معماری کامپیوتر (۱۳-۱۱-۰۱۳)

جلسه‌ی چهارم

دانشگاه شهید بهشتی

دانشکده‌ی مهندسی برق و کامپیوتر

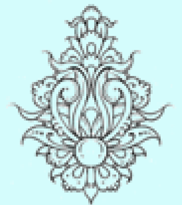
زمستان ۱۳۹۰

احمد محمودی ازناوه

http://faculties.sbu.ac.ir/~a_mahmoudi/

فهرست مطالب

- قانون Amdahl
- رابطه‌ی توان مصرفی و بار محاسباتی
- MIPS معیاری دیگر برای ارزیابی کامپیوترها
- آشنایی با زبان اسمبلی MIPS



قانون Amdahl

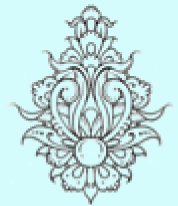
- برنامه‌ای را در نظر بگیرید که زمان پاسخ آن ۱۰۰ ثانیه است، ۸۰ ثانیه‌ی مربوط به عملیات ضرب می‌باشد، سرعت عملیات ضرب چند برابر شود تا سرعت برنامه پنچ بر ابر شود؟

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

$$20 = \frac{80}{n} + 20$$

دام: نباید انتظار داشت متناسب با بهبود یک بخش عملکرد کلی بهبود یابد شدنی نیست!

Amdahl's law

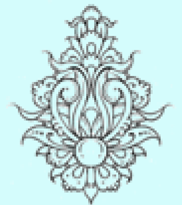


قانون **Amdhal**
بهبود کارایی سیستم، هنگامی که بخشی از آن بهبود یابد

نتیجه: بهتر است قسمت‌های پرکاربرد بهبود یابند

- توان مصرفی X4
  - در صورت به‌کارگیری ۱۰۰ درصدی 295W
  - در صورت به‌کارگیری ۵۰ درصدی 246W
  - در صورت به‌کارگیری ۱۰ درصدی 180W
- سرورهای گوگل اغلب با ده تا پنجاه درصد ظرفیت کار می‌کنند و تنها یک درصد اوقات بار آن به صددرصد می‌رسد.

Energy-proportional computing



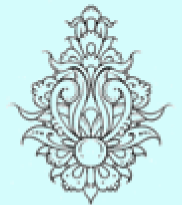
معیاری دیگر برای ارزیابی کارایی

MIPS: million instructions per second

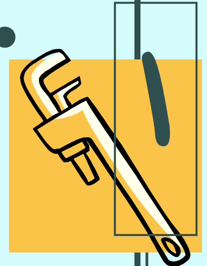
$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

- در ISAهای متفاوت، توانایی‌ها فرق می‌کند.
- MIPS در یک کامپیوتر خاص برای برنامه‌های متفاوت، مقادیر متفاوتی خواهد داشت.

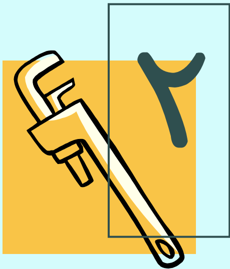
$$\text{MIPS} = \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$



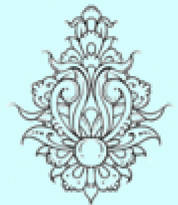
اصول طراحی سخت‌افزار- استفاده از ثبات‌ها



- نظم (Regularity) منجر به ارائه‌ی مداری ساده‌تر (Simplicity) می‌شود.
  - مدار ساده‌تر معادل به دست آوردن مداری با کارایی بالا و قیمتی پایین خواهد شد.
  - سخت‌افزار برای تعداد عملوند متغیر پیچیده‌تر است.



  - طرح هرچه کوچک‌تر باشد، سریع‌تر خواهد بود.



حافظه به عنوان عملوند

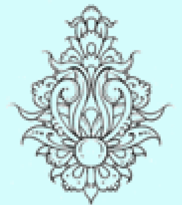
- ساختارهای پیچیده‌تر در حافظه ذخیره می‌شوند.
  - مانند آرایه‌ها، ساختارها و ...
- برای اعمال دستورهای حسابی
  - داده از حافظه به ثبات منتقل شده و پس از انجام محاسبات، حاصل در حافظه نوشته می‌شود.
- هر خانه‌ی حافظه به یک بایت اشاره می‌کند.
- کلمات در حافظه هم‌تراز شده‌اند، شروع هر کلمه مضربی از چهار است.

alignment restriction

- در MIPS، خانه‌ی حافظه با آدرس کوچک‌تر، حاوی بایت پرارزش‌تر است.

big-endian

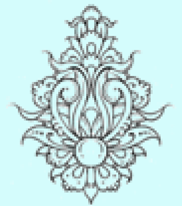
بخش پرارزش‌تر در خانه‌ی اول قرار می‌گیرد



ثبات و حافظه

- دستیابی به محتوای ثبات‌ها بسیار سریع‌تر از محتوای حافظه می‌باشد.
- برای هر بار دستیابی به حافظه، اجرای دستورات lw و sw لازم است. یعنی تعداد دستورات بیشتر است.
- کامپایلر باید تا جایی که ممکن است از رجیسترها به عنوان متغیر استفاده کنند.
  - در صورت در اختیار نداشتن ثبات، از بین متغیرها، آن‌هایی که کم‌تر مورد استفاده قرار می‌گیرند، از ثبات خارج می‌شوند.
  - استفاده بهینه از فضای ثبات‌ها مهم است.

spilling register



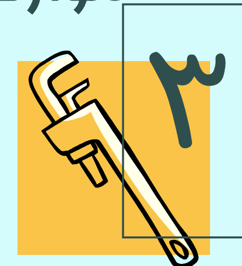
استفاده از اعداد ثابت

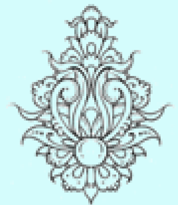
- در بسیاری موارد لازم است، از اعداد ثابت در برنامه‌ها استفاده کرد. چه راهی پیشنهاد می‌دهید؟
  – به عنوان مثال در صورتی که بخواهیم به $s3 چهار واحد اضافه کنیم؟

```
lw  $t0,AddrConstant4($s1) # $t0= constant 4
add $s3,$s3,$t0
```

  – با توجه به استفاده مکرر از چنین دستوراتی (بر اساس آزمون SPEC2006 نیمی از دستورات MIPS دارای عملوند ثابت هستند) و این اصل که
  – موارد پر استفاده، باید سریع‌تر باشند.

```
addi $s3, $s3, 4
```

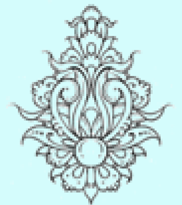




قالب دستورهای MIPS

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- فیلدهای دستور:
  - op: دستور را مشخص می‌کند.
  - rs: ثبات منبع اول
  - rt: دومین ثبات منبع
  - rd: ثبات مقصد
  - shamt: میزان شیفت
  - funct: نوع خاصی از دستور



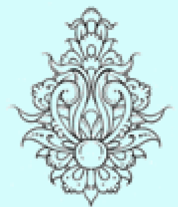
مثال

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

add $t0, $s1, $s2

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

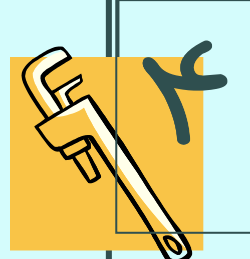
$00000010001100100100000000100000_2 = 02324020_{16}$



دستورهای با عملوند ثابت

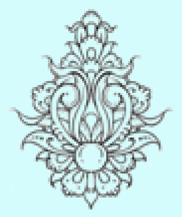
op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- برای دستورات lw، st و دستوراتی که نیاز به استفاده از ثابت‌ها دارند، قالب دیگری مطرح می‌شود.
  - rt: ثبات منبع یا مقصد
  - بدین ترتیب می‌توان ثابتی از -2^{15} تا $+2^{15}-1$ را در این گونه دستورها به کار برد.
  - همچنین، برای دستوراتی که با آدرس حافظه کار می‌کنند، بخش آخر دربردارنده‌ی آدرس می‌باشد.



Constant

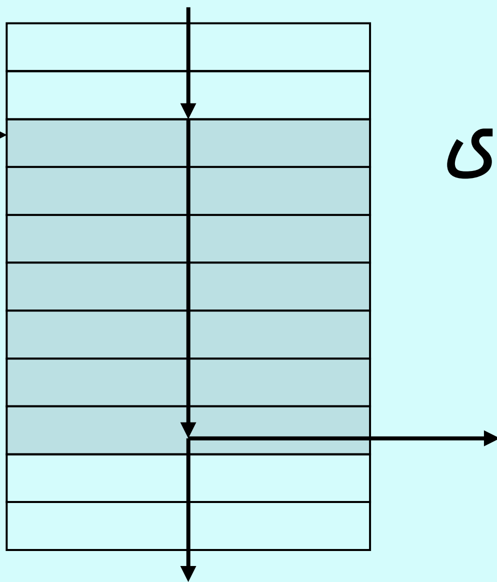
- در یک طراحی خوب، معادل حل یک مسأله‌ی بهینه‌سازی است.
  - استفاده از قالب‌های متنوع طراحی را پیچیده می‌کند، در عوض طول دستورات ثابت مانده است.
  - با این حال، باید تا جایی که شدنی است، قالب‌ها مشابه باشند.

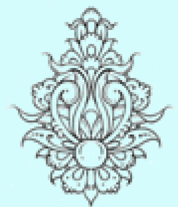


بلوک پایه

Basic Block

- یک بلوک پایه مجموعه‌ای از دستورهاست که
  - دارای دستورات پرش نیستند (به جز انتها)
  - دارای برچسب نیستند (به جز ابتدا)
- کامپایلر برای بهینه‌سازی بلوک‌های پایه را شناسایی می‌کند.
- پردازنده‌های پیشرفته می‌توانند اجرای بلوک‌های پایه را تسریع بخشند.

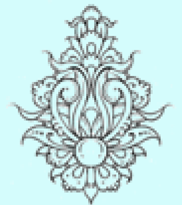




Pseudo instructions

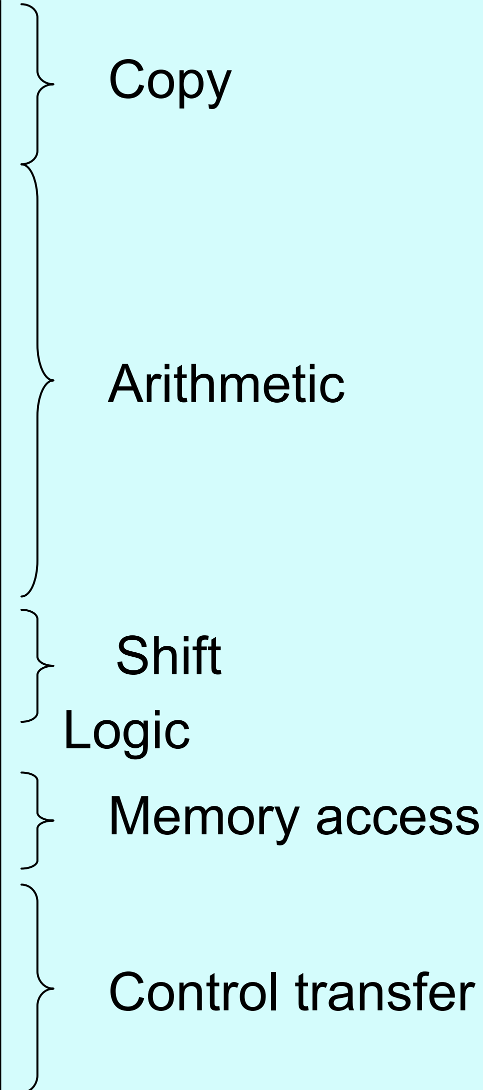
طراحی دستورات شرطی

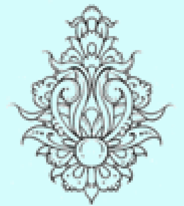
- دستورهای blt و bge
  - سخت‌افزار مدارهای $<$ و یا $\leq$ نسبت به $=$ یا $\neq$ کندتر هستند.
  - هنگامی که با پرش همراه شوند، به زمان بیشتری نیاز دارند و در نتیجه پالس‌ساعت کندتر خواهد شد.
  - بدین ترتیب تمام دستورها کند می‌شوند.
- در MIPS دستور پرش در حالتی رجیستری از دیگری کوچک‌تر باشد، تعبیه نشده است. چنین دستوری پیچیده است. استفاده از دو دستور ساده ترجیح داده شده است.



اسمبلر (شبه‌دستور)

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`	





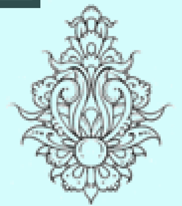
اسمبلر (شبه‌دستور)

```
not  $s0                    # complement ($s0)
```

```
nor  $s0,$s0,$zero    # complement ($s0)
```

```
abs  $t0,$s0          # put |($s0)| into $t0
```

```
add  $t0,$s0,$zero   # copy x into $t0
       slt  $at,$t0,$zero   # is x negative?
       beq  $at,$zero,+4    # if not, skip next instr
       sub  $t0,$zero,$s0   # the result is 0 – x
```



مقایسه و علامت

- دستورات slt و slti در مقایسه، اعداد را به صورت مکمل ۲ در نظر می‌گیرند، برای مقایسه‌ی بدون علامت از دستورهای `sltu` و `sltui` استفاده می‌شود.

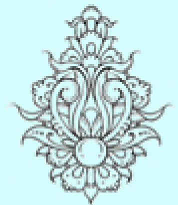
```
$s0 = 1111 1111 1111 1111 1111 1111 1111 1111
$s1 = 0000 0000 0000 0000 0000 0000 0000 0001
```

```
slt  $t0, $s0, $s1  # signed
```

$-1 < +1 \Rightarrow \$t0 = 1$

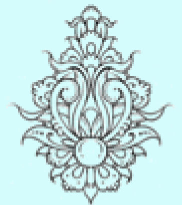
```
sltu $t0, $s0, $s1  # unsigned
```

$+4,294,967,295 > +1 \Rightarrow \$t0 = 0$



فراخوانی روال

- برای فراخوانی یک روال، مراحل زیر انجام می‌شود:
  - ارسال پارامترها به روال
  - انتقال کنترل به روال
  - تخصیص حافظه‌ی مورد نیاز
  - اجرای روال
  - انتقال نتیجه‌ی به دست آمده به برنامه‌ی اصلی
  - بازگرداندن کنترل به برنامه‌ی اصلی



ارسال پارامترها

- در MIPS، برای انتقال پارامترها از ثبات‌ها استفاده می‌شود.

arguments

– $a0 – $a3:

– برای پارامترهای ارسالی (ثبات شماره ۴ تا ۷)

result values

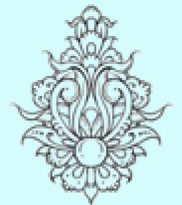
– $v0, $v1:

– برای مقادیری فرستاده شده(ثبات شماره ۲ تا ۳)

return address

– $ra:

– آدرس بازگشت در این ثبات ذخیره می‌شود. (ثبات شماره ۳۱)



سایر ثبات‌ها

– $t0 – $t9: temporaries

– ثبات‌های موقت، رویه می‌تواند از این ثبات‌ها استفاده کند.

– $s0 – $s7: saved

– رویه‌ی فراخوانی شده، نباید مقادیر این ثبات‌ها را تغییر دهد.

– $gp: global pointer

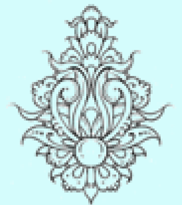
– اشاره‌گر عمومی برای داده‌های ایستا (شماره ۲۸)

– $sp: stack pointer

– اشاره‌گر به پشته (شماره ۲۹)

– $fp frame pointer

– اشاره‌گر قاب (شماره ۳۰)



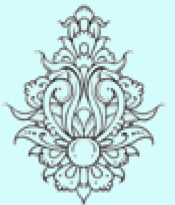
دستور فراخوانی تابع

```
jal ProcedureLabel
```

jump-and-link instruction

- با اجرای این دستور، افزون بر پرش به آدرس شروع رویه، آدرس بازگشت در $ra قرار می‌گیرد.
- برای بازگشت به برنامه کافیست از دستور پرشی که پیش از این با آن اشنا شدیم، استفاده کنیم.

```
jr $ra
```



program counter (PC) or instruction address

ثباتی که آدرس بخشی از برنامه را که بناست اجرا شود، نگه می‌دارد