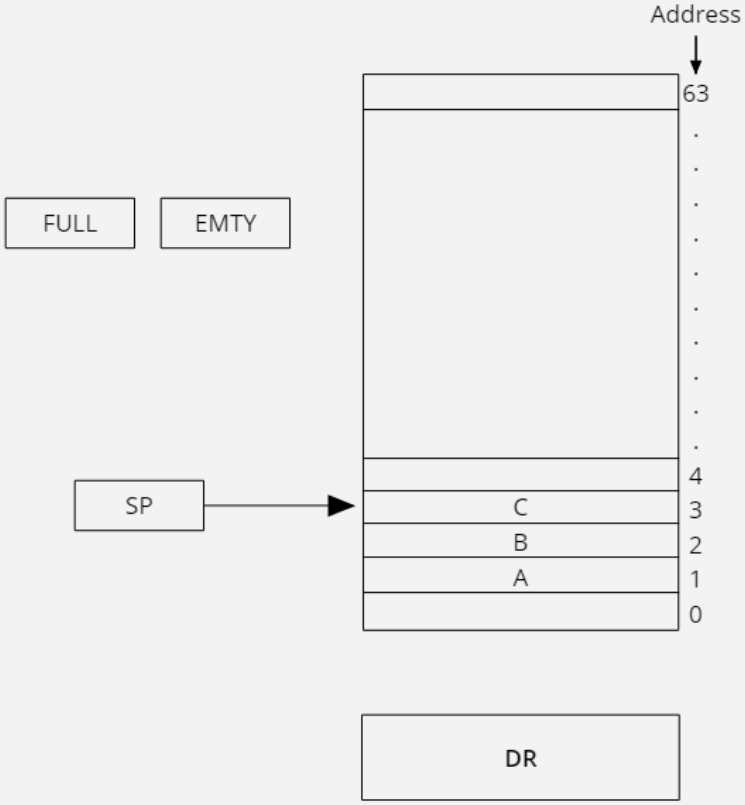
پیاده سازی پشته بوسیله رجیسترها



Block diagram of a 64-word stack.

- هنگامی که رجیسترهای پردازنده به صورت ساختار پشته‌ای سازمان‌دهی شوند، به آن پشته رجیستری (Register Stack) گفته می‌شود.
- در نمودار روبرو، یک پشته رجیستری با ظرفیت ۶۴ کلمه نشان داده شده است.
- اشاره‌گر پشته (SP) آدرس بالاترین عنصر موجود در پشته را ذخیره می‌کند.

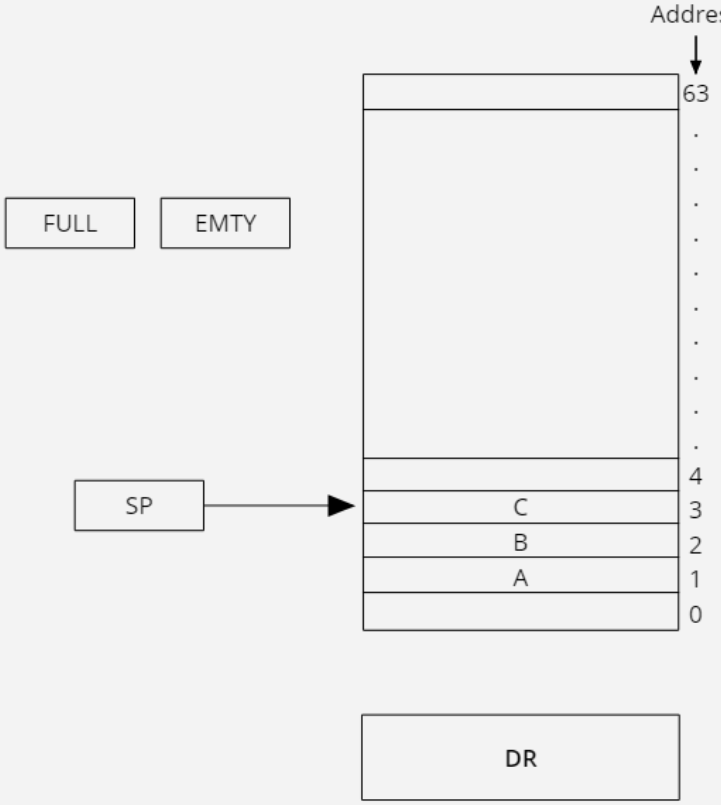
پیاده سازی پشته بوسیله رجیسترها



Block diagram of a 64-word stack.

- زمانی که پشته خالی باشد، پرچم EMPTY برابر ۱ تنظیم می‌شود و وقتی پشته پر باشد، پرچم FULL برابر ۱ می‌شود.
- رجیستر داده (DR) داده‌ای را که در حال برداشتن (Pop) یا اضافه کردن (Push) به پشته است، نگهداری می‌کند.

پیاده سازی پشته بوسیله رجیسترها



Block diagram of a 64-word stack.

- مزایای دیگر پشته رجیستری شامل زمان دسترسی سریع‌تر و کاهش تداخل در باس حافظه است، که آن را برای وظایف محاسباتی نیازمند دستکاری سریع داده مناسب می‌سازد.
- به عنوان مثال، در تصویر، سه آیتم (A، B، و C) در پشته قرار گرفته‌اند که آیتم C در بالاترین موقعیت قرار دارد. بنابراین اشاره‌گر پشته SP آدرس C را نگهداری می‌کند (SP=۳).

عملیات PUSH (افزودن به پشته):

هنگامی که عملیات PUSH برای اضافه کردن یک عنصر (مثلاً E) به پشته انجام می‌شود، مراحل زیر اجرا می‌شوند:

- مرحله ۱:
  - ✓ اشاره‌گر پشته (SP) را یک واحد افزایش دهید تا به یک جای خالی اشاره کند.
  - ✓ SP←SP+۱ (افزایش اشاره‌گر پشته)

- مرحله ۲:
- مقدار موجود در رجیستر داده (DR) را در آدرسی که توسط SP اشاره شده است ذخیره کنید.
- M[SP]←DR (ذخیره آیتم در بالای پشته)

- مرحله ۳:
- شرایط مرزی را بررسی کنید.
- اگر SP=۰، آنگاه
- FULL ← ۱ تنظیم می‌شود، به این معنی که پشته پر است.
- EMPTY ← ۰ تنظیم می‌شود، که نشان می‌دهد پشته خالی نیست.

➢ Push

- ✓ SP←SP+۱
- ✓ M[SP] ← DR ,EMTY ← ۰
- ✓ if(SP=۰) Then (FULL ← ۱)

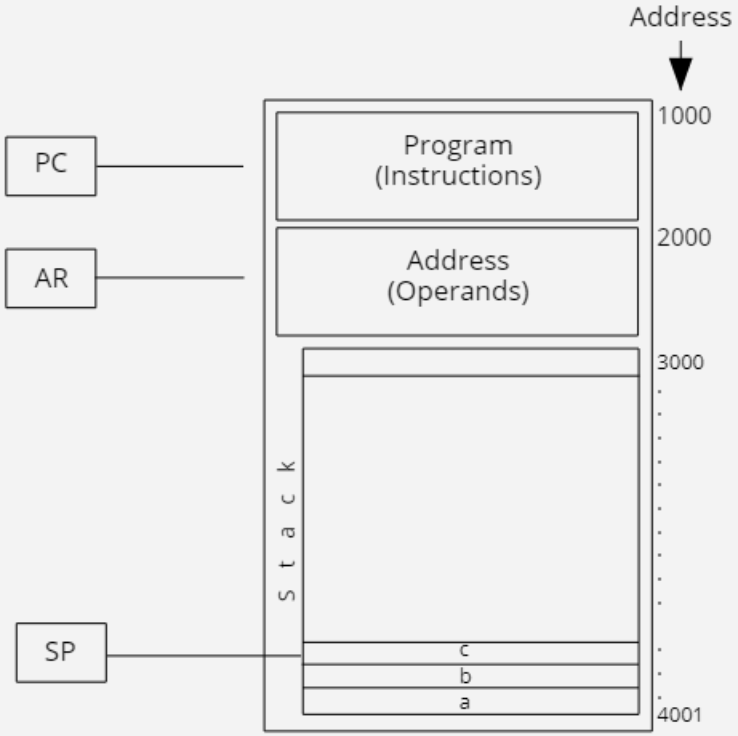
عملیات POP (حذف از پشته):

هنگامی که عملیات POP برای حذف کردن یک عنصر (مثلاً E) از پشته انجام می‌شود، مراحل زیر اجرا می‌شوند:

- مرحله ۱:
  - ✓ داده را از آدرسی که توسط اشاره‌گر پشته (SP) مشخص شده است، بازیابی کرده و در رجیستر داده (DR) ذخیره کنید.
  - ✓ DR←M[SP] (بازیابی آیتم از بالای پشته)
- مرحله ۲:
- اشاره‌گر پشته (SP) را یک واحد کاهش دهید.
- SP←SP-۱ (کاهش اشاره‌گر پشته)
- مرحله ۳:
- شرایط مرزی را بررسی کنید.
- اگر SP=۰، آنگاه
- FULL ← ۰ تنظیم می‌شود، به این معنی که پشته دیگر پر نیست.
- EMPTY ← ۱ تنظیم می‌شود، که نشان می‌دهد پشته خالی است.

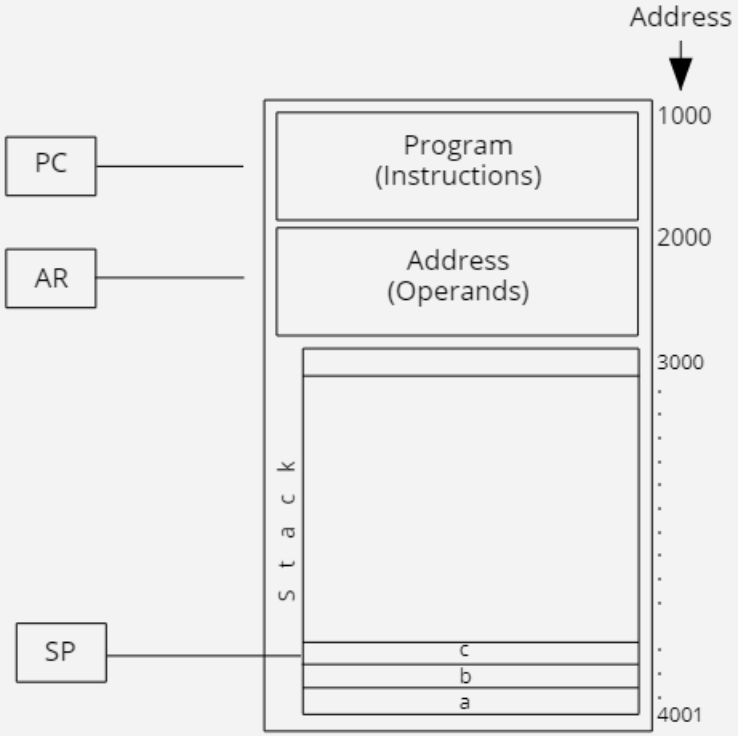
- POP
  - ✓ DR ← M[SP] , FULL ← ۰
  - ✓ SP ← SP-۱
  - ✓ if(SP=۰) Then (EMTY ← ۱)

پیاده سازی پشته در قسمتی از حافظه RAM



- هنگامی که حافظه اصلی (RAM) به صورت یک ساختار پشته سازمان‌دهی شود، به آن پشته حافظه‌ای (Memory Stack) گفته می‌شود.
- شمارنده برنامه (PC) آدرس دستور بعدی در برنامه را نشان می‌دهد.
- رجیستر آدرس (AR) به یک آرایه داده در داخل پشته حافظه اشاره می‌کند.
- پایین پشته (SP) محل بالای پشته را شناسایی می‌کند.
- در شکل نشان داده شده، مقدار اولیه SP برابر با ۴۰۰۱ است و پشته با کاهش آدرس‌ها رشد می‌کند. به همین دلیل، اولین آیتم ذخیره شده در پشته در آدرس ۴۰۰۰ قرار دارد، دومین آیتم در آدرس ۳۹۹۹ و آخرین آیتم در آدرس ۳۰۰۰ قرار می‌گیرد.

پیاده سازی پشته در قسمتی از حافظه RAM



- هنگامی که حافظه اصلی (RAM) به صورت یک ساختار پشته سازمان‌دهی شود، به آن پشته حافظه‌ای (Memory Stack) گفته می‌شود.
- شمارنده برنامه (PC) آدرس دستور بعدی در برنامه را نشان می‌دهد.
- رجیستر آدرس (AR) به یک آرایه داده در داخل پشته حافظه اشاره می‌کند.
- پایین پشته (SP) محل بالای پشته را شناسایی می‌کند.
- در شکل نشان داده شده، مقدار اولیه SP برابر با ۴۰۰۱ است و پشته با کاهش آدرس‌ها رشد می‌کند. به همین دلیل، اولین آیتم ذخیره شده در پشته در آدرس ۴۰۰۰ قرار دارد، دومین آیتم در آدرس ۳۹۹۹ و آخرین آیتم در آدرس ۳۰۰۰ قرار می‌گیرد.

عملیات روی پشته حافظه‌ای

- Push
  - ✓ SP←SP-۱
  - ✓ M[SP] ← DR
- POP
  - ✓ DR ← M[SP]
  - ✓ SP ← SP+۱

- عملیات PUSH شامل کاهش مقدار پایینی پشته (SP) برای تخصیص فضای جدید برای یک آیتم و ذخیره مقدار موجود در رجیستر داده‌ها (DR) در آدرسی است که توسط SP به‌روز شده اشاره می‌کند.
- عملیات POP آیتم را از بالای پشته بازیابی می‌کند، به‌طوری‌که داده‌ها از آدرسی که توسط SP نشان داده می‌شود به DR کپی می‌شوند. سپس، SP افزایش می‌یابد تا فضای پشته آزاد شود.
- هنگام استفاده از پشته حافظه‌ای:

باید مواظب بود که بزرگ شدن پشته باعث دست اندازی آن به داده های سایر برنامه ها در RAM نشود و 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 قرار می 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.

۱. ابتدا عبارت بصورت RPN نوشته می شود (معمولا این کار توسط کامپایلر انجام می شود)

۲. در هنگام محاسبه

- با برخورد به عملوندها آنها را در پشته PUSH می کنیم.
- با برخورد به عملگرها ، دو داده موجود در بالای پشته POP شده و عمل مورد نظر بر روی آنها انجام و حاصل در پشته PUSH می شود.

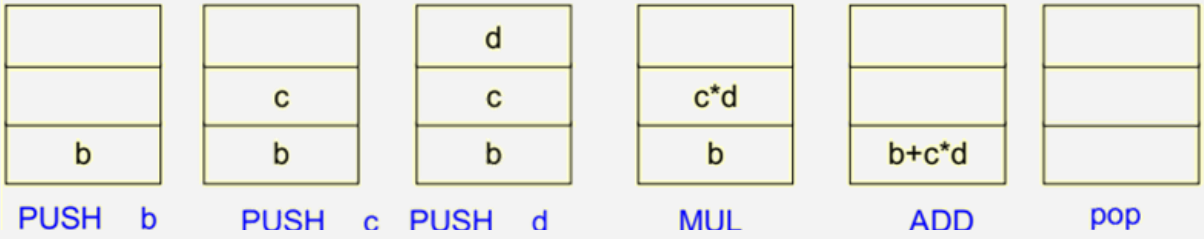
مثال

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

a = b + c * d; → **bcd*+** →

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

در اینصورت محتوی پشته بصورت زیر خواهد بود:



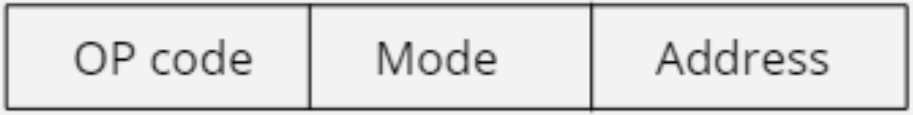
فرمت دستورالعمل

- معمول ترین فرمت یك دستورالعمل

۱. قسمت Opcode مشخص کننده نوع عملیات دستور

۲. قسمت آدرس، مشخص کننده آدرس یك خانه حافظه یا ثبات پردازنده

۳. قسمت حالت آدرس دهی، تعیین کننده عملوند یا آدرس مؤثر



دستورات سه آدرسی

در کامپیوترهای سه آدرسی، هر قسمت آدرس، برای مشخص نمودن یك ثبات پردازنده و یا آدرس یك عملوند در حافظه تخصیص داده می شود.

ADD R۱, A, B	R۱ ← M[A] + M[B]
ADD R۲, C, D	R۲ ← M[C] + M[D]
MUL X, R۱, R۲	M[X] ← R۱ × R۲

دستورات دو آدرسی

دستورات دوآدرسی معمول ترین فرمت دستور در کامپیوترها هستند. قسمت آدرس می تواند یك ثبات پردازنده یا یك خانه حافظه را مشخص نماید.

MOV R۱, A	R۱ ← M[A]
ADD R۱, B	R۱ ← R۱ + M[B]
ADD R۲, D	R۲ ← R۲ + M [D]
MUL R۱, R۲	R۱ ← R۱ × R۲
MOV X, R۱	M[A] ← R۱

دستورات یك آدرسی

دستورات یك آدرسی، برای تمام عملیات بر روی داده ها، ثبات AC را به کار می برد.

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، برای تمام عملیات بر روی داده ها، از پشته استفاده می کند.

Push A

Push B

ADD

Push C

Push D

ADD

MUL

POP X

حالات آدرس دهی

دستورات يك كامپيوتر عملی را بر روی داده ذخيره شده در حافظه و يا رجيسترهای CPU انجام می دهند. روش مشخص كردن عملوند يك دستورالعمل حالات آدرس دهی و يا addressing mode ناميده می شود.

اصولا حالات مختلف آدرس دهی عملوند دستور، تسهيلات زير را در سيستم فراهم می آورد:

۱. قابليت ايجاد شمارنده برای برنامه حلقه، و شاخص بندی در داده ها و هم چنين ايجاد اشاره گر حافظه و جابجايی برای كاربر فراهم می شود

۲. امكان تقليل تعداد بيت های قسمت آدرس دستور، فراهم می شود.

انواع حالت های آدرس دهی

- آدرس دهی ضمنی
- آدرس دهی بلادرنگ
- آدرس دهی ثبات
- آدرس دهی غیر مستقیم بکمك 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

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

- مثل دستور CMA که محتوی آکومولاتور را متمم می کند.
- دستورهای بدون آدرس و تمام دستورهایی که از اکومولاتور استفاده می‌کنند، دستورهای حالت ضمنی هستند. به عنوان مثال، دستور "complement" یک دستور حالت ضمنی است زیرا عملوند در آکومولاتور قرار دارد.

در این حالت، عملوند مستقیماً در خود دستور مشخص می‌شود و به جای فیلد آدرس، فیلد عملوند وجود دارد.

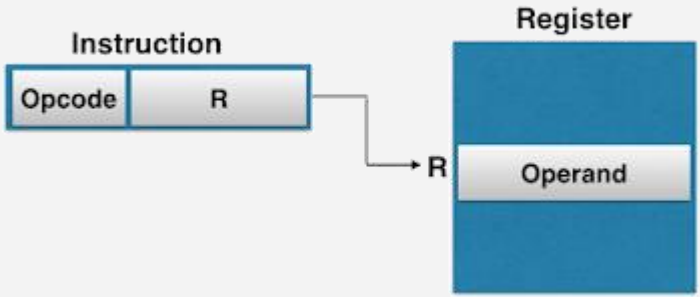
به عنوان مثال:

ADD ۱۰, ۲۰.
MOV CX, 1024

این حالت زمانی استفاده می‌شود که داده‌ها در رجیسترهای پردازنده ذخیره شده باشند و بخش آدرس دستور حاوی آدرس رجیستر پردازنده باشد.

ADD R2, R1

- ✓ در این حالت، داده‌ها نیازی به دسترسی به حافظه وجود ندارد، به این معنی که: EA=(R) به عبارت دیگر، کنترل ابتدا یک دستورالعمل را از حافظه واکشی می‌کند و سپس از آدرس آن برای دسترسی به رجیستر استفاده کرده و در رجیستر (R) به دنبال آدرس مؤثر عملوند در حافظه می‌گردد.

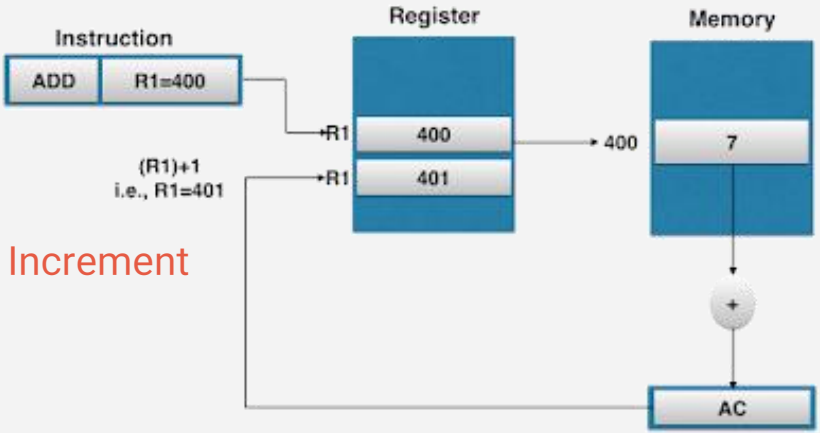


آدرس دهی غیر مستقیم به کمك ثبات

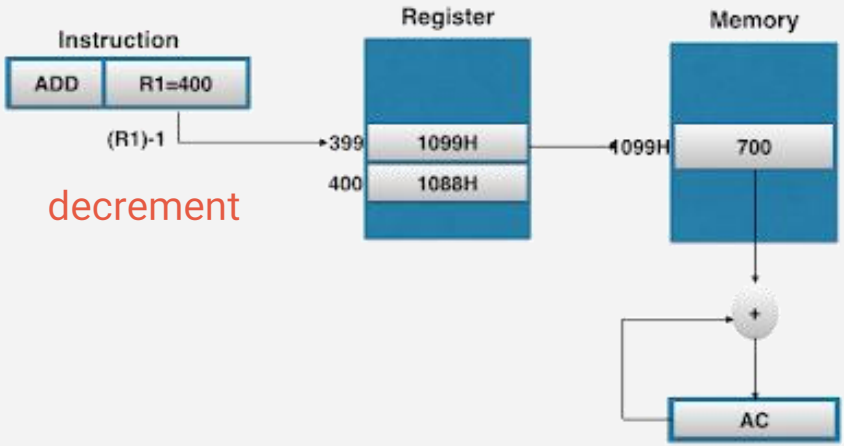
دستور دارای آدرس یک رجیستر پردازنده است که آدرس عملوند را در حافظه نگه‌داری می‌کند.

MOV BX,[SI]

این حالت زمانی استفاده می‌شود که یک سری داده‌ها از حافظه بازیابی شوند و بخش آدرس دستور آدرس شروع را مشخص می‌کند. این آدرس به ترتیب افزایش یا کاهش می‌یابد تا داده‌های بعدی از حافظه بازیابی شوند.



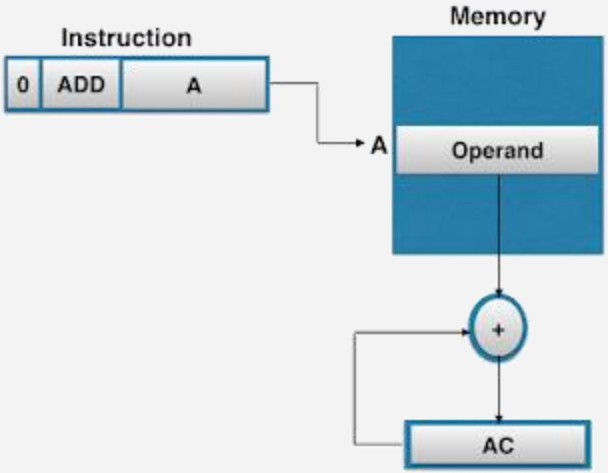
این حالت زمانی استفاده می‌شود که یک سری داده‌ها از حافظه بازیابی شوند و بخش آدرس دستور آدرس شروع را مشخص می‌کند. این آدرس به ترتیب افزایش یا کاهش می‌یابد تا داده‌های بعدی از حافظه بازیابی شوند.



آدرس دهی مستقیم

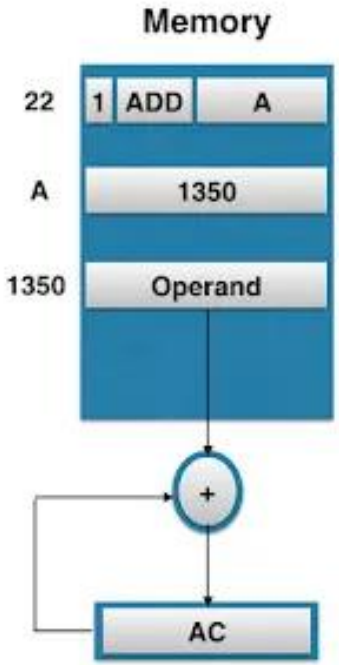
در این روش آدرس عملوند در داخل دستورالعمل ذکر می شود.

MOV AX,[۳۰۰۰]



آدرس دهی غیرمستقیم

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 .

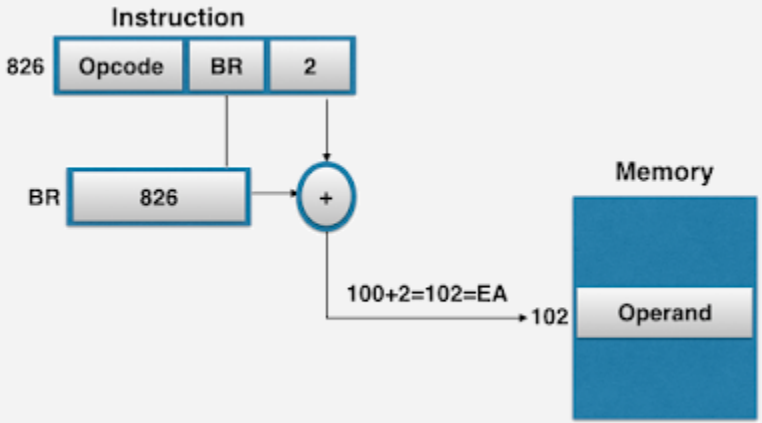


آدرس دهی نسبی

- 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

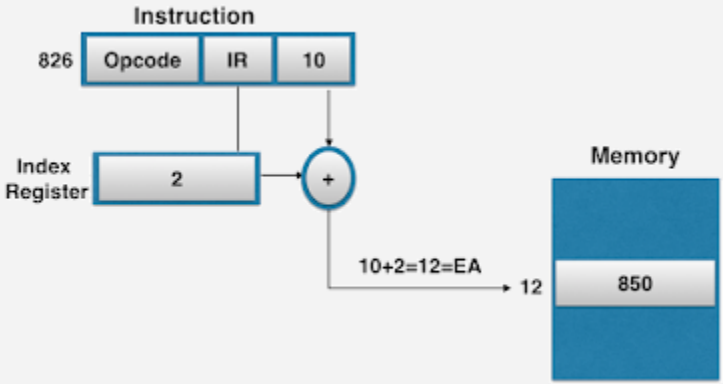


آدرس دهی شاخص

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

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

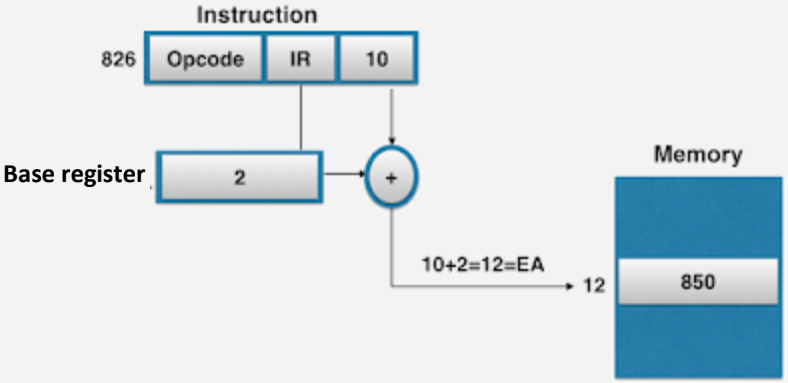
معمولا از این روش برای دسترسی به داده های یك آرایه استفاده می شود که محل شروع داده ها در حافظه در دستورالعمل مشخص می شود و فاصله داده مورد نظر تا محل شروع توسط رجیستر Index تعیین می گردد.



آدرس دهی با ثبات پایه

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



مثال عددی

مقدار اکومولاتور در صورت اجرای دستور موجود در آدرس ۲۰۰ برای حالتهای مختلف آدرس دهی چیست؟

Address	Memory	
200	Load to AC	Mode
201	Address = 500	
202	Next instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

Addressing Mode	Effective Address	Content of AC
Direct address		
Immediate operand		
Indirect address		
Relative address		
Indexed address		
Register		
Register indirect		
Autoincrement		
Autodecrement		

PC = 200 | XR = 100

R1 = 400 | AC

مثال عددی

مقدار اکومولاتور در صورت اجرای دستور موجود در آدرس ۲۰۰ برای حالتهای مختلف آدرس دهی چیست؟

Address	Memory	
200	Load to AC	Mode
201	Address = 500	
202	Next instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

Addressing Mode	Effective Address		Content of AC
Direct address	500	/* AC ← (500) */	800
Immediate operand			
Indirect address			
Relative address			
Indexed address			
Register			
Register indirect			
Autoincrement			
Autodecrement			

PC = 200

XR = 100

R1 = 400

AC

مثال عددی

مقدار اکومولاتور در صورت اجرای دستور موجود در آدرس ۲۰۰ برای حالتهای مختلف آدرس دهی چیست؟

Address	Memory	
200	Load to AC	Mode
201	Address = 500	
202	Next instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

Addressing Mode	Effective Address		Content of AC
Direct address	500	/* AC ← (500) */	800
Immediate operand	-	/* AC ← 500 */	500
Indirect address			
Relative address			
Indexed address			
Register			
Register indirect			
Autoincrement			
Autodecrement			

PC = 200

XR = 100

R1 = 400

AC

مثال عددی

مقدار اکومولاتور در صورت اجرای دستور موجود در آدرس ۲۰۰ برای حالتهای مختلف آدرس دهی چیست؟

Address	Memory	
200	Load to AC	Mode
201	Address = 500	
202	Next instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

Addressing Mode	Effective Address		Content of AC
Direct address	500	/* AC ← (500) */	800
Immediate operand	-	/* AC ← 500 */	500
Indirect address	800	/* AC ← ((500)) */	300
Relative address			
Indexed address			
Register			
Register indirect			
Autoincrement			
Autodecrement			

PC = 200 | XR = 100

R1 = 400 | AC

مثال عددی

مقدار اکومولاتور در صورت اجرای دستور موجود در آدرس ۲۰۰ برای حالتهای مختلف آدرس دهی چیست؟

Address	Memory	
200	Load to AC	Mode
201	Address = 500	
202	Next instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

Addressing Mode	Effective Address		Content of AC
Direct address	500	/* AC ← (500) */	800
Immediate operand	-	/* AC ← 500 */	500
Indirect address	800	/* AC ← ((500)) */	300
Relative address	702	/* AC ← (PC+500) */	325
Indexed address			
Register			
Register indirect			
Autoincrement			
Autodecrement			

PC = 200 | XR = 100

R1 = 400 | AC

مثال عددی

مقدار اکومولاتور در صورت اجرای دستور موجود در آدرس ۲۰۰ برای حالتهای مختلف آدرس دهی چیست؟

Address	Memory	
200	Load to AC	Mode
201	Address = 500	
202	Next instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

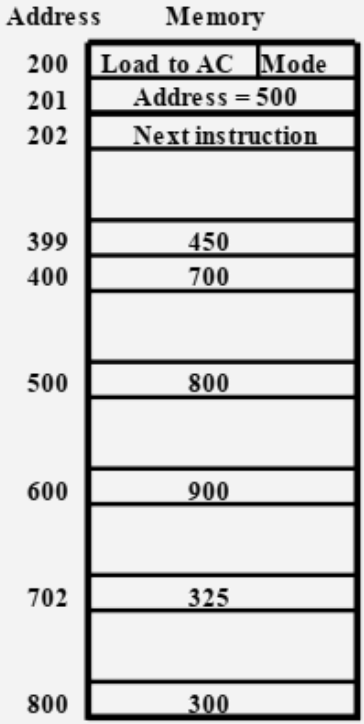
Addressing Mode	Effective Address		Content of AC
Direct address	500	/* AC ← (500) */	800
Immediate operand	-	/* AC ← 500 */	500
Indirect address	800	/* AC ← ((500)) */	300
Relative address	702	/* AC ← (PC+500) */	325
Indexed address	600	/* AC ← (XR+500) */	900
Register			
Register indirect			
Autoincrement			
Autodecrement			

PC = 200 | XR = 100

R1 = 400 | AC

مثال عددی

مقدار اکومولاتور در صورت اجرای دستور موجود در آدرس ۲۰۰ برای حالتهای مختلف آدرس دهی چیست؟



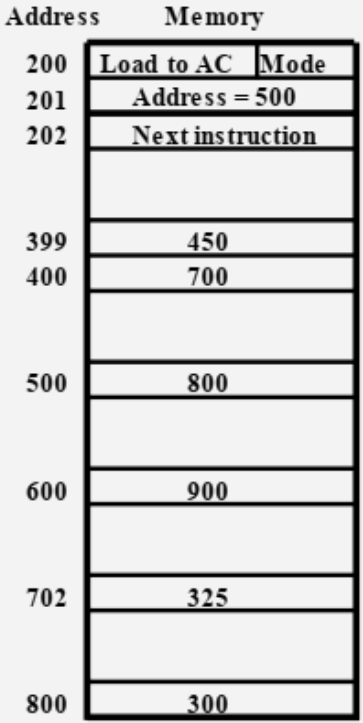
Addressing Mode	Effective Address		Content of AC
Direct address	500	/* AC ← (500) */	800
Immediate operand	-	/* AC ← 500 */	500
Indirect address	800	/* AC ← ((500)) */	300
Relative address	702	/* AC ← (PC+500) */	325
Indexed address	600	/* AC ← (XR+500) */	900
Register	-	/* AC ← R1 */	400
Register indirect			
Autoincrement			
Autodecrement			

PC = 200 | XR = 100

R1 = 400 | AC

مثال عددی

مقدار اکومولاتور در صورت اجرای دستور موجود در آدرس ۲۰۰ برای حالتهای مختلف آدرس دهی چیست؟



Addressing Mode	Effective Address			Content of AC
Direct address	500	/* AC ← (500)	*/	800
Immediate operand	-	/* AC ← 500	*/	500
Indirect address	800	/* AC ← ((500))	*/	300
Relative address	702	/* AC ← (PC+500)	*/	325
Indexed address	600	/* AC ← (XR+500)	*/	900
Register	-	/* AC ← R1	*/	400
Register indirect	400	/* AC ← (R1)	*/	700
Autoincrement				
Autodecrement				

PC = 200

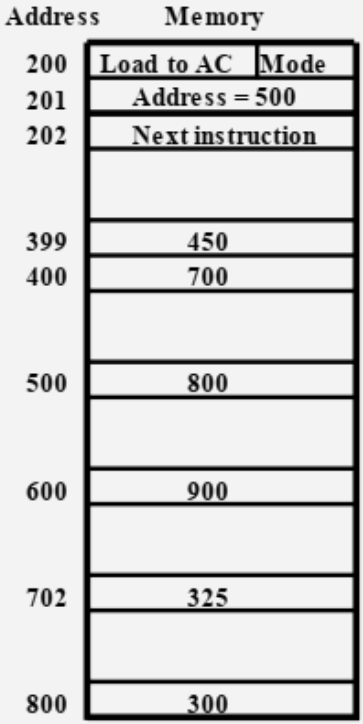
XR = 100

R1 = 400

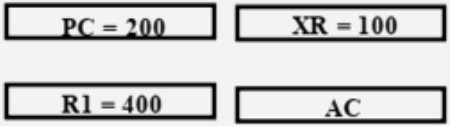
AC

مثال عددی

مقدار اکومولاتور در صورت اجرای دستور موجود در آدرس ۲۰۰ برای حالتهای مختلف آدرس دهی چیست؟

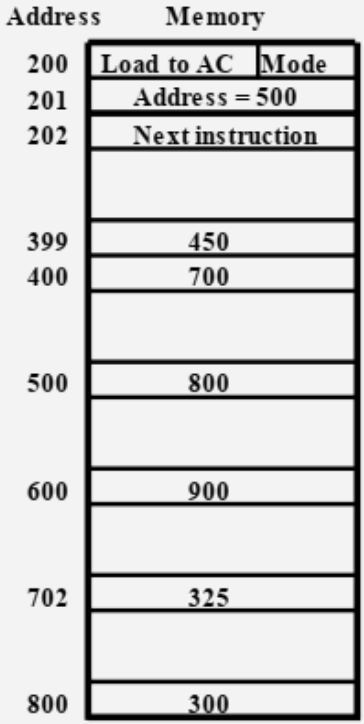


Addressing Mode	Effective Address		Content of AC
Direct address	500	/* AC ← (500) */	800
Immediate operand	-	/* AC ← 500 */	500
Indirect address	800	/* AC ← ((500)) */	300
Relative address	702	/* AC ← (PC+500) */	325
Indexed address	600	/* AC ← (XR+500) */	900
Register	-	/* AC ← R1 */	400
Register indirect	400	/* AC ← (R1) */	700
Autoincrement	400	/* AC ← (R1)+ */	700
Autodecrement			



مثال عددی

مقدار اکومولاتور در صورت اجرای دستور موجود در آدرس ۲۰۰ برای حالتهای مختلف آدرس دهی چیست؟



Addressing Mode	Effective Address		Content of AC
Direct address	500	/* AC ← (500) */	800
Immediate operand	-	/* AC ← 500 */	500
Indirect address	800	/* AC ← ((500)) */	300
Relative address	702	/* AC ← (PC+500) */	325
Indexed address	600	/* AC ← (XR+500) */	900
Register	-	/* AC ← R1 */	400
Register indirect	400	/* AC ← (R1) */	700
Autoincrement	400	/* AC ← (R1)+ */	700
Autodecrement	399	/* AC ← -(R) */	450

PC = 200

XR = 100

R1 = 400

AC

مجموعه دستورات کامپیوتر

- دستورات مورد استفاده در کامپیوترهای مختلف از لحاظ تعداد، عملکرد، نشانه‌های مورد استفاده برای اسمبلی، و کد باینری بسیار متفاوت هستند با این وجود تمامی آنها دارای دستوراتی از انواع زیر می‌باشند:

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

هشت حالت آدرس دهی برای دستور بارکردن

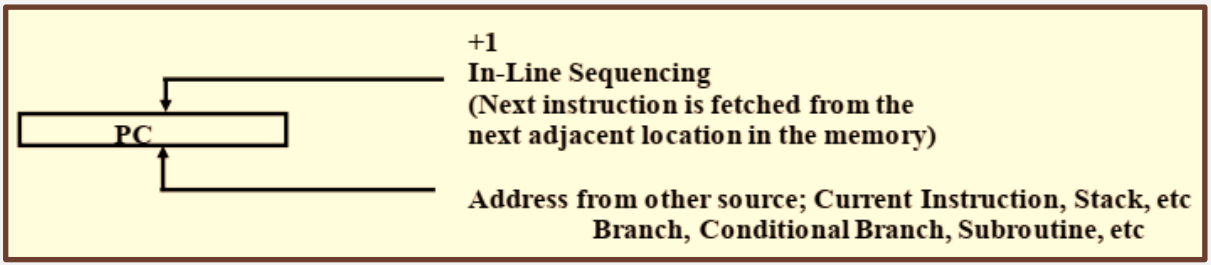
Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	AC ← M[ADR]
Indirect address	LD @ADR	AC ← M[M[ADR]]
Relative address	LD $ADR	AC ← M[PC + ADR]
Immediate operand	LD #NBR	AC ← NBR
Index addressing	LD ADR(X)	AC ← M[ADR + XR]
Register	LD R1	AC ← R1
Register indirect	LD (R1)	AC ← M[R1]
Autoincrement	LD (R1)+	AC ← M[R1], R1 ← R1 + 1
Autodecrement	LD -(R1)	R1 ← R1 - 1, AC ← M[R1]

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with Carry	ADDC
Subtract with Borrow	SUBB
Negate(2's Complement)	NEG

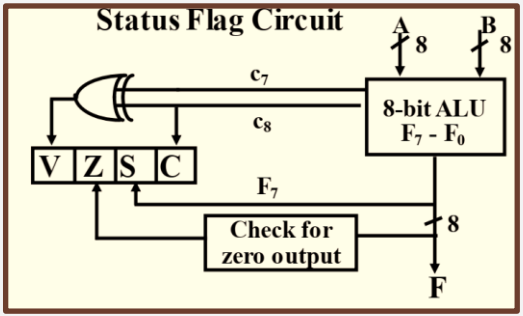
Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right thru carry	RORC
Rotate left thru carry	ROLC

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RTN
Compare(by -)	CMP
Test (by AND)	TST



Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RTN
Compare(by -)	CMP
Test (by AND)	TST



Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	Z = 1
BNZ	Branch if not zero	Z = 0
BC	Branch if carry	C = 1
BNC	Branch if no carry	C = 0
BP	Branch if plus	S = 0
BM	Branch if minus	S = 1
BV	Branch if overflow	V = 1
BNV	Branch if no overflow	V = 0
	Unsigned compare conditions (A - B)	
BHI	Branch if higher	A > B
BHE	Branch if higher or equal	A ≥ B
BLO	Branch if lower	A < B
BLOE	Branch if lower or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B
	Signed compare conditions (A - B)	
BGT	Branch if greater than	A > B
BGE	Branch if greater or equal	A ≥ B
BLT	Branch if less than	A < B
BLE	Branch if less or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B

وقفه

- وقفه (Interrupt) یک مکانیسم مهم است که به پردازنده اجازه می‌دهد کار فعلی خود را متوقف کند و به یک درخواست فوری از یک دستگاه خارجی یا یک رویداد خاص پاسخ دهد.
- این مکانیسم شبیه این است که وقتی کسی در حال انجام کاری است، زنگ در به صدا درمی‌آید و او مجبور می‌شود به طور موقت کارش را کنار بگذارد و به زنگ پاسخ دهد.
- منظور از وقفه انتقال کنترل برنامه از یك برنامه در حال اجرا به یك برنامه سرویس دهی خاص در نتیجه یك درخواست داخلی یا خارجی است که پس از اجرای روال سرویس دهی، کنترل به برنامه اصلی باز می گردد.

مقایسه وقفه و زیرروال

- روال وقفه مشابه فراخوانی زیرروال است با این تفاوت که :

✓ برنامه فرعی (زیرروال) در اثر اجرای یك دستور شروع می شود، اما روال وقفه به علت اعمال یك 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 زیرروال، تنها محتوای PC ذخیره می شود؛ در صورتی که به هنگام اجرای یك 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 یك دستگاه I/O اعمال می شود.
- وقفه های داخلی
  - در اثر استفاده غلط و نابجای یك دستور حاصل می‌شود. مثل رخ‌دادن Overflow انجام یك تقسیم بر صفر، سرریز شدن پشته که متناسب با هر کدام از آنها یك سرویس‌دهی وقفه اجرا می شود که معمولا پیغامی است در جهت تصحیح داده یا دستورالعمل .این وقفه گاهی trap نامیده می‌شود.
- وقفه های نرم افزاری
  - نوع خاصی از صدا زدن برنامه فرعی هستند . و در اثر یك دستورالعمل اجرا می شود. برای انجام عملیات خاصی نظیر تغییر مد برنامه از حالت user به supervisor بکار می‌روند. تفاوت آن با اجرای زیرروال آن است که علاوه بر ذخیره محتوای PC وضعیت کلی CPU را نیز ذخیره می‌کند.

کامپیوترهای با مجموعه دستورات کاهش یافته RISC

- دستورات ساده و کمی دارند
- هدف این است که بتوان دستورات را به سرعت اجرا کرد. اغلب دستورات RISC در یك 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 کوچك 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 تکنیك 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

کامپیوترهای با مجموعه دستورات پیچیده CISC

- معماری یك 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 گاهاً تا ۲۰۰ دستور و تعداد بسیار زیادی مد آدرس دهی داشتند. این کامپیوتر ها Complex Instruction Set Computer (CISC) را می نامند.

ویژگی های کامپیوترهای CISC

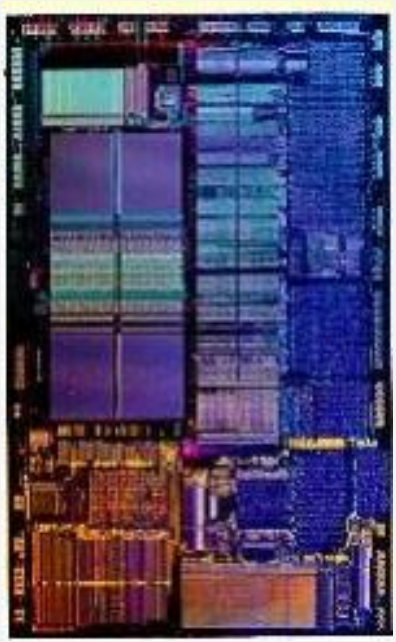
- تعداد زیادی دستورالعمل دارند (۱۰۰ تا ۲۰۰ عدد)
- دستوراتی برای انجام کارهای ویژه دارند که معمولا به ندرت مورد استفاده قرار می گیرند.
- تعداد زیادی مد آدرس دهی دارند.
- طول دستورالعمل ها متغیر است پس کدگشایی (Decoding) آن پیچیده است.
- دستوراتی دارند که عملیاتی را بر روی اپراندهای موجود در حافظه انجام می دهند و مستقیما داده های حافظه را دستکاری می کنند.
- ۶۳برای اجرای دستورات به چندین کلاك نیاز هست.

مقایسه معماری RISC و CISC

ویژگی‌ها	CISC	RISC
تعداد دستورات	تعداد زیادی دستور پیچیده	تعداد کمی دستور ساده و محدود
پیچیدگی دستورها	دستورات پیچیده، گاهی چندین کار در یک دستور	دستورات ساده، هر دستور یک کار مشخص انجام می‌دهد
طول دستورها	طول متغیر	طول ثابت (معمولاً یک کلمه یا چند بایت)
اجرا در هر سیکل	ممکن است اجرای برخی دستورات چندین سیکل طول بکشد	بیشتر دستورات در یک سیکل کلاک اجرا می‌شوند
واحد کنترل	کنترل میکروبرنامه‌ای پیچیده‌تر و کندتر	کنترل سخت‌افزاری ساده‌تر و سریع‌تر
حافظه	بیشتر از حافظه استفاده می‌کند	استفاده بیشتر از ثبات‌ها برای عملیات
کد ماشین	کدهای ماشین پیچیده و بزرگ‌تر	کدهای ماشین ساده و کم‌حجم‌تر
مصرف انرژی	مصرف انرژی بیشتر	مصرف انرژی کمتر
کارایی	مناسب برای کارهای پیچیده با دستورات متنوع	مناسب برای کارهایی با نیاز به پردازش سریع و یکنواخت
مثال پردازنده‌ها	Intel x۸۶، AMD	ARM، MIPS، PowerPC

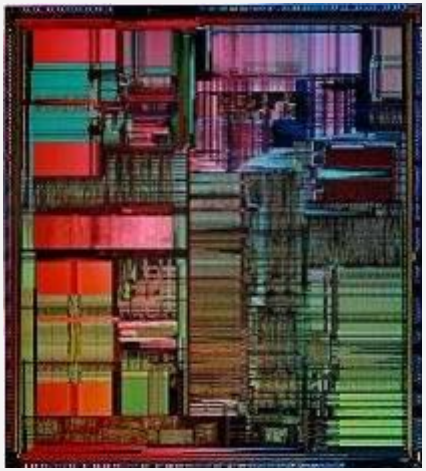
Intel ۴۸۶™ DX CPU

- ✓ Design ۱۹۸۶ – ۱۹۸۹
- ✓ ۲۵ MHz, ۳۳ MHz
- ✓ ۱.۲ M transistors
- ✓ ۱.۰ micron
- ✓ ۵ stage pipeline
- ✓ Unified ۸ KByte code/data cache (write-through)
- ✓ First IA-۳۲ processor capable of executing ۱ instruction per clock cycle



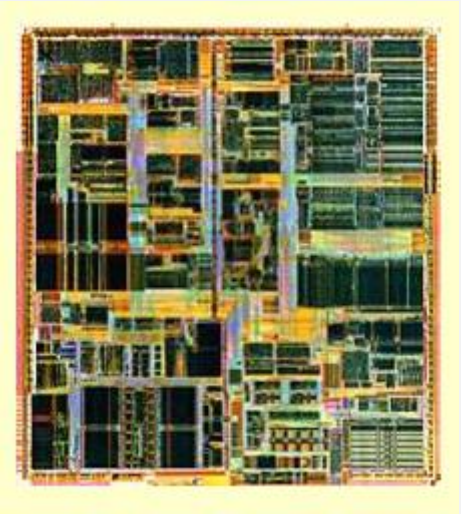
Pentium® Processor

- Design ۱۹۸۹ – ۱۹۹۳
- ۶۰ MHz, ۶۶ MHz
- ۳.۱ M transistors
- ۰.۸ micron
- ۵ stage pipeline
- ۸ KByte instruction and ۸ Kbyte data caches (writeback)
- Branch predictor
- Pipelined floating point
- First superscalar IA-۳۲: capable of executing ۲ instructions per clock



Pentium® II Processor

- Design ۱۹۹۵ – ۱۹۹۷
- ۲۳۳ MHz, ۲۶۶ MHz, ۳۰۰ MHz
- ۷.۵ M transistors
- ۰.۳۵ micron
- ۱۶ KByte L۱I, ۱۶ KByte L۱D, ۵۱۲ KByte off-die L۲
- First compaction of P۶ microarchitecture

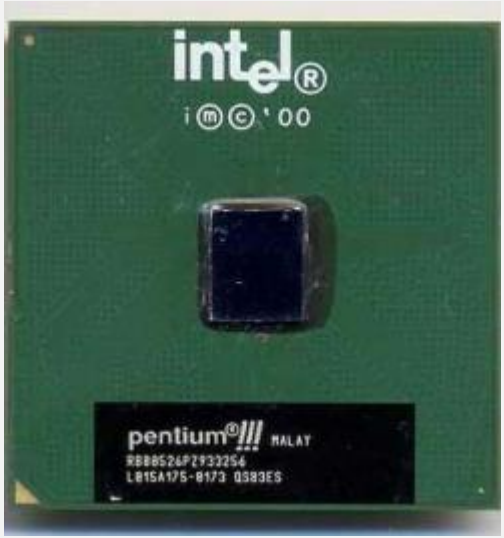


Pentium® III Processor (Katmai)

- ✓ Introduced: ۱۹۹۹
- ✓ ۴۵۰ MHz, ۵۰۰ MHz, ۵۳۳ MHz, ۶۰۰MHz
- ✓ ۹.۵ M transistors
- ✓ ۰.۲۵ micron
- ✓ ۱۶ KByte L۱I, ۱۶ Kbyte L۱D, ۵۱۲ KByte offchip L۲
- ✓ Addition of SSE instructions.
- ✓ SSE: Intel Streaming SIMD Extensions to the x۸۶ ISA

Pentium® III Processor (Coppermine)

- ✓ Introduced: ۱۹۹۹
- ✓ ۵۰۰MHz ... ۱۱۳۳MHz
- ✓ ۲۸ M transistors
- ✓ ۰.۱۸ micron
- ✓ ۱۶ KByte L۱I, ۱۶ Kbyte L۱D, ۲۵۶KByte on-chip L۲
- ✓ Integrate L۲ cache on chip, It topped out at ۱GHz.



Pentium® IV Processor

- ✓ Introduced: ۲۰۰۰
- ✓ ۱.۳GHz … ۲GHz … ۳.۴GHz
- ✓ ۴۲M … ۵۵M … ۱۲۵ M transistors
- ✓ ۰.۱۸ … ۰.۱۳ … ۰.۰۹ micron
- ✓ Latest one: ۱۶ KByte L۱I, ۱۶ KByte L۱D, ۱M on-chip L۲
- ✓ Very high clock speed and SSE performance

Intel® Itanium® Processor

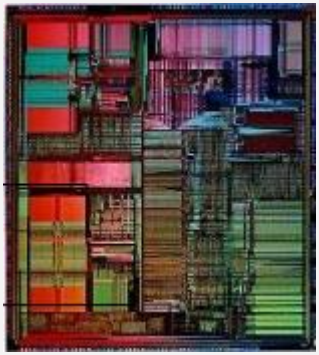
- Design ۱۹۹۳ – ۲۰۰۰
- ۷۳۳ MHz, ۸۰۰ MHz
- ۲۵ M transistors
- ۰.۱۸ micron
- ۳ levels of cache
  - ✓ ۱۶ KByte L۱I, ۱۶ KByte L۱D
  - ✓ ۹۶ KByte L۲
  - ✓ ۴ MByte off-die L۳
- Superscalar degree ۶, in-order machine
- First implementation of ۶۴-bit Itanium architecture

Intel® Itanium ۲® Processor

- Introduced: ۲۰۰۲
- ۱GHz
- ۲۲۱ M transistors
- ۰.۱۸ micron
  - ✓ ۳ levels of cache
  - ✓ ۳۲ KByte I&D L۱
  - ✓ ۲۵۶ KByte L۲
- integrated ۱.۵MByte L۳
- Based on EPIC architecture
- Enhanced Machine Check Architecture (MCA) with extensive Error Correcting Code (ECC)

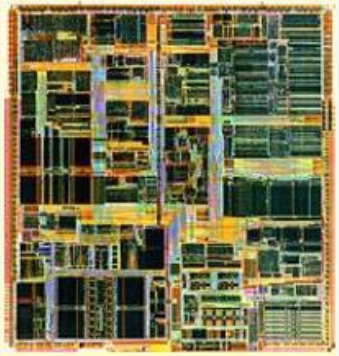
Cache Size Becoming Larger and Larger

۱۹۹۳: Pentium



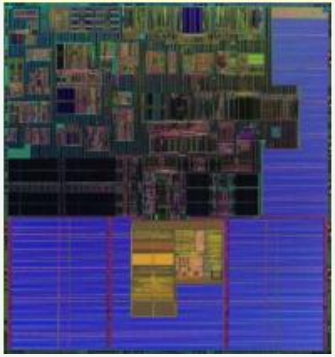
8 KByte I-cache
and 8 KByte D-cache

۱۹۹۷: Pentium-II

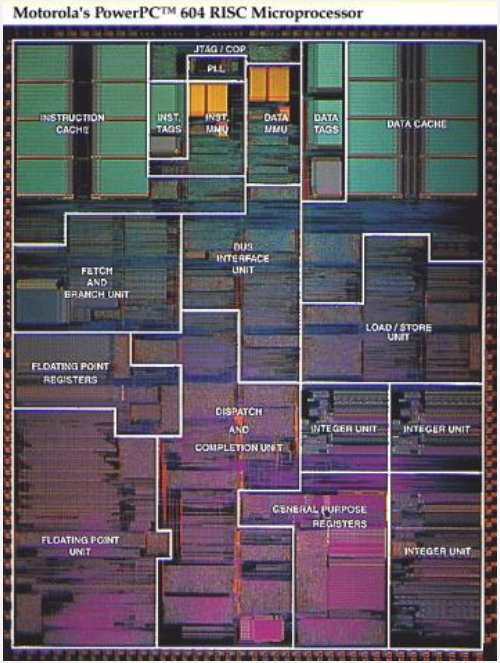


16 KByte L1I, 16
KByte L1D
512 KByte off-die L2

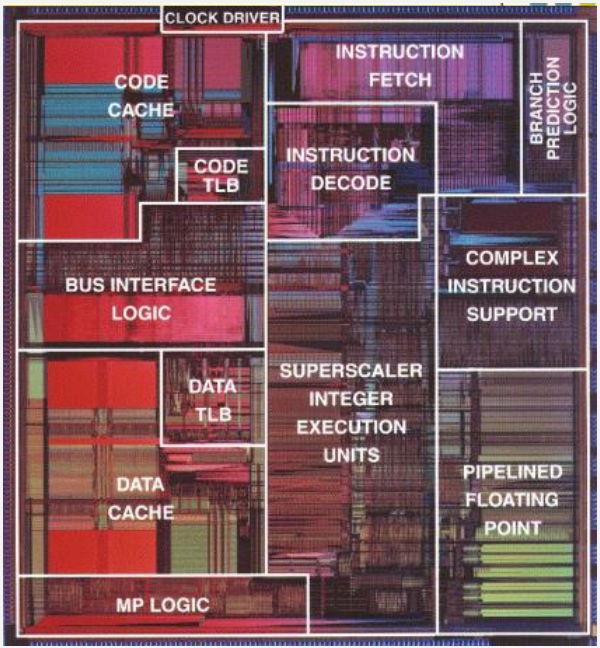
۲۰۰۲: Itanium-۲



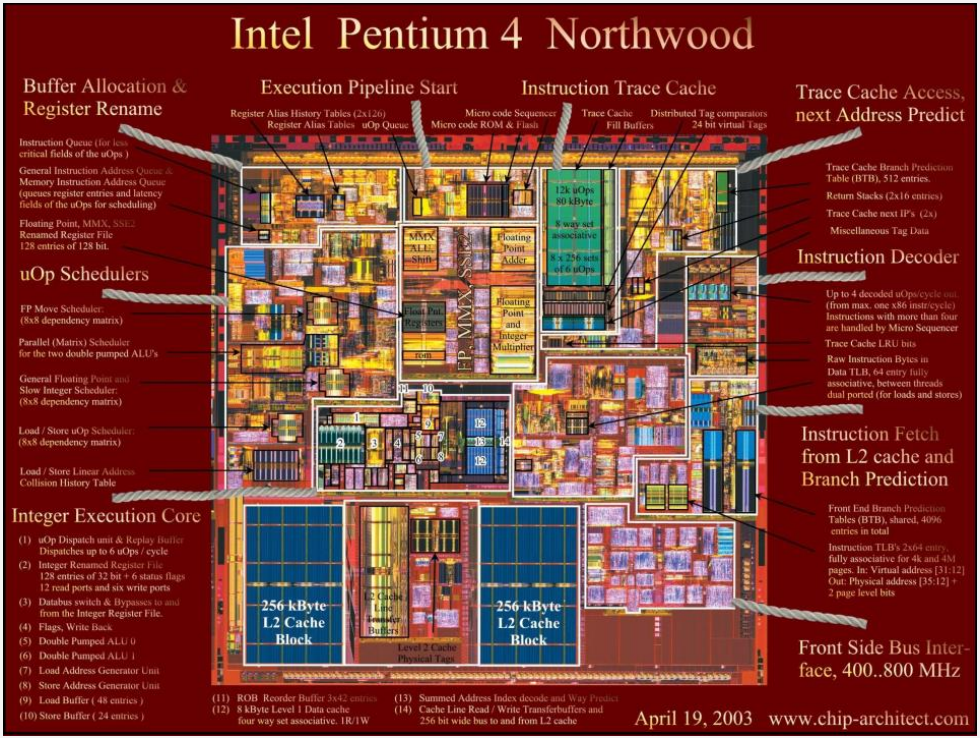
Level 1: 16K KByte Icache,
16 KByte D-cache
Level 2: 256 KB
Level 3: integrated 3 MB
or 1.5 MB

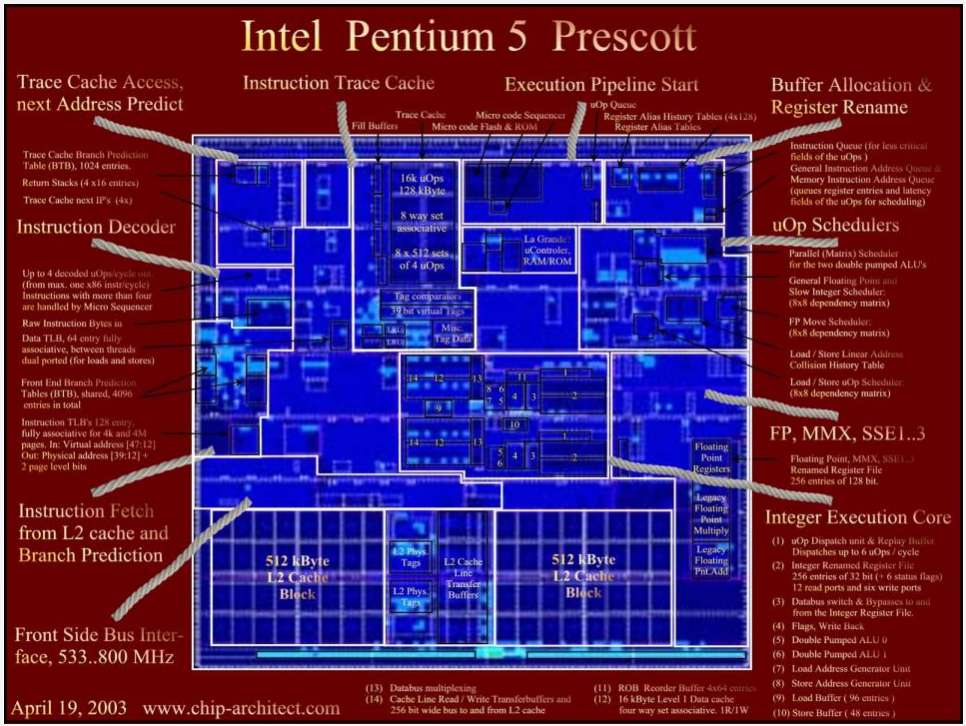


Motorola's PowerPC 604



Pentium


Intel Pentium 4 Northwood
Buffer Allocation & Register Rename
Execution Pipeline Start
Instruction Trace Cache
Trace Cache Access, next Address Predict
Register Alias History Tables (2x126)
Register Alias Tables
uOp Queue
Micro code Sequencer
Micro code ROM & Flash
Trace Cache Fill Buffers
Distributed Tag comparators 24 bit virtual Tags
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.
12k uOps 80 kByte
8 way set associative
6 x 256 sets of 6 uOps
Trace Cache Branch Prediction Table (BTB), 512 entries.
Return Stacks (2x16 entries)
Trace Cache next IP's (2x)
Miscellaneous Tag Data
Instruction Decoder
uOp Schedulers
MMX ALU Shift
Floating Point Adder
Quad Pump Register
Floating Point and Integer Multiplier
FP MMX SSE2
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)
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
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
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)
256 kByte L2 Cache Block
L2 Cache Line Transfer Buffers
256 kByte L2 Cache Block
Level 2 Cache Physical Tags
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
Instruction Trace Cache
Execution Pipeline Start
Buffer Allocation & Register Rename
Trace Cache Fill Buffers
Micro code Sequencer
Micro code Flash & ROM
uOp Queue
Register Alias History Tables (4x128)
Register Alias Tables
Trace Cache Branch Prediction Table (BTB), 1024 entries.
Return Stacks (4 x16 entries)
Trace Cache next IP's (4x)
16k uOps 128 kByte
8 way set associative
8 x 512 sets of 4 uOps
La Grande uController, RAM/ROM
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
uOp Schedulers
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)
Tag comparators
39 bit virtual Tags
Misc. Tag Data
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)
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
FP, MMX, SSE1..3
Floating Point Registers
Floating Point, MMX, SSE1..3 Renamed Register File 256 entries of 128 bit.
Legacy Floating Point Multiply
Legacy Floating Pnt Add
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)
Instruction Fetch from L2 cache and Branch Prediction
512 kByte L2 Cache Block
L2 Phys. Tags
L2 Cache Line Transfer Buffers
512 kByte L2 Cache Block
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

razeghizade@gmail.com

Razeghizade.pudica.ir

CREADITS: This presentation was created by M.Razeghizadeh
Please keep this slide for attribution