معماری مجموعه دستورالعمل ۴

معماری کامپیوتر

۱۳۹۵-۱۱-۱۳

جلسه‌ی هفتم

دانشگاه شهید بهشتی

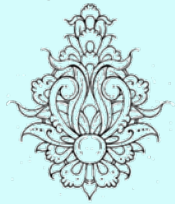
دانشکده‌ی مهندسی برق و کامپیوتر

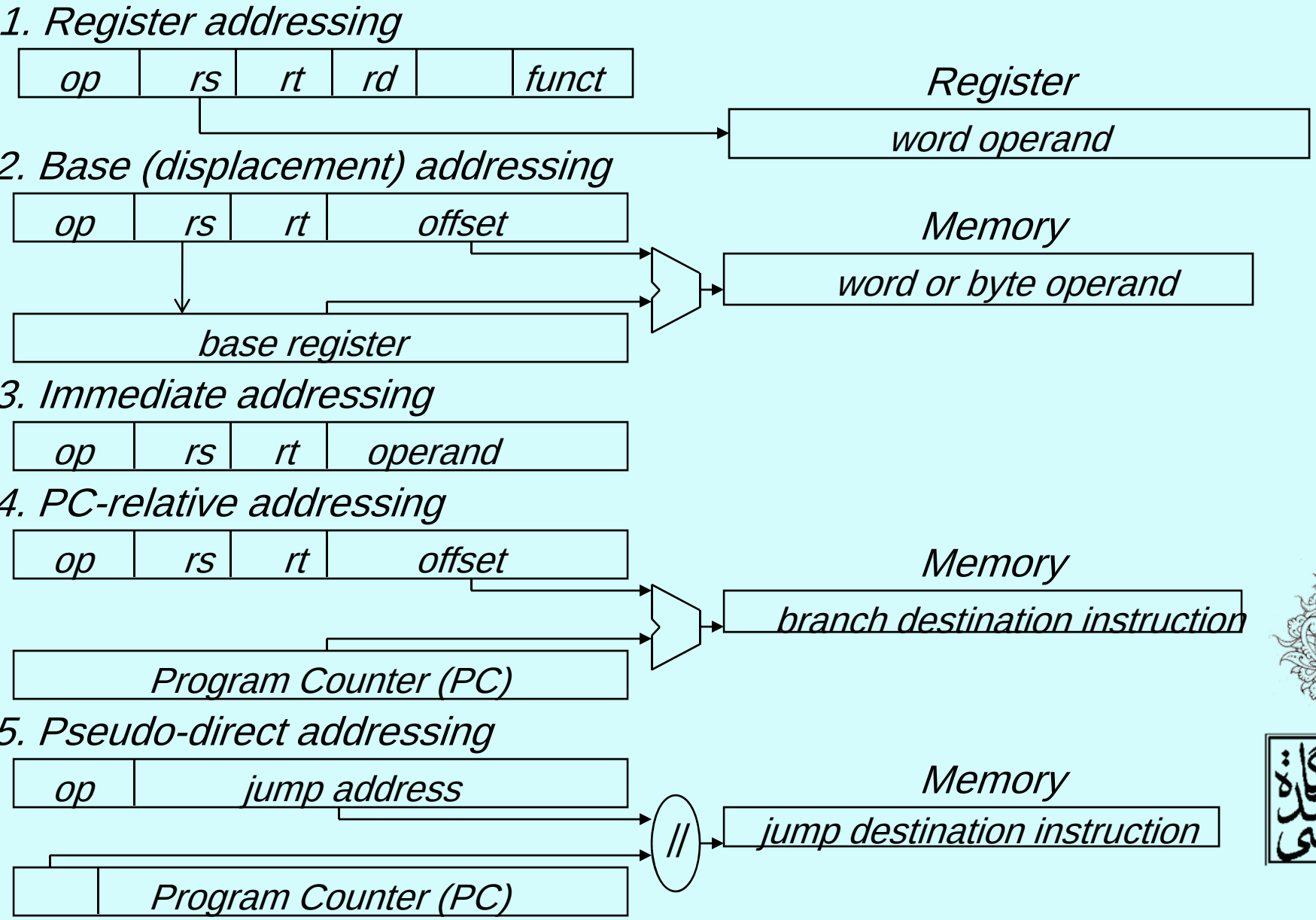
زمستان ۱۳۹۵

احمد محمودی ازناوه

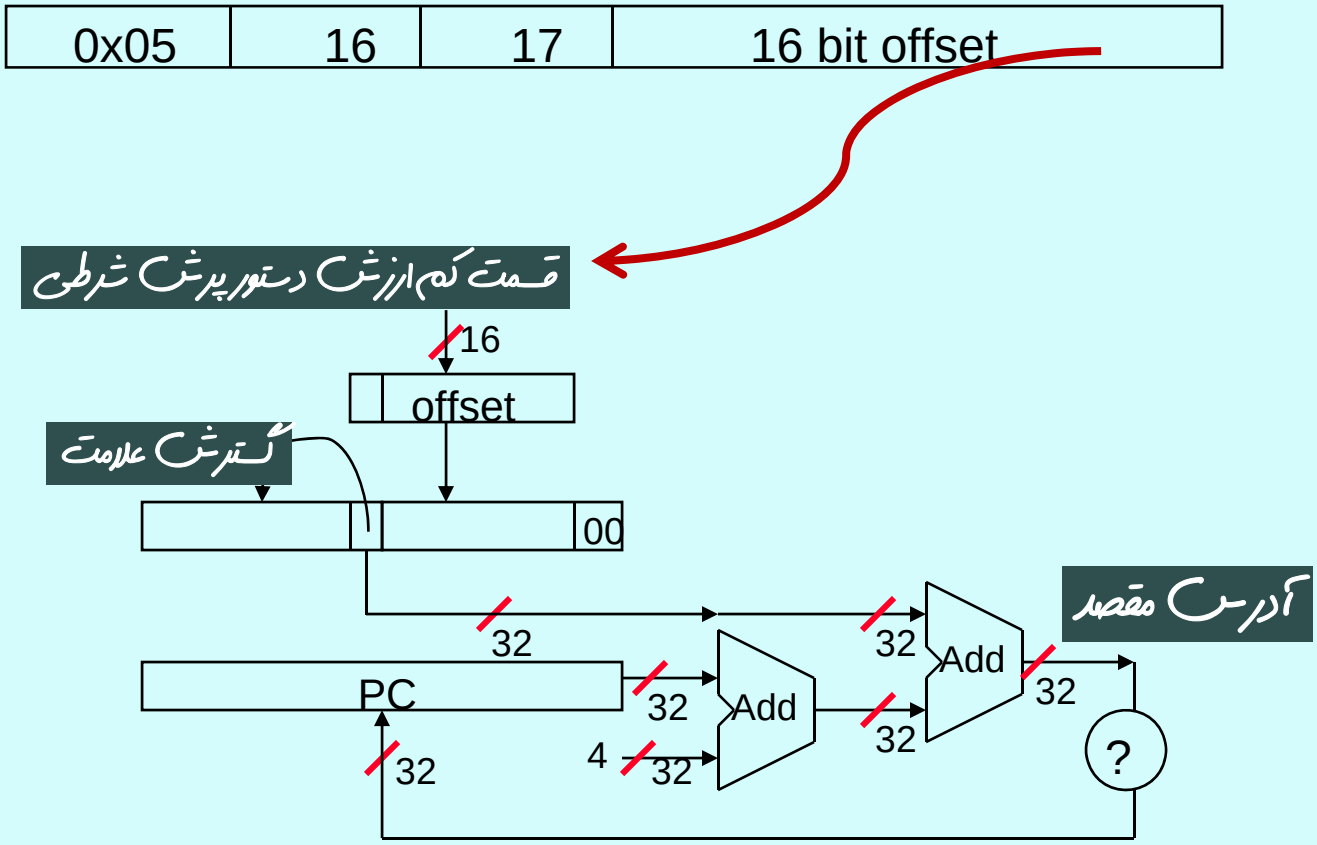
فهرست مطالب

- مروری برجلسه‌ی پیش
- انحصار متقابل
- سایر پردازنده‌ها

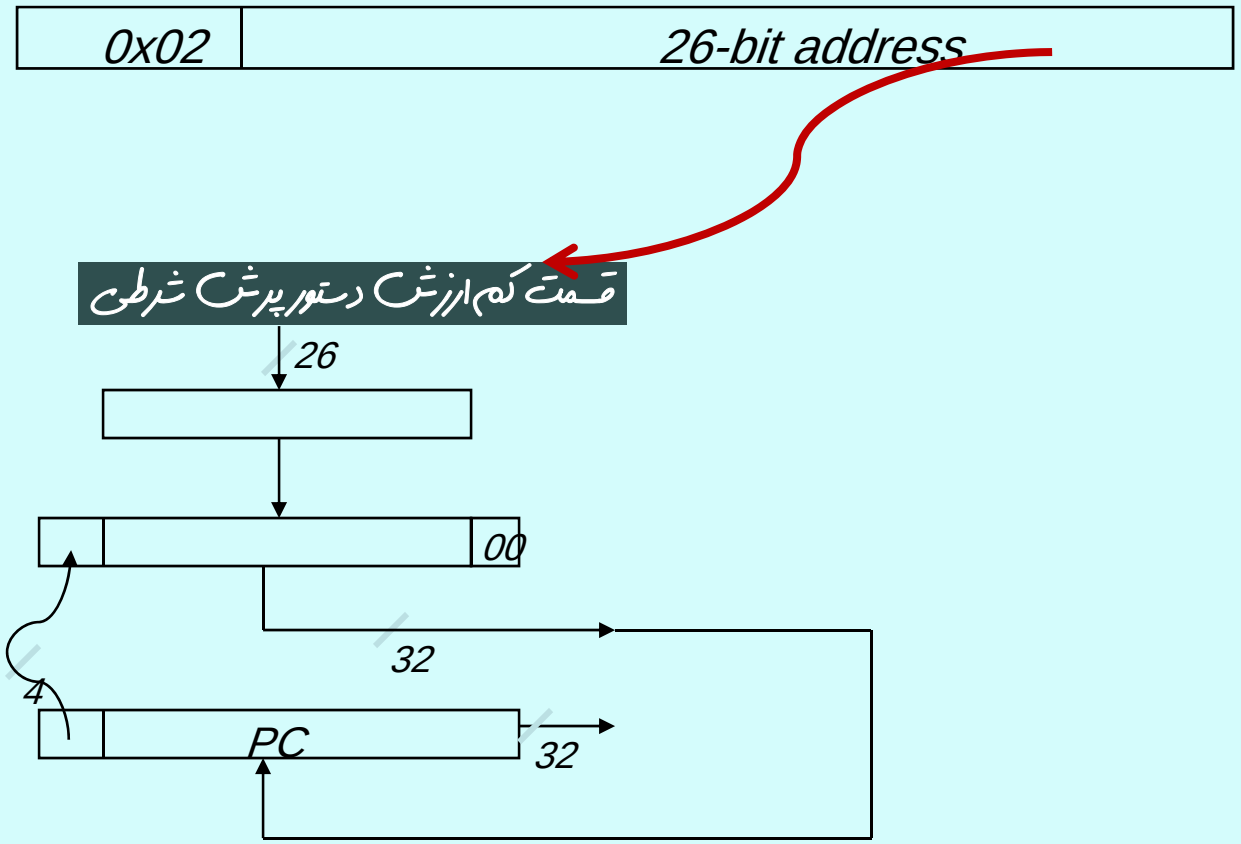



1. Register addressing
op
rs
rt
rd
funct
Register
word operand
2. Base (displacement) addressing
op
rs
rt
offset
Memory
word or byte operand
base register
3. Immediate addressing
op
rs
rt
operand
4. PC-relative addressing
op
rs
rt
offset
Memory
branch destination instruction
Program Counter (PC)
5. Pseudo-direct addressing
op
jump address
Memory
jump destination instruction
//
Program Counter (PC)

سبک‌های آدرس‌دهی (PC relative)



سبک‌های آدرس‌دهی(شبه مستقیم)

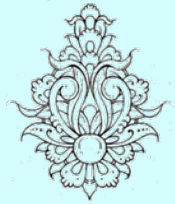


پردازش موازی و همگام‌سازی

- در باره‌ی مزایای پردازش موازی، سخن‌های فروانی گفته شد، در این بخش به یکی از چالش‌های فراروی اجرای موازی فرآیندها خواهیم پرداخت.
- دو فرآیند را تصور کنید که بخشی از فضای حافظه را به صورت مشترک مورد استفاده قرار می‌دهند.
  - P1 می‌نویسد، P2 می‌خواند.
  - P1 و P2 باید هماهنگ باشند، در غیر این صورت رقابت داده پیش خواهد آمد.

Data race

- پاسخ به ترتیب دسترسی دو فرآیند به حافظه وابسته است.



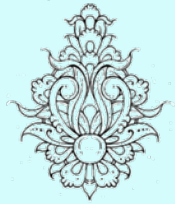
نه تنها در سیستم‌های چند پردازنده‌ای، بلکه در یک سیستم تک‌پردازنده‌ای با قابلیت multitasking نیز امکان رقابت داده وجود دارد.

پردازش موازی و همگام‌سازی (ادامه...)

- سازوکارهای همگام‌سازی، معمولا در لایه‌ی روال‌های کاربر انجام‌می‌پذیرد.
- اما این روال‌ها به پشتیبانی دستورات سخت‌افزاری وابسته هستند.
  - یکی از دستوراتی که برای همگام‌سازی استفاده می‌شوند، نوشتن و خواندن **تجزیه‌ناپذیر** است.
  - تجزیه‌ناپذیری، بدین معناست که بین این دو کار، عملیات دیگری نمی‌تواند انجام شود.
  - این دو، یک دستور انگاشته می‌شوند.

Atomic Operation

- بدون پشتیبانی سخت‌افزاری، هزینه‌ی همگام‌سازی بسیار بالا خواهد بود و با تعداد پردازنده‌ها نیز افزایش خواهد یافت.
- چنین دستوراتی برای برنامه‌نویسان سیستم در نظر گرفته شده است.



پردازش موازی و همگام‌سازی (ادامه...)

- یکی از این دستورات **جابه‌جایی تجزیه‌ناپذیر** است.

Atomic Swap/exchange

- با جابجایی تجزیه‌ناپذیر، محتوای ثبات و یک خانه‌ی حافظه به صورت تجزیه‌ناپذیر جابه‌جا خواهد شد.
- با استفاده از چنین دستوری می‌توان lock را به گونه‌ای طراحی نمود که در صورت 0 بودن به معنای آزاد بودن قفل و در غیر این صورت به معنای در دسترس نبودن آن است.

mutual exclusion (Mutex)

انحصار متقابل: الگوریتمی است که در برنامه‌نویسی همروند برای جلوگیری از اخلال در استفاده از منابع مشترک.

پردازش موازی و همگام‌سازی (ادامه...)

- یک فرآیند، با استفاده از جابه‌جایی تجزیه‌ناپذیر اقدام به تغییر قفل می‌کند. در صورتی که پیش از این، فرآیند دیگری قفل را در اختیار گرفته باشد، مقدار 0 وگرنه مقدار 1 را برمی‌گرداند.
- در صورت رها بودن قفل، آن را مقداردهی کرده و مقدار 0 را باز می‌گرداند.
- بدین‌ترتیب دو فرآیند، به طور همزمان نمی‌توانند قفل را در اختیار بگیرند.
- نکته‌ای که به همگام‌سازی کمک می‌کند، تجزیه‌ناپذیری دستور است.

چالش‌های دستورات تجزیه‌ناپذیر

- اجرای چنین دستوری، مستلزم خواندن، بررسی مقدار و در صورت نیاز نوشتن در خانه‌ی حافظه طی انجام یک دستور بی‌وقفه است.
- به جای این کار، می‌توان از دو دستور متوالی بهره جست، به گونه‌ای دو دستور بر روی هم تجزیه‌ناپذیر باشند.

```
ll rt, offset(rs)
```

load link

```
sc rt, offset(rs)
```

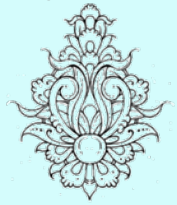
store conditional

در صورتی که خانه‌ای که توسط ll خوانده شده است، تغییر نکند اجرای sc موفقیت‌آمیز انگاشته شده؛ خانه‌ی حافظه را با مقدار rt مقداردهی کرده و در ثبات مزبور مقدار 1 را قرار می‌دهد
در صورت شکست، مقدار 0 در rt قرار خواهد گرفت.

جابجایی تجزیه‌ناپذیر

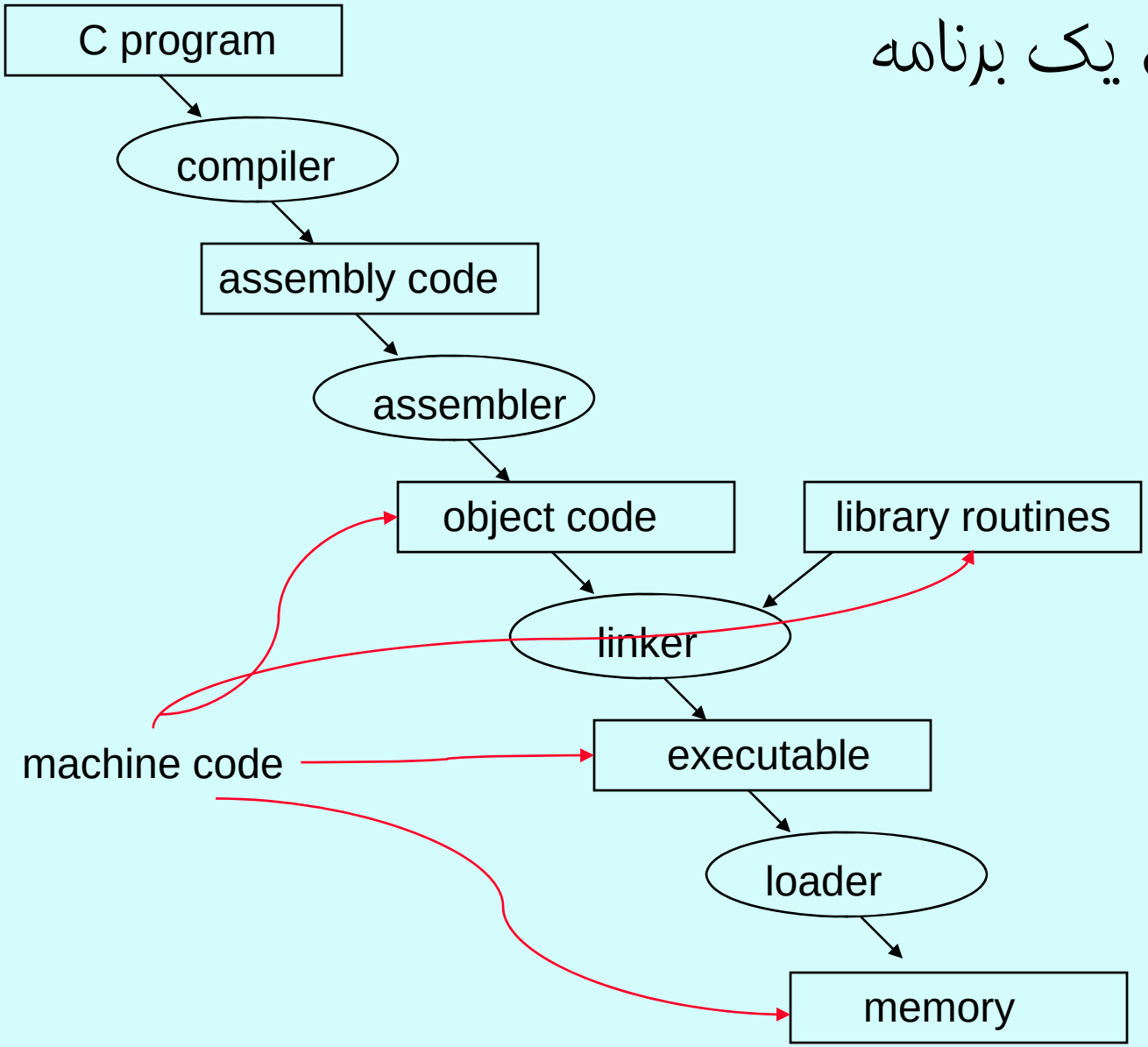
```
try: add $t0,$zero,$s4 ;copy exchange value
     ll  $t1,0($s1)    ;load linked
     sc  $t0,0($s1)    ;store conditional
     beq $t0,$zero,try ;branch store fails
     add $s4,$zero,$t1 ;put load value in
$s4
```

- از این دو دستور، برای پیاده‌سازی تجزیه‌ناپذیر دستورات زیر می‌توان بهره جست:
  - مقایسه و جابه‌جایی
  - واکشی و افزایش



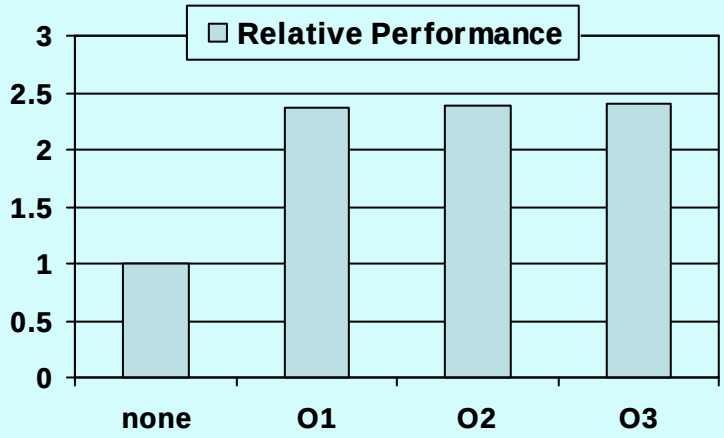
Pseudoinstruction	Usage		
Move	`move`	`regd,regs`	Copy
Load address	`la`	`regd,address`	
Load immediate	`li`	`regd,anyimm`	
Absolute value	`abs`	`regd,regs`	Arithmetic
Negate	`neg`	`regd,regs`	
Multiply (into register)	`mul`	`regd,reg1,reg2`	
Divide (into register)	`div`	`regd,reg1,reg2`	
Remainder	`rem`	`regd,reg1,reg2`	
Set greater than	`sgt`	`regd,reg1,reg2`	
Set less or equal	`sle`	`regd,reg1,reg2`	
Set greater or equal	`sge`	`regd,reg1,reg2`	
Rotate left	`rol`	`regd,reg1,reg2`	Shift
Rotate right	`ror`	`regd,reg1,reg2`	
NOT	`not`	`reg`	Logic
Load doubleword	`ld`	`regd,address`	Memory access
Store doubleword	`sd`	`regd,address`	
Branch less than	`blt`	`reg1,reg2,L`	Control transfer
Branch greater than	`bgt`	`reg1,reg2,L`	
Branch less or equal	`ble`	`reg1,reg2,L`	
Branch greater or equal	`bge`	`reg1,reg2,L`	

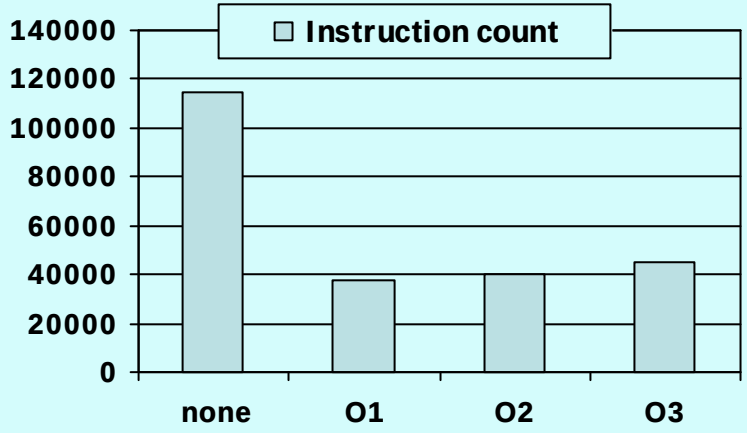
ترجمه‌ی یک برنامه

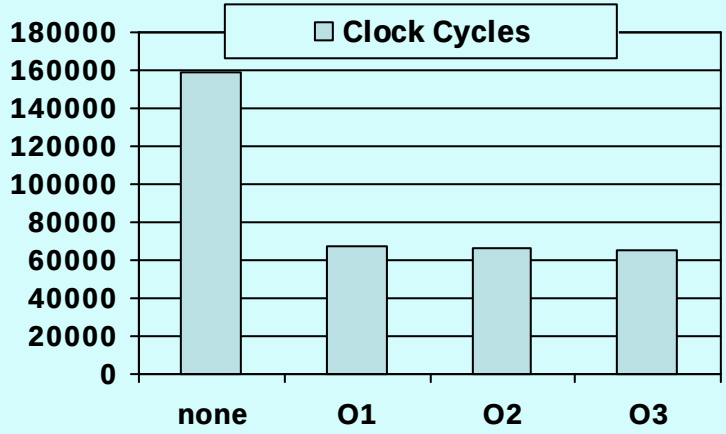


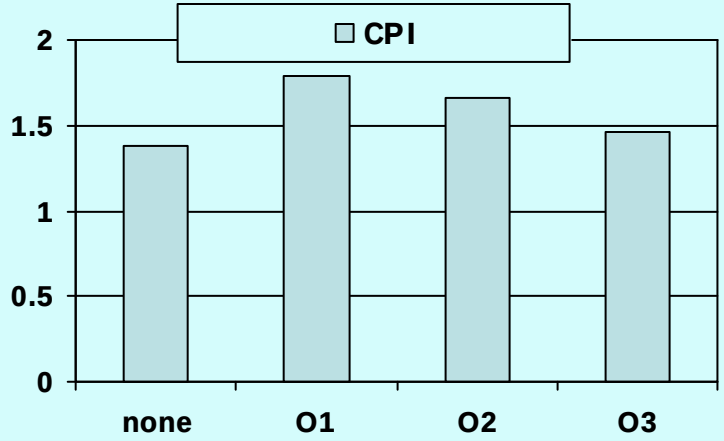
بهینه‌سازی کامپایلر

Compiled with gcc for Pentium 4 under Linux

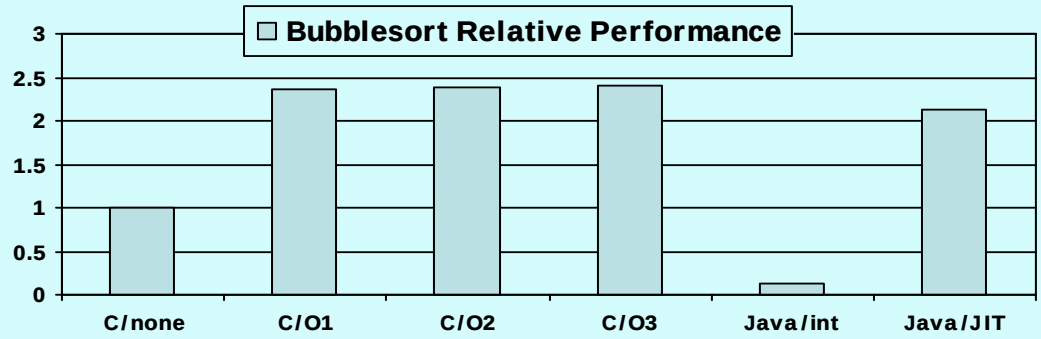


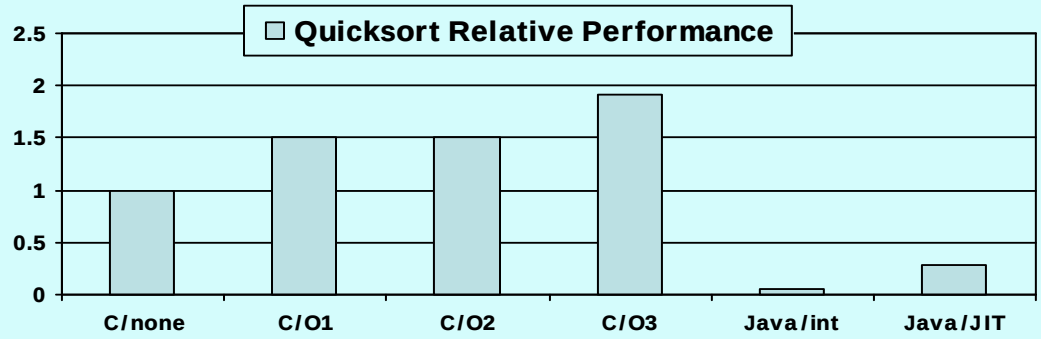


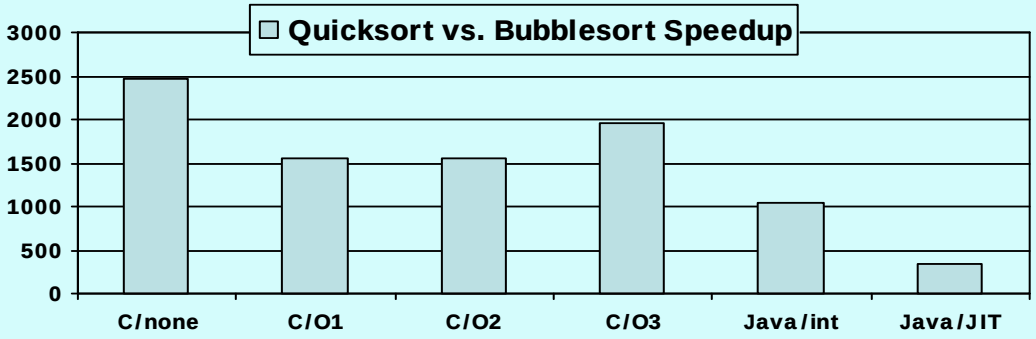




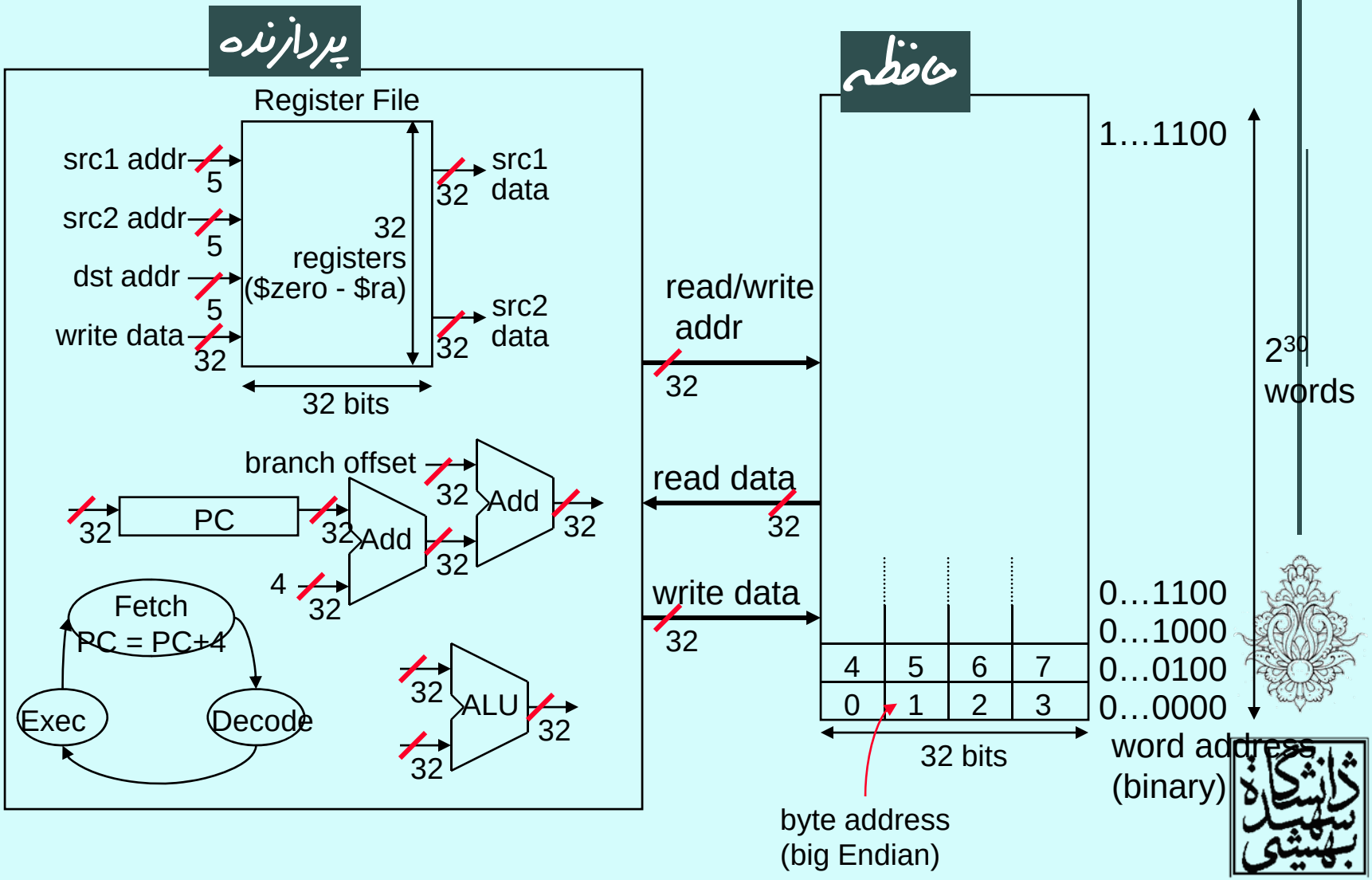
نقش زبان برنامه‌نویسی در کارایی







ساختار پردازنده‌های MIPS



مجموعه دستورات
x86 و ARM

ARM Partnership Model

ARM پردازنده نمی سازد

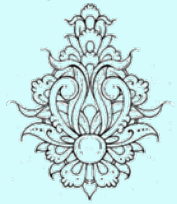
ARM Powered Products

THE ARCHITECTURE FOR THE DIGITAL WORLD

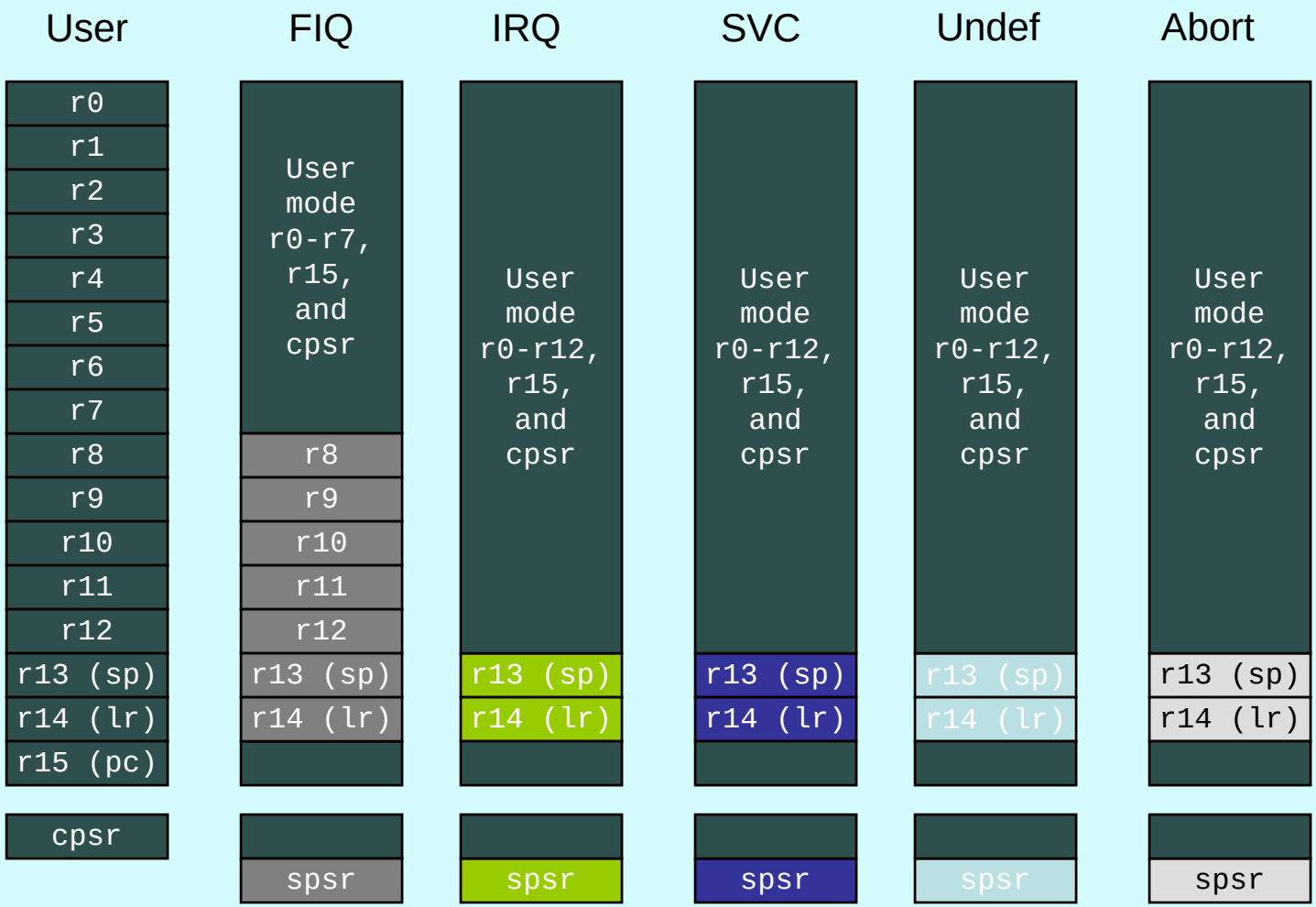
- پردازنده‌های ARM دو مدل دستور دارند:
  - مجموعه دستورات سی‌ودو بیتی
  - مجموعه دستورات شانزده بیتی (thumb)
- برخی هسته‌های ARM توانایی اجرای سخت‌افزاری java byte code را دارند.

Jazelle DBX (Direct Bytecode eXecution)

- پردازنده‌های ARM هفت اسلوب کاری دارند.



ثبات‌ها در ARM



اسلوب system، از ثبات‌های کاربر استفاده می‌کند.

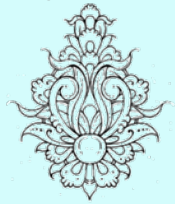
شباهت‌های MIPS و ARM

- ARM متداول‌ترین پردازنده برای سیستم‌های درون‌کار می‌باشد.

	ARM	MIPS
Date announced	1985	1985
Instruction size	32 bits	32 bits
Address space	32-bit flat	32-bit flat
Data alignment	Aligned	Aligned
Data addressing modes	9	3
Registers	15 × 32-bit	31 × 32-bit
Input/output	Memory mapped	Memory mapped

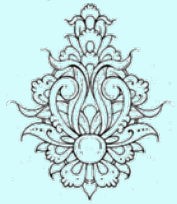
مقایسه و پرش شرطی در ARM

- در ARM از پرچم‌های وضعیت برای دستورات پرش استفاده می‌شود:
  - Negative, zero, carry, overflow
    - این پرچم‌ها در ثبات PSW ذخیره می‌شوند.
    - بعد از دستورات ریاضی و منطقی، مقدار پرچم‌ها می‌تواند تغییر کند.
- دستورات مقایسه، بدون نگهداری نتیجه مقدار پرچم‌ها را تغییر می‌دهند.

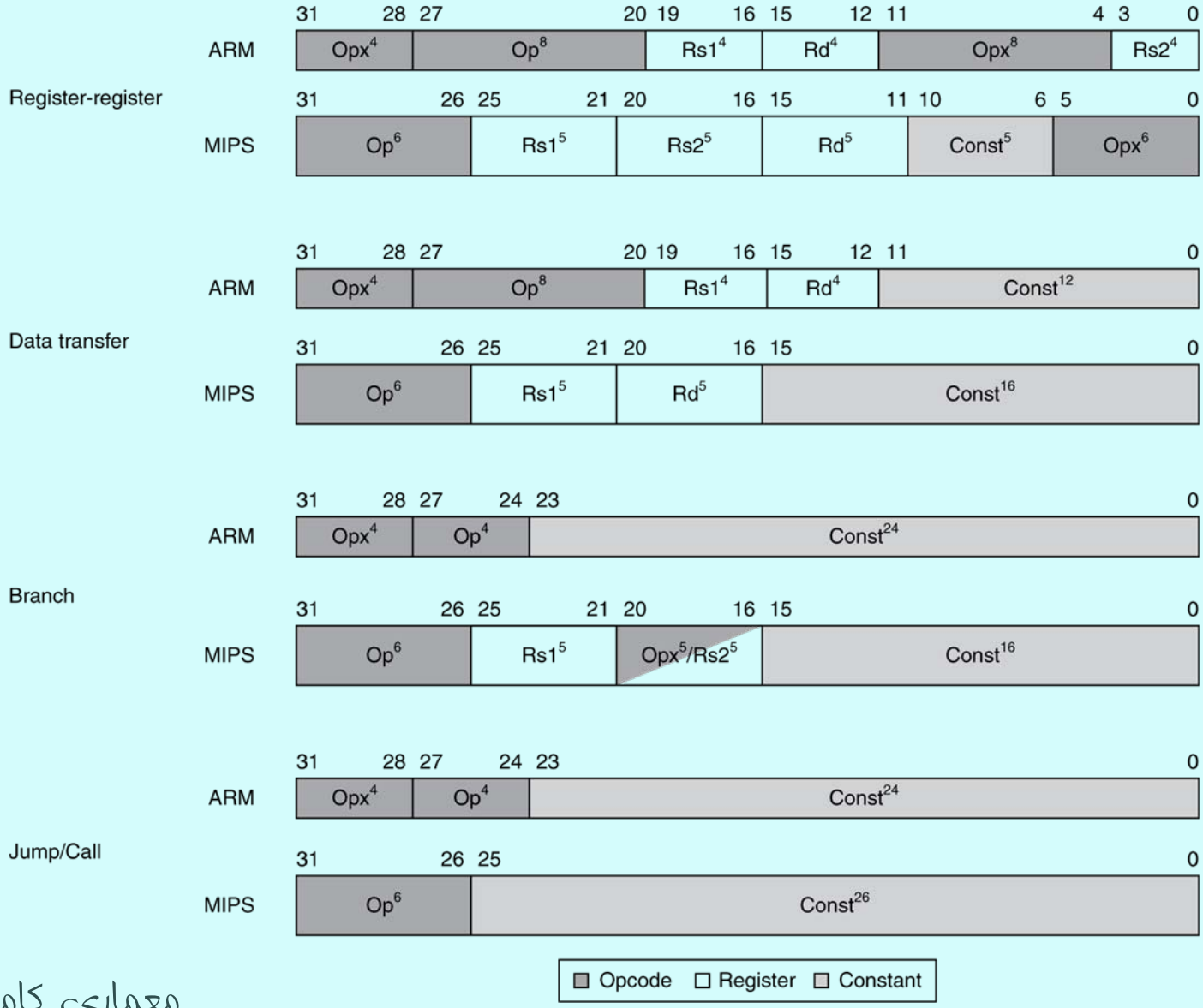


مقایسه و پرش شرطی در ARM (ادامه...)

- تمامی دستورات در ARM قابلیت اجرای مشروط را دارند. چهار بیت پرارزش دستور شرط را معین می‌کند.
- بدین‌ترتیب برای شرطی که روی یک دستور اعمال می‌شود، نیازی به دستورات شرطی نیست.
- بسته به شرط دستور به گونه‌ای خاص و یا به صورت nop اجر می‌شود.



قالب دستورها

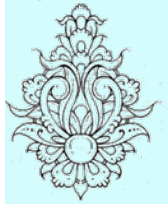


ARM

- ARM دارای دستوری است که می‌تواند گروهی از ثبات‌ها را ذخیره کند.

همچنین، محتوی ثبات دوم در دستورهای حسابی و منطقی قابلیت شیفت دادن را دارد.

Name	Definition	ARM v.4	MIPS
Load immediate	Rd = Imm	mov	addi, $0,
Not	Rd = ~(Rs1)	mvn	nor, $0,
Move	Rd = Rs1	mov	or, $0,
Rotate right	Rd = Rs i >> i $Rd_{0...i-1} = Rs_{31-i...31}$	ror	
And not	Rd = Rs1 & ~(Rs2)	bic	
Reverse subtract	Rd = Rs2 - Rs1	rsb, rsc	
Support for multiword integer add	CarryOut, Rd = Rd + Rs1 + OldCarryOut	adcs	—
Support for multiword integer sub	CarryOut, Rd = Rd – Rs1 + OldCarryOut	sbcs	—



مقایسه‌ی مودهای آدرس‌دهی

Addressing mode	ARM v.4	MIPS
Register operand	X	X
Immediate operand	X	X
Register + offset (displacement or based)	X	X
Register + register (indexed)	X	—
Register + scaled register (scaled)	X	—
Register + offset and update register	X	—
Register + register and update register	X	—
Autoincrement, autodecrement	X	—
PC-relative data	X	—

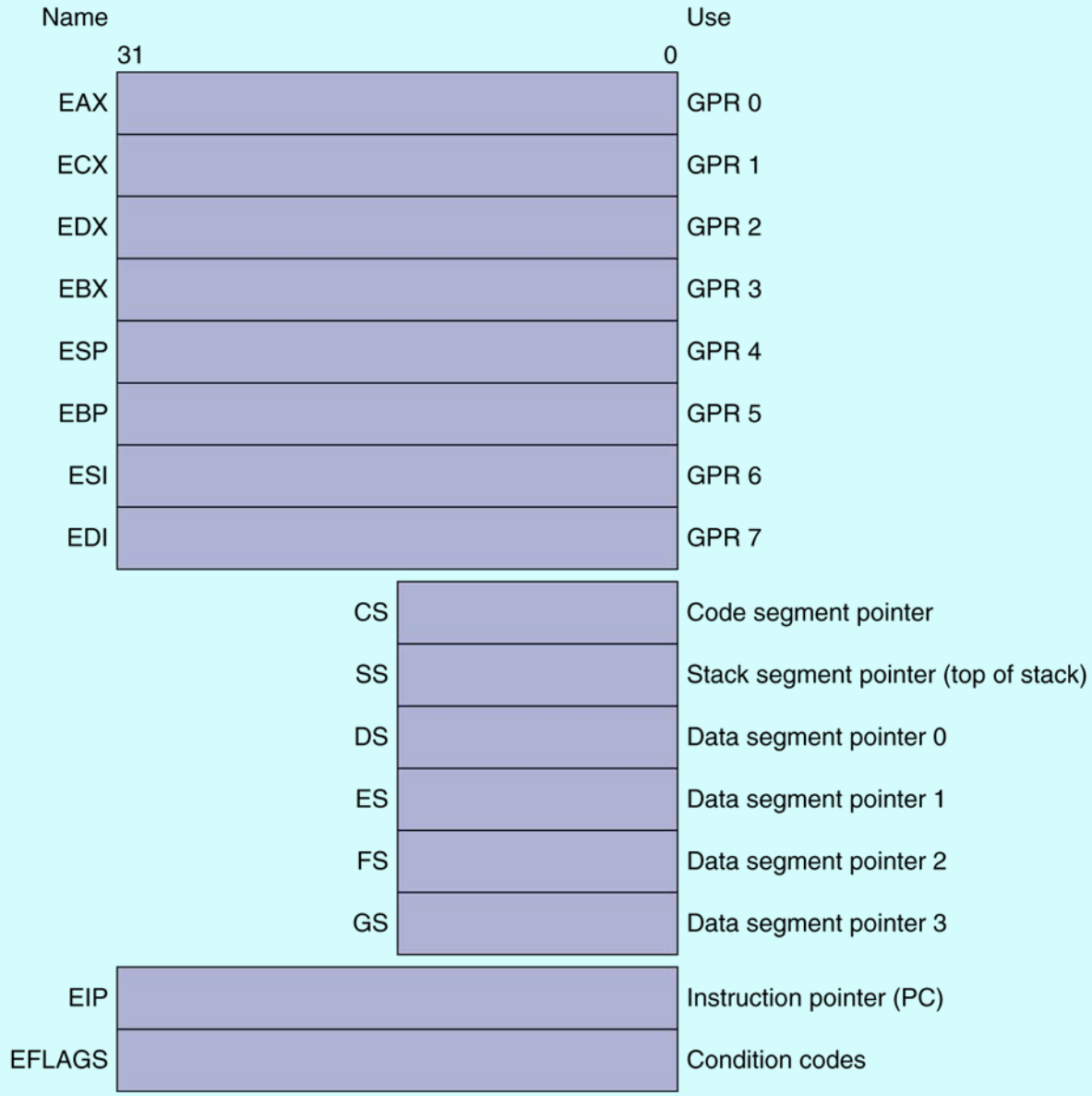
مجموعه دستورات خانواده‌ی X86

- روند تکامل با حفظ سازگاری

– 8080	۱۹۷۴	۸بیتی
– 8086	۱۹۷۸	۱۶بیتی
– 8087	۱۹۸۰	کمک‌پردازنده‌ی ممیز شناور
– 80286	۱۹۸۲	آدرس ۲۴بیتی همراه با MMU
– 80386	۱۹۸۵	۳۲ بیتی، اضافه شدن مودهای آدرس‌دهی جدید
– i486	۱۹۸۹	دارای خط لوله، حافظه‌ی نهان
– Pentium	۱۹۹۳	superscaler
– Pentium Pro	۱۹۹۵	ریزمعماری جدید
– Pentium III	۱۹۹۹	
– Pentium 4	۲۰۰۱	

Technical elegance ≠ market success

ثبات‌های خانواده‌ی x86



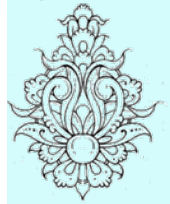
حالت‌های آدرس‌دهی حافظه:

- Address in register
- Address = R_{base} + displacement
- Address = R_{base} + 2^{scale} × R_{index} (scale = 0, 1, 2, or 3)
- Address = R_{base} + 2^{scale} × R_{index} + displacement

Mode	Description	Register restrictions	MIPS equivalent
Register indirect	Address is in a register.	Not ESP or EBP	`lw $s0,0($s1)`
Based mode with 8- or 32-bit displacement	Address is contents of base register plus displacement.	Not ESP	`lw $s0,100($s1) # <= 16-bit # displacement`

Base plus scaled index	The address is Base + (2^{Scale} x Index) where Scale has the value 0, 1, 2, or 3.	Base: any GPR Index: not ESP	`mul $t0,$s2,4`
			`add $t0,$t0,$s1`
			`lw $s0,0($t0)`

Base plus scaled index with 8- or 32-bit displacement	The address is Base + (2^{Scale} x Index) + displacement where Scale has the value 0, 1, 2, or 3.	Base: any GPR Index: not ESP	`mul $t0,$s2,4`
			`add $t0,$t0,$s1`
			`lw $s0,100($t0) # ð16-bit # displacement`

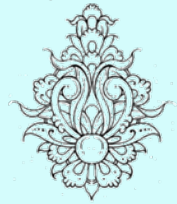


نمونه‌ای از دستورات x86

Instruction	Function
je name	if equal(condition code) {EIP=name}; EIP-128 <= name < EIP+128
jmp name	EIP=name
call name	SP=SP-4; M[SP]=EIP+5; EIP=name;
movw EBX,[EDI+45]	EBX=M[EDI+45]
push ESI	SP=SP-4; M[SP]=ESI
pop EDI	EDI=M[SP]; SP=SP+4
add EAX,#6765	EAX= EAX+6765
test EDX,#42	Set condition code (flags) with EDX and 42
movsl	M[EDI]=M[ESI]; EDI=EDI+4; ESI=ESI+4

نمونه‌ای از دستورات x86 (ادامه...)

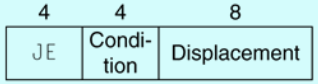
Instruction	Meaning
Control	**Conditional and unconditional branches**
`jnz, jz`	Jump if condition to EIP + 8-bit offset; `JNE` (for `JNZ`), `JE` (for `JZ`) are alternative names
`jmp`	Unconditional jump—8-bit or 16-bit offset
`call`	Subroutine call—16-bit offset; return address pushed onto stack
`ret`	Pops return address from stack and jumps to it
`loop`	Loop branch—decrement ECX; jump to EIP + 8-bit displacement if ECX ≠ 0
Data transfer	**Move data between registers or between register and memory**
`move`	Move between two registers or between register and memory
`push, pop`	Push source operand on stack; pop operand from stack top to a register
`les`	Load ES and one of the GPRs from memory
Arithmetic, logical	**Arithmetic and logical operations using the data registers and memory**
`add, sub`	Add source to destination; subtract source from destination; register-memory format
`cmp`	Compare source and destination; register-memory format
`shl, shr, rcr`	Shift left; shift logical right; rotate right with carry condition code as fill
`cbw`	Convert byte in eight rightmost bits of EAX to 16-bit word in right of EAX
`test`	Logical AND of source and destination sets condition codes
`inc, dec`	Increment destination, decrement destination
`or, xor`	Logical OR; exclusive OR; register-memory format
String	**Move between string operands; length given by a repeat prefix**
`movs`	Copies from string source to destination by incrementing ESI and EDI; may be repeated
`lods`	Loads a byte, word, or doubleword of a string into the EAX register



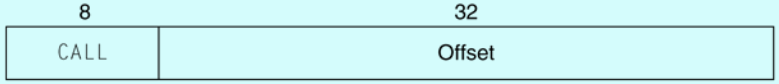
قالب دستورها

- طول دستورها متغیر است.
- سخت‌افزار دستورات را به دستورهای ساده‌تری (ریز دستور) ترجمه می‌کند.

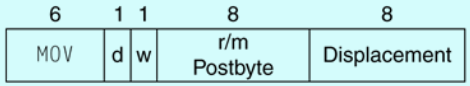
a. JE EIP + displacement



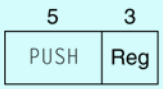
b. CALL



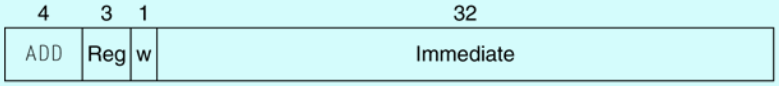
c. MOV EBX, [EDI + 45]



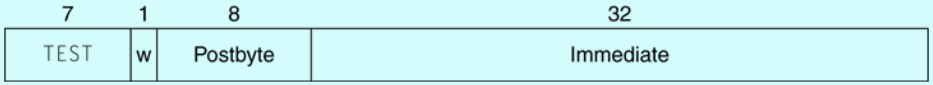
d. PUSH ESI



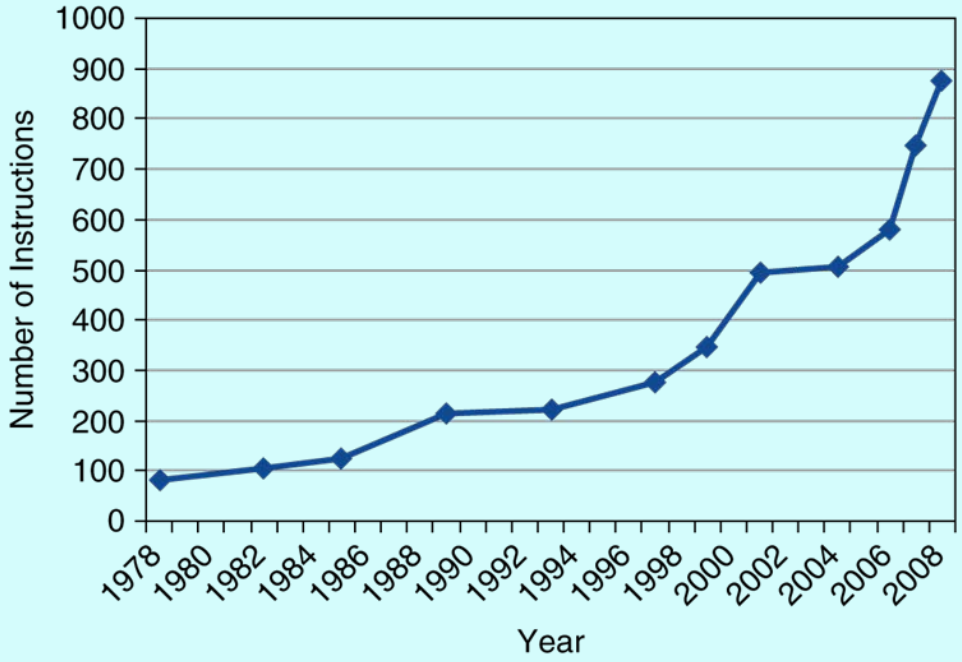
e. ADD EAX, #6765



f. TEST EDX, #42



- آیا دستورات پیچیده به معنای کارایی بالاتر است؟
- آیا نوشتن برنامه به زبان اسمبلی، کارایی را افزایش می‌دهد؟



نتیجه‌گیری

- نظم منجر به سادگی بیشتر می‌شود.
- کوچک‌تر یعنی سریع‌تر
- سرعت داده به موارد پر استفاده
- طراحی خوب یعنی مصالحه‌ی خوب