مقدمه

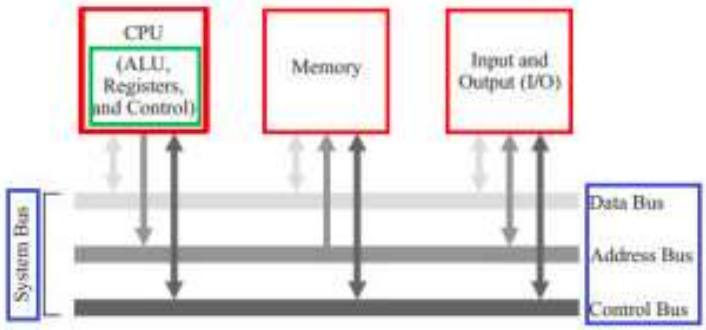
به طور کلی میکروپروسسور یک پکیج IC است به طوری که در آن چندین function یکپارچه شده و روی یک چیپ نیمه هادی سیلیکونی ساخته شده است. معماری میکروپروسسورها شامل یک واحد پردازش مرکزی (CPU)، ماژول‌های حافظه، واحدهای ورودی خروجی و سیستم باس (Bus) است.

سیستم باس به واحدهای مختلف متصل می‌شود تا موجب تسهیل تبادل اطلاعات شود. انواع مختلف آن شامل باس داده (Data bus)، باس آدرس (Address bus) و باس کنترل (Control bus) است که برای انتقال مناسب اطلاعات به کار می‌روند.

CPU از یک یا چند واحد منطق و محاسبات (ALU)، رجیسترها و واحد کنترل تشکیل شده. بر اساس رجیسترها می‌توان خانواده‌های میکروپروسسور را طبقه‌بندی کرد. یک پروسسور شامل رجیسترهای اهداف کلی و نوع خاصی از رجیسترها است که در زمان اجرای برنامه، دستورالعمل‌ها را اجرا کرده و آدرس یا داده‌هایی را ذخیره کند. ALU تمامی محاسبات ریاضی و منطقی داده‌ها را انجام می‌دهد و ظرفیت میکروپروسسور را مشخص می‌کند. برای مثال ۱۶ بیت یا ۳۲ بیت.

واحد حافظه (Memory)، برنامه و داده‌ها را نگهداری می‌کند و دارای سه نوع حافظه اولیه، حافظه ثانویه و حافظه پردازنده است.

واحدهای ورودی و خروجی، واسط بین I/O دستگاه‌های جانبی میکروپروسسور هستند که اطلاعات را دریافت و منتقل می‌کنند.



Architecture of Microprocessor

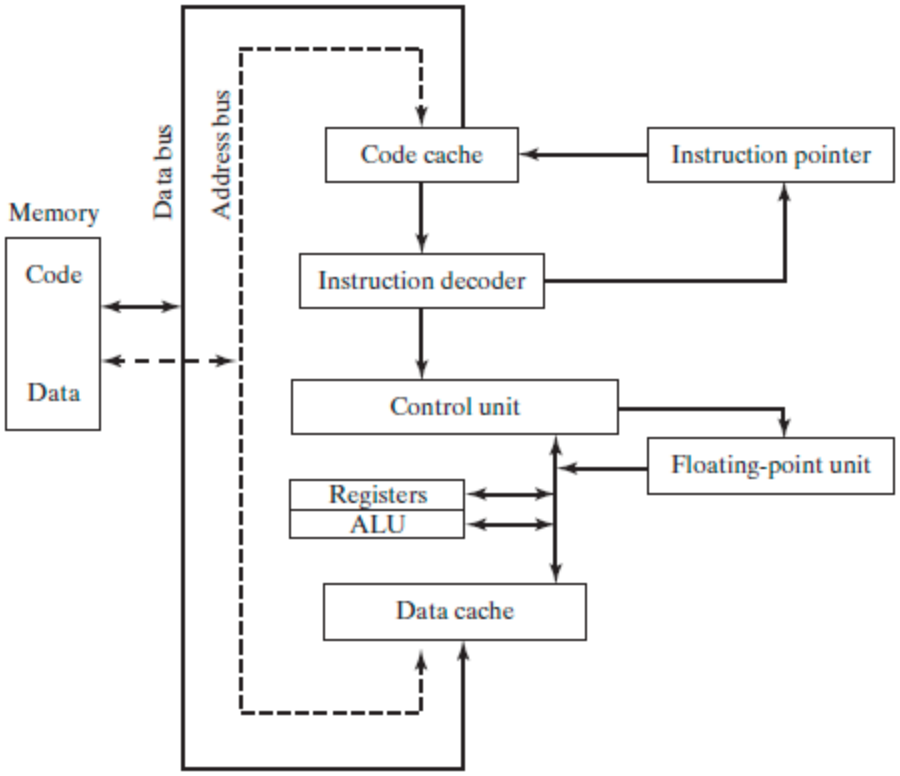
چرخه‌ی اجرای دستورالعمل[1]

اجرای هر دستورالعمل را می‌توان به دنباله‌ای از عملیات‌های جداگانه، تفکیک کرد که به آن چرخه اجرای دستورالعمل گفته می‌شود. قبل از اجرای هر دستورالعمل، برنامه در حافظه load می‌شود. اشاره‌گر دستورالعمل شامل آدرس دستورالعمل بعدی است که قرار است اجرا شود. اجرای هر دستورالعمل شامل سه بخش اصلی است: fetch، decode و execute. در صورتی که دستورالعملی شامل عملوند حافظه (memory operand) باشد، دو مرحله‌ی fetch operand و output operand نیز اضافه می‌شود.

- مرحله fetch: واحد کنترل، دستورالعمل اجرایی بعدی را دریافت می‌کند و به اشاره‌گر دستورالعمل (IP) یک واحد اضافه می‌شود. درواقع IP شمارنده‌ی برنامه است.
- مرحله decode: واحد کنترل، دستورالعمل را decode می‌کند تا مشخص شود چه دستوری باید انجام گیرد. عملوندهای ورودی دستور به ALU منتقل می‌شوند و سیگنال‌هایی که به ALU فرستاده شده‌اند، مشخص می‌کنند که چه عملیاتی باید صورت گیرد.
- مرحله fetch operand: اگر دستورالعمل از عملوند ورودی که در حافظه قرار گرفته استفاده کند، واحد کنترل، عملیات خواندن (read) برای کپی کردن عملوند از حافظه به رجیسترهای داخلی را انجام می‌دهد.
- مرحله execute: واحد ALU با استفاده از رجیسترها و رجیسترهای داخلی به عنوان عملوند، دستورالعمل را اجرا کرده و خروجی را به حافظه یا رجیستر می‌فرستد.
- مرحله stored output operand: اگر خروجی عملوند در حافظه باشد، واحد کنترل از عملیات نوشتن (write) برای ذخیره داده استفاده می‌کند.

شکل زیر ارتباط بین بخش‌های مختلف پردازنده که در طول چرخه اجرای دستورالعمل با یکدیگر تعامل دارند را نشان می‌دهد. اگر قرار باشد دستورالعمل از حافظه خوانده شود، یک آدرس روی address bus قرار می‌گیرد سپس کنترل‌کننده‌ی حافظه، کد مربوطه را روی data bus قرار می‌دهد که داخل code cache قابل دسترس باشد. مقدار اشاره‌گر دستورالعمل مشخص می‌کند که کدام دستورالعمل باید اجرا شود.

[1] Instruction Execution Cycle



با آنالیز دستورالعمل توسط Instruction decoder سیگنال‌های مناسب به واحد کنترل فرستاده می‌شود که واحدهای ALU و floating-point را هماهنگ می‌کند. در شکل بالا control bus که حامل سیگنال‌های system clock برای هماهنگ سازی انتقال داده بین بخش‌های مختلف CPU است، نشان داده نشده.

معماری x86

X86 یک خانواده از معماری مجموعه دستورات (ISA[2]) برای پردازنده‌ی محاسباتی است که بیانگر روشی است که CPU با آن، اطلاعات را مدیریت می‌کند و توسط شرکت Intel در سال ۱۹۷۹ معرفی شد. درواقع مجموعه دستورالعمل‌ها، زبانی است که مغز کامپیوتر برای درک آن طراحی شده و به وسیله‌ی آن به پروسسور فرمان می‌دهد که در در هر لحظه چه کاری انجام دهد.

x86 بر اساس میکروپروسسور x86 اینتل شکل گرفت که یک معماری ۱۶ بیتی برای میکروپروسسور ۱۶ بیتی بود که طی چند سال، به مجموعه دستورات ۳۲ بیتی توسعه پیدا کرد که کاملا قابلیت سازگار شدن با ورژن قبلی را داشت.

[2] Instruction Set Architecture

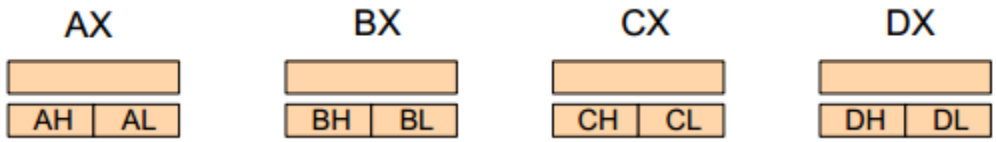
- رجیسترهای x86

برای نوشتن کدهای اسمبلی ISA آشنایی با رجیسترها یعنی محلی که داده‌ها در آن قرار می‌گیرند لازم است.

- چهار رجیستر ۱۶ بیتی برای اهداف کلی :

AX , BX , CX , DX

که هر کدام از آن‌ها شامل بخش ۸ بیت "Low bit" و ۸ بیت "High bit" هستند که در ISA هر دو بخش را می‌توان جداگانه استفاده کرد.



- دو رجیستر ۱۶ بیتی index که معمولا شامل آدرس هستند و به عنوان pointer استفاده می‌شوند:

SI , DI

این دو رجیستر قابل تفکیک به دو قسمت ۸ بیتی نیستند.

- دو رجیستر ۱۶ بیتی خاص:

BP -> Base pointer

SP -> Stack pointer

- چهار رجیستر ۱۶ بیتی segment که برای آدرس‌دهی حافظه استفاده می‌شوند:

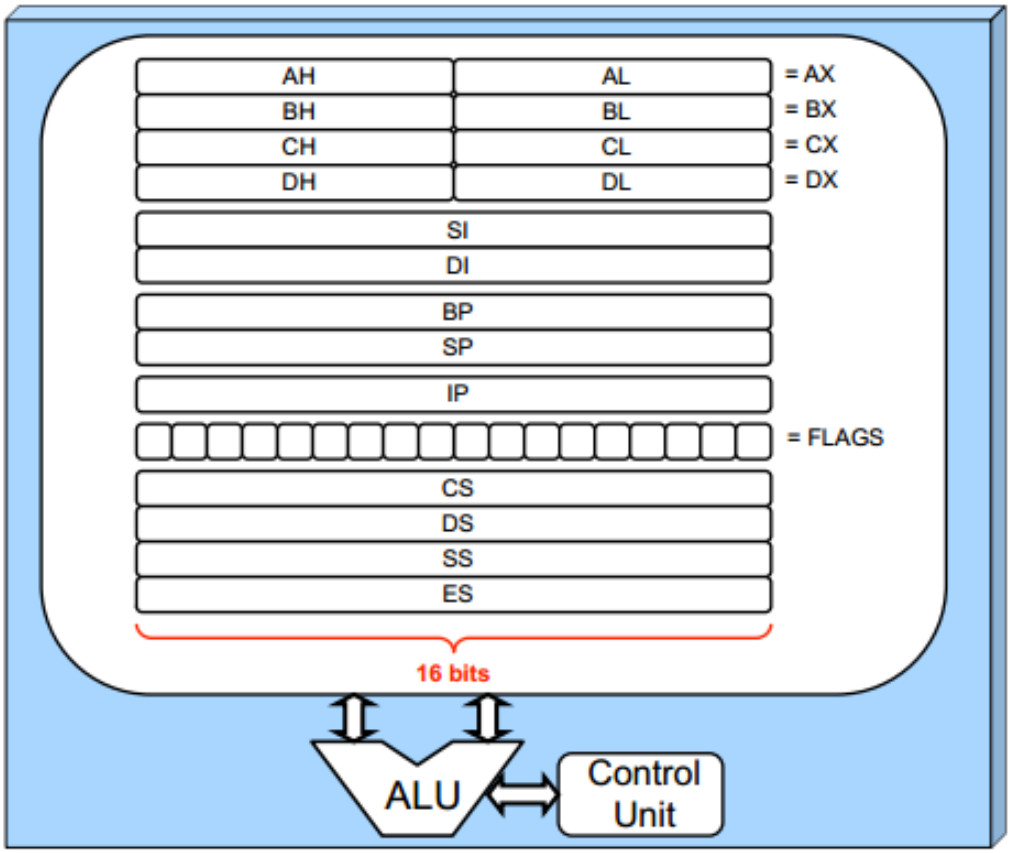
CS -> Code segment

DS -> Data segment

SS -> Stack segment

ES -> Extra segment

- رجیستر ۱۶ بیتی IP (Instruction Pointer) که نشان دهنده‌ی دستورالعمل بعدی برای اجرا است.
- رجیسترهای ۱۶ بیتی Flag که اطلاعات در هر یک از بیت‌ها می‌تواند ذخیره شود.



شرکت اینتل پروسسورهایی را با address bus ۲۰ بیتی و نحوه استفاده از سگمنت در آدرس‌دهی را با خانواده x86 معرفی کرد که می‌توانست تا ۴ سگمنت ۶۴ کیلو بیتی را آدرس‌دهی کند.

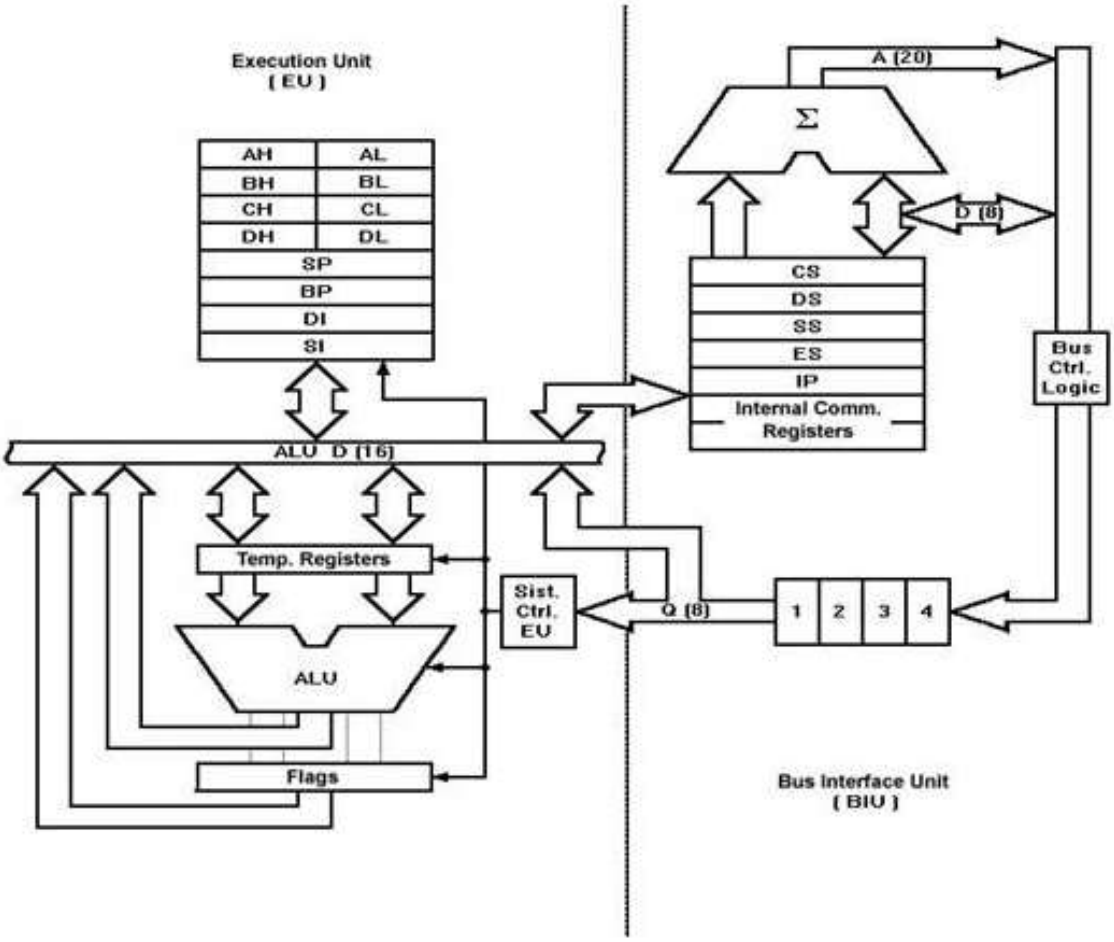
دو بخش کلی CPU در x86 در شکل زیر نشان داده شده. بخش BIU (bus interface unit) که وظیفه آدرس‌دهی حافظه و I/O برای انتقال داده‌ها بین دنیای خارج CPU و بخش EU (execution unit) را دارد.

قسمت EU نیز کدهای دستورالعمل و داده‌ها را از BIU دریافت کرده و دستورالعمل را اجرا می‌کند و نتیجه را در رجیسترهای عمومی ذخیره می‌کند یا به بخش BIU می‌فرستد که در حافظه ذخیره شود.

همچنین معماری مجموعه دستورات x86 از نوع CISC[3] است که در آن هر دستورالعمل می‌تواند چندین عملیات low-level (بارگیری از حافظه، عملیات محاسباتی و ذخیره در حافظه) را انجام دهد. نقطه مقابل این نوع

[3] Complex Instruction Set Computer

معماری، معماری RISC[4] است که در آن طول تقریبا همه دستورالعمل‌ها برای اجرای عملیات مختلف یکسان است و برای عملیات بارگیری و ذخیره کردن، دستورالعمل‌های جداگانه استفاده می‌شود.



کاربردها

در اکثر کامپیوترهای شخصی، لپتاپ‌ها و کنسول‌های بازی از معماری x86 استفاده می‌شود. همچنین در کامپیوترهای صنعتی و در زمینه‌ی ذخیره اطلاعات ابری پرکاربردترین پردازنده است.

[4] Reduced Instruction Set Computer

معماری MIPS

اولین نسخه معماری[5]MIPS توسط شرکت MIPS Technologies در سال ۱۹۸۵ برای میکروپروسور R2000 ۳۲ بیتی همین شرکت معرفی شد. معماری MIPS از نوع ماژولار است که می‌تواند تا ۴ coprocessor (CP0/1/2/3) را پشتیبانی کند. coprocessor یک پردازنده محاسباتی است که به عنوان مکمل CPU اصلی استفاده می‌شود. عملیاتی که توسط coprocessor انجام می‌شوند، شامل: محاسبات floating point، گرافیکی، پردازش سیگنال، رمز نگاری و رابط I/O هستند. با جدا کردن این محاسبات از پروسسور اصلی، coprocessor می‌تواند سرعت اجرای سیستم را بالا ببرد.

در ادبیات MIPS، CP0، coprocessor کنترل سیستم، CP1 واحد floating-point (FPU) و CP2/3 با توجه به نوع کاربرد تعریف می‌شوند. برای مثال در کنسول بازی play station، CP2 موتور تبدیل هندسی (Geometric Transformation Engine) است که باعث سرعت بخشیدن به پردازش هندسی در محاسبات گرافیکی سه بعدی می‌شود.

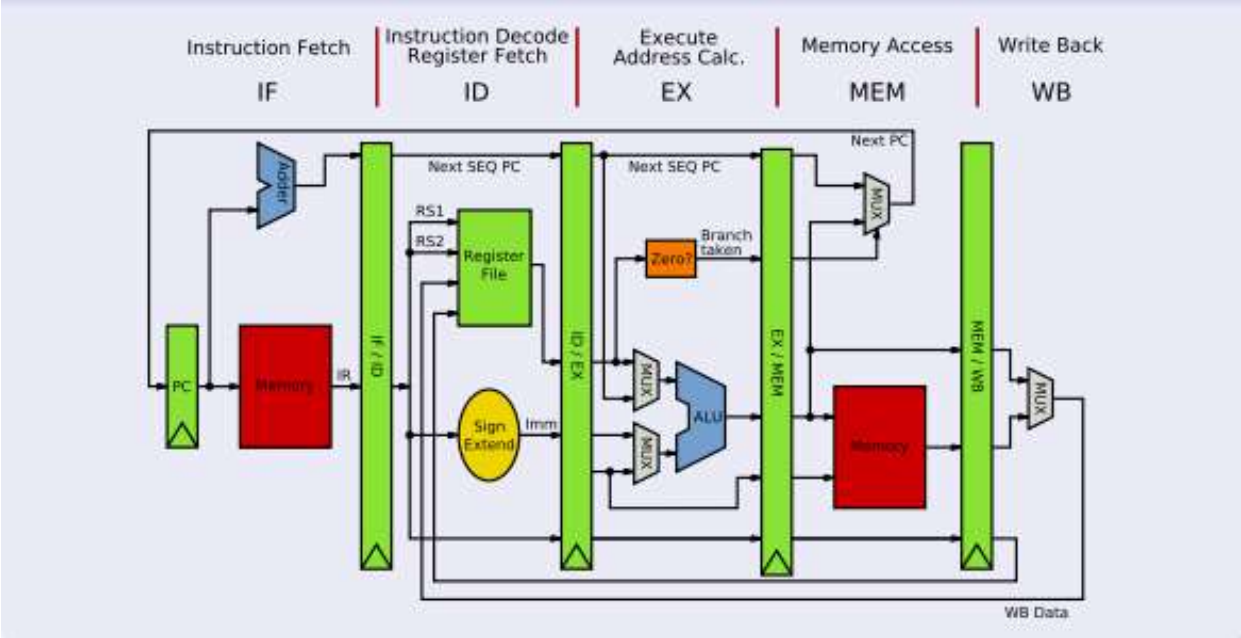
معماری مجموعه دستورالعمل‌های MIPS از نوع RISC است و به جز دستورالعمل‌های مربوط به دسترسی به حافظه، از مدل Load/store استفاده می‌کند. در مدل Load/store، مجموعه دستورات به دو دسته‌ی کلی تقسیم می‌شوند:

۱- دسترسی به حافظه (بارگیری و ذخیره‌سازی بین حافظه و رجیسترها)
۲- عملیات ALU (فقط بین رجیسترها)

برخی معماری‌های نوع RISC مانند SPARK، ARM، Power PC و MIPS به صورت Load/store هستند.

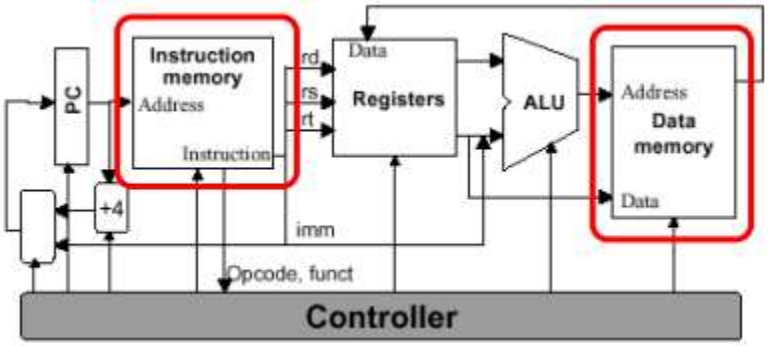
شکل زیر معماری MIPS را به طور کلی نشان می‌دهد.

[5] Microprocessor without Interlocked Pipelined Stages

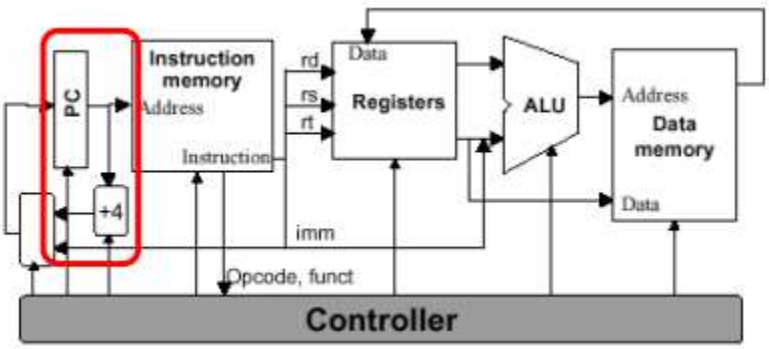


در MIPS داده‌ها و برنامه‌ها به صورت جداگانه در حافظه قرار دارند:

- Text Memory: بخشی از حافظه که برنامه‌ها را ذخیره می‌کند و به صورت read only است. (Machine code).
- Data Memory: بخشی از حافظه به صورت read/write برای ذخیره داده‌ها که توسط برنامه تغییر می‌کنند.



رجیستر Program counter (PC) یا IP آدرس دستورالعمل بعدی که باید fetch شود را ذخیره می‌کند. از آنجا که در MIPS هر دستورالعمل ۳۲ بیت طول دارد و هر آدرس در یک بایت (۸ بیت) قرار می‌گیرد، پس مقدار PC در هر Clock cycle با ۴ جمع می‌شود که آدرس دستورالعمل بعدی در آن قرار گیرد.



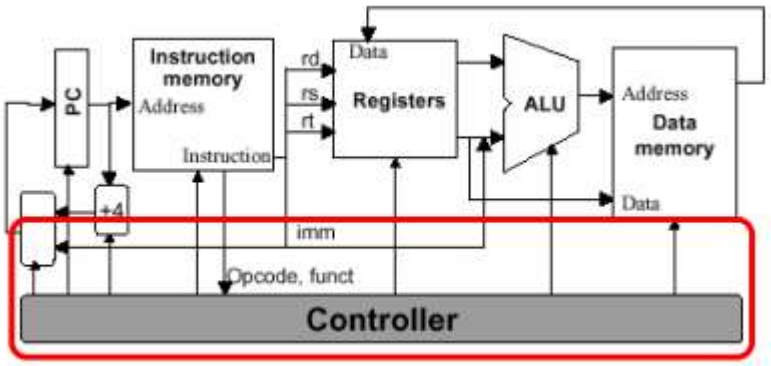
رجیستر دستورالعمل (IR[6])، دستورالعملی که در حال decode شدن است را در خود نگه می‌دارد.

MIPS دارای ۳۲ رجیستر ۳۲ بیتی برای اهداف کلی و خاص (مانند IR و PC) است که در جدول زیر نشان داده شده است.

Register Number	Conventional Name	Usage
$0	$zero	Hard-wired to 0
$1	$at	Reserved for pseudo-instructions
$2 - $3	$v0, $v1	Return values from functions
$4 - $7	$a0 - $a3	Arguments to functions - not preserved by subprograms
$8 - $15	$t0 - $t7	Temporary data, not preserved by subprograms
$16 - $23	$s0 - $s7	Saved registers, preserved by subprograms
$24 - $25	$t8 - $t9	More temporary registers, not preserved by subprograms
$26 - $27	$k0 - $k1	Reserved for kernel. Do not use.
$28	$gp	Global Area Pointer (base of global data segment)
$29	$sp	Stack Pointer
$30	$fp	Frame Pointer
$31	$ra	Return Address
$f0 - $f3	-	Floating point return values
$f4 - $f10	-	Temporary registers, not preserved by subprograms
$f12 - $f14	-	First two arguments to subprograms, not preserved by subprograms
$f16 - $f18	-	More temporary registers, not preserved by subprograms
$f20 - $f31	-	Saved registers, preserved by subprograms

[6] Instruction Register

بخش کنترل، مسیر داده در یک چرخه دستورالعمل را کنترل می‌کند. ورودی آن سیگنال‌های شرطی و خروجی آن سیگنال‌های کنترلی هستند.



انواع دستورالعمل‌ها در MIPS

- دستورالعمل نوع R:

زمانی استفاده می‌شود که مقادیر داده‌هایی که دستورالعمل با آن کار می‌کند، در رجیسترها موجود باشد. دستورالعمل‌های نوع R فرمتی به صورت زیر دارند:

opcode	rs	rt	rd	shift (shamt)	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

مقدار opcode نشان دهنده‌ی کد دستورالعمل و rs، rt و rd به ترتیب رجیسترهای source، targetو destination هستند. Shift برای دستورالعمل‌های حاوی شیفت و funct بیانگر نوع عملیات در ALU است.

- دستورالعمل نوع I:

زمانی استفاده می‌شود که دستورالعمل باید روی immediate value و مقدار رجیستر عملیات انجام دهد. فرمت آن‌ها به صورت زیر است:

opcode	rs	rt	IMM
6 bits	5 bits	5 bits	16 bits

- دستورالعمل نوع J:

زمانی استفاده می‌شود که باید پرشی به یک آدرس مشخص انجام شود. شامل ۶ بیت opcode و ۲۶ بیت آدرس کوتاه شده‌ی مقصد پرش است.

Opcode	Pseudo-Address

بخشی از جدول opcode های MIPS در به صورت زیر است:

Mnemonic	Meaning	Type	Opcode	Funct
add	Add	R	0x00	0x20
addi	Add Immediate	I	0x08	NA
addiu	Add Unsigned Immediate	I	0x09	NA
addu	Add Unsigned	R	0x00	0x21
and	Bitwise AND	R	0x00	0x24
andi	Bitwise AND Immediate	I	0x0C	NA
beq	Branch if Equal	I	0x04	NA
blez	Branch if Less Than or Equal to Zero	I	0x06	NA
bne	Branch if Not Equal	I	0x05	NA
bgtz	Branch on Greater Than Zero	I	0x07	NA
div	Divide	R	0x00	0x1A
divu	Unsigned Divide	R	0x00	0x1B
j	Jump to Address	J	0x02	NA
jal	Jump and Link	J	0x03	NA
jalr	Jump and Link Register	J	0x00	0x09

کاربردها

پروسسورهای MIPS با هدف کلی انجام عملیات محاسباتی در دهه ۸۰ و ۹۰ میلادی طراحی شدند و در کامپیوترهای شخصی، صنعتی و سرورها توسط شرکت‌های مختلف استفاده می‌شدند. در کنسول‌های بازی Nintendo64، Sony playStation و playStation 2 از پروسسور MIPS استفاده شده. همچنین در دهه ۹۰ برای استفاده در supercomputerها کاربرد داشته است. در سال‌های اخیر از این پروسسور بیشتر برای سیستم‌های embedded در زمینه‌های خودرویی، روترهای بی‌سیم و میکروکنترلرها استفاده می‌شود.

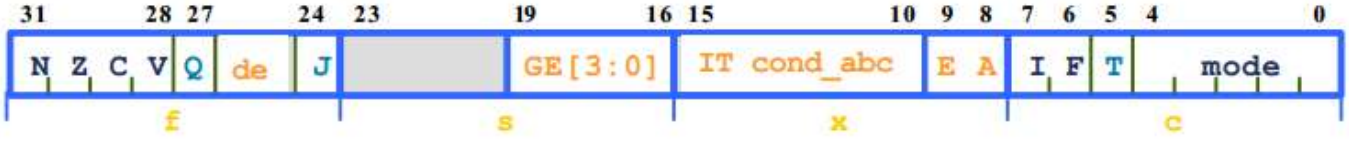
معماری ARM

معماری ARM (Advanced RISC Machine) خانواده‌ای از معماری RISC به صورت load/store است که اولین نسخه آن در ۱۹۹۰ توسط شرکت Acorn معرفی شد. هسته پردازنده ۳۲ بیتی و ۳۷ رجیستر ۳۲ بیتی دارد و تا ۱۶ coprocessor را پشتیبانی می‌کند.

ARM شامل ۷ مد عملیاتی به صورت زیر است:

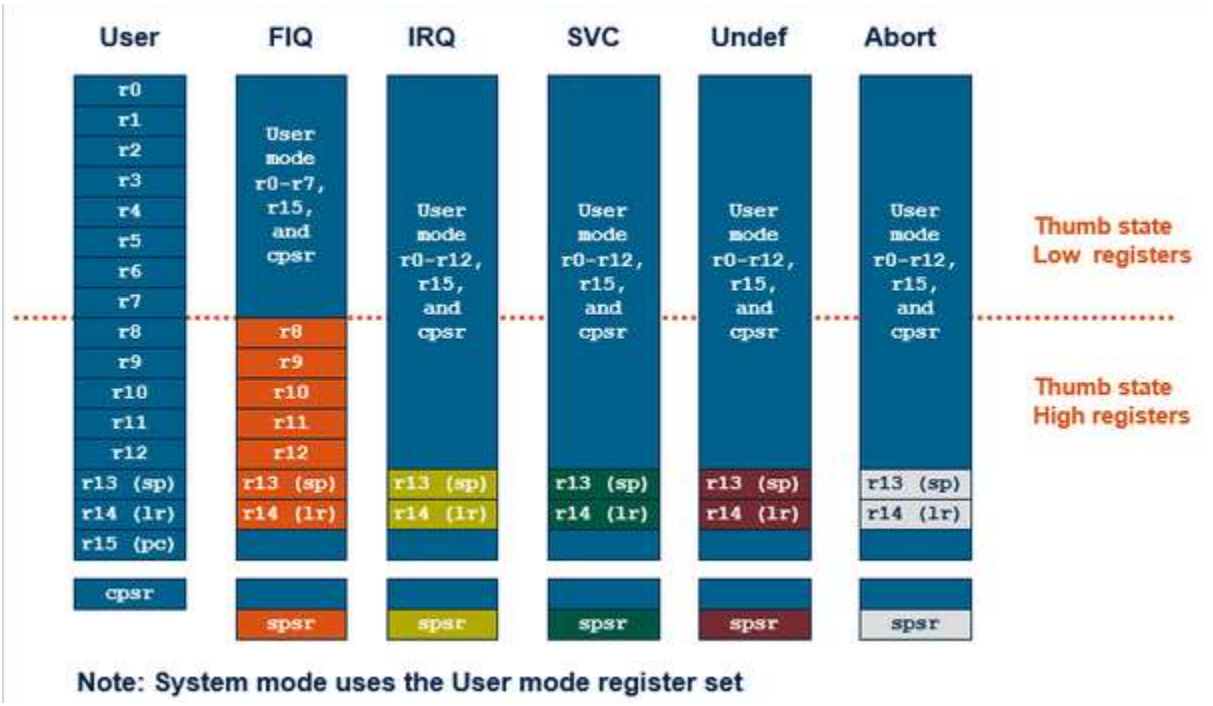
Mode	Description	
Supervisor (SVC)	**Entered on reset and when a Software Interrupt instruction (SWI) is executed**	Privileged modes
FIQ	**Entered when a high priority (fast) interrupt is raised**	
IRQ	**Entered when a low priority (normal) interrupt is raised**	
Abort	**Used to handle memory access violations**	
Undef	**Used to handle undefined instructions**	
System	**Privileged mode using the same registers as User mode**	
User	**Mode under which most Applications / OS tasks run**	Unprivileged mode

مد user قابلیت دسترسی به ۱۶ رجیستر را دارد و بقیه مدها علاوه بر این رجیسترها، از رجیسترهای دیگری نیز استفاده می‌کنند. ساختار رجیستر مشترک cpsr که نشان دهنده‌ی وضعیت برنامه است به صورت زیر است:



- Condition code flags
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation oVerflowed
- Sticky Overflow flag - Q flag
  - Architecture 5TE and later only
  - Indicates if saturation has occurred
- J bit
  - Architecture 5TEJ and later only
  - J = 1: Processor in Jazelle state
- Interrupt Disable bits
  - I = 1: Disables IRQ
  - F = 1: Disables FIQ
- T Bit
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state
  - Introduced in Architecture 4T
- Mode bits
  - Specify the processor mode
- New bits in V6
  - GE[3:0] used by some SIMD instructions
  - E bit controls load/store endianness
  - A bit disables imprecise data aborts
  - IT [abcde] IF THEN conditional execution of Thumb2 instruction groups

مجموعه رجیسترها در مدهای مختلف عملکرد ARM به صورت زیر است:



مجموعه دستورالعمل‌های ARM شامل ARM Instruction set ۳۲ بیتی و Thumb Instruction set ۱۶ بیتی است. برای هسته‌های جدیدتر ARM نیز مجموعه دستورالعمل Thumb-2 که ترکیب دستورهای ۱۶ و ۳۲ بیتی است نیز معرفی شده است.

فرمت دستورالعمل‌های شرطی ARM به صورت جدول زیر است:

Mnemonic	Condition	Mnemonic	Condition
CS	*C*arry *S*et	CC	*C*arry *C*lear
EQ	*Eq*ual (Zero Set)	NE	*N*ot *E*qual (Zero Clear)
VS	O*v*erflow *S*et	VC	O*v*erflow *C*lear
GT	*G*reater *T*han	LT	*L*ess *T*han
GE	*G*reater Than or *E*qual	LE	*L*ess Than or *E*qual
PL	*Pl*us (Positive)	MI	*Mi*nus (Negative)
HI	*Hi*gher Than	LO	*Lo*wer Than (aka CC)
HS	*H*igher or *S*ame (aka CS)	LS	*L*ower or *S*ame

فرمت دستورالعمل‌های ریاضی به صورت جدول زیر است:

Syntax: <instruction>{<cond>}{S} Rd, Rn, N

ADC	add two 32-bit values and carry	$Rd = Rn + N +$ carry
ADD	add two 32-bit values	$Rd = Rn + N$
RSB	reverse subtract of two 32-bit values	$Rd = N - Rn$
RSC	reverse subtract with carry of two 32-bit values	$Rd = N - Rn -$!(carry flag)
SBC	subtract with carry of two 32-bit values	$Rd = Rn - N -$!(carry flag)
SUB	subtract two 32-bit values	$Rd = Rn - N$

برای اجرای دستورالعمل‌ها در یک سیکل، ARM7 از سه مرحله اصلی fetch، decode و execution در پردازش استفاده می‌کند که در سری ARM9 دو مرحله‌ی memory access و write back را اضافه کرده است و باعث افزایش توان عملیاتی تا ۱۳٪ نسبت به ARM7 شده است. در سال ۲۰۱۱ شرکت ARM نسخه ARMv8 را معرفی کرد که معماری ۶۴ بیتی داشت و شامل دو مرحله‌ی execution به نام‌های AArch32 و AArch64 برای اجرای کدهای ۳۲ بیتی و ۶۴ بیتی بود.

کاربردها

خانواده ARM سه پروفایل معماری اصلی به نام M، A و R دارد که از پروفایل M برای کاربردهای مقرون به صرفه که در آن‌ها مصرف توان، بهینه‌سازی انرژی و اندازه مهم است، استفاده می‌شود مانند تجهیزات IOT ساده و چیپ‌های deeply embedded.

پروفایل R برای کاربردهایی که در آن‌ها پاسخ real-time نیاز است، مانند برنامه‌های مهم امنیتی، تجهیزات پزشکی و هدایت خودروها به کار می‌رود.

پروفایل A نیز در زمینه‌هایی که نیاز به محاسبات پیچیده دارد مانند تجهیزات شبکه، سرورها، موبایل‌ها و لپتاپ استفاده می‌شود.

علت فراگیر شدن ARM در بازار پردازنده‌های امروزی

برای سال‌ها پروسسورهای شرکت intel در کامپیوترها، لپتاپ‌ها، تلفن‌های هوشمند و ... استفاده می‌شد تا زمانی که دیگر نیاز اصلی سریعتر بودن پردازنده نبود، بلکه کارآمد و قابل حمل بودن شد و ARM توانست به جایگاهی برسد که بازار پروسسورها را به خود اختصاص دهد. بنابراین در ادامه، تفاوت‌های اصلی معماری ARM و x86 بررسی می‌شود.

- یکی از مهم‌ترین تفاوت‌های دو معماری ذکر شده، تفاوت در RISC و CISC بودن آن‌ها است. در معماری RISC هر دستور، عمل مشخصی برای اجرا به CPU اعمال می‌کند و دستورات نسبتا ساده هستند. در حالی که دستورات در معماری CISC دستورات پیچیدگی بیشتری دارند و CPU را ملزم به اجرای اعمال گسترده‌تری می‌کند و این به معنی تجزیه‌ی هر دستور به یک سری ریز-فرمان توسط CPU است. بنابراین معماری در CISC می‌توان جزییات خیلی بیشتری در هر دستور رمزنگاری (encode) کرد که این امر موجب بهبود کارایی می‌شود. اما این ویژگی با اینکه باعث می‌شود عملیات پیچیده‌تر و بزرگتر در زمان کمتری انجام شود، اما همین پیچیدگی، به معنی استفاده از برنامه‌نویسی بیشتر و تخصصی‌تر و همچنین مصرف توان بیشتر می‌شود که به معنی افزایش هزینه است. بنابراین کارایی بهتر را به تنهایی نمی‌توان ملاک انتخاب قرار داد.
  با توجه به توضیحات ذکر شده، استفاده از معماری ARM دقیقا مناسب پروسسورهای سیستم‌های قابل حمل مانند تلفن همراه که مصرف توان کمی (ماکزیمم ۲ الی ۳ وات) دارند، است.
  تفاوت‌های کلی معماری‌های RISC و CISC در جدول زیر آورده شده است.

CISC	RISC
Push complexity to hardware	Push complexity to software
Many different types and formats for instructions	Instructions follow similar format
Few internal registers	Many internal registers
Complex decoding to break up instruction parts	Complex compiler to write code with granular instructions
Complex forms of memory interaction	Few forms of memory interaction
Instructions take different number of cycles to finish	All instructions finish in one cycle
Difficult to divide and parallelize work	Easy to parallelize work

- مشخصه اصلی دیگری که ARM را متمایز می‌کند، معماری محاسباتی ناهمگون[۷] big.LITTLE است. این طراحی زمانی که از دو پروسسور روی یک چیپ استفاده می‌شود، خود را نشان می‌دهد. یکی از هسته‌ها قدرتمند با مصرف توان بالا و دیگری با مصرف توان کم و ضعیفتر. چیپ باید بهره‌وری از سیستم را آنالیز کند تا مشخص شود کدام هسته را باید فعال کند. زمانی که چیپ در حالت idle است یا محاسبات ابتدایی‌ای انجام می‌دهد از هسته کم مصرف استفاده می‌کند و هسته‌ی دیگر غیرفعال است. ARM بیان می‌کند که این کار باعث ۷۵٪ صرفه‌جویی در مصرف توان می‌شود. اگرچه که CPUهای معمول، زمانی که بار کمتری روی آن‌ها است، مصرف توانشان را کاهش می‌دهند اما برخی قسمت‌ها به طور کامل خاموش نمی‌شوند. ولی ARM این قابلیت را دارد که هسته‌ی پرمصرف را کاملا خاموش کند.

[7] Heterogeneous computing architecture