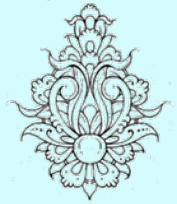
معماری کامپیوتر (۱۱۳۵-۱۱-۱۳)

جلسه‌ی نخست

دانشگاه شهید بهشتی
دانشکده‌ی مهندسی برق و کامپیوتر
زمستان ۱۳۹۰
احمد محمودی ازناوه

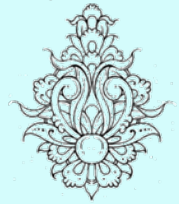
فهرست مطالب

- دیباچه
- بارم‌بندی
- مراجع
- چگونگی اجرای یک برنامه



دیباچه

- عملکرد هر نرم‌افزار به به سخت‌افزاری که روی آن اجرا می‌شود، بستگی دارد. از این رو، شناخت ویژگی‌های ریزپردازنده‌ها و تعامل سخت‌افزار و نرم‌افزار برای بهبود کارایی برنامه ضروری است.
- از سوی دیگر؛ برای چیرگی بر محدودیت‌های موجود از نظر توان مصرفی و فن‌آوری ساخت، «پردازش موازی» پیشنهاد شده است.
- استفاده از GPU برای کارهایی با حجم پردازش بالا مؤیدی بر این ادعاست.



بارم‌بندی

توجه: بارم‌بندی فوق تقریبی است و با توجه به شرایط قابل تغییرات است :D

- درصد نمرات
  - میان‌ترم (۴ اردیبهشت) ۵-۴ نمره
  - تکالیف ۲ نمره
  - پروژه ۶-۴ نمره
  - کوییز ۳-۲ نمره
  - پایان‌ترم ۸-۶ نمره
  - دستیاران:
    - آقای سینا آقاسی و آقای محسن فاریابی
    - آقایان:

سینا آقاسی و محسن فاریابی

- زمان پیشنهادی برای کلاس حل تمرین: سه‌شنبه ساعت ۱۲ تا ۱۴

تذکرات

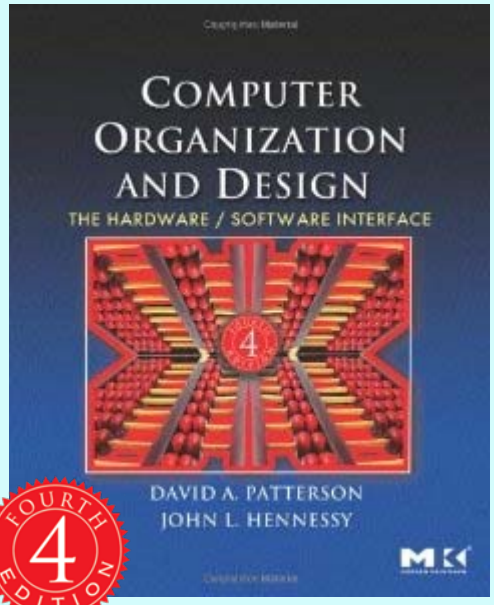
- برای امور رسمی، شناسه‌ی ایمیل مناسبی انتخاب کنید.
- عنوان ایمیل، بیانگر محتوای آن باشد.
  - حتما عنوان مناسبی برای ایمیل خود در نظر بگیرید.
- در صورتی که در رابطه با مطلبی، ایمیل می‌زنید، لطفا در پایان ایمیل نام خود را هم بنویسید، به ویژه اگر از نام مستعار برای شناسه‌ی ایمیل خود استفاده می‌کنید.
- نام درس و گروه فراموش نشود.
- از نوشتن به صورت فینگیلیش بپرهیزید.

از همکاری شما پیشاپیش سپاسگزارم!

- **Computer Organization and Design: The Hardware/Software Interface**

by David A. Patterson

& John L. Hennessy



فراگیری علوم و مهندسی کامپیوتر، افزون بر مفاهیم پایه می‌باید پیشرفت‌های کنونی را نیز در بر گیرد.

جمله‌ی نخست دیباچه‌ی کتاب

مراجع (ادامه...)

Read carefully	Read if have time	Reference
Review or read	Read for culture	

Chapter or appendix	Sections	Software focus	Hardware focus
1. Computer Abstractions and Technology	1.1 to 1.9	✓	✓
	1.10 (History)		
2. Instructions: Language of the Computer	2.1 to 2.14	✓	✓
	2.15 (Compilers & Java)	✓	
	2.16 to 2.19	✓	✓
	2.20 (History)		
E. RISC Instruction-Set Architectures	E.1 to E.19	✓	
3. Arithmetic for Computers	3.1 to 3.9	✓	✓
	3.10 (History)	✓	✓
C. The Basics of Logic Design	C.1 to C.13		✓
	4.1 (Overview)	✓	✓
	4.2 (Logic Conventions)		✓
	4.3 to 4.4 (Simple Implementation)	✓	✓

در این کتاب، یکی از اهداف نشان دادن ارتباط نرم‌افزار و سخت‌افزار است

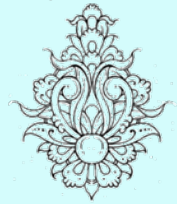
مراجع (ادامه...)

- Computer Organization and Architecture:
  Designing for Performance
  by: William Stallings
- Computer Architecture From Microprocessors to Supercomputers
  by: Behrooz Parhami
- Computer System Architecture, Third Edition
  by: M. Morris Mano
- See MIPS Run
  by: Dominic Sweetman
- Professional Assembly Language
  by: Richard Blum

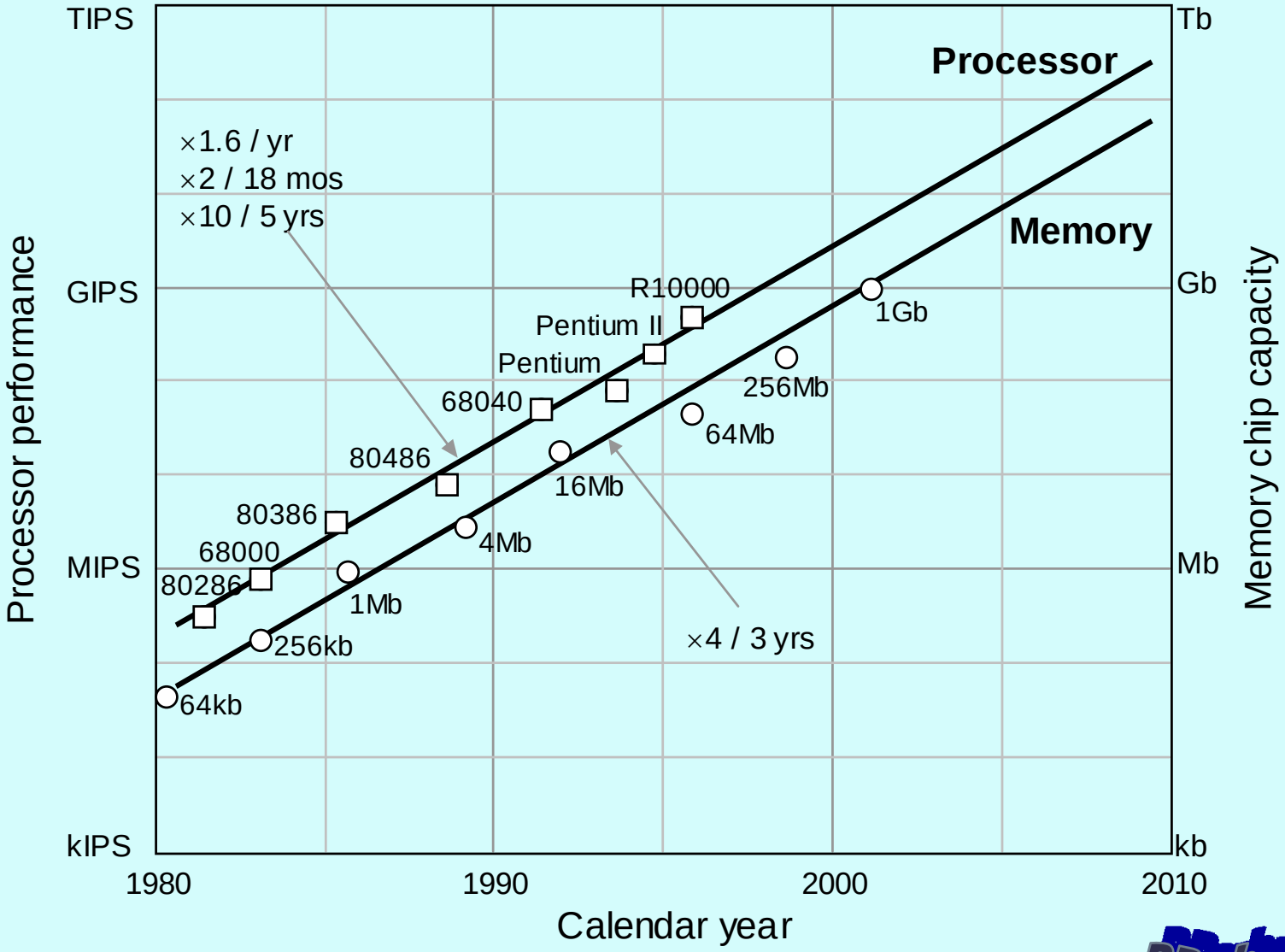
انقلاب کامپیوتری

- **پیشرفت خارق‌العاده‌ی فن‌آوری دیجیتال**
  - که رشد آن از قانون Moore پیروی می‌کند، این قانون را به خاطر دارید؟
- **تقریبا در همه جا می‌توان اثری از کامپوتر یافت**
  - تلفن همراه
  - خودرو
  - Xbox
  - وسایل آشپزخانه-
  - اسباب‌بازی‌ها و .....
- **کامپیوتر چیست؟**

حضور کامپوترها، حضوری فراگیر است و به کامپوترهای شخصی محدود نمی‌شود



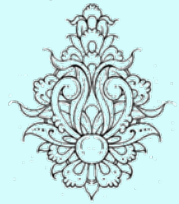
Moore قانون


TIPS
GIPS
MIPS
kIPS
Processor performance
Tb
Gb
Mb
kb
Memory chip capacity
1980
1990
2000
2010
Calendar year
Processor
Memory
×1.6 / yr
×2 / 18 mos
×10 / 5 yrs
×4 / 3 yrs
R10000
Pentium II
Pentium
68040
80486
80386
68000
80286
1Gb
256Mb
64Mb
16Mb
4Mb
1Mb
256kb
64kb

انواع کامپیوترها

- کامپیوترهای رومیزی (شخصی)
  - چندمنظوره
  - امکان اجرای نرم‌افزارهای متفاوت
  - برای استفاده‌ی یک شخص
- سرورها
  - سرعت و حافظه‌ی و قابلیت اعتماد بالا
  - قابل استفاده توسط چندین کاربر به صورت همزمان
- رایانه‌های درون‌کار
  - بزرگ‌ترین دسته از بین گروه‌بندی بالاست
  - خاص‌منظوره

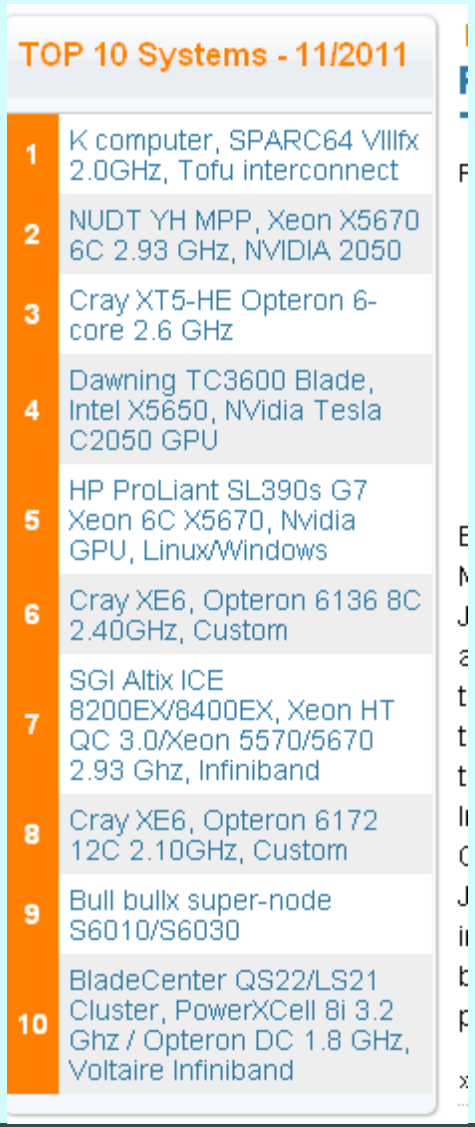
embedded computer



ابرکامپیوتر

- کامپیوتری که از نظر قدرت در زمان معرفی خود پیشتاز باشد.

- رتبه‌ی نخست کنونی:
  - کشور سازنده: ژاپن
  - حافظه‌ی اصلی: 229376 GB
  - قدرت پردازش: 8.162 Petaflops
  - تعداد هسته‌ها: 548352
  - سیستم عامل: لینوکس



TOP 10 Systems - 11/2011

1	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect
2	NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050
3	Cray XT5-HE Opteron 6-core 2.6 GHz
4	Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU
5	HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows
6	Cray XE6, Opteron 6136 8C 2.40GHz, Custom
7	SGI Altix ICE 8200EX/8400EX, Xeon HT QC 3.0/Xeon 5570/5670 2.93 Ghz, Infiniband
8	Cray XE6, Opteron 6172 12C 2.10GHz, Custom
9	Bull bullx super-node S6010/S6030
10	BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband

http://www.top500.org/

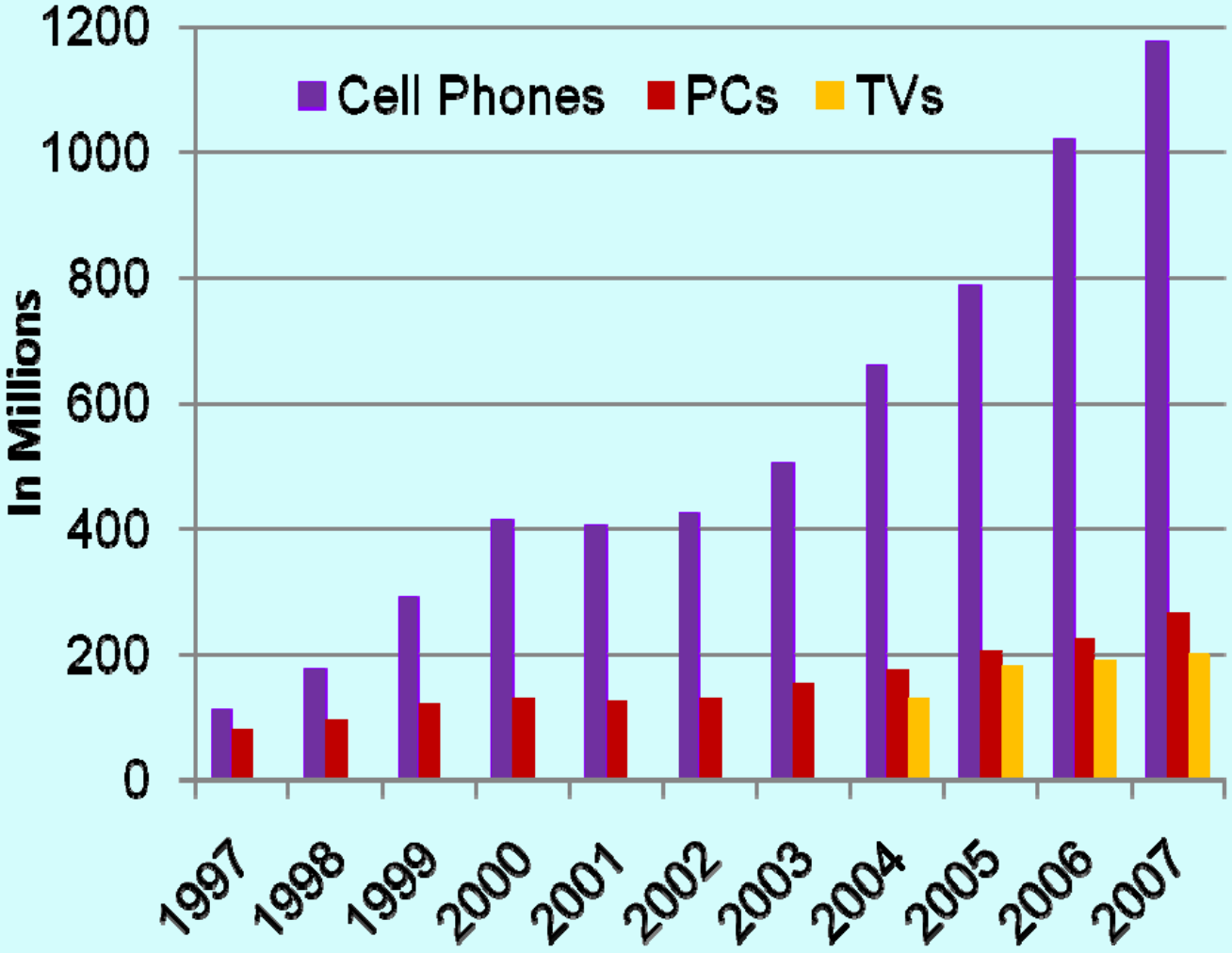
FLoating point OPerations per Second

ابرکامپیوتر

- کامپیوتری که از نظر قدرت در زمان معرفی خود پیشتاز باشد.
- رتبه‌ی نخست کنونی:
  - کشور سازنده: چین
  - هزینه: نود میلیون و چهارصد هزار
  - حا[illegible] اصلی: 229376 GB
  - قد[illegible] پرد[illegible]ش: 2.566 Pe[illegible]aflo[illegible]s
  - تعداد هست[illegible]ها: 18636[illegible]
  - سیستم عامل: لینوکس

http://www.top500.org/

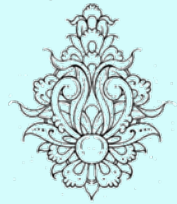
بازار پردازنده‌ها



آن چه خواهیم آموخت

- برنامه‌هایی که می‌نویسیم، چگونه اجرا می‌شود؟
  - زبان ماشین (اسمبلی) چیست؟
- ارتباط بین نرم‌افزار و سخت‌افزار چگونه است؟
- کارایی یک برنامه چگونه تعیین می‌شود؟
  - چگونه می‌توان آن را ارتقا داد؟
- پردازش موازی چیست؟ و علت حرکت از پردازش ترتیبی به پردازش موازی چیست؟

multicore microprocessor



معماری کامپیوتر (۱۳۹۰-۱۱-۱۳)

جلسه‌ی دوم

دانشگاه شهید بهشتی

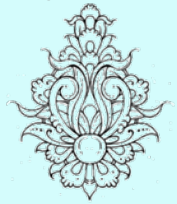
دانشکده‌ی مهندسی برق و کامپیوتر

زمستان ۱۳۹۰

احمد محمودی ازناوه

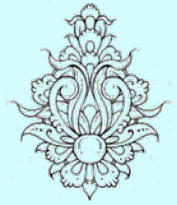
فهرست مطالب

- اجزای داخلی پردازنده
- تجرید
- کارایی



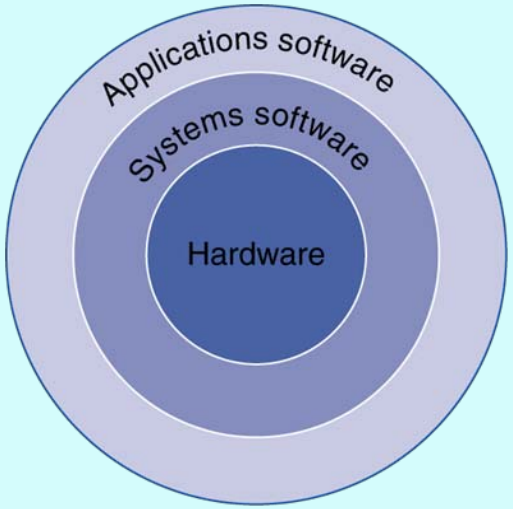
کارایی اجرای برنامه

- الگوریتم
  - تعداد دستورالعمل‌ها و تعداد عملیات I/O
- زبان برنامه‌نویسی، کامپایلر و معماری
  - تعداد دستورالعمل زبان ماشین به ازای دستورالعمل‌های زبان سطح بالا
- پردازنده و حافظه
  - سرعت اجرای هر دستور چقدر است؟
- سرعت انجام عملیات I/O



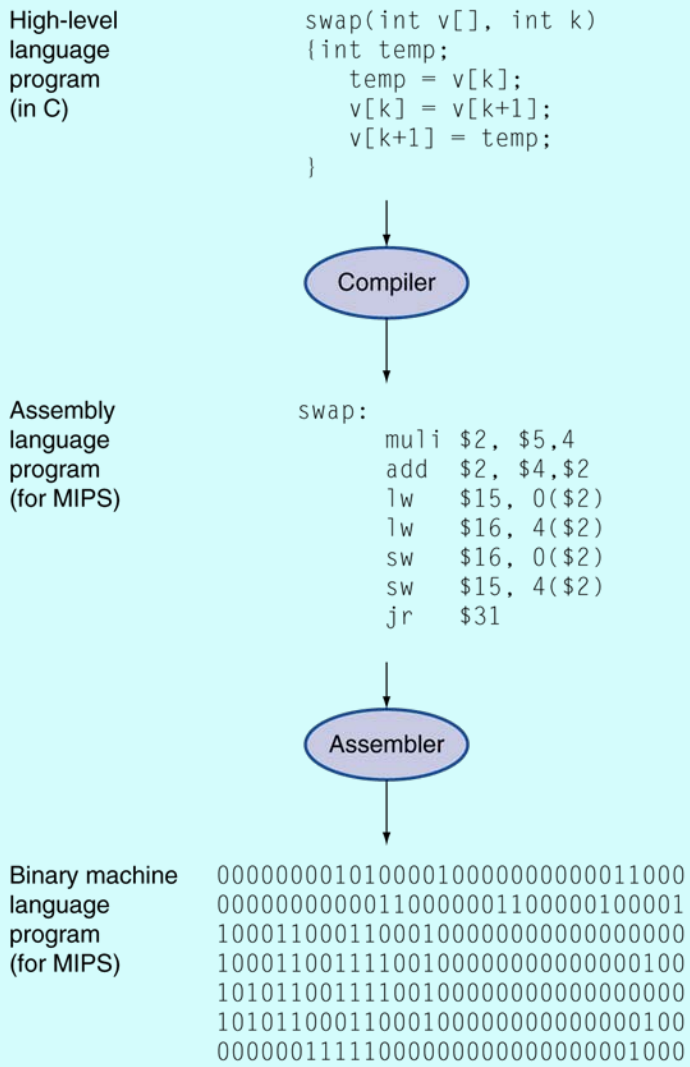
برنامه

- **برنامه‌های کاربردی**
  - به زبان‌های سطح بالا نوشته می‌شوند.
- **برنامه‌های سیستمی**
  - کامپایلر
  - سیستم‌عامل
    - مدیریت حافظه و ذخیره‌سازی
    - اشتراک منابع
    - مدیریت ورودی و خروجی



- **سخت‌افزار**
  - پردازنده، حافظه و ورودی-خروجی

سطوح زبان‌های برنامه‌نویسی



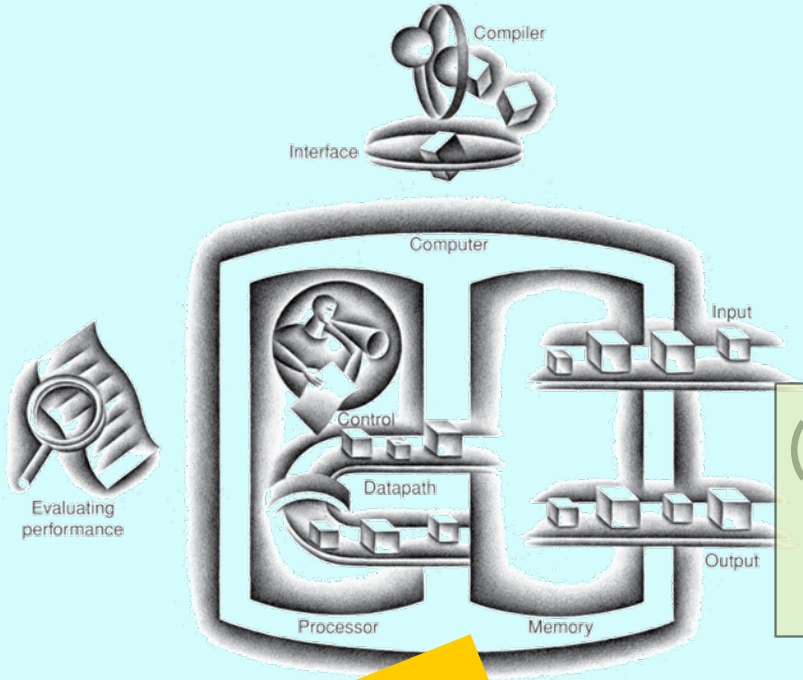
- **زبان‌های سطح بالا**
  - سطحی از تجرید که به زبان طبیعی نزدیک‌تر است
  - کارایی و قابلیت حمل برنامه را افزایش می‌دهد.
- **زبان اسمبلی**
  - نمادهایی که جایگزین زبان ماشین می‌شوند.
- **زبان ماشین**
  - دنباله‌ای از صفر و یک

instruction

دستوری که سخت‌افزار می‌فهمد و می‌تواند آن را اجرا کند

بخش‌های یک کامپیوتر

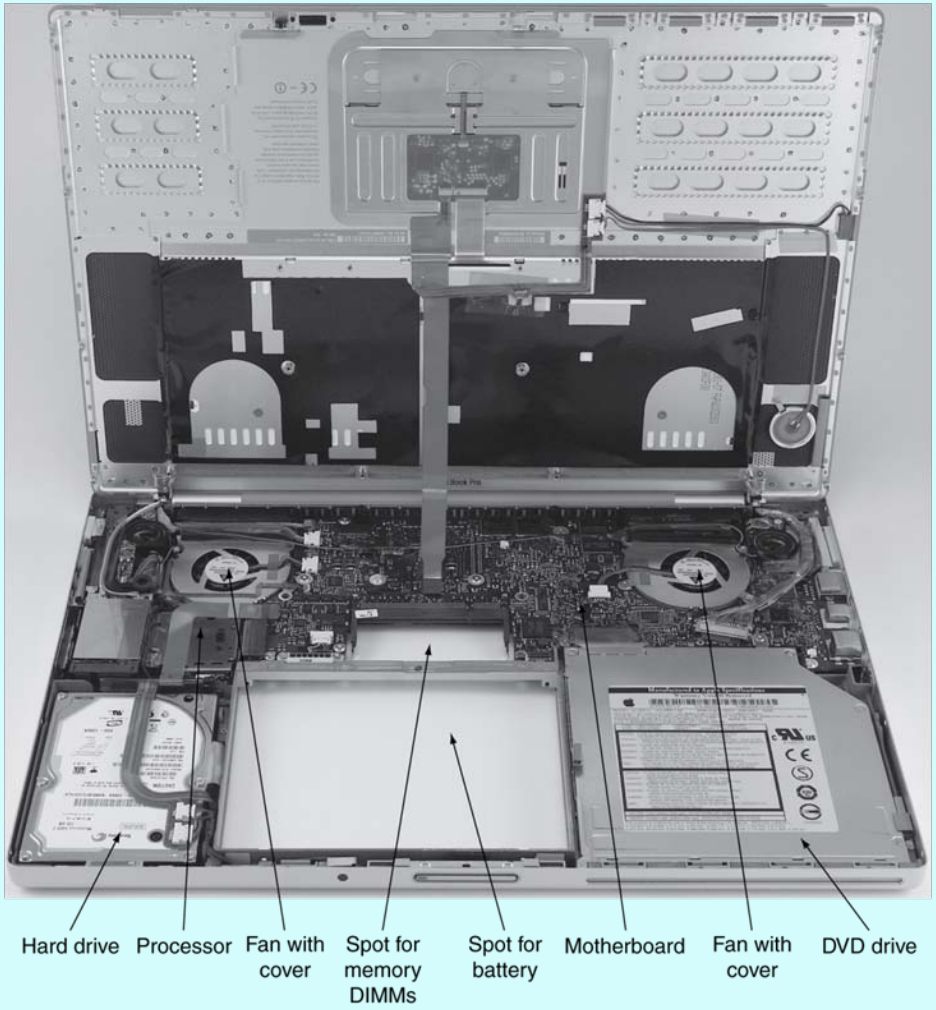
- گذشته از نوع معماری، هر کامپوتر دارای پنج بخش پایه است:
- ورودی
- خروجی
- حافظه
- مسیر گذار داده (datapath)
- کنترل

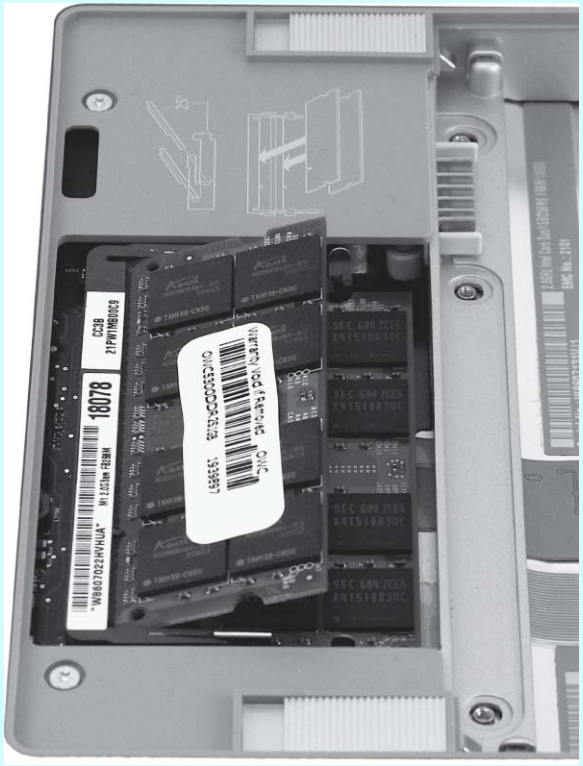


دورنما

پردازنده

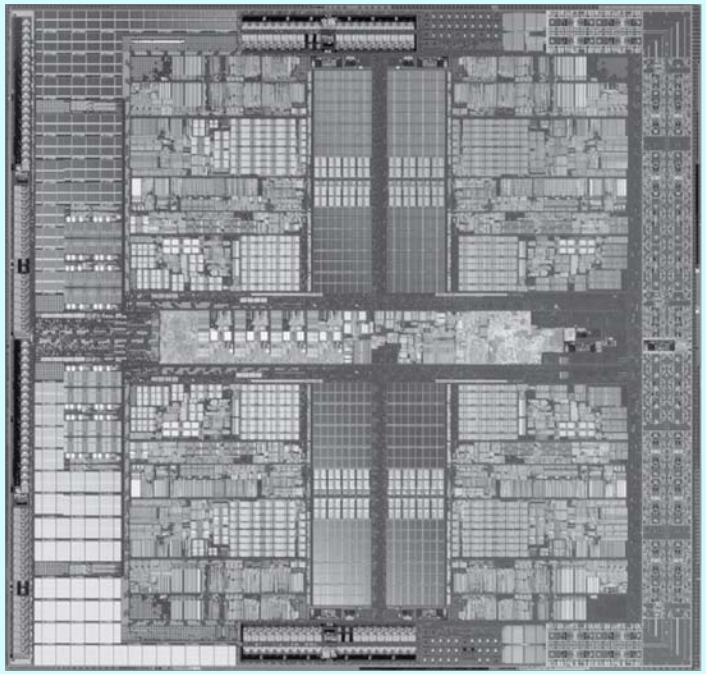
اجزای کامپیوتر

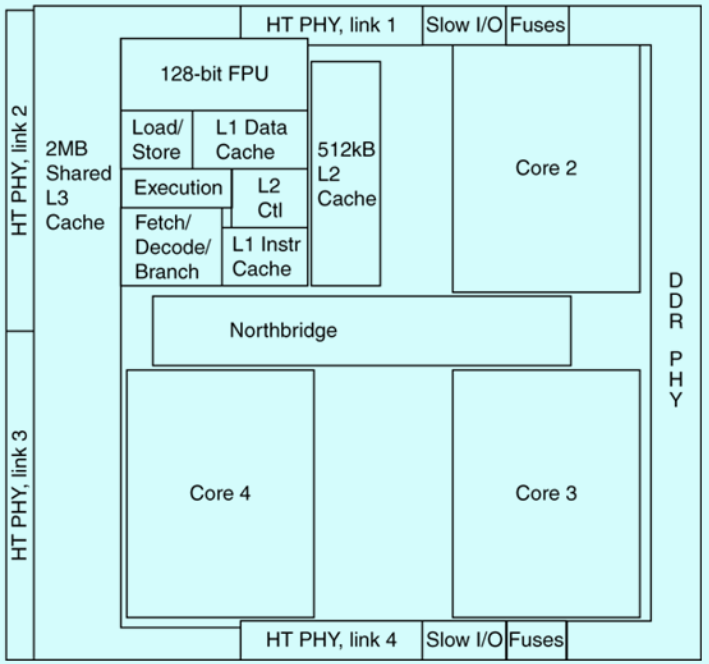




اجزای داخلی پردازنده

- بخش محاسباتی
- واحد کنترل
- حافظه پنهانی (cache memory)





AMD Barcelona: 4 processor cores

تجرید

- «تجرید» به برخورد با سیستم‌های پیچیده کمک می‌کند.
  - جزییات لایه‌های پایین را از دیده‌ها پنهان می‌کند.
- رابط انتزاعی میان سخت‌افزار و نرم‌افزار

Instruction Set Architecture (ISA)

- ISA همراه با رابط سیستم‌عامل

Application binary interface (ABI)

دورنما

یکی از کلیدی ترین واسط های بین سطوح تجرید، معماری مجموعهٔ دستورالعمل (ISA) یا همان واسط بین سخت افزار و نرم افزار سطح پایین است. چنین واسط مجرّدی است که این امکان را فراهم آورده تا پیاده سازی های متعدد با قیمت و کارآیی متفاوت از یک سخت‌افزار خاص وجود داشته باشد و همهٔ آنها بتوانند نرم افزار واحدی را اجرا کنند.

حافظه‌ی فرار

Volatile main memory

- حافظه‌ی اصلی
  - در صورت قطع منبع تغذیه، حافظه پاک می‌شود.

(main memory)

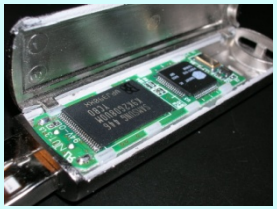
(primary memory)

حافظه‌ی غیرفرار

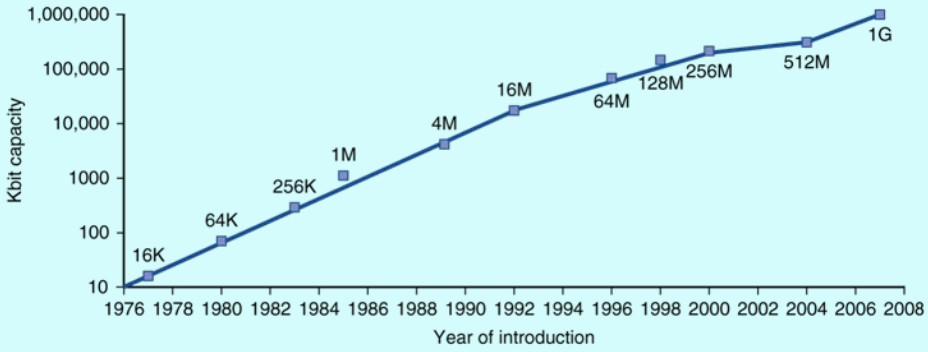
Non-volatile secondary memory

- حافظه‌ی ثانویه
  - دیسک‌های مغناطیسی
  - سی‌دی و دی‌وی‌دی
  - flash memory

(secondary memory)



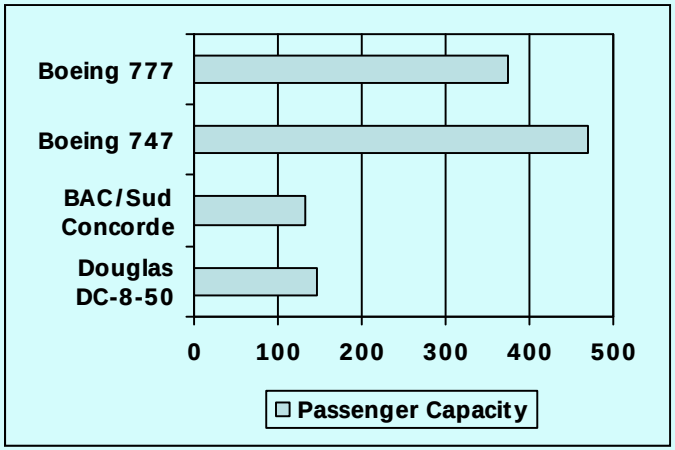
روند به کارگیری فناوری

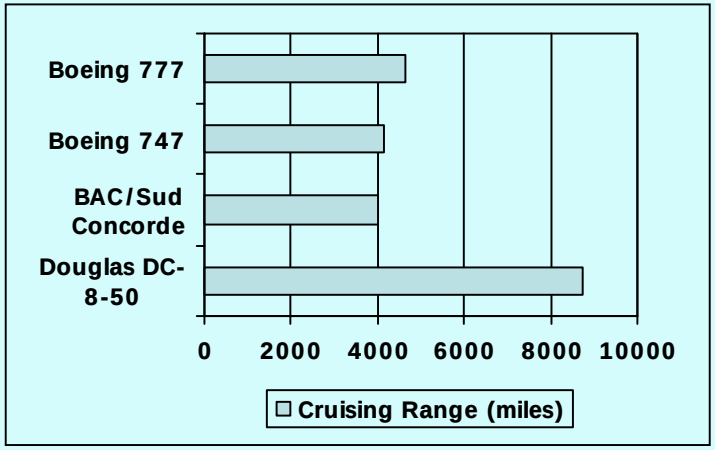


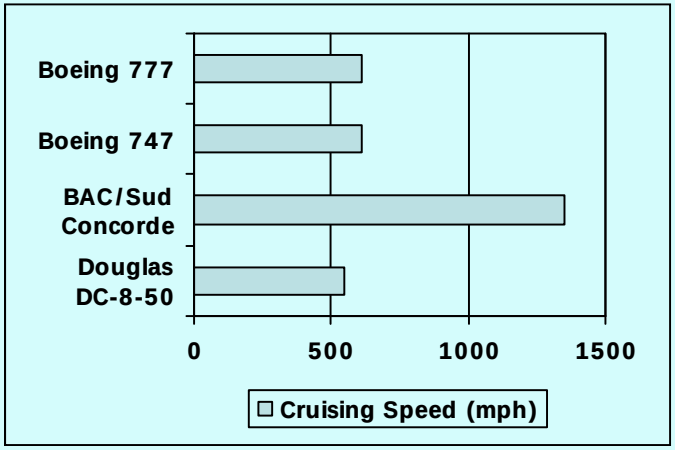
DRAM capacity

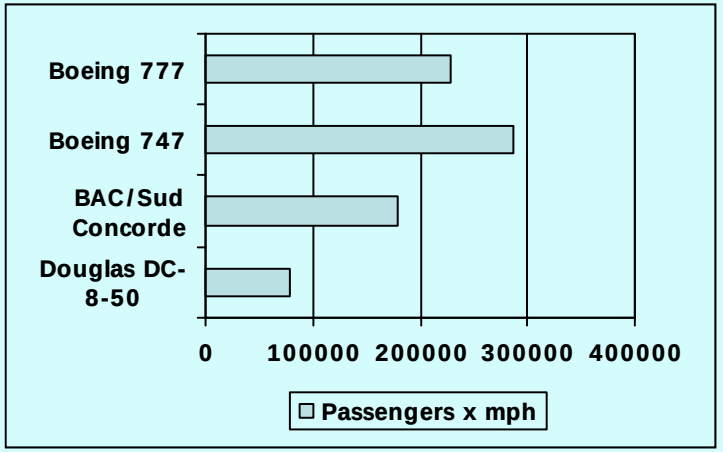
Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2005	Ultra large scale IC	6,200,000,000

کارایی اجرا


Boeing 777
Boeing 747
BAC/Sud Concorde
Douglas DC-8-50
0
100
200
300
400
500
Passenger Capacity


Boeing 777
Boeing 747
BAC/Sud Concorde
Douglas DC-8-50
0
2000
4000
6000
8000
10000
Cruising Range (miles)


Boeing 777
Boeing 747
BAC/Sud Concorde
Douglas DC-8-50
0
500
1000
1500
Cruising Speed (mph)


Boeing 777
Boeing 747
BAC/Sud Concorde
Douglas DC-8-50
0
100000
200000
300000
400000
Passengers x mph

کارایی سیستم در برابر کارایی پردازنده

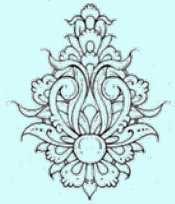
- **زمان پاسخ:**
  - بازه‌ی زمانی که برای تکمیل یک کار صرف می‌شود، شامل پردازش، عملیات I/O و ...
  - بیان‌گر کارایی سیستم می‌باشد.

wall clock time, response time, elapsed time

- **زمان اجرای cpu:**
  - زمانی که صرف پردازش می‌شود. زمان سایر فعالیت‌ها در نظر گرفته نمی‌شود.

cpu execution time

  - شامل user cpu time و system cpu time



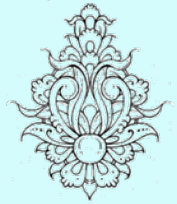
معماری کامپیوتر (۱۳۵۱۰-۱۱-۱۳۳)

جلسه‌ی سوم

دانشگاه شهید بهشتی
دانشکده‌ی مهندسی برق و کامپیوتر
زمستان ۱۳۹۰
احمد محمودی ازناوه

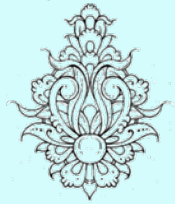
فهرست مطالب

- کارایی
- توان مصرفی
- پردازش موازی
- فرآیند ساخت تراشه



ظرفیت گذردهی در برابر زمان پاسخ

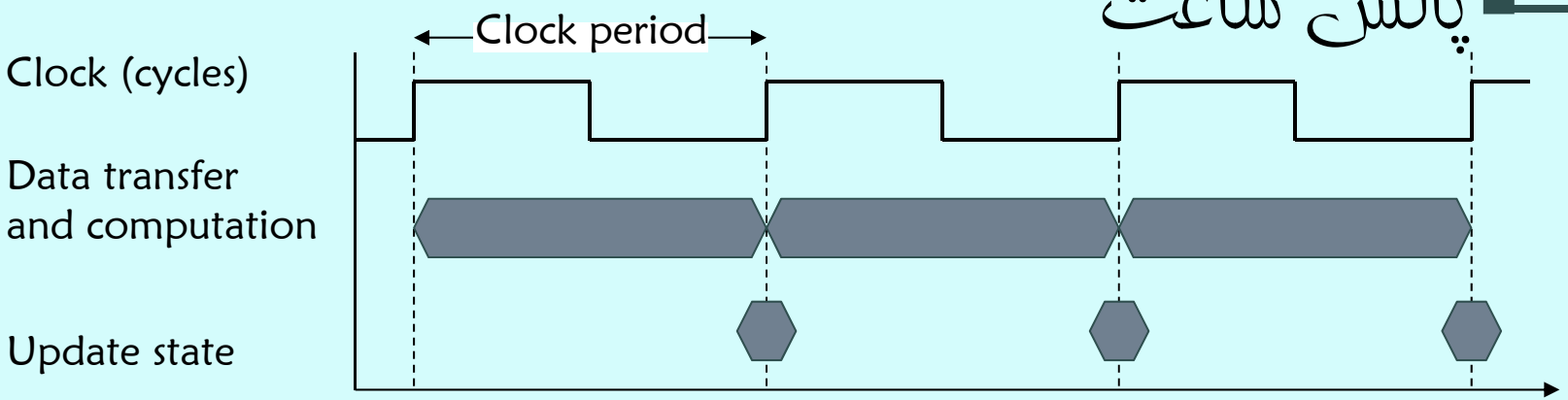
- **زمان پاسخ (اجرا):**
  - زمانی که طول می‌کشد تا یک کامپیوتر کاری را تمام کند.
- **ظرفیت گذردهی (توان عملیاتی):**
  - تعداد کارهایی که در واحد زمان انجام می‌شوند.



- «کارایی» به صورت معکوس زمان پاسخ تعریف می‌شود.
- X از Y، n بار سریع‌تر است اگر:

$$\text{Performance}_X / \text{Performance}_Y = \text{Execution time}_Y / \text{Execution time}_X = n$$

پالس ساعت

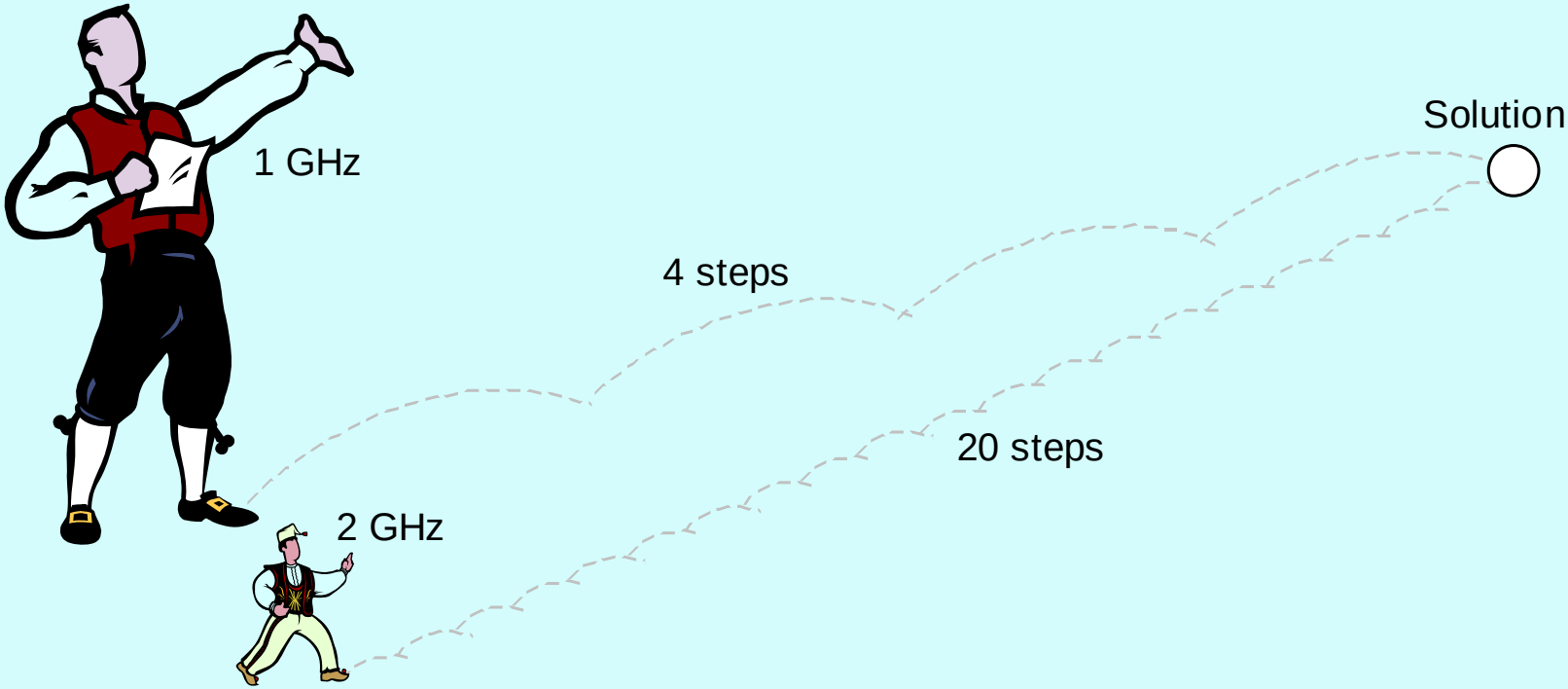


$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$
$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- برای افزایش سرعت cpu
  - فرکانس پالس ساعت را افزایش داد.
  - تعداد پالس به ازای هر دستورالعمل را کاهش داد.

سرعت کامپیوترها

بالا بودن ساعت سریع‌تر به معنای سرعت اجرای بیشتر نیست



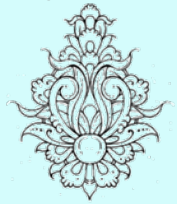
مثال

- کامپیوتر A با 2GHz clock به ده ثانیه برای اجرای برنامه‌ای خاص نیاز دارد. می‌خواهیم کامپیوتر B را به گونه‌ای طراحی کنیم که زمان اجرا را به شش ثانیه تقلیل دهد. با این فرض که در صورت افزایش سرعت پالس ساعت مجبور به تغییر طراحی خواهیم شد به طوری که تعداد سیکل‌های لازم برای اجرای دستورات ۱,۲ برابر می‌شود. فرکانس کامپیوتر B را حساب کنید؟

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$
$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$



کارایی دستورالعمل

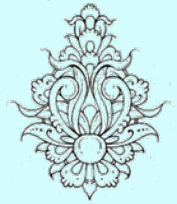
clock cycle per instruction

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- تعداد متوسط دستورالعمل توسط
  - برنامه، ISA و کامپایلر تعیین می‌شود
- تعداد سکیل به ازای هر دستورالعمل توسط سخت‌افزار تعیین می‌شود.



مثالی از CPI

- دو کامپیوتر را در نظر بگیرید:
  - کامپیوتر A با پالس ساعت 250ps و CPI=2 برای برنامه‌ای خاص
  - کامپیوتر B با پالس ساعت 500ps و CPI=1.2 برای همان برنامه
- کدام سریع‌ترند؟

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$
$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$
$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$
$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$
$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2$$

کامپیوتر A سریع‌تر است

این قدر

دستورات با سیکل‌های متفاوت

- در صورتی که یک cpu دارای دستورات با سیکل‌های متفاوت باشد، تعداد پالس ساعت برای اجرای یک برنامه به صورت زیر محاسبه می‌شود.

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

- و CPI کلی با میانگین وزن‌دهی شده به دست می‌آید

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count}} \right)$$

Relative frequency

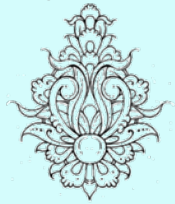
مثال

- برای اجرای یک تکه کد، دو شیوه قابل به کارگیری است، کدامیک سریع‌تر خواهد بود؟

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

- Sequence 1: IC = 5
  - Clock Cycles
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0

- Sequence 2: IC = 6
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5

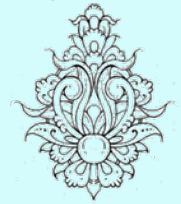


جمع‌بندی در مورد کارایی

دورنما

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- کارایی یک برنامه به عوامل زیر بستگی دارد:
  - الگوریتم IC CPI
  - زبان برنامه‌نویسی IC CPI
  - کامپایلر IC CPI
  - معماری مجموعه‌ی دستورالعمل‌ها (ISA) IC CPI T



تمرین کلاسی

Op	Freq	CPI_i	Freq x CPI_i			
ALU	50%	1	.5	.5	.5	.25
Load	20%	5	1.0	.4	1.0	1.0
Store	10%	3	.3	.3	.3	.3
Branch	20%	2	.4	.4	.2	.4
			$\Sigma =$ 2.2	1.6	2.0	1.95

- چنانچه که با به‌کارگیری که حافظه‌ی نهان بهتر دستورات خواندن به میانگین دو سیکل برسند، سرعت تا چه‌میزان بهبود می‌یابد؟

CPU time new = 1.6 x IC x CC so 2.2/1.6 means 37.5% faster

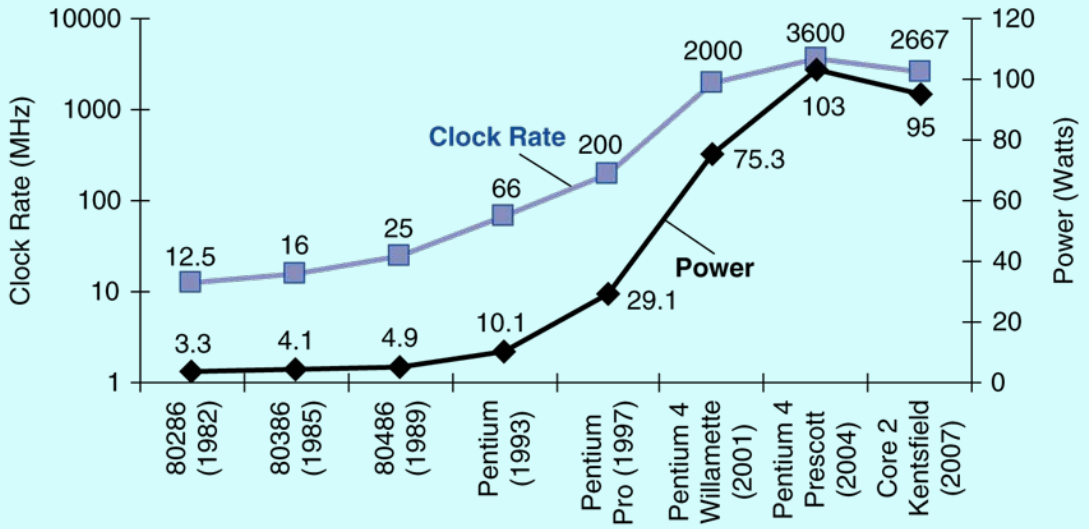
- در صورتی که با پیش‌بینی دستورات پرش میانگین اجرای دستورات پرش به نصف کاهش یابد، سرعت چه تغییری خواهد کرد؟

CPU time new = 2.0 x IC x CC so 2.2/2.0 means 10% faster

- در صورتی که دو دستور همزمان در واحد محاسباتی اجرا شوند چه؟

CPU time new = 1.95 x IC x CC so 2.2/1.95 means 12.8% faster

رویه تغییرات توان مصرفی



- در تکنولوژی CMOS

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency Switched}$$

×30

5V → 1V

×1000

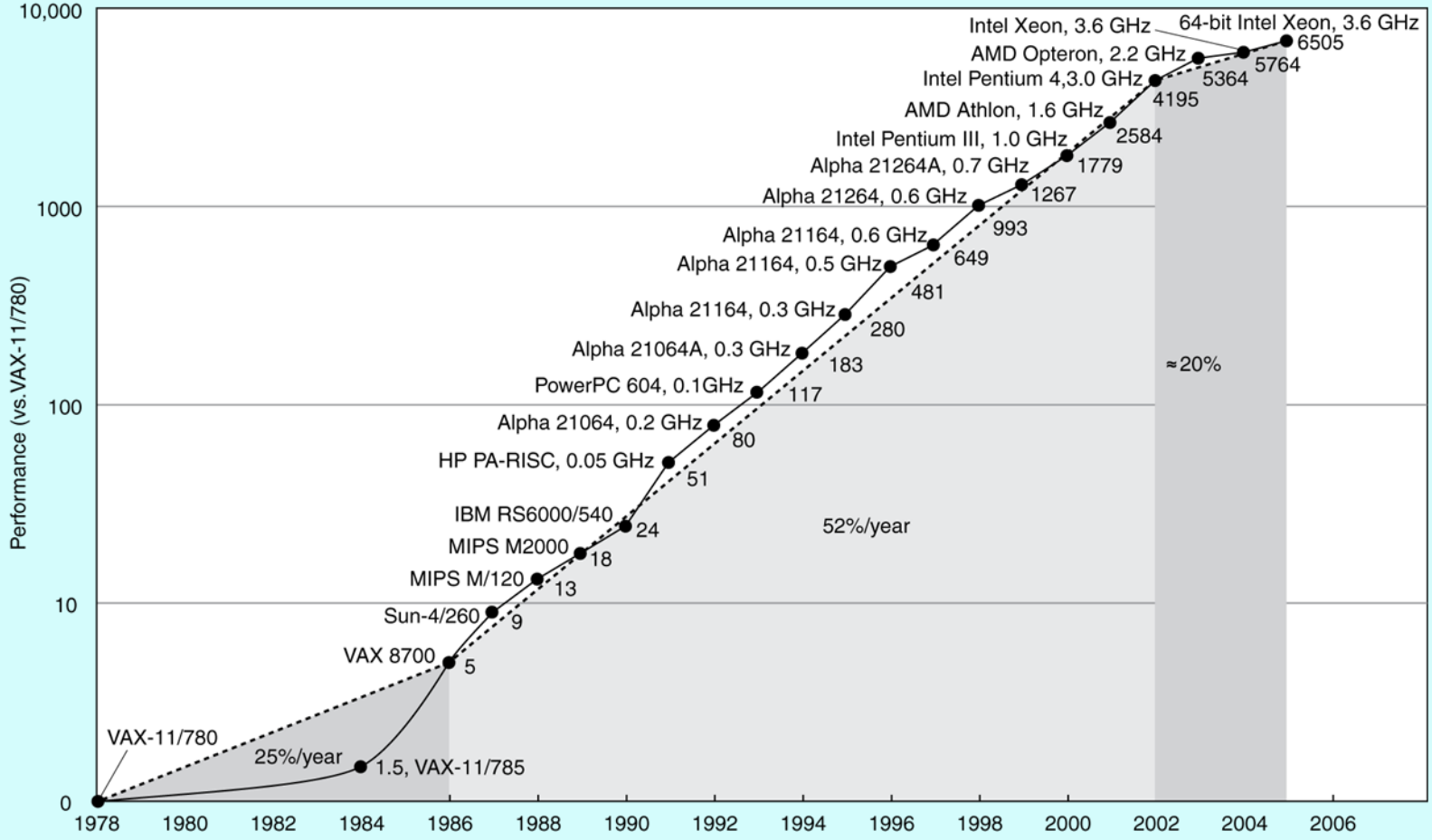
مثال- توان نسبی

- در صورتی که در یک پردازنده طراحی به گونه‌ای تغییر کند که ولتاژ، فرکانس پالس ساعت و بار خازنی هر یک پانزده‌درصد کاسته شوند، توان مصرفی چه تغییری می‌کند؟

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

- بیش از این نمی‌توان ولتاژ را کاهش داد.
- روش‌های متفاوتی برای خنک کردن cpu به کار گرفته شده است.

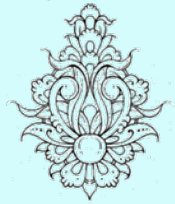
حرکت به سوی پردازش موازی



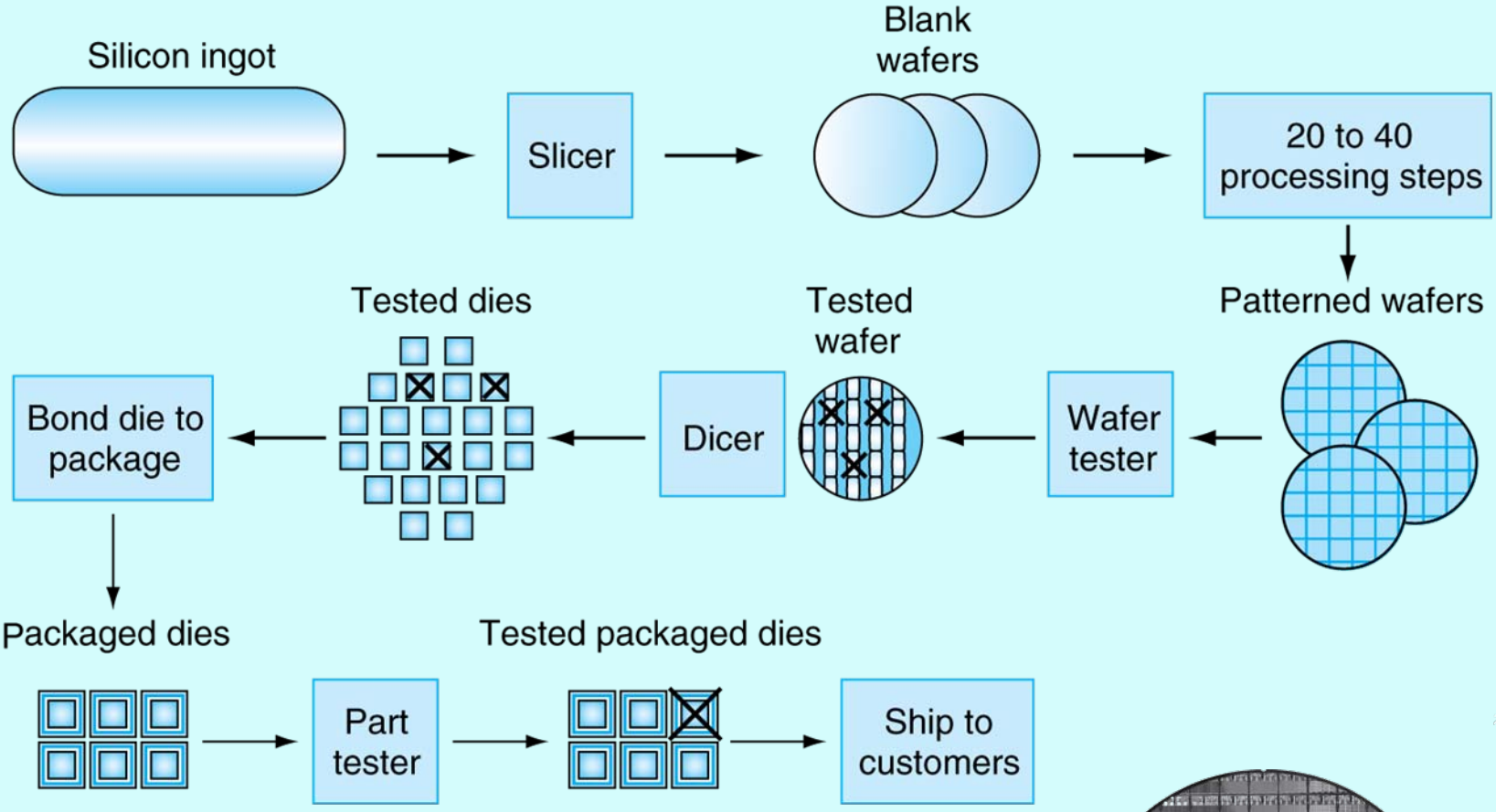
چندپردازشی

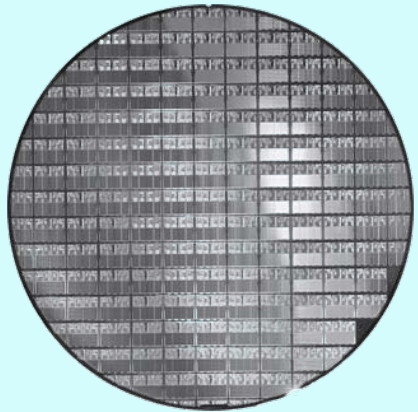
multiprocessing

- پردازنده‌های چند هسته‌ای
  - بیش از یک پردازنده در یک تراشه
- نیاز به نوشتن برنامه‌های که قابلیت اجرا شدن به صورت موازی را دارند.
- دشواری‌های برنامه‌نویسی به صورت موازی
  - باید کارا باشد، در این حالت تنها پاسخ درست مد نظر نیست
  - توزیع بار (Load balancing)
  - کاهش ارتباط و نیاز به هماهنگ کردن



ساخت تراشه



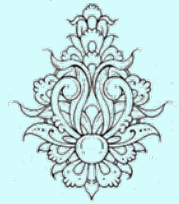


300mm wafer, 117 chips, 90nm technology

محک آزمون برای بررسی کارایی

- بار کاری (workload):
  - مجموعه برنامه‌ای که بر روی یک کامپیوتر اجرا می‌شود.
- محک (benchmarks):
  - مجموعه برنامه‌هایی که برای ارزیابی یک سیستم کامپیوتری مورد استفاده قرار می‌گیرد.
- SPEC برای ارزیابی کارایی یک سری محک عرضه می‌کند.
  - SPEC CPU2006، عملیات ورودی خروجی را در نظر نمی‌گیرد، و در نتیجه بر روی کارایی پردازنده تمرکز دارد.

SPEC: (System Performance Evaluation Cooperative)



AMD Opteron X4

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

Name	Description	IC×10^9	CPI	Tc (ns)	Exec time	Ref time	SPECratio
perl	Interpreted string processing	2,118	0.75	0.40	637	9,777	15.3
bzip2	Block-sorting compression	2,389	0.85	0.40	817	9,650	11.8
gcc	GNU C Compiler	1,050	1.72	0.47	24	8,050	11.1
mcf	Combinatorial optimization	336	10.00	0.40	1,345	9,120	6.8
go	Go game (AI)	1,658	1.09	0.40	721	10,490	14.6
hmmer	Search gene sequence	2,783	0.80	0.40	890	9,330	10.5
sjeng	Chess game (AI)	2,176	0.96	0.48	37	12,100	14.5
libquantum	Quantum computer simulation	1,623	1.61	0.40	1,047	20,720	19.8
h264avc	Video compression	3,102	0.80	0.40	993	22,130	22.3
omnetpp	Discrete event simulation	587	2.94	0.40	690	6,250	9.1
astar	Games/path finding	1,082	1.79	0.40	773	7,020	9.1
xalancbmk	XML parsing	1,058	2.70	0.40	1,143	6,900	6.0
							11.7

High cache miss rates

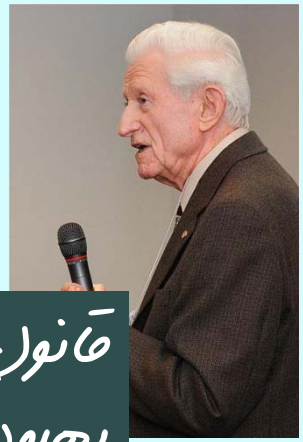
قانون Amdahl

- 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 پنج بر ابر شود؟

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

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

Amdahl's law



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

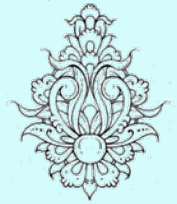
قانون Amdhal

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.
- برای یک برنامه با تعداد دستورات بیشتر اما سریع‌تر معیار خوبی نخواهد بود.

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

معماری کامپیوتر (۱۳۹۰-۱۱-۱۳)
جلسه‌ی چهارم

دانشگاه شهید بهشتی
دانشکده‌ی مهندسی برق و کامپیوتر
زمستان ۱۳۹۰
احمد محمودی ازناوه

فهرست مطالب

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

قانون Amdahl

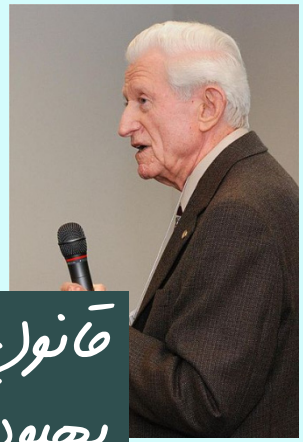
- برنامه‌ای را در نظر بگیرید که زمان پاسخ آن ۱۰۰ ثانیه است، ۸۰ ثانیه‌ی مربوط به عملیات ضرب می‌باشد، سرعت عملیات ضرب چند برابر شود تا سرعت برنامه پنج بر ابر شود؟

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

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

Amdahl's law

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



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

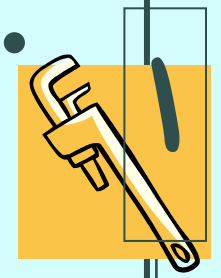
MIPS: million instructions per second

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

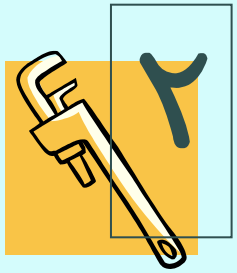
- در ISAهای متفاوت، توانایی‌ها فرق می‌کند.
- MIPS در یک کامپیوتر خاص برای برنامه‌های متفاوت، مقادیر متفاوتی خواهد داشت.

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

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.

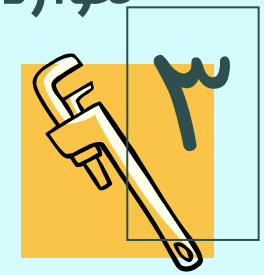
spilling register

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

- **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

- 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.

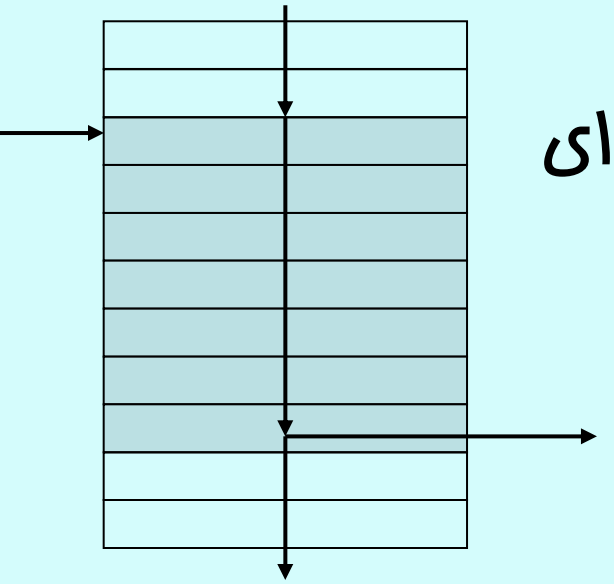
Constant

- 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 می‌تواند اجرای بلوک‌های پایه را تسریع بخشند.



Pseudo instructions

طراحی دستورات شرطی

- دستورهای blt و bge
  - سخت‌افزار مدارهای > و یا ≤ نسبت به = یا ≠ 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.

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 ⇒ $t0 = 1

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

+4,294,967,295 > +1 ⇒ $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

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

معماری کامپیوتر (۱۳۹۰-۱۱-۱۳)
جلسه‌ی پنجم

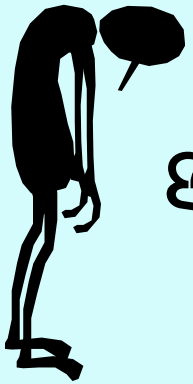
دانشگاه شهید بهشتی
دانشکده‌ی مهندسی برق و کامپیوتر
زمستان ۱۳۹۰
احمد محمودی ازناوه

فهرست مطالب

- شکل‌های مختلف دستور
- شیوه‌های آدرس‌دهی

شیوه‌های آدرس‌دهی

- حالتی را تصور کنید که به یک داده‌ی ثابت سی‌ودو بیتی نیاز داشته باشیم!
- یا بخواهیم به یک آدرس سی‌ودو بیتی دسترسی پیدا کنیم!
- چه راه حلی پیشنهاد می‌دهید؟
- برای نمونه بخواهیم عدد ۸۱۹۴۳۰۴ را در جمع ثابت استفاده کنیم.



0000 0000 0111 1101 0000 1001 0000 0000

ثابت‌های سی‌ودوبیتی

- بیشتر ثابت‌هایی که مورد استفاده قرار می‌گیرند، کوچک هستند و در شانزده بیت می‌گنجند.
- به ندرت مواردی پیش می‌آید که به ثابت‌هایی بزرگ نیاز داشته باشیم.
- در این موارد می‌توان داده‌ی مورد نیاز را در ثبات بارگذاری نمود و از دستورات که دارای عملوند ثبات هستند، استفاده کرد.
- برای این منظور دستور زیر پیش‌بینی شده است:

```
lui rt, constant
```

load upper immediate

این دستور، مقدار ثابت را در شانزده بیت پرارزش قرار می‌دهد و بخش کم ارزش را صفر می‌کند.

ثابت‌های سی‌ودوبیتی (ادامه...)

```
lui $s0, 61
```

0000 0000 0111 1101	0000 0000 0000 0000

برای قرار دادن بخش کم ارزش چه پیشنهادی دارید؟

نظر شما درباره‌ی دستور addi چیست؟

```
ori $s0, $s0, 2304
```

0000 0000 0111 1101	0000 1001 0000 0000

The machine language version of `lui $t0, 255` `# $t0 is register 8:`

001111	00000	01000	0000 0000 1111 1111

Contents of register `$t0` after executing `lui $t0, 255`:

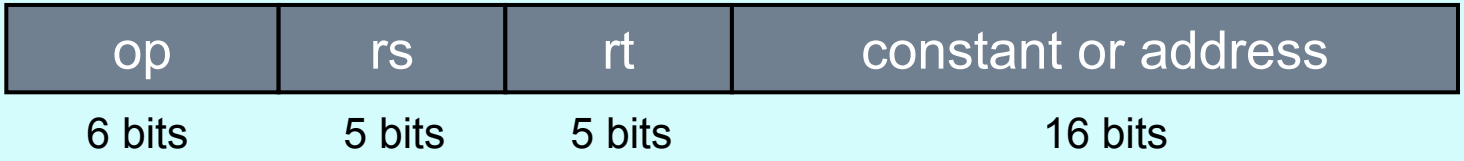
0000 0000 1111 1111	0000 0000 0000 0000

ثابت‌های سی‌ودوبیتی (ادامه...)

- ثابت‌های بزرگ توسط کامپایلر و یا اسمبلر به اعداد کوچک‌تر شکسته شده و در یک ثبات جمع‌آوری می‌شوند.
- کوچک بودن اندازه‌ی فیلد داده‌های ثابت، برای آدرس‌ها و به ویژه در دستورات lw و st ایجاد دشورای می‌کند.
- برای جمع‌آوری آدرس، اسمبلر به یک ثبات موقت نیاز دارد. ثبات $at برای این منظور در نظر گرفته می‌شود.

آدرس‌دهی در دستورات پرش شرطی

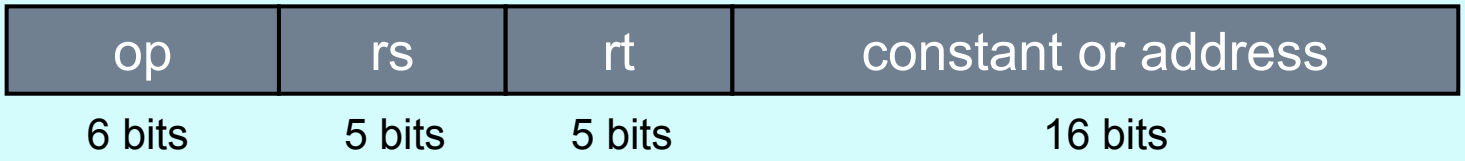
- دستورات پرش شرطی از نوع I هستند.
- آدرس شانزده‌بیتی میزان پرش را محدود می‌کند. در این حالت، طول برنامه برای کاربردهای امروزی معقول نیست.
- معمولاً محدوده‌ی مقصد پرش نزدیک محل دستور پرش است.



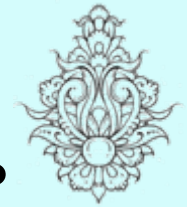
PC = a register + branch address

آدرس‌دهی نسبی بر مبنای PC

- Target address = PC + offset × 4



- بدین‌ترتیب، طول برنامه می‌تواند تا چهار گیگابایت افزایش یابد.
- هنگام اجرای هر دستور، PC به آدرس بعدی اشاره می‌کند.
- آدرسی که در دستورات پرش استفاده می‌شود، آدرس کلمه است، نه آدرس بایت

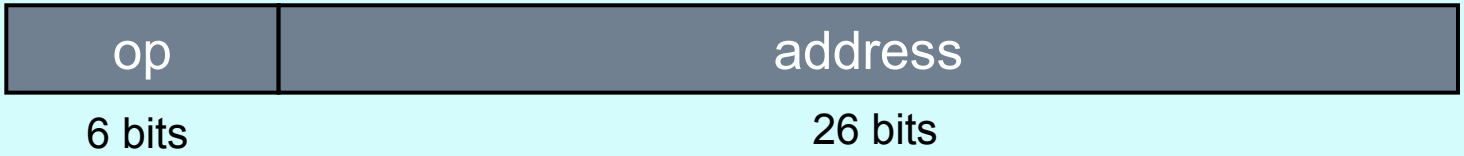


آدرس‌دهی شبه‌مستقیم

```
j L1
```

```
jal ProcedureLabel
```

- این دستورات از کدام نوع هستند؟
  - نوع I؟
- آخرین نوع دستورها در MIPS، نوع J می‌باشد:




```
Target address = PC31…28 : (address × 4)
```

مثال

```
while (save[i] == k) i += 1;
```

```
Loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j    Loop
Exit: …
```

80000	0	0	19	9	4	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	19	1		
80020	2	20000				
80024						

پیاده‌سازی پرش شرطی

- در صورتی که نیاز به پرش شرطی داشتیم که فاصله‌ی نسبی آن با آدرس فعلی به بیش از شانزده بیت نیاز داشت،چه باید کرد؟

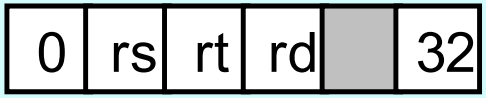
```
beq $s0,$s1, L1
```

مثلاً آدرس L1 خیلی دور است!

```
    bne $s0,$s1, L2
    j L1
L2:     …
```

البته زحمت انجام این کار به عهده‌ی اسمبلر است!

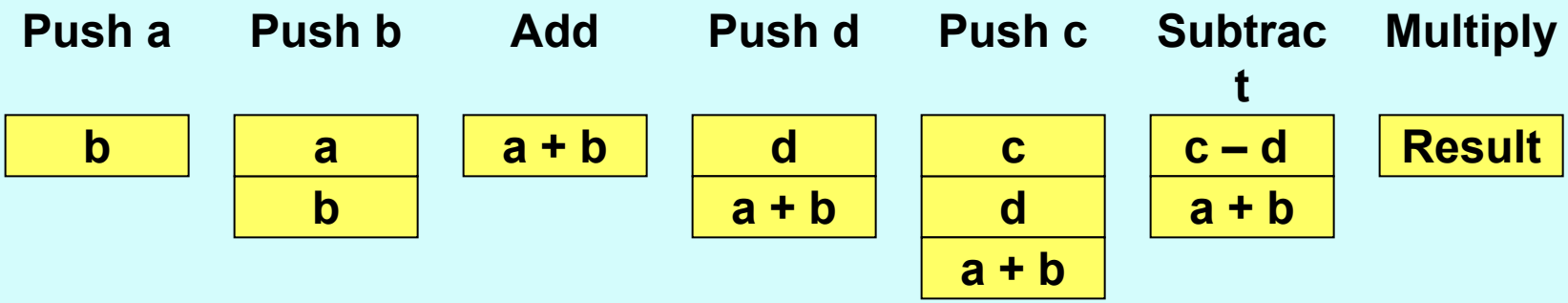
شکل‌های مختلف دستور

Category	Format	Opcode
0-address		`syscall`
1-address		`j`
2-address		`mult`
3-address		`add`

مثالی دستورات بدون عملوند

مثال: ارزیابی عبارت

$(a + b) \times (c - d)$



Reverse Polish string: b a + d c – ×

postfix notation

مثالی معماری تک عملوند

مثال: ارزیابی عبارت

$$(a + b) \times (c - d)$$

انباره ثباتی است که یکی از عملوندها را در خود نگه می‌دارد و همیشه پاسخ در آن ذخیره می‌شود.

```
Load      a
add       b
Store     t
load      c
subtract  d
multiply  t
```

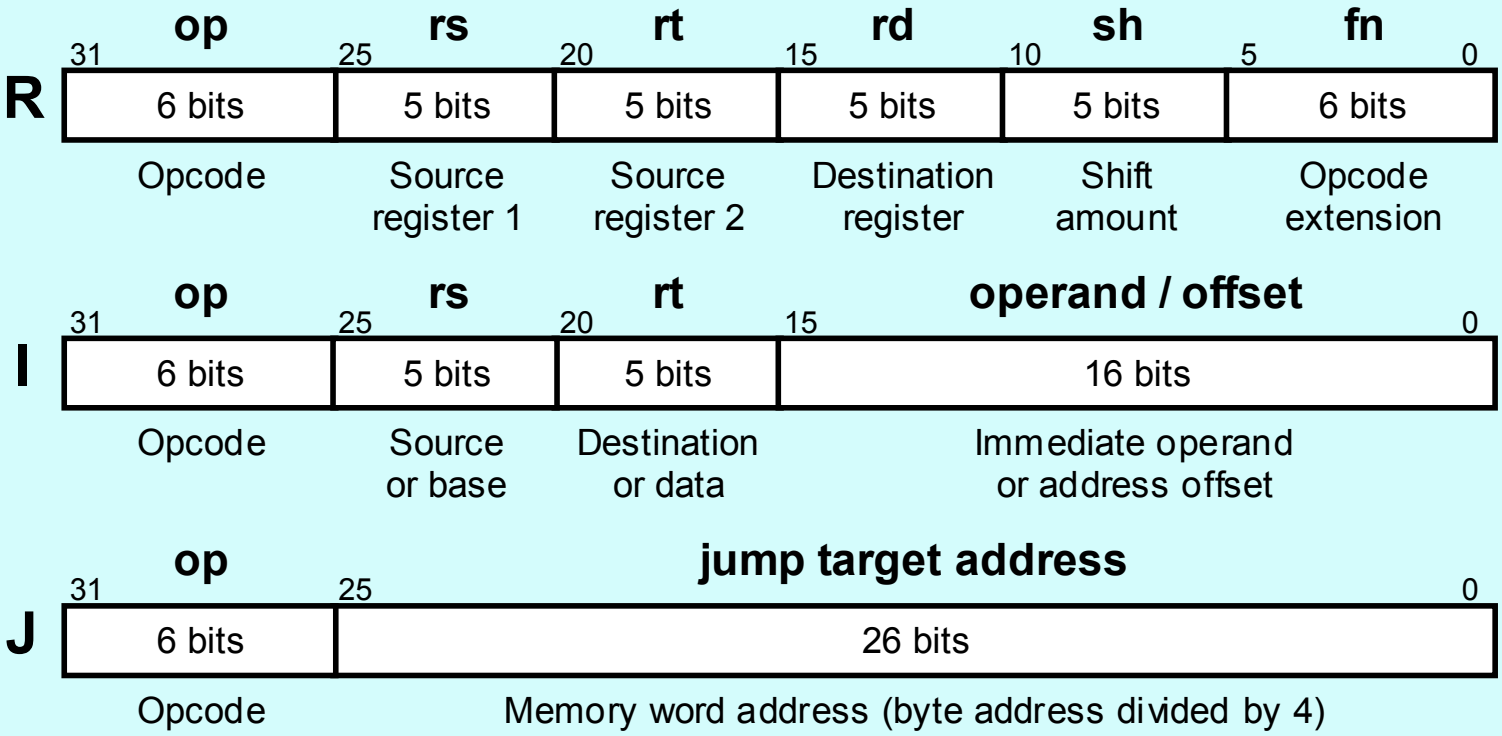
مثالی معماری دستورالعمل با دو عملوند

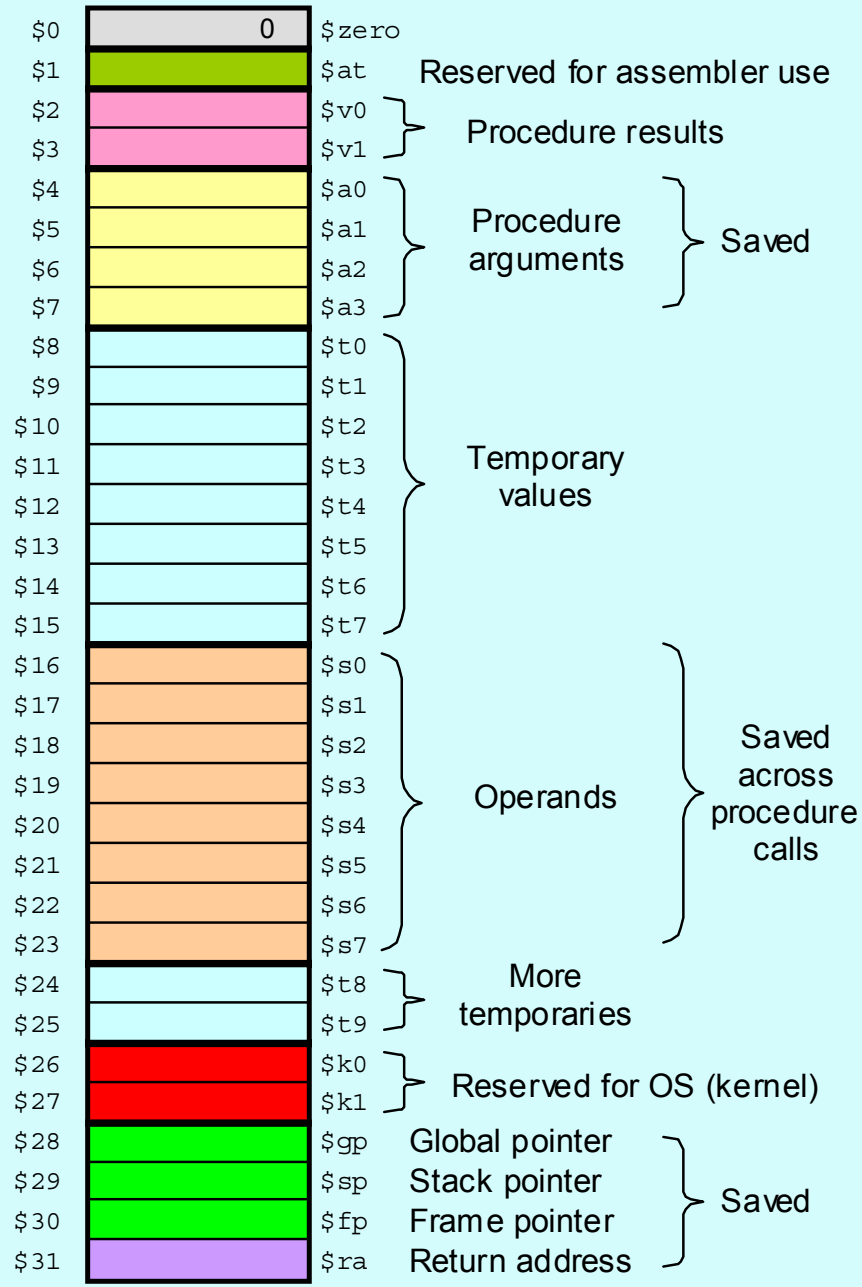
مثال: ارزیابی عبارت

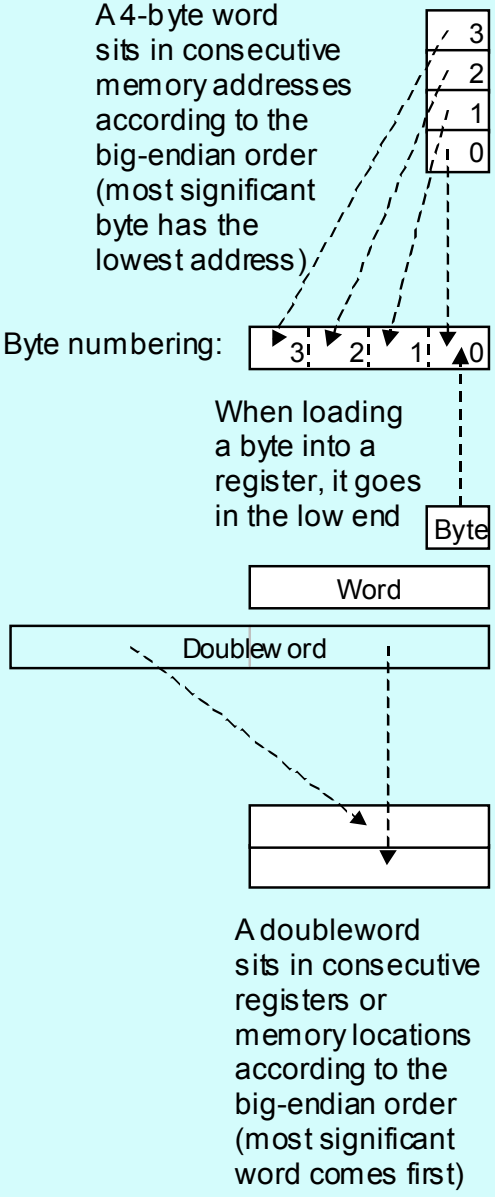
$$(a + b) \times (c - d)$$

```
load      $1,a
add       $1,b
load      $2,c
subtract  $2,d
multiply  $1,$2
```

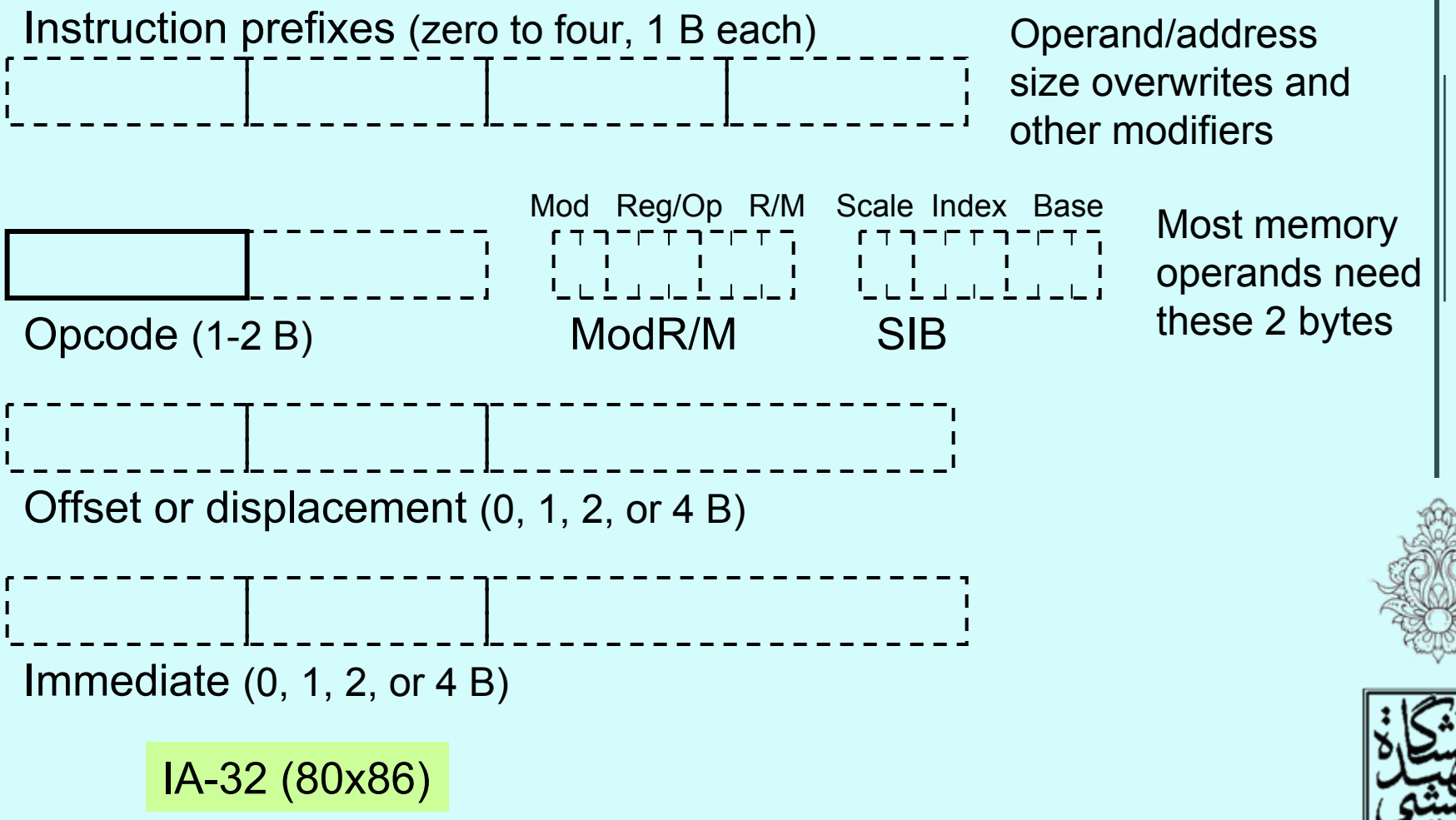
قالب‌های دستور در یک نگاه



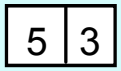
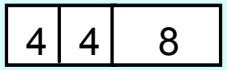
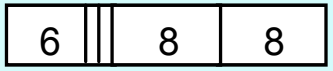
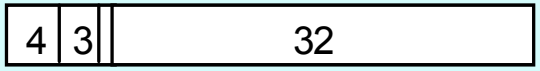

$0 0 $zero
$1 $at Reserved for assembler use
$2 $v0
$3 $v1 Procedure results
$4 $a0
$5 $a1
$6 $a2
$7 $a3 Procedure arguments
Saved
$8 $t0
$9 $t1
$10 $t2
$11 $t3
$12 $t4
$13 $t5
$14 $t6
$15 $t7 Temporary values
$16 $s0
$17 $s1
$18 $s2
$19 $s3
$20 $s4
$21 $s5
$22 $s6
$23 $s7 Operands
Saved across procedure calls
$24 $t8
$25 $t9 More temporaries
$26 $k0
$27 $k1 Reserved for OS (kernel)
$28 $gp Global pointer
$29 $sp Stack pointer
$30 $fp Frame pointer
$31 $ra Return address
Saved


A 4-byte word sits in consecutive memory addresses according to the big-endian order (most significant byte has the lowest address)
3
2
1
0
Byte numbering:
3 2 1 0
When loading a byte into a register, it goes in the low end
Byte
Word
Doubleword
A doubleword sits in consecutive registers or memory locations according to the big-endian order (most significant word comes first)

مثالی ازیک ساختار پیچیده

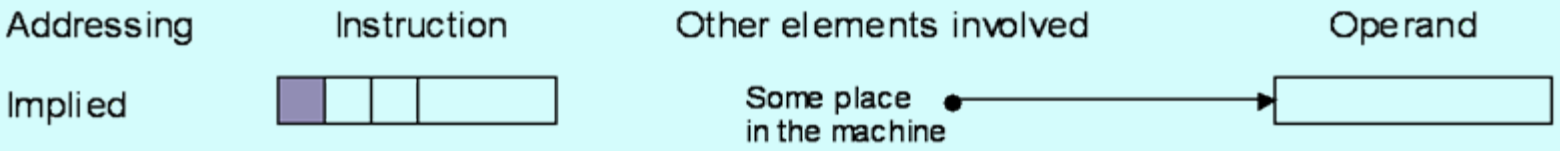


مثال

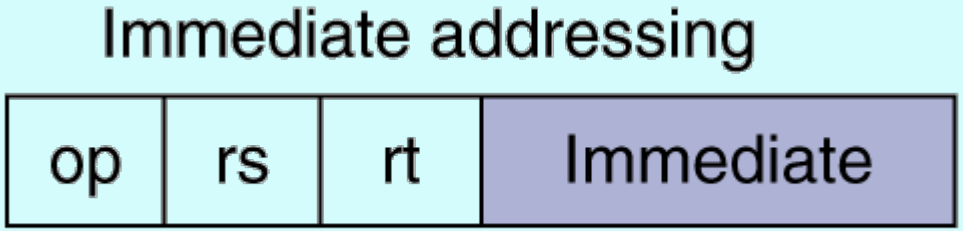
Type	Format (field widths shown)	Opcode	Description of operand(s)
1-byte		PUSH	3-bit register specification
2-byte		JE	4-bit condition, 8-bit jump offset
3-byte		MOV	8-bit register/mode, 8-bit offset
4-byte		XOR	8-bit register/mode, 8-bit base/index, 8-bit offset
5-byte		ADD	3-bit register spec, 32-bit immediate
6-byte		TEST	8-bit register/mode, 32-bit immediate

implied addressing

- **نشانی‌دهی ضمنی:**
  - عملوند آشکارا گفته نمی‌شود، هنگامی اجرا به صورت ضمنی برای پردازنده مشخص می‌شود.



- **نشانی‌دهی بی‌واسطه:**
  - داده‌ی ثابت به صورت مستقیم مورد استفاده قرار می‌گیرد.

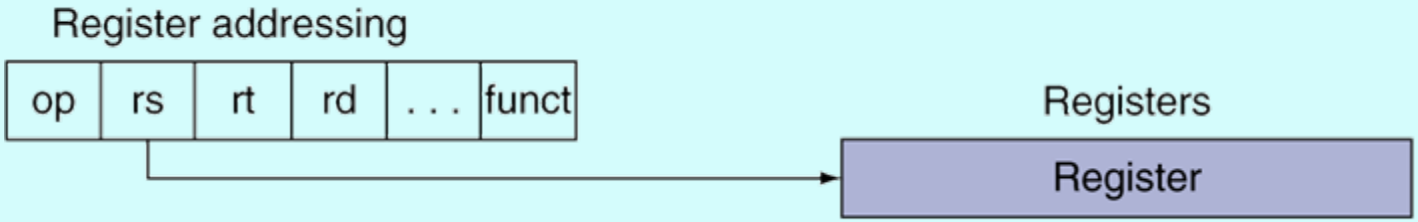


Operand=A

Immediate/literal/constant addressing

انواع نشانی‌دهی (ادامه...)

- **آدرس‌دهی ثبات:**
  - **در فیلد آدرس شماره ثبات مورد نظر آورده می‌شود.**



Register addressing

با توجه به محدودیت ثبات، فیلد آدرس کوچک خواهد بود
روش سریعی می‌باشد که پیاده‌سازی راحتی هم دارد

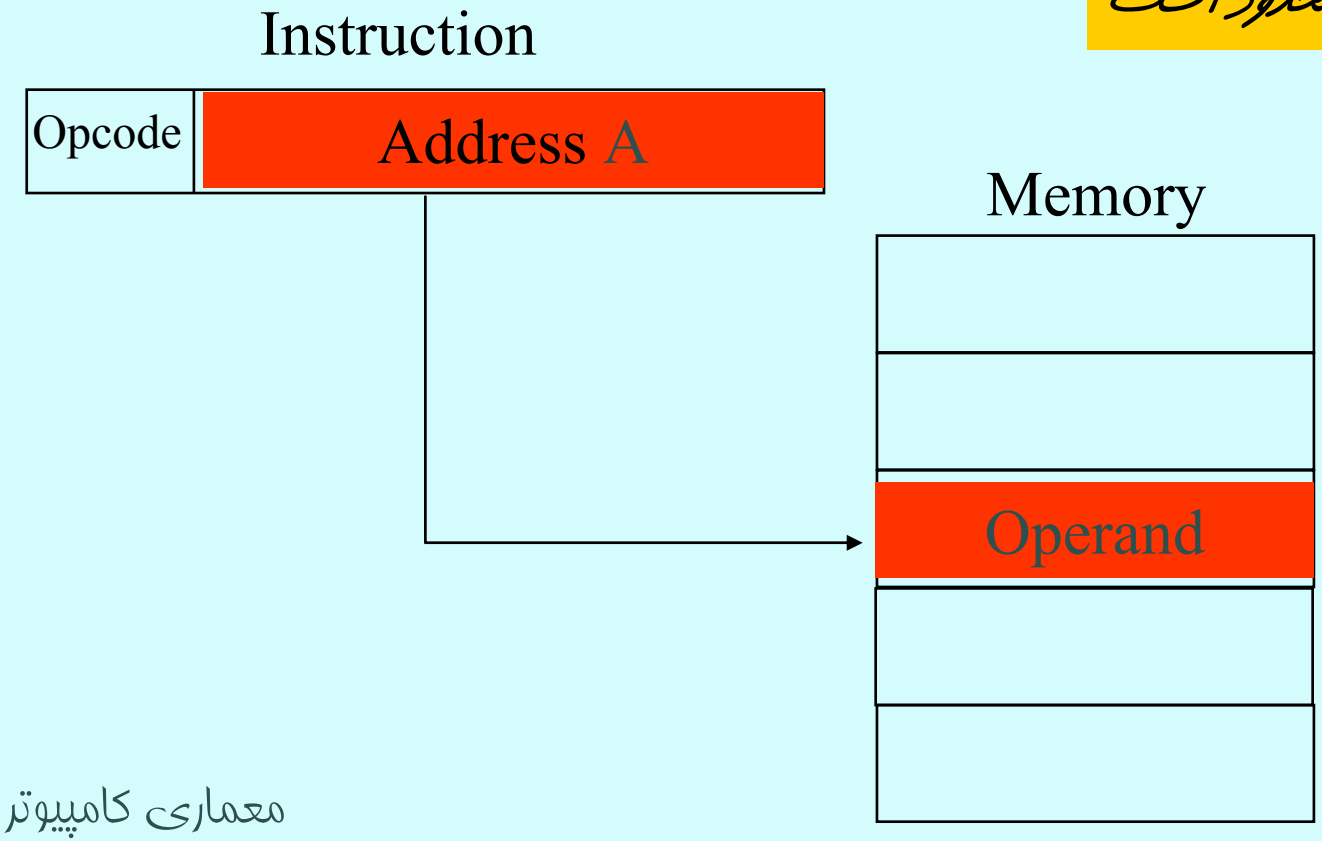
$EA = R$

Effective address

انواع نشانی‌دهی (ادامه...)

- **آدرس‌دهی مستقیم:**
  - در فیلد آدرس آدرس خانه‌ی حافظه آورده می‌شود.

فضای آدرس‌دهی محدود است

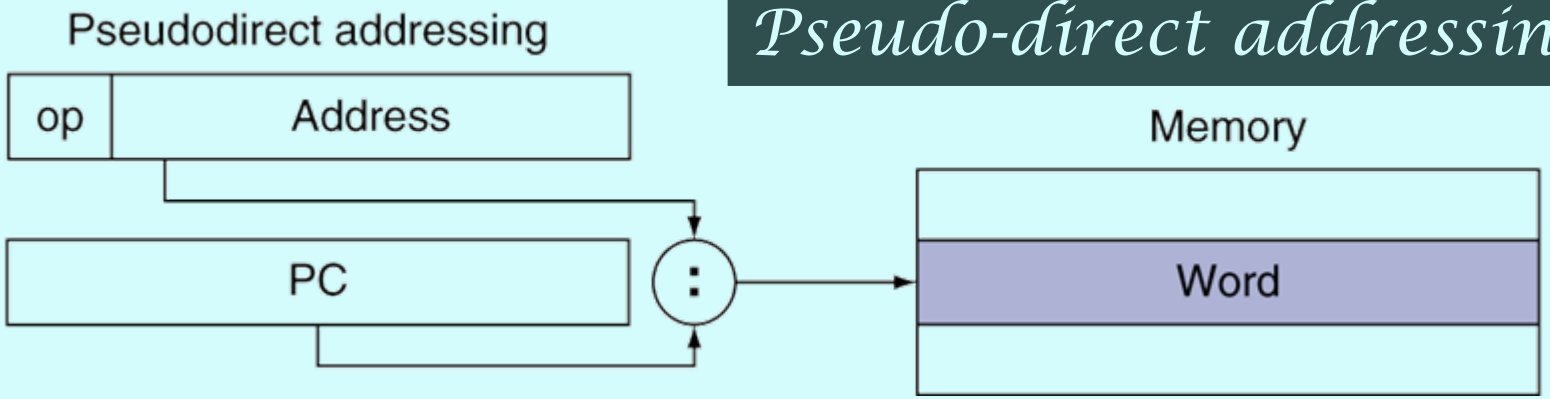


EA = A

انواع نشانی‌دهی (ادامه...)

- **آدرس‌دهی شبه مستقیم:**
  - بخشی از بیت‌های آدرس از PC برداشته می‌شود.

Pseudo-direct addressing

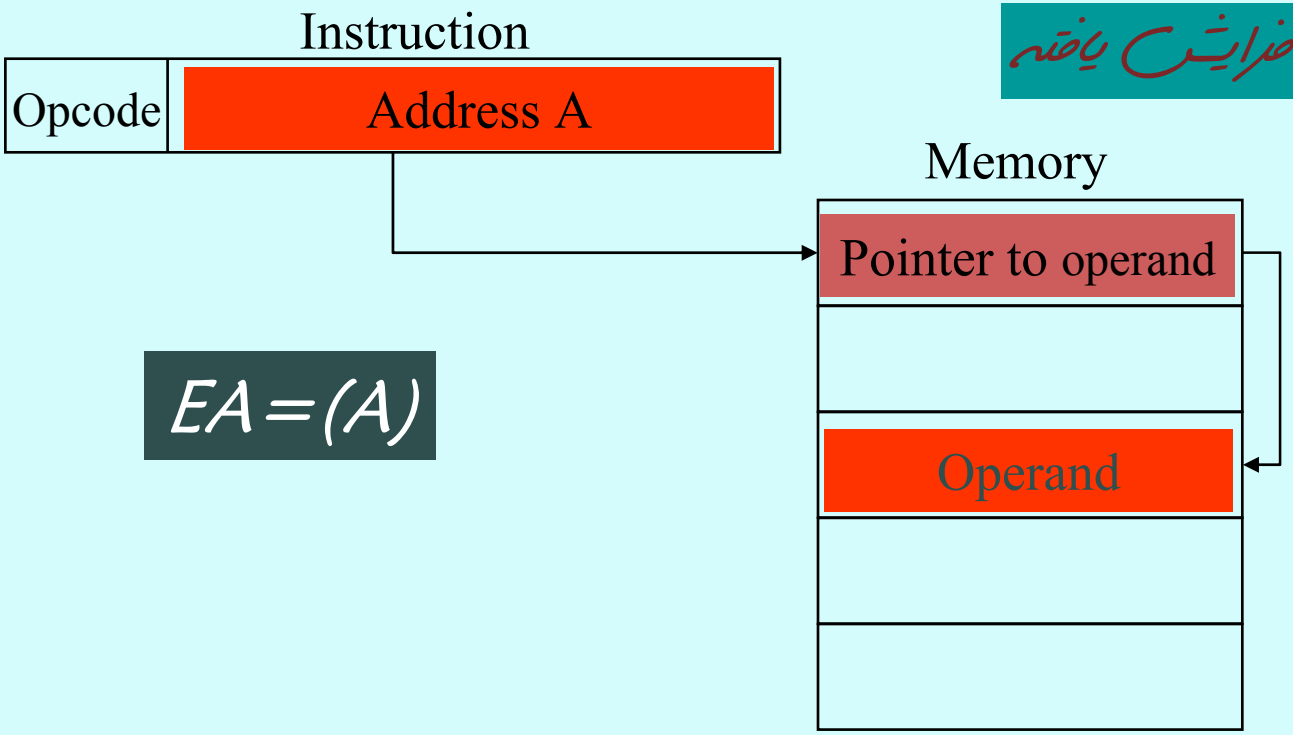


برخی منابع آن را augmented addressing هم می‌نامند.

انواع نشانی‌دهی (ادامه...)

- **آدرس‌دهی غیرمستقیم:**
  - در فیلد آدرس، آدرس خانه‌ای از حافظه آورده می‌شود که حاوی آدرس عملوند است.

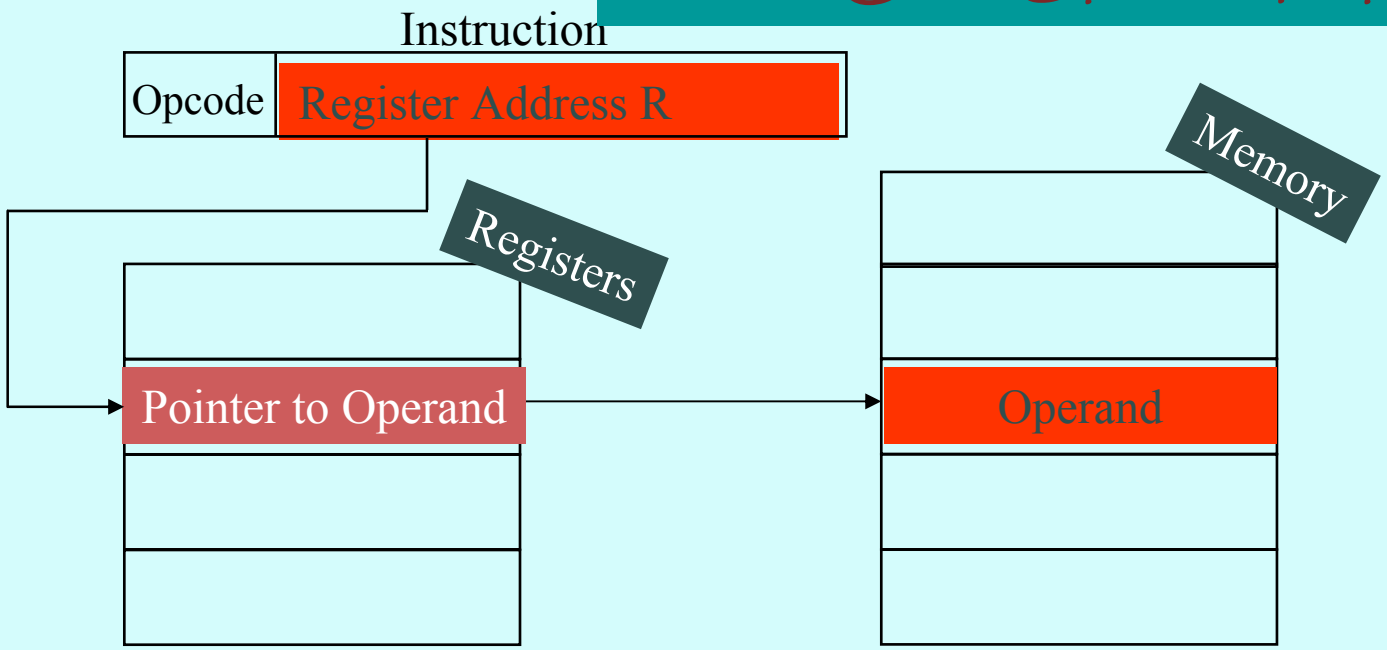
فضای آدرس‌دهی افزایش یافته



انواع نشانی‌دهی (ادامه...)

- آدرس‌دهی غیرمستقیم از طریق ثبات: در این شیوه آدرس خانه‌ی حافظه در یک ثبات قرار دارد.

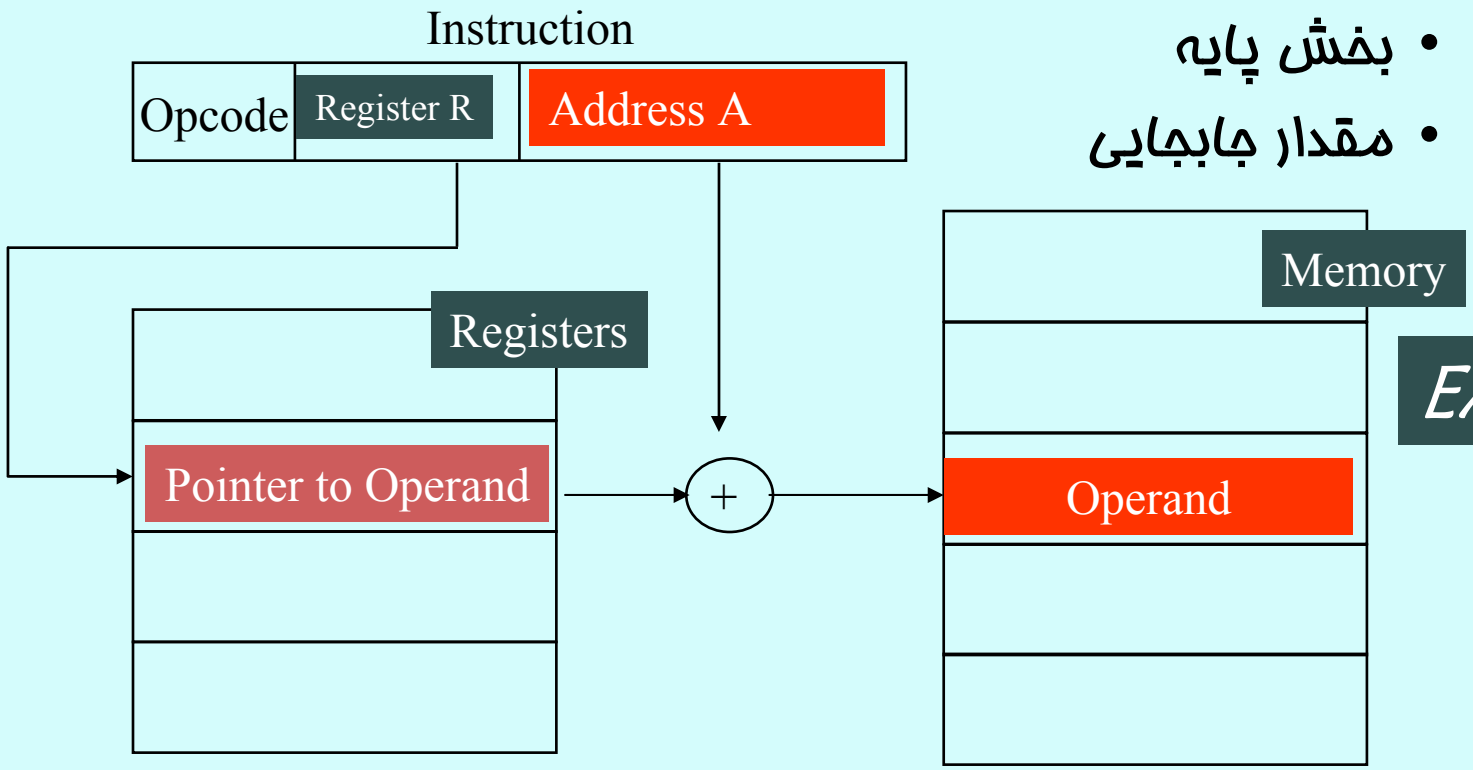
فضای آدرس‌دهی بزرگ است
مراجعه به حافظه نسبت به حالت قبل کاهش یافته است



$EA = (R)$

انواع نشانی‌دهی (ادامه...)

- آدرس‌دهی بر اساس آدرس پایه یا جانشین‌سازی:
  - آدرس از دو بخش تشکیل شده است:
    - بخش پایه
    - مقدار جابجایی

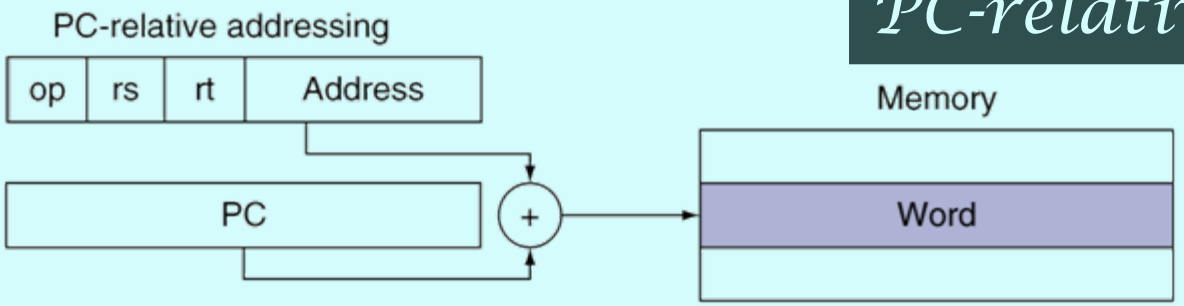


$EA = A + (R)$

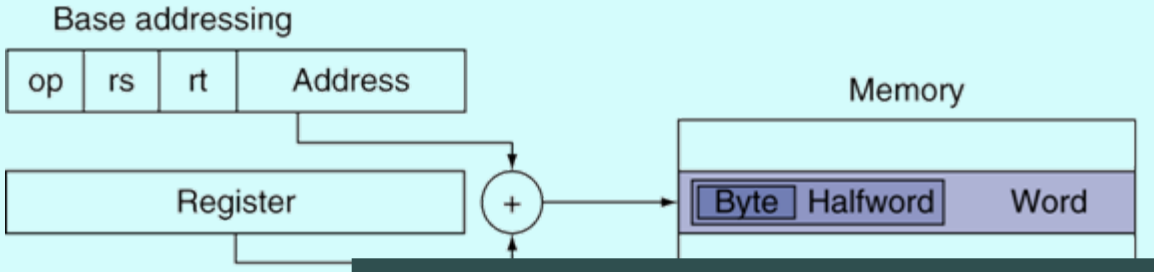
آدرس دهی نسبی (Relative) : نوع خاصی از این نوع آدرس دهی است که PC را به عنوان بخش پایه استفاده می‌کند.

انواع نشانی‌دهی (ادامه...)

- آدرس دهی نسبی (نسبت به PC):
  - مانند آدرس‌های پرش شرطی که نسبت به آدرس دستور بعدی سنجیده می‌شوند



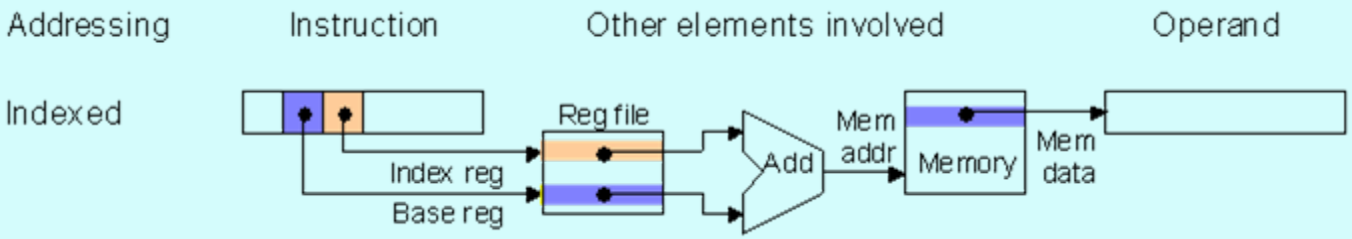
PC-relative addressing



Base or displacement addressing

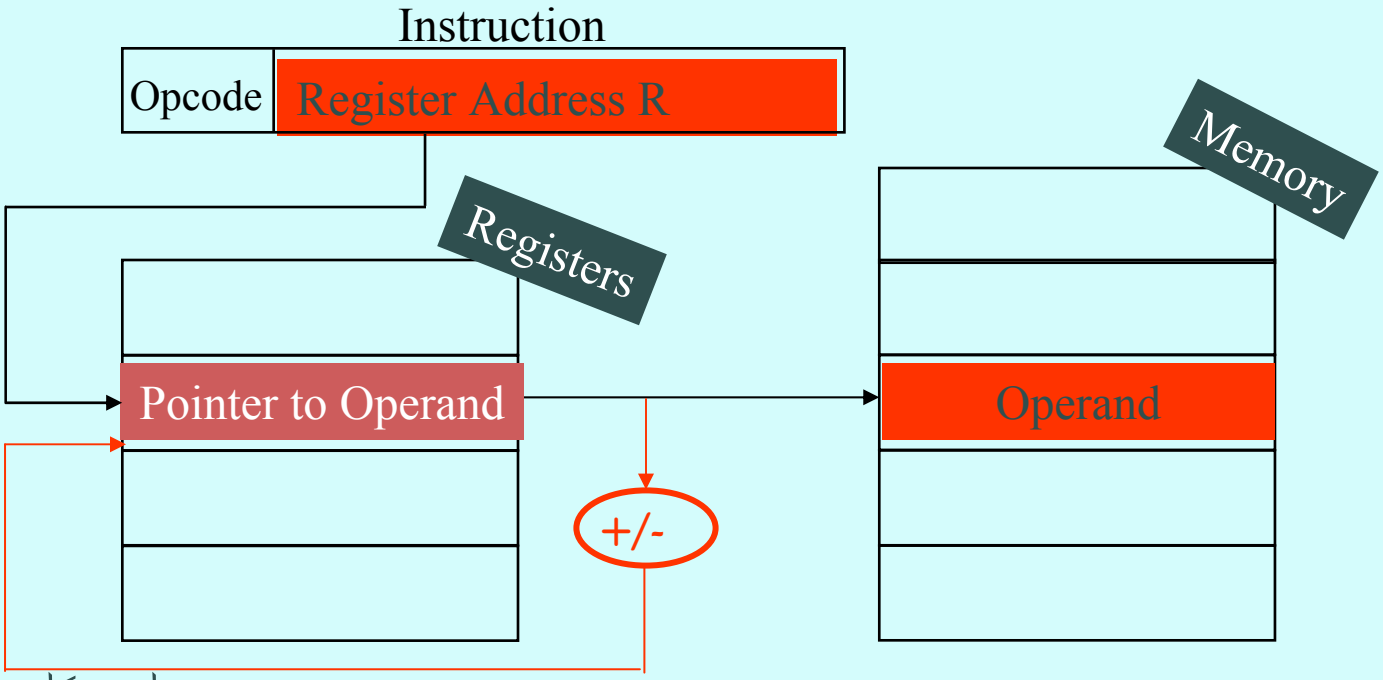
Indexed Addressing

- **آدرس‌دهی شاخص:**
  - شبیه‌آدرس‌دهی بر اساس آدرس پایه است، با این تفاوت که بخش آدرس، ابتدای آدرس بخشی از حافظه را نشان می‌دهد در حالی که بخش شاخص اختلاف از آن بخش را نشان می‌دهد.

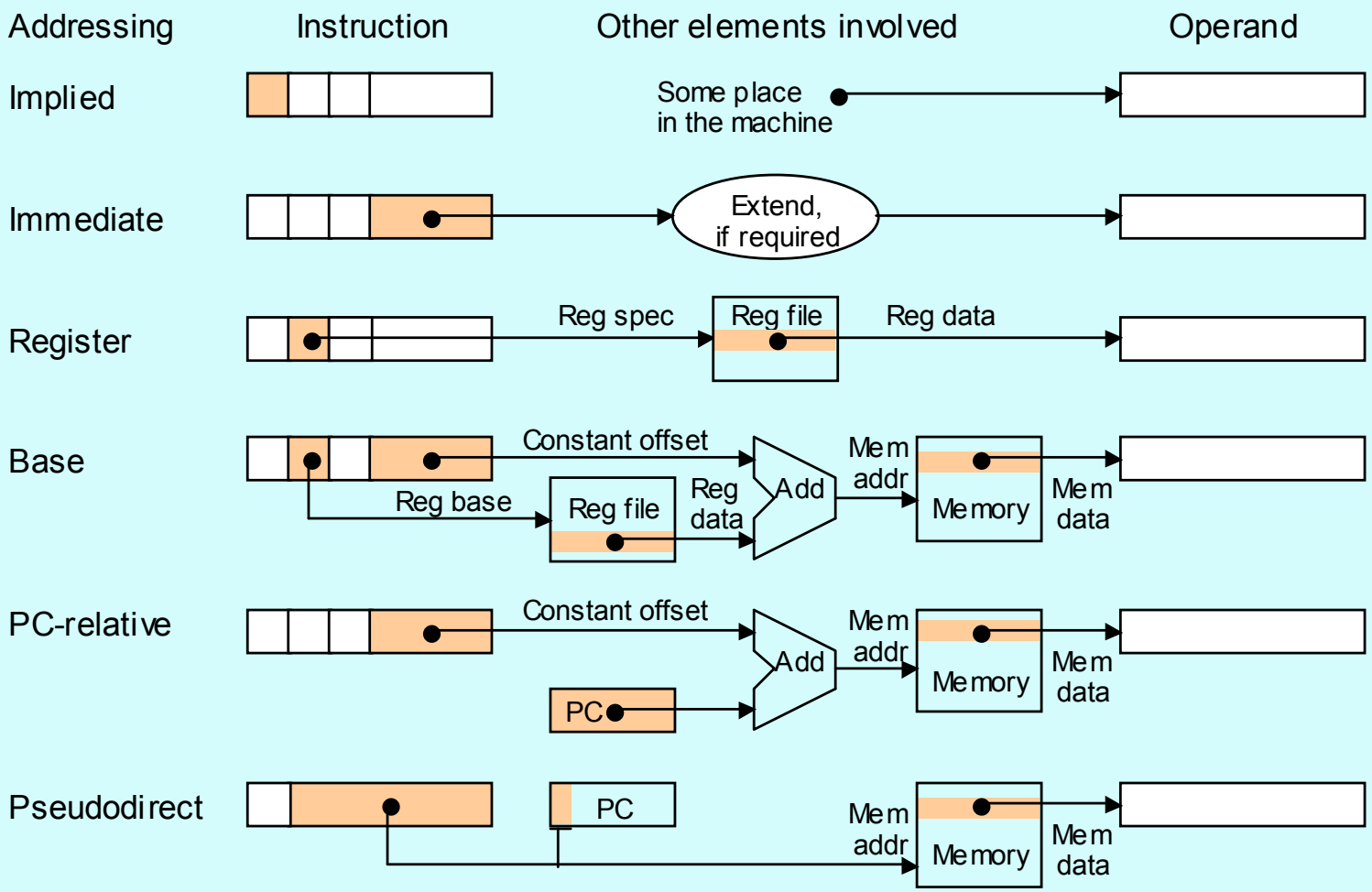


انواع نشانی‌دهی (ادامه...)

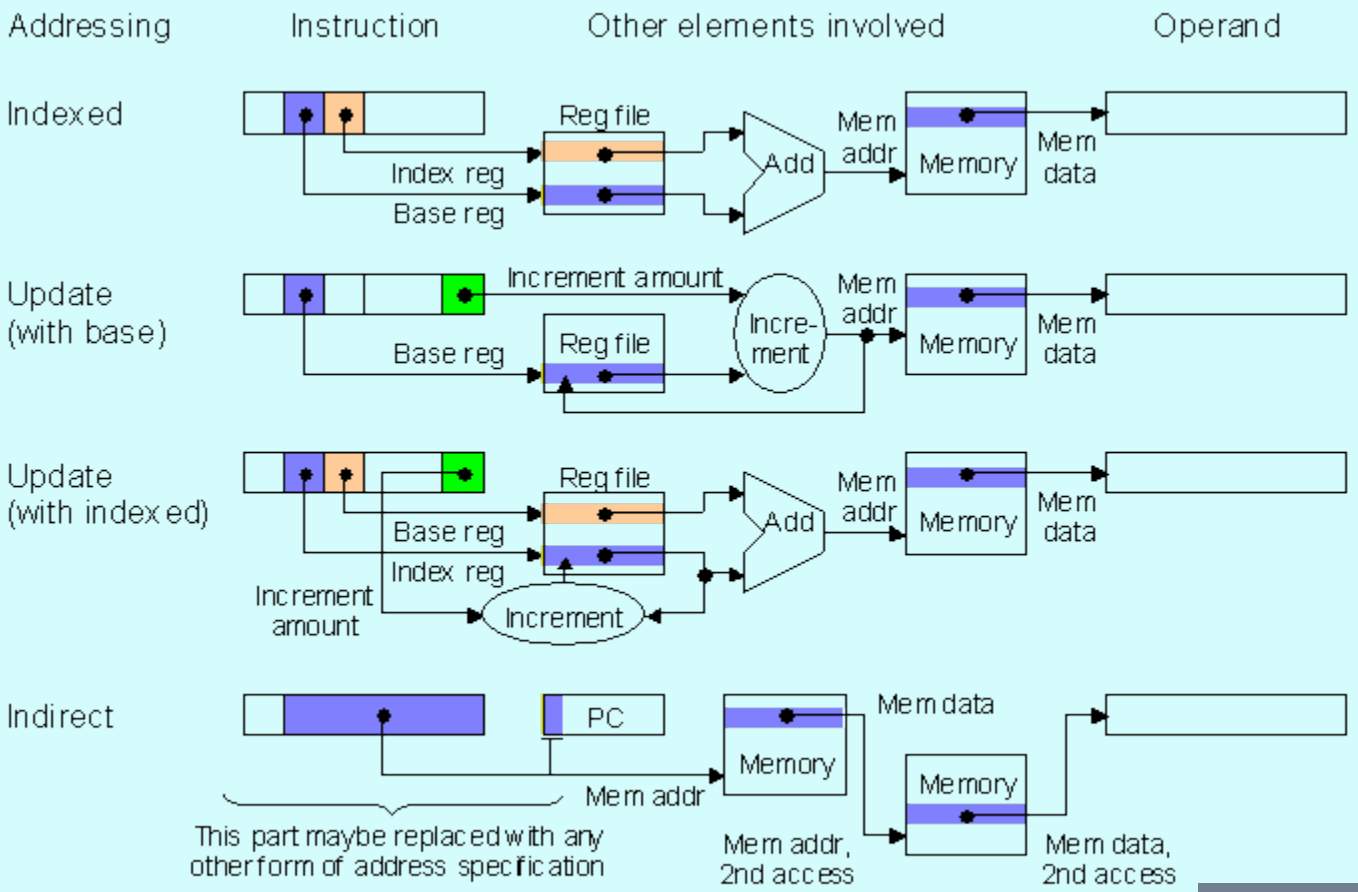
- در این شیوه‌ی محتوای ثبات هر بار یکی افزوده می‌شود



آدرس‌دهی در MIPS



سایر انواع آدرس‌دهی




```
x := B[i]
```

```
x := Mem[p]
p := p + 1
```

```
x := B[i]
i := i + 1
```

```
t := Mem[p]
x := Mem[t]
```

```
x := Mem[Mem[p]]
```

معماری کامپیوتر (۱۳۹۰-۱۱-۱۳)
جلسه‌ی ششم

دانشگاه شهید بهشتی
دانشکده‌ی مهندسی برق و کامپیوتر
زمستان ۱۳۹۰
احمد محمودی ازناوه

فهرست مطالب

- پردازش موازی، همگام‌سازی و نقش سخت‌افزار
- نقش کامپایلر در کارایی
  - اشاره‌گر و آرایه
- نگاهی به دو نوع معماری
- مدارهای حسابی

پردازش موازی و همگام‌سازی

Synchronization

- 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 تک پردازنده‌ای 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 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.

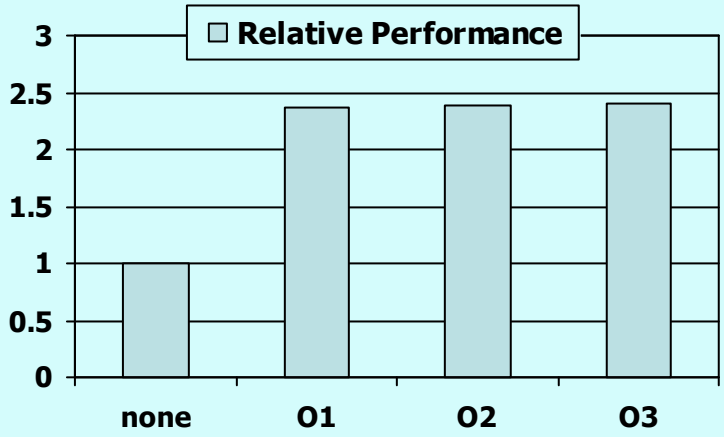
جابجایی تجزیه‌ناپذیر

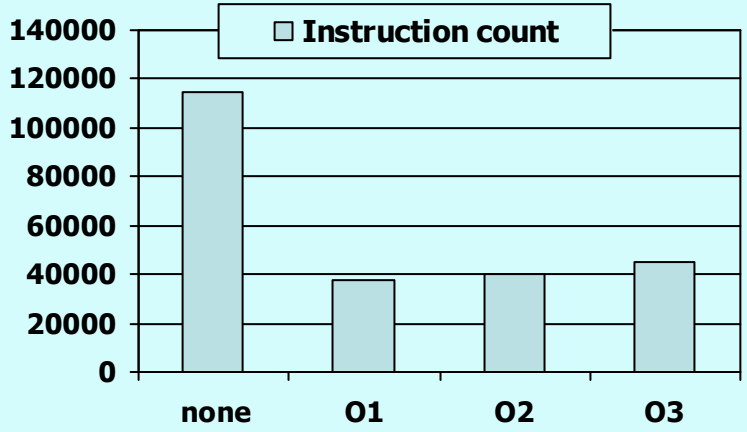
```
try: add $t0,$zero,$s4 ;copy exchange value
     ll  $t1,0($s1)    ;load linked
     sc  $t0,0($s1)    ;store conditional
     beq $t0,$zero,try ;branch store fails
     add $s4,$zero,$t1 ;put load value in $s4
```

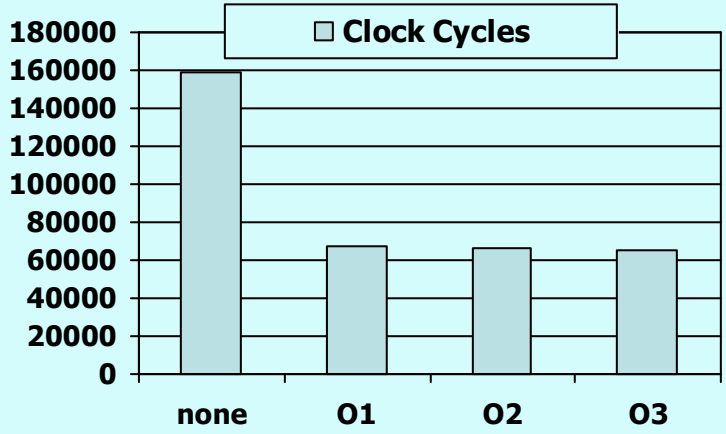
- از این دو دستور، برای پیاده‌سازی تجزیه‌ناپذیر دستورات زیر می‌توان بهره جست:
  - مقایسه و جابه‌جایی
  - واکشی و افزایش

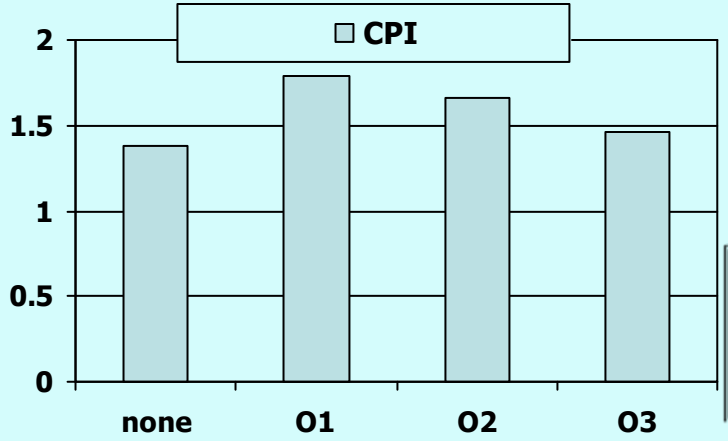
بهینه‌سازی کامپایلر

Compiled with gcc for Pentium 4 under Linux

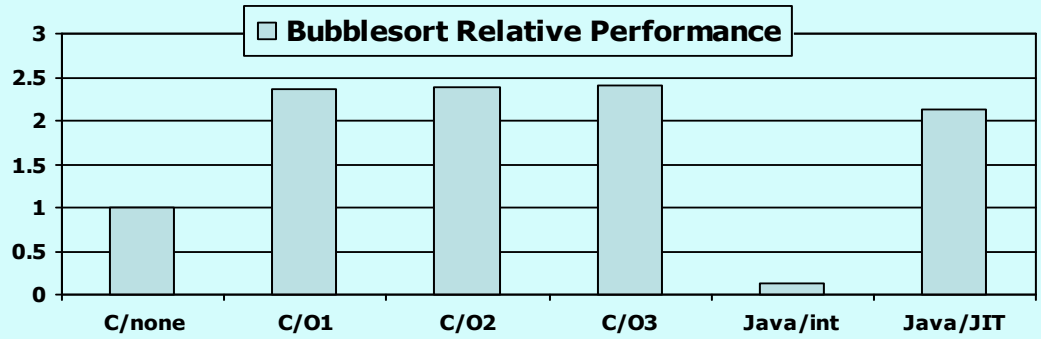


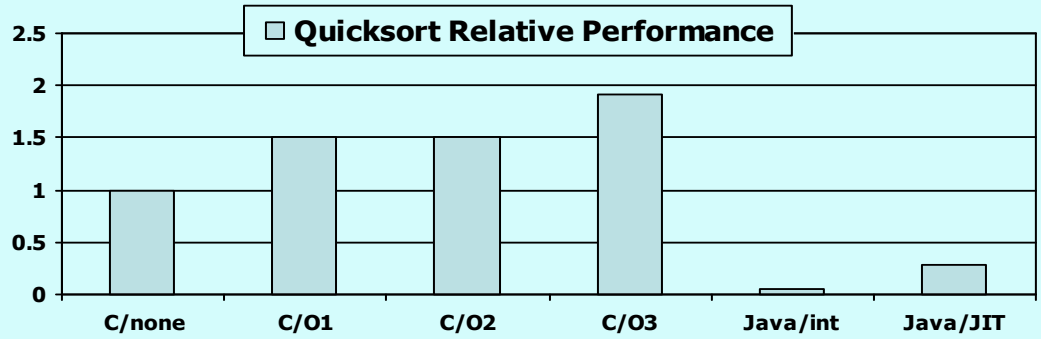


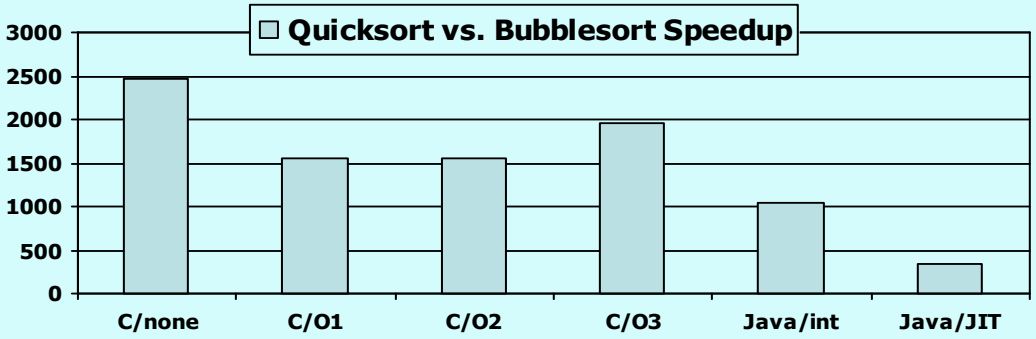




نقش زبان برنامه‌نویسی در کارایی







آرایه در برابر اشاره‌گر

```
clear1(int
array[], int size)
{
  int i;
  for (i = 0; i <
size; i += 1)
    array[i] = 0;
}
```

array در $a0، size در $a1، i در $t0 قرار داده می‌شوند.

```
      move $t0,$zero   # i = 0
loop1: sll $t1,$t0,2    # $t1 = i * 4
       add $t2,$a0,$t1  # $t2 =
                        #   &array[i]
       sw $zero, 0($t2) # array[i] = 0
       addi $t0,$t0,1   # i = i + 1
       slt $t3,$t0,$a1  # $t3 =
                        #   (i < size)
       bne $t3,$zero,loop1 # if (…)
                           # goto loop1
```

آرایه در برابر اشاره‌گر (ادامه...)

```
clear2(int *array, int
size) {
  int *p;
  for (p = &array[0];
p < &array[size];
       p = p + 1)
    *p = 0;
}
```

array در $a0، size در $a1، p در $t0 قرار داده می‌شوند.

```
      move   $t0,$a0    # p = & array[0]#
loop2: sw $zero,0($t0) # Memory[p] = 0
      addi $t0,$t0,4  # p = p + 4
      sll $t1,$a1,2   # $t1 = size * 4
      add $t2,$a0,$t1 # $t2 = &array[size]
      slt $t3,$t0,$t2 # $t3 = (p<&array[size])
      bne $t3,$zero,loop2 # if (…) goto loop2
```

آرایه در برابر اشاره‌گر (ادامه...)

```
clear1(int array[], int size) {
  int i;
  for (i = 0; i < size; i += 1)
    array[i] = 0;
}
```

```
clear2(int *array, int size) {
  int *p;
  for (p = &array[0]; p < &array[size];
       p = p + 1)
    *p = 0;
}
```

```
      move $t0,$zero   # i = 0
loop1: sll $t1,$t0,2   # $t1 = i * 4
      add $t2,$a0,$t1  # $t2 =
                       #   &array[i]
      sw $zero, 0($t2) # array[i] = 0
      addi $t0,$t0,1   # i = i + 1
      slt $t3,$t0,$a1  # $t3 =
                       #   (i < size)
      bne $t3,$zero,loop1 # if (...)
                          # goto loop1
```

```
      move $t0,$a0     # p = & array[0]
      sll $t1,$a1,2    # $t1 = size * 4
      add $t2,$a0,$t1  # $t2 =
                       #   &array[size]
loop2: sw $zero,0($t0) # Memory[p] = 0
      addi $t0,$t0,4   # p = p + 4
      slt $t3,$t0,$t2  # $t3 =
                       #(p<&array[size])
      bne $t3,$zero,loop2 # if (...)
                          # goto loop2
```

مجموعه دستورات
x86 و ARM

ARM Partnership Model

ARM پردازنده نمی سازد

ARM Powered Products

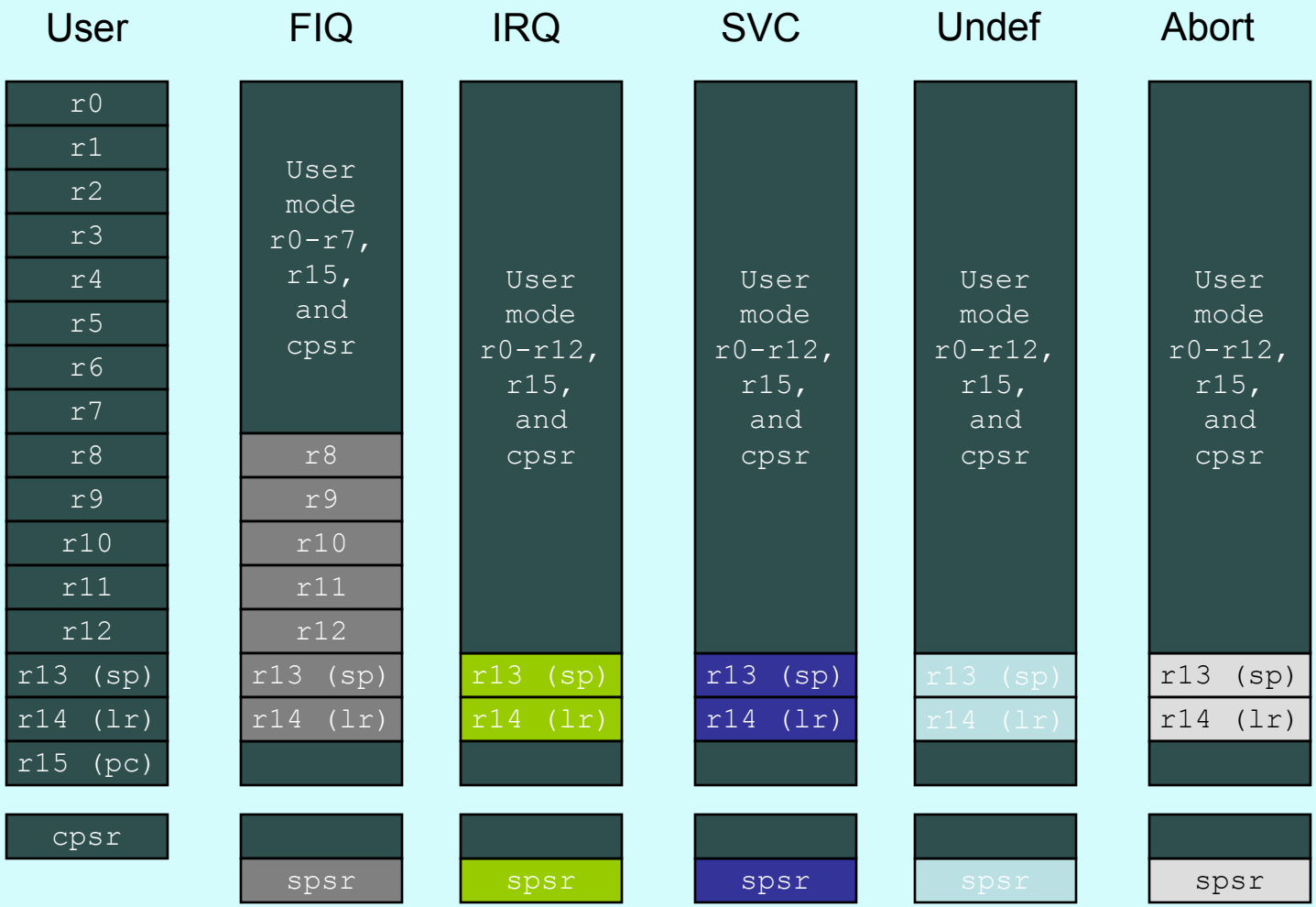
THE ARCHITECTURE FOR THE DIGITAL WORLD

- 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

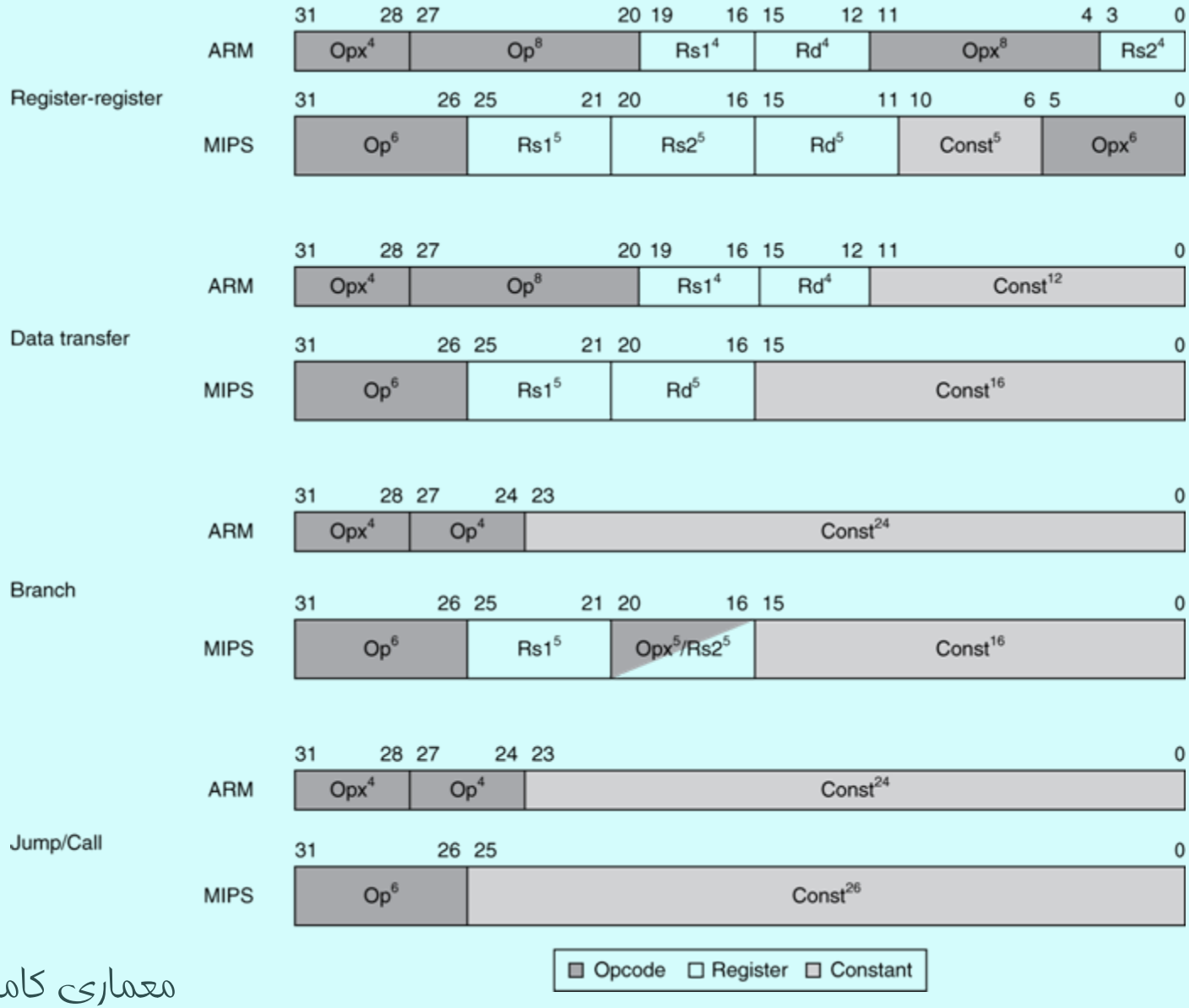
مقایسه و پرش شرطی در ARM

- 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

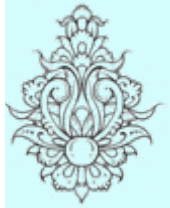


ARM

- 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	—



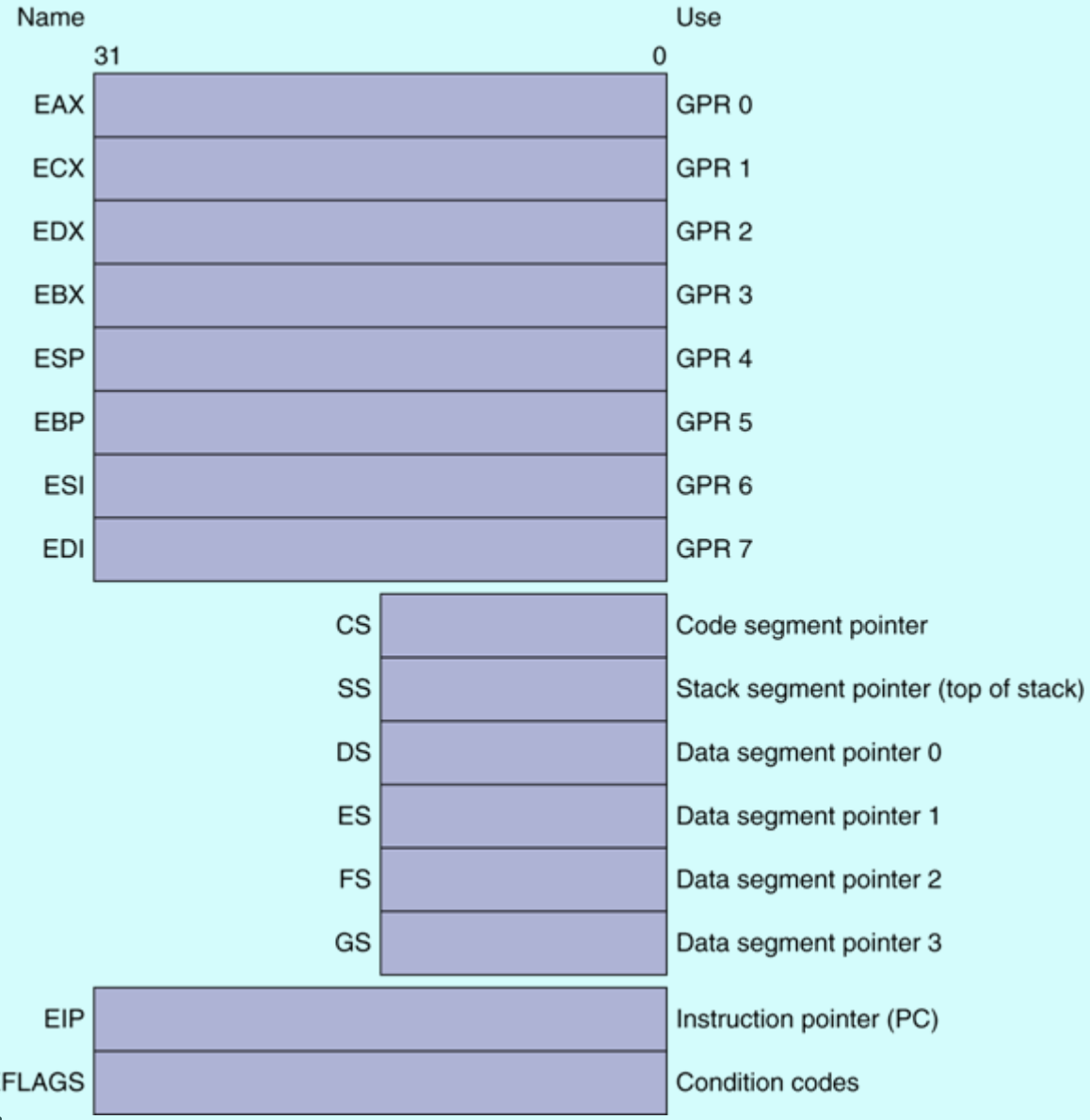
مجموعه دستورات خانواده‌ی X86

- 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



آدرس‌دهی در x86

- **هر دستور دو عملوند دارد:**

Source/dest operand	Second source operand
Register	Register
Register	Immediate
Register	Memory
Memory	Register
Memory	Immediate

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

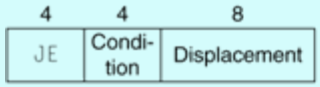
نمونه‌ای از دستورات x86 (ادامه...)

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

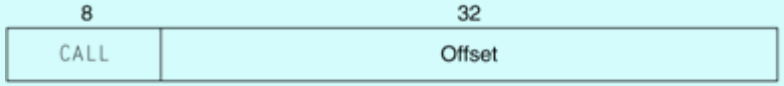
قالب دستورها

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

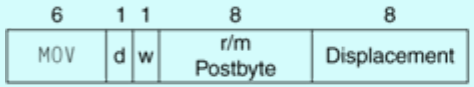
a. JE EIP + displacement



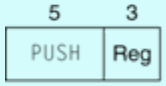
b. CALL



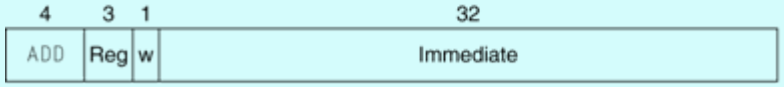
c. MOV EBX, [EDI + 45]



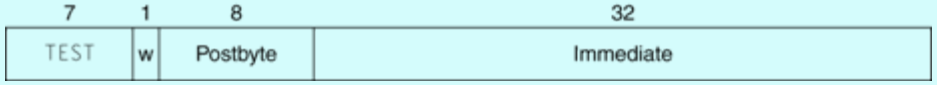
d. PUSH ESI



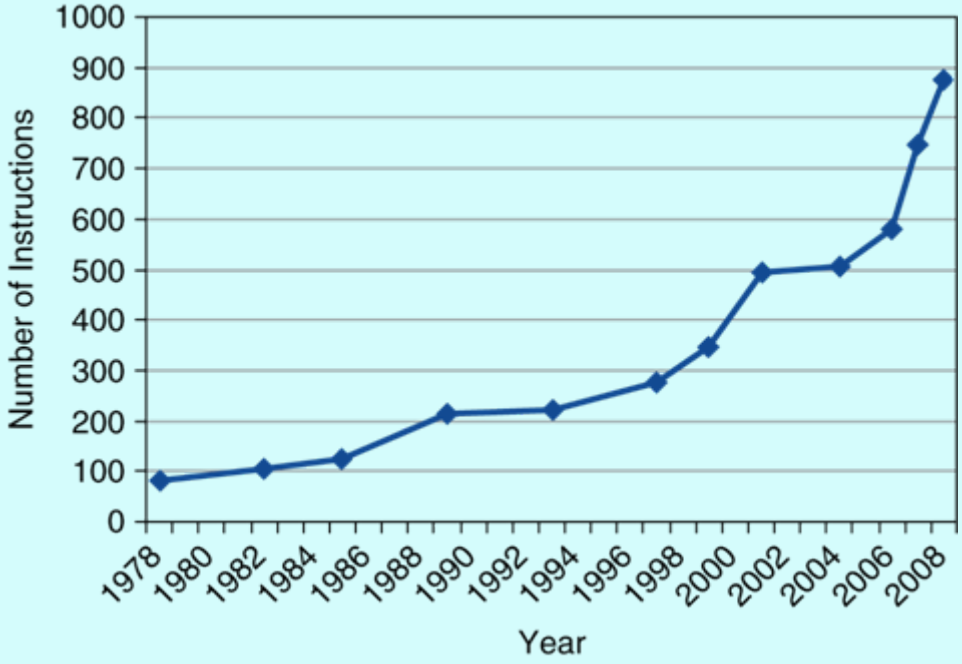
e. ADD EAX, #6765



f. TEST EDX, #42



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



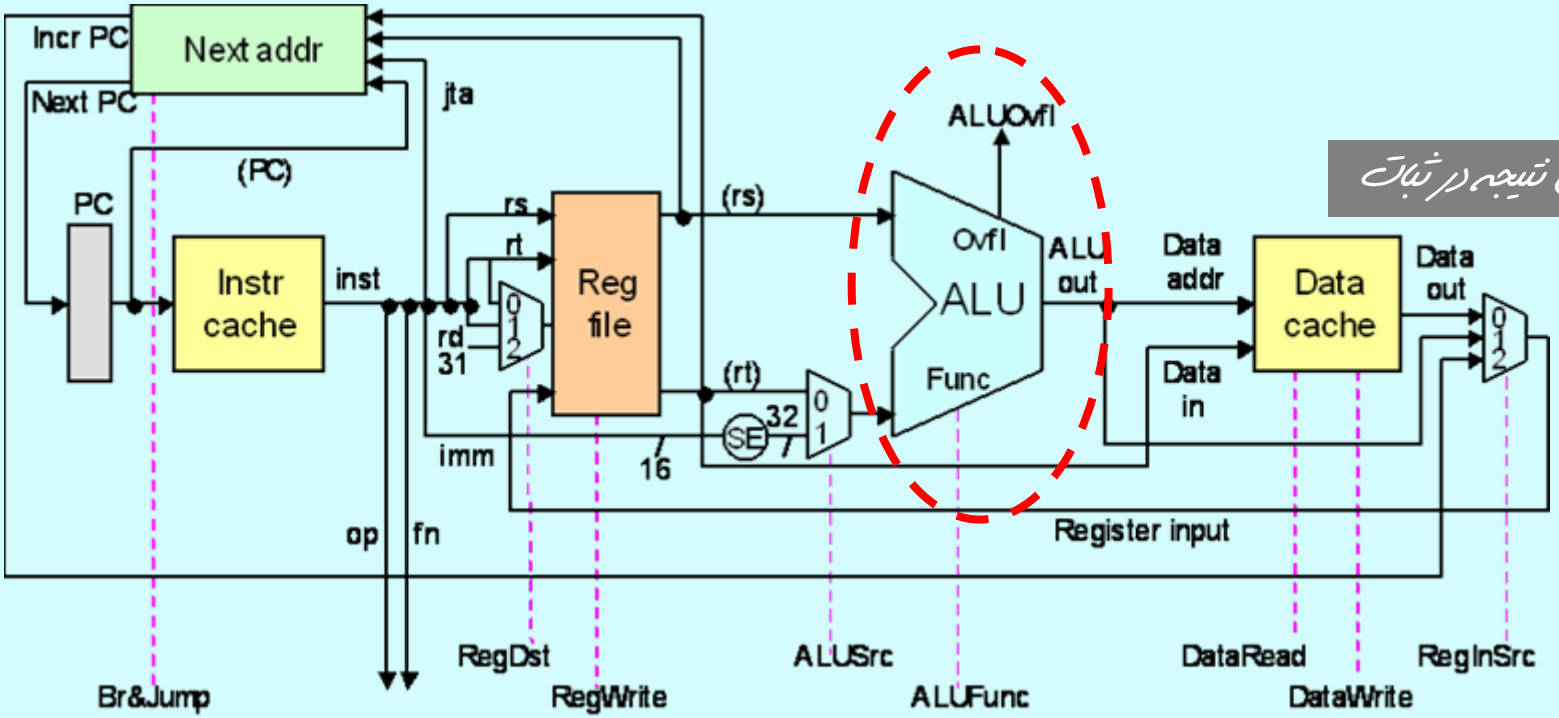
نتیجه‌گیری

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

مدارهای حساب

Numerical Precision is the very soul of science

نمایی از واحد حساب



نوشتن نتیجه در ثبات

واکشی دستورات

دسترسی به محتوای ثبات

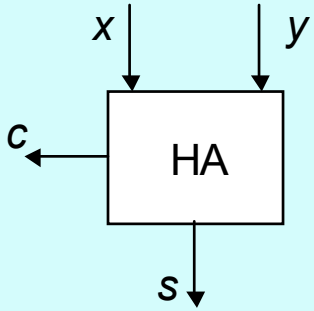
عملیات ALU

دسترسی به حافظه

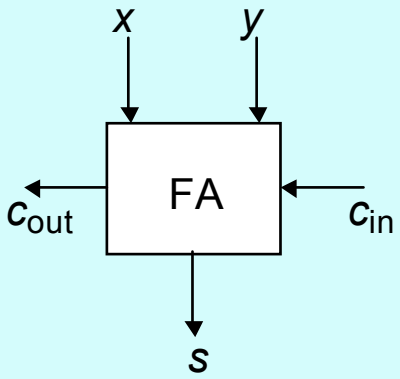
یادآوری

نیم‌جمع‌کننده و تمام‌جمع‌کننده

Inputs		Outputs	
x	*y*	*c*	*s*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

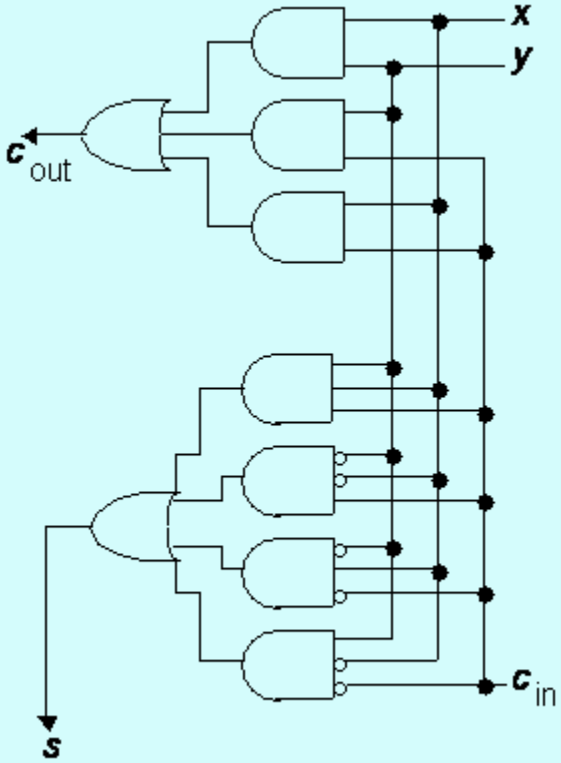


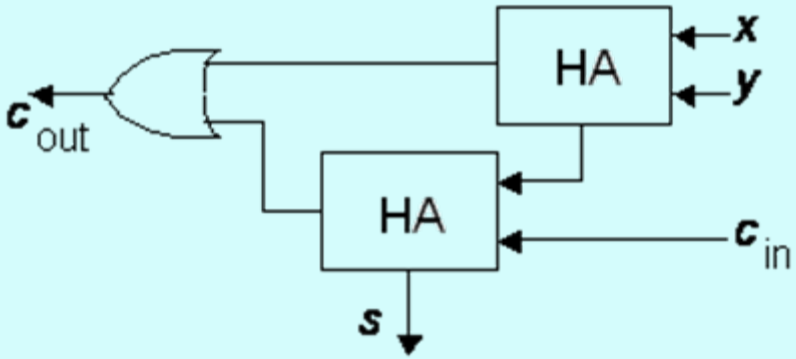
Inputs			Outputs	
x	*y*	c_{in}	c_{out}	*s*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

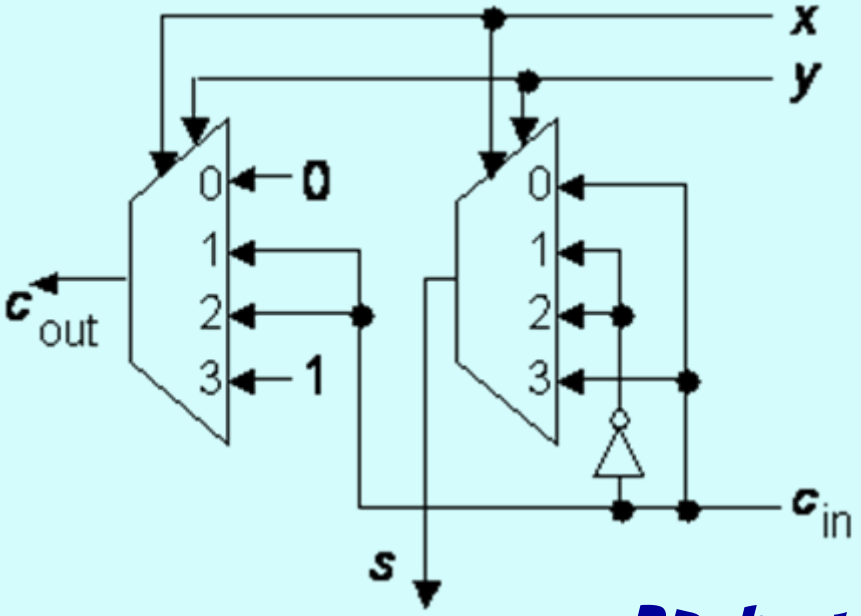


یادآوری

شیوه‌های گوناگون ساخت تمام‌جمع‌کننده







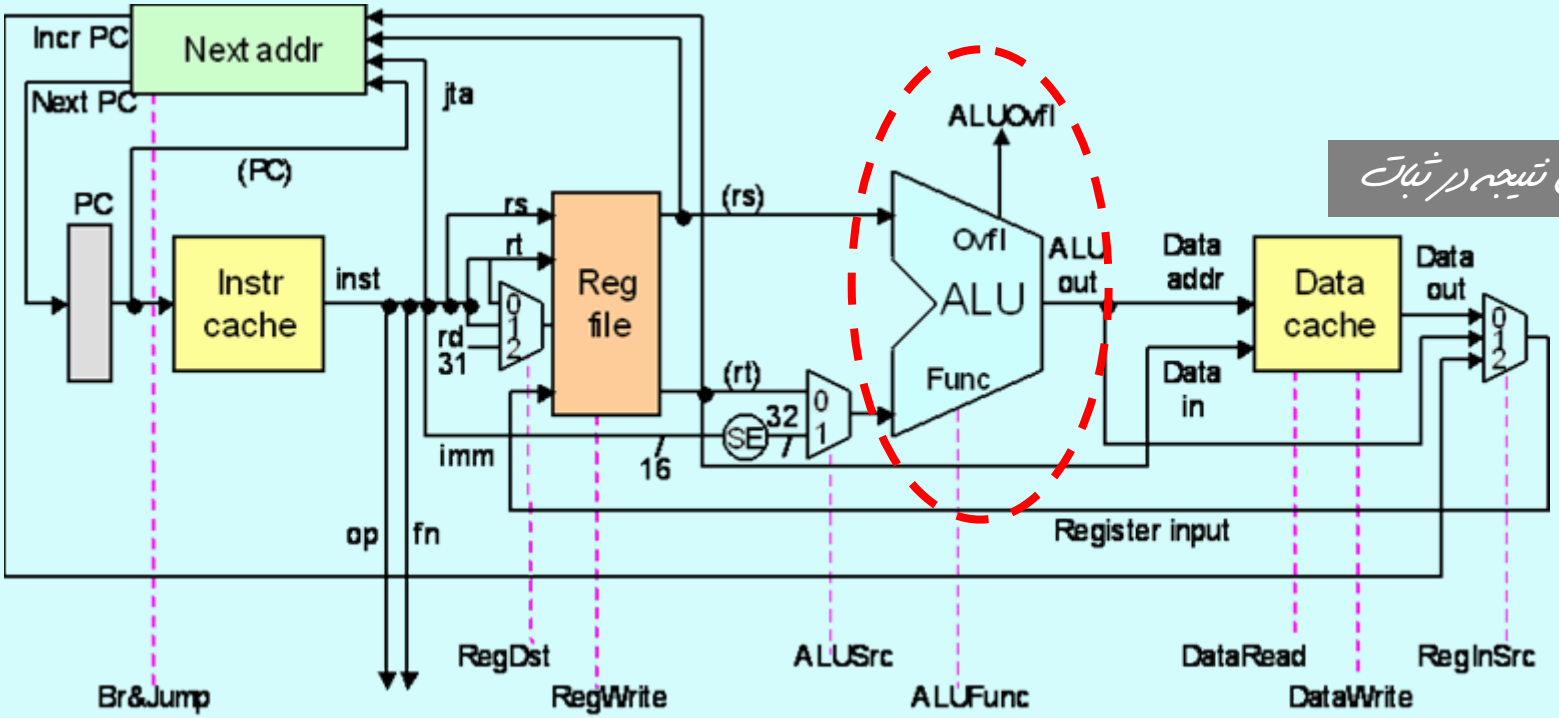
معماری کامپیوتر (۱۳۹۰-۱۱-۱۳)
جلسه‌ی هفتم

دانشگاه شهید بهشتی
دانشکده‌ی مهندسی برق و کامپیوتر
زمستان ۱۳۹۰
احمد محمودی ازناوه

فهرست مطالب

- جمع‌کننده‌ها
  - جمع‌کننده‌های سریع

نمایی از واحد حساب



نوشتن نتیجه در ثبات

واکشی دستورات

دسترسی به محتوای ثبات

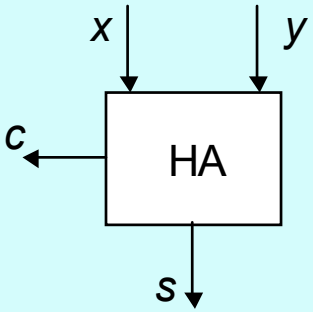
عملیات ALU

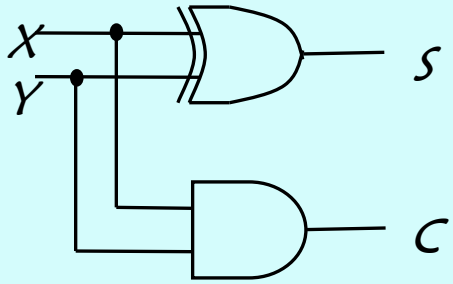
دسترسی به حافظه

یادآوری

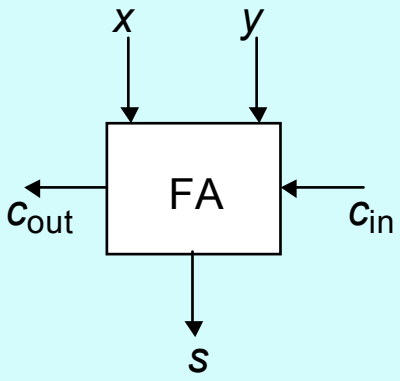
نیم‌جمع‌کننده و تمام‌جمع‌کننده

Inputs		Outputs	
x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



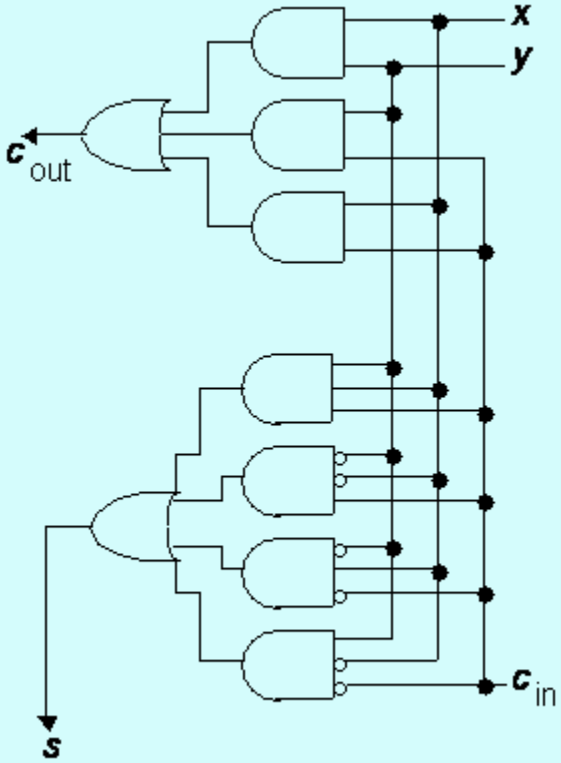


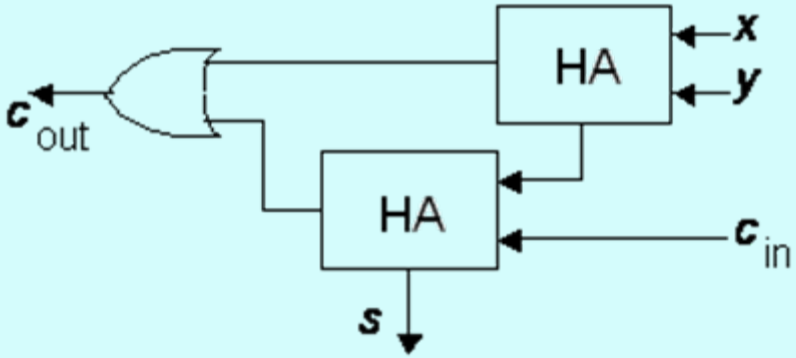
Inputs			Outputs	
x	y	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

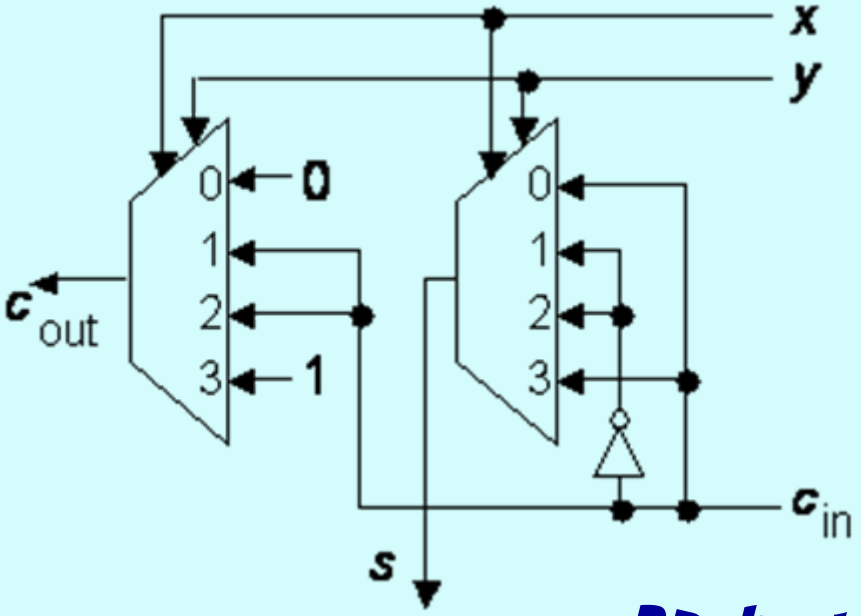


یادآوری

شیوه‌های گوناگون ساخت تمام‌جمع‌کننده

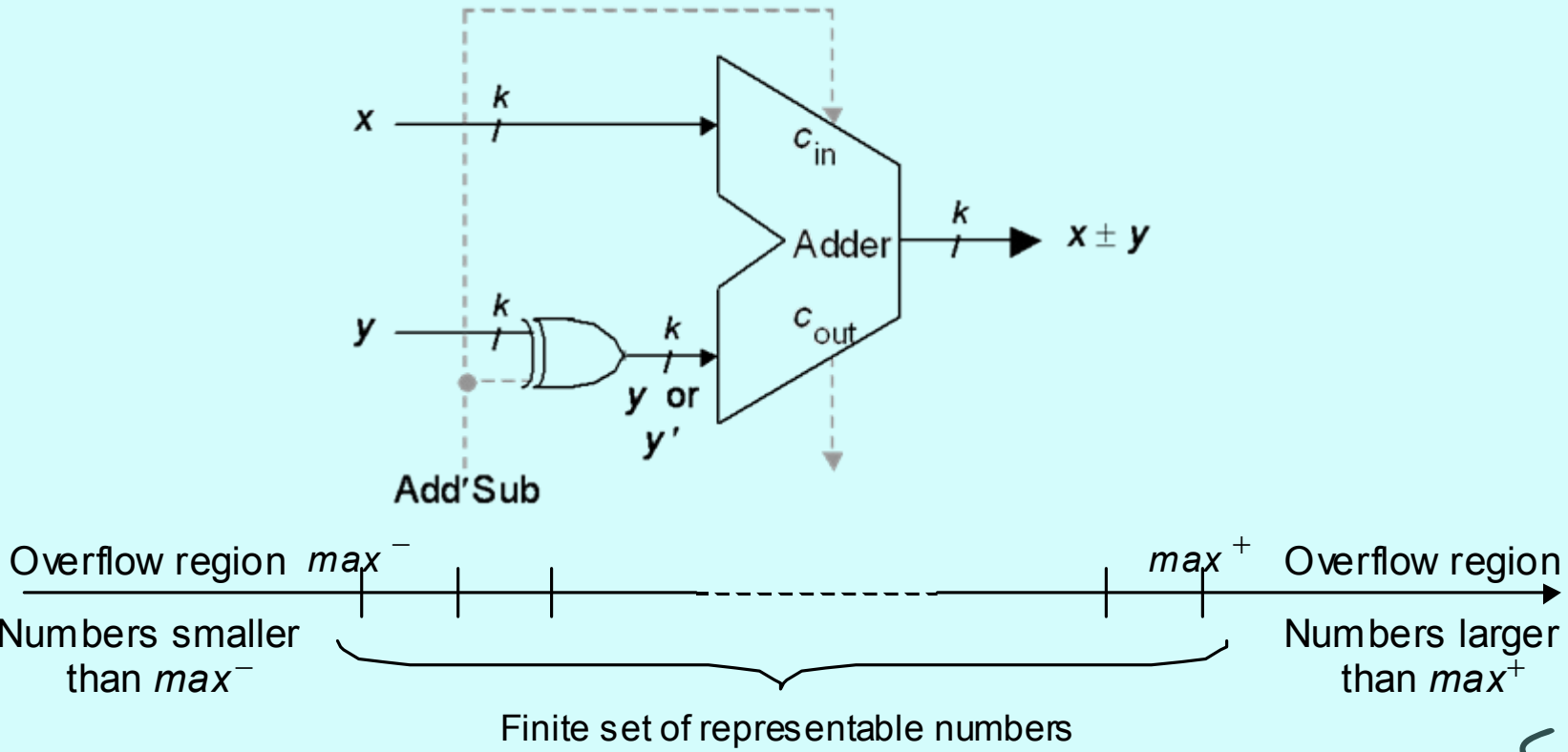






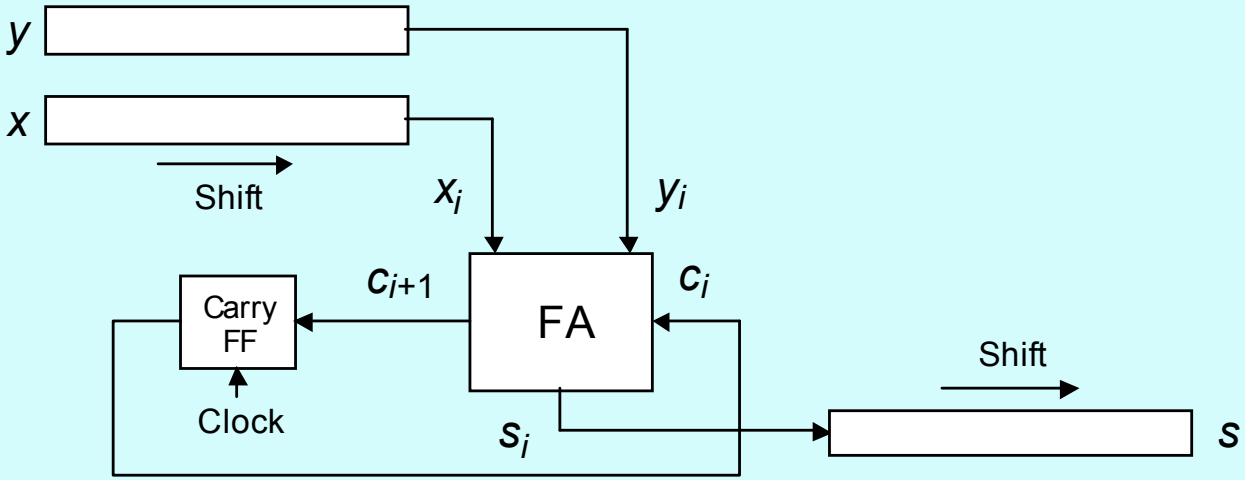
یادآوری

جمع و تفریق اعداد مکمل ۲

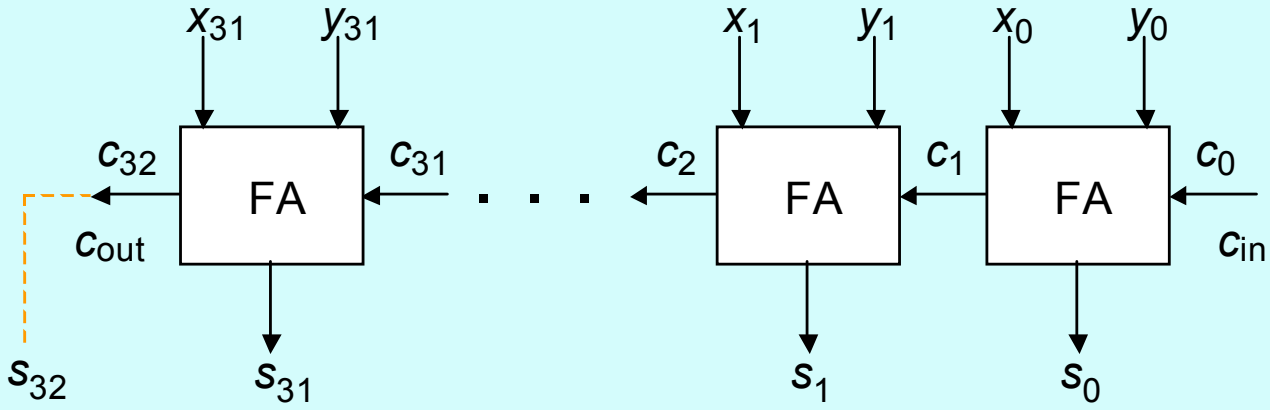


آیا به خاطر دارید چگونه می‌توان وقوع سرریز را تشخیص داد؟

ساخت جمع‌کننده با FA



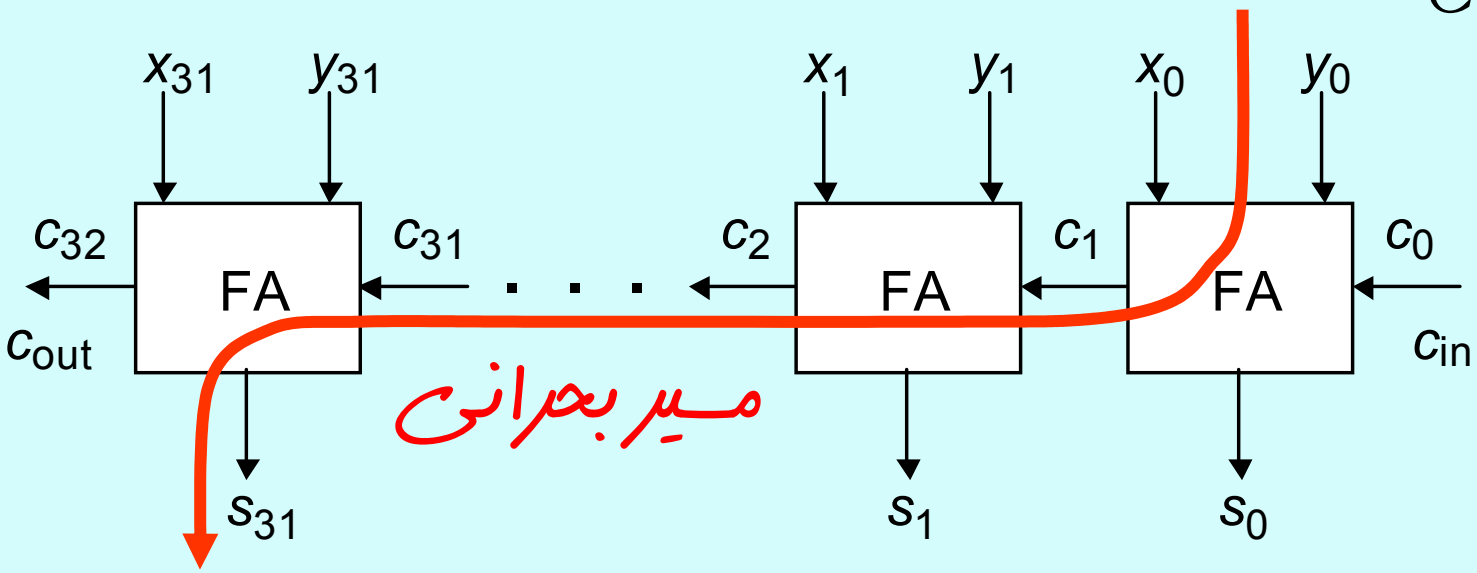
(a) Bit-serial adder.



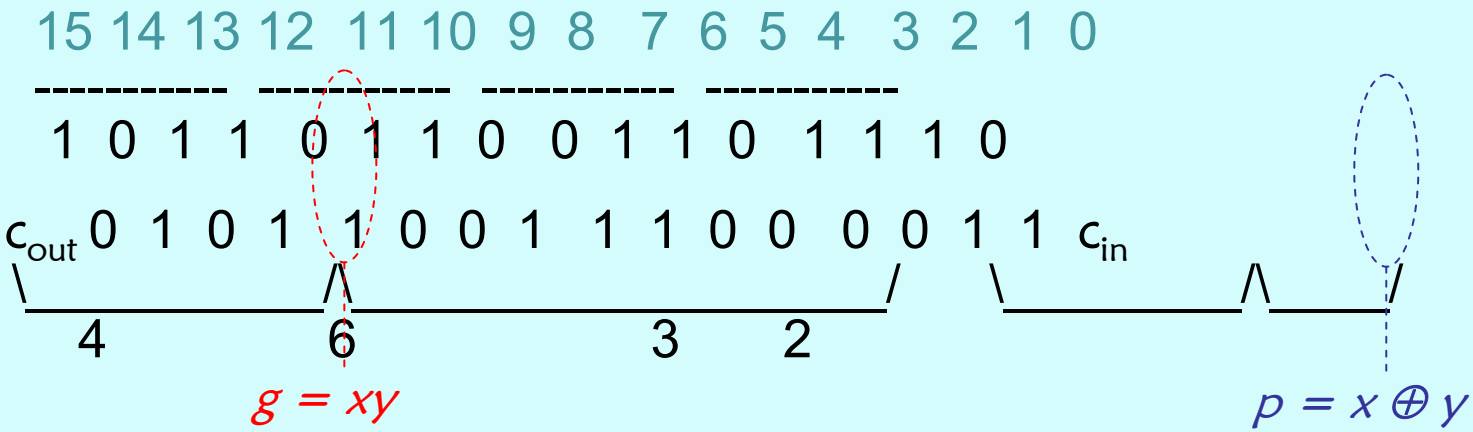
(b) Ripple-carry adder.

یادآوری

جمع‌کننده با انتشار رقم‌نقلی

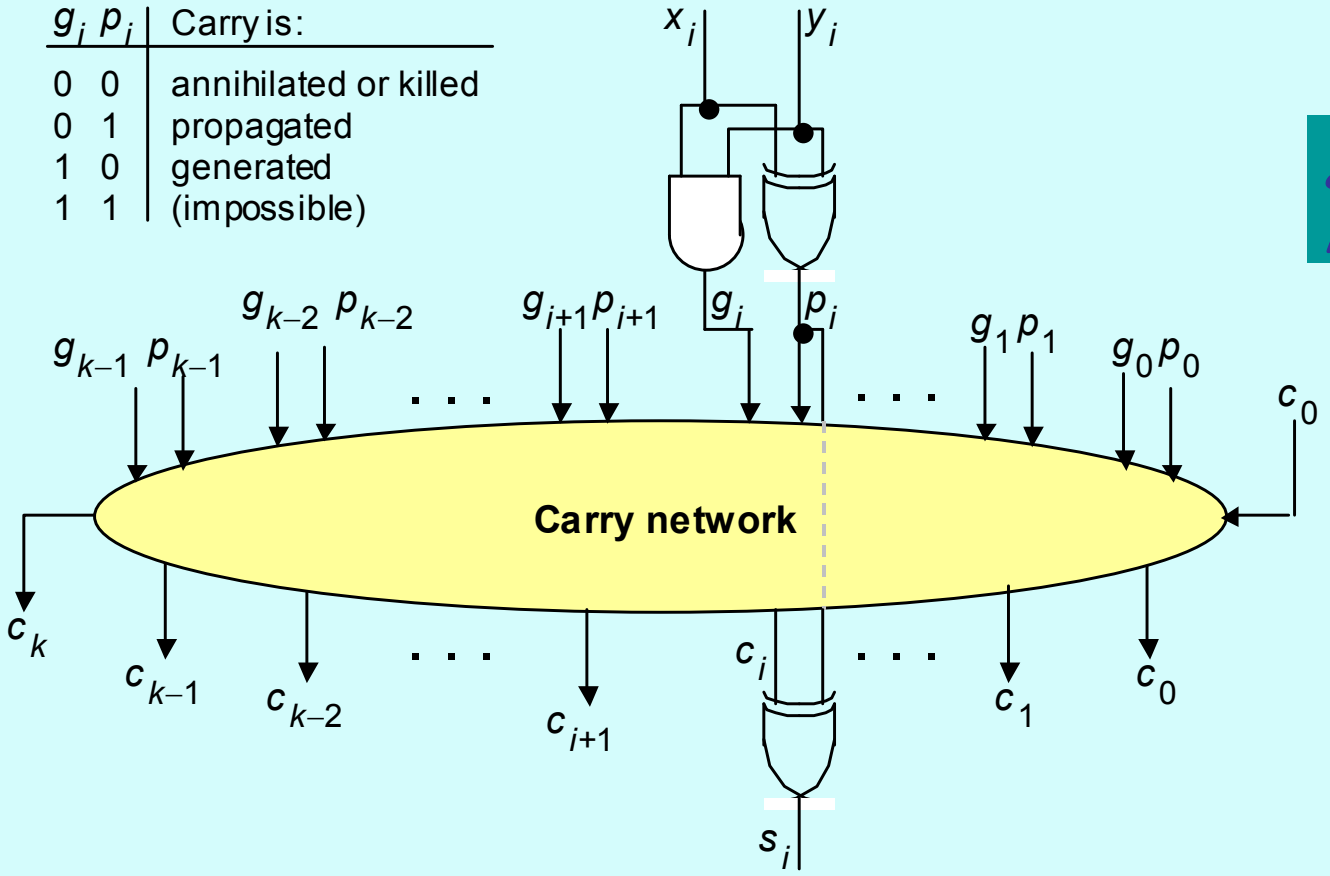


موقعیت بیت‌ها



زنجیره‌ی انتقال رقم نقلی

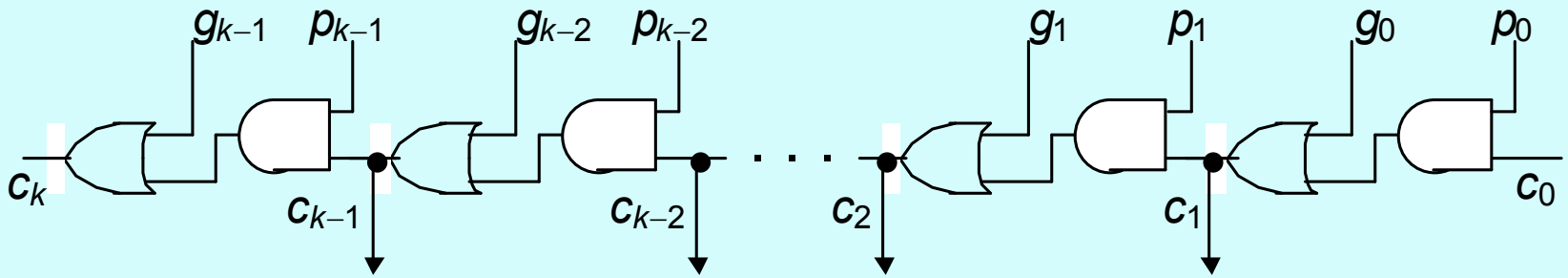
g_i	p_i	Carry is:
0	0	annihilated or killed
0	1	propagated
1	0	generated
1	1	(impossible)



$$g_i = x_i\, y_i$$
$$p_i = x_i \oplus y_i$$

The carry recurrence: $c_{i+1} = g_i \vee p_i\, c_i$

زنجیره‌ی انتقال رقم نقلی (ادامه...)

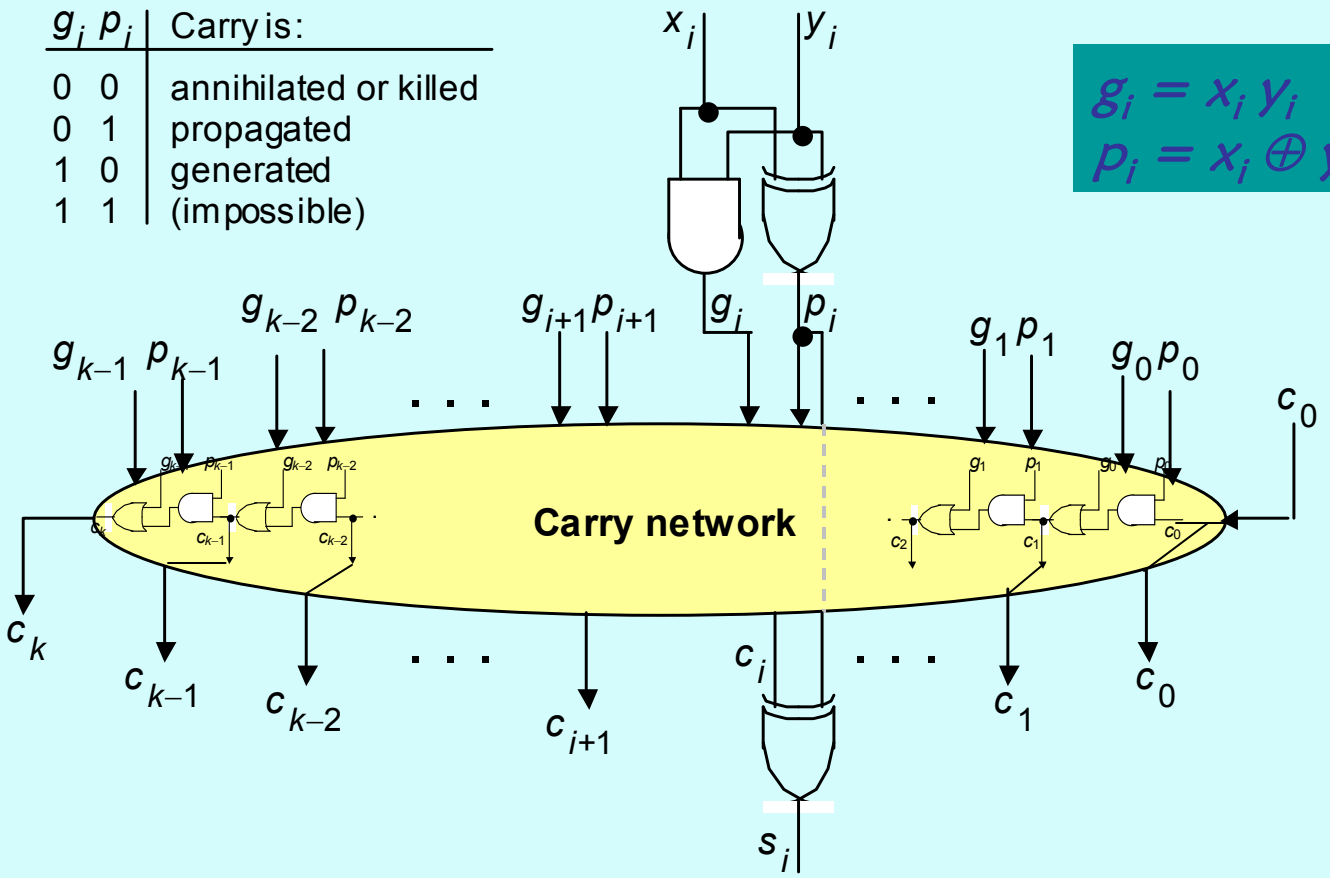


تاخیر انتشار بیت نقلی در یک جمع‌کننده‌ی k بیتی چقدر است؟

زنجیره‌ی انتقال رقم نقلی (ادامه...)

g_i	p_i	Carry is:
0	0	annihilated or killed
0	1	propagated
1	0	generated
1	1	(impossible)

$$g_i = x_i\, y_i$$
$$p_i = x_i \oplus y_i$$



جمع‌کننده با پیش‌بینی رقم نقلی

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

$$c_{i+1} = x_i y_i + (x_i + y_i) c_i$$

$$c_{i+1} = g_i + p_i c_i$$

$$g_i = x_i y_i$$

$$p_i = (x_i + y_i)$$

$$c_{i+1} = g_i + p_i (g_{i-1} + p_{i-1} c_{i-1})$$

$$= g_i + p_i g_{i-1} + p_i p_{i-1} c_{i-1}$$

Generate function

Propagate function

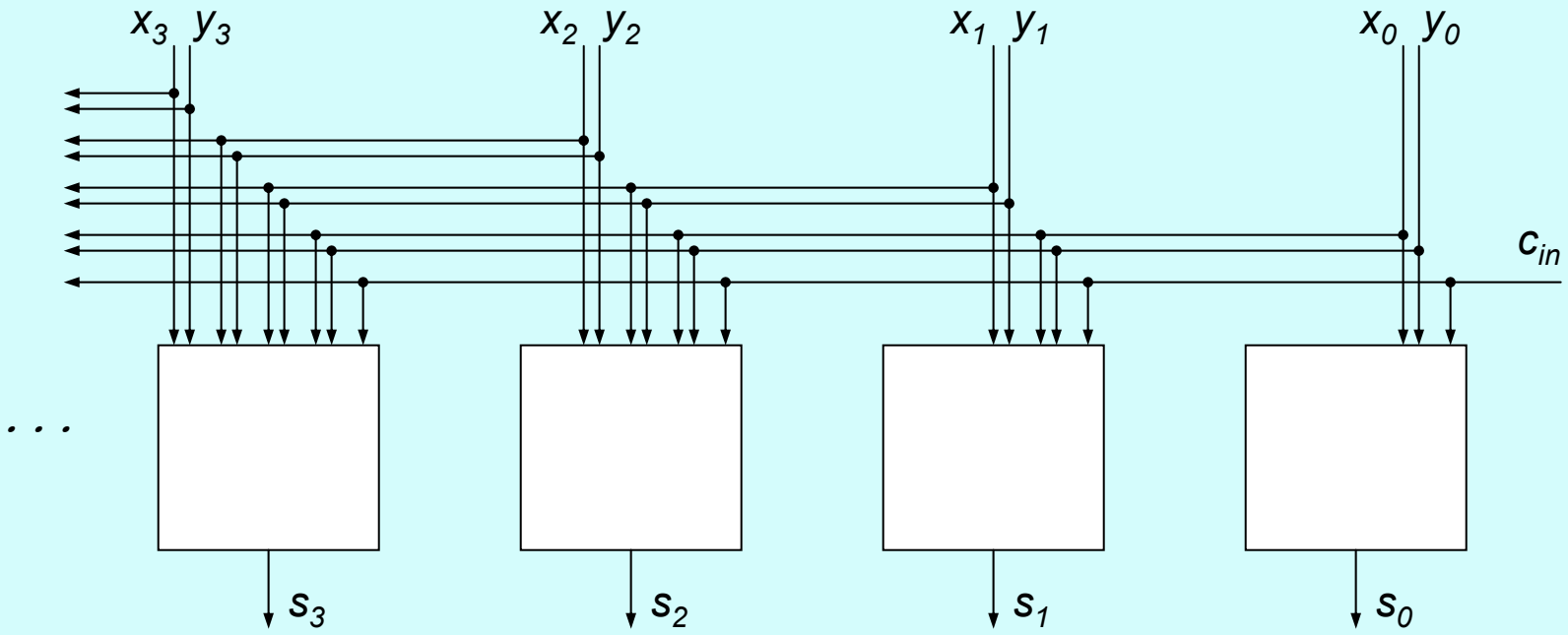
پیش‌بینی رقم نقلی (ادامه...)

$$c_1 = g_0 + p_0 c_0$$

$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$$

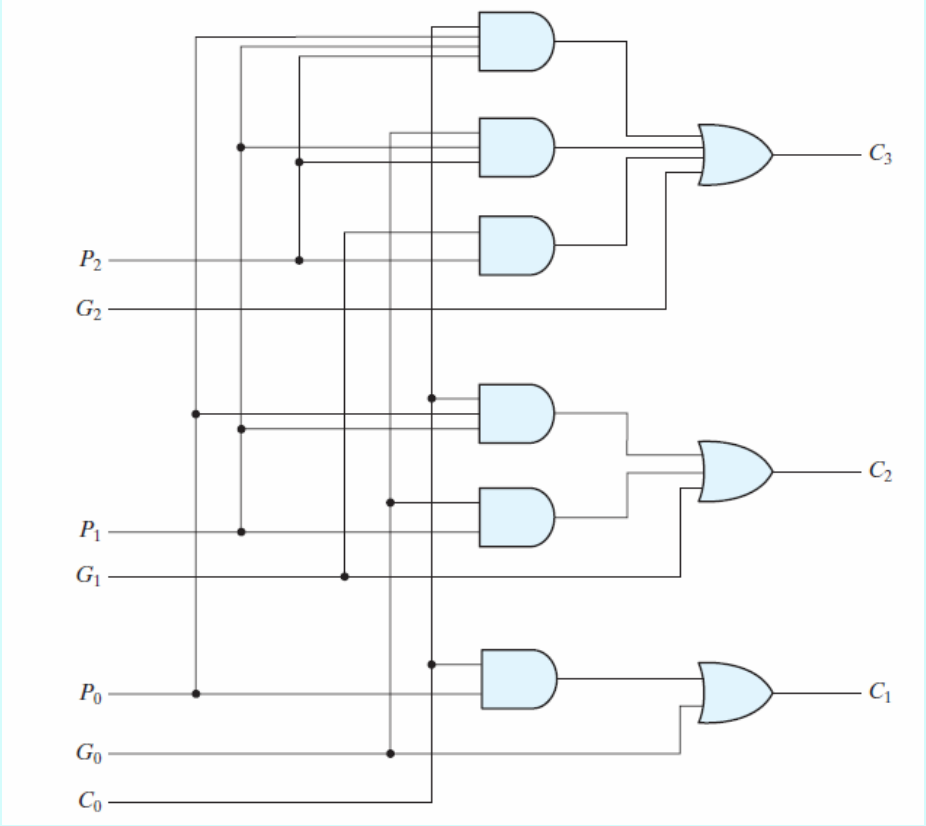
$$c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$$

$$c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$$

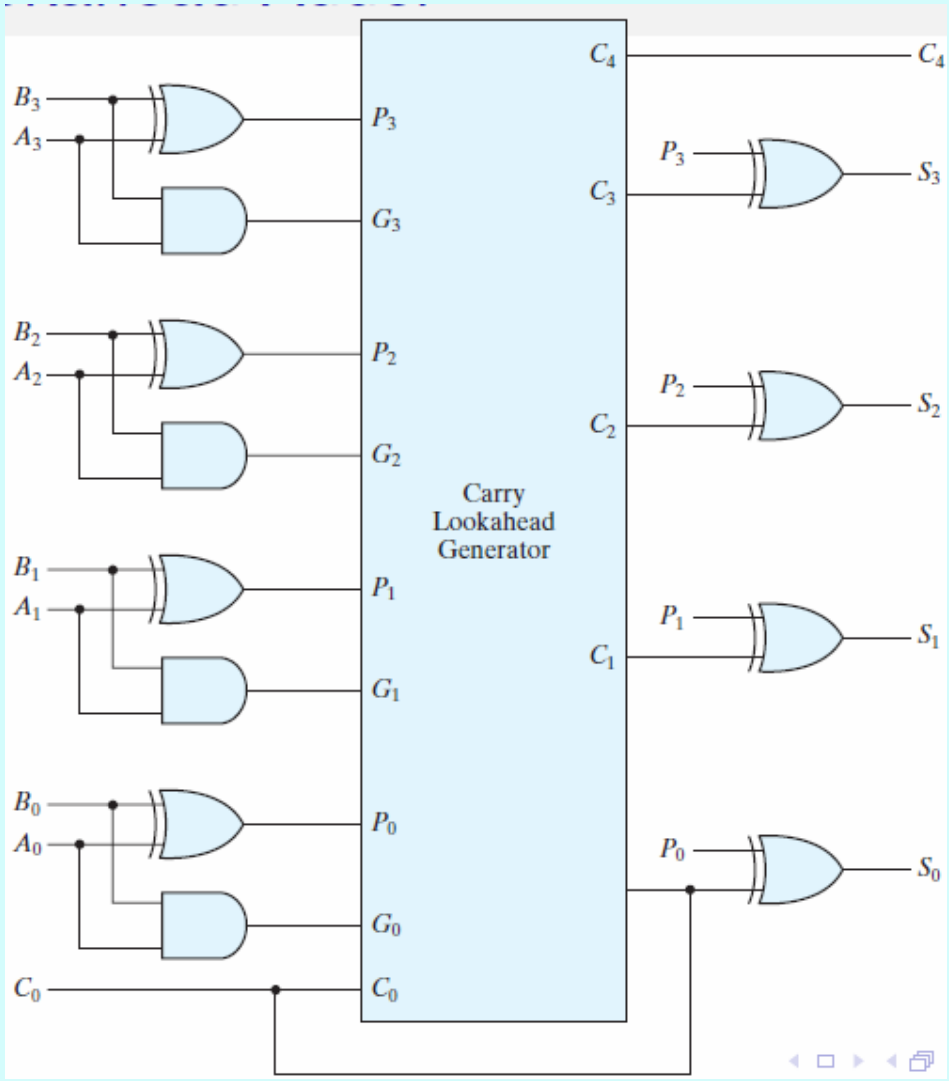


پیش‌بینی رقم نقلی (ادامه...)

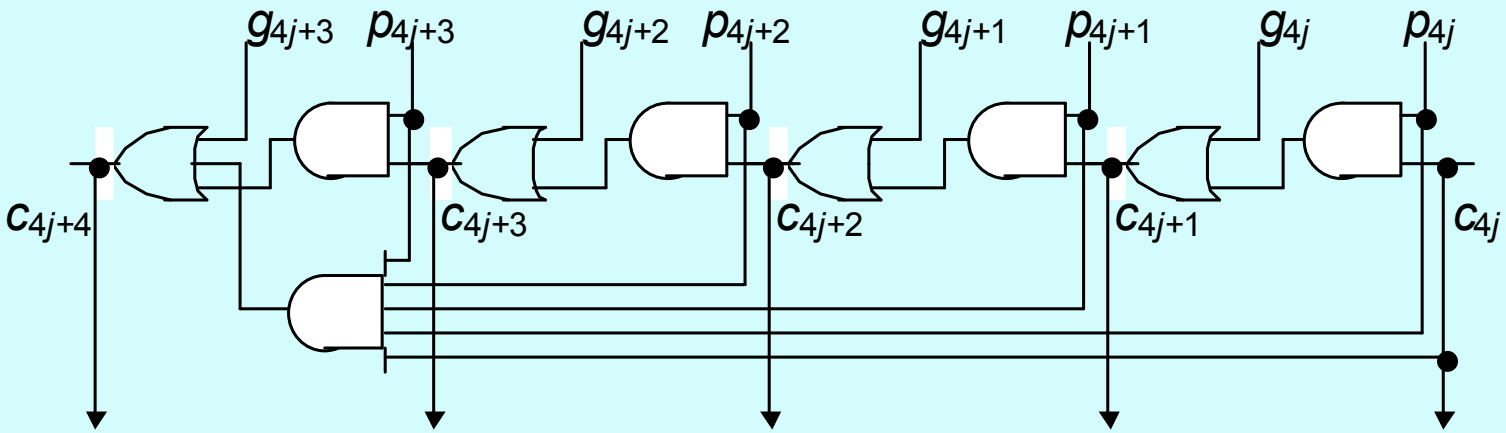
$$c_1 = g_0 + c_0 p_0$$
$$c_2 = g_1 + g_0 p_1 + c_0 p_0 p_1$$
$$c_3 = g_2 + g_1 p_2 + g_0 p_1 p_2 + c_0 p_0 p_1 p_2$$
$$c_4 = g_3 + g_2 p_3 + g_1 p_2 p_3 + g_0 p_1 p_2 p_3 + c_0 p_0 p_1 p_2 p_3$$

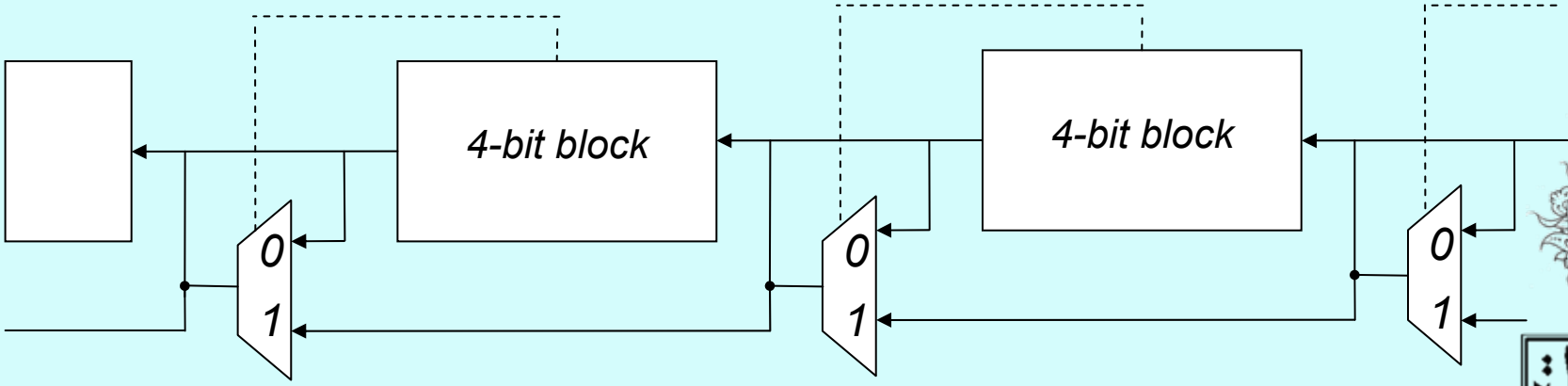


پیش‌بینی رقم نقلی (ادامه...)

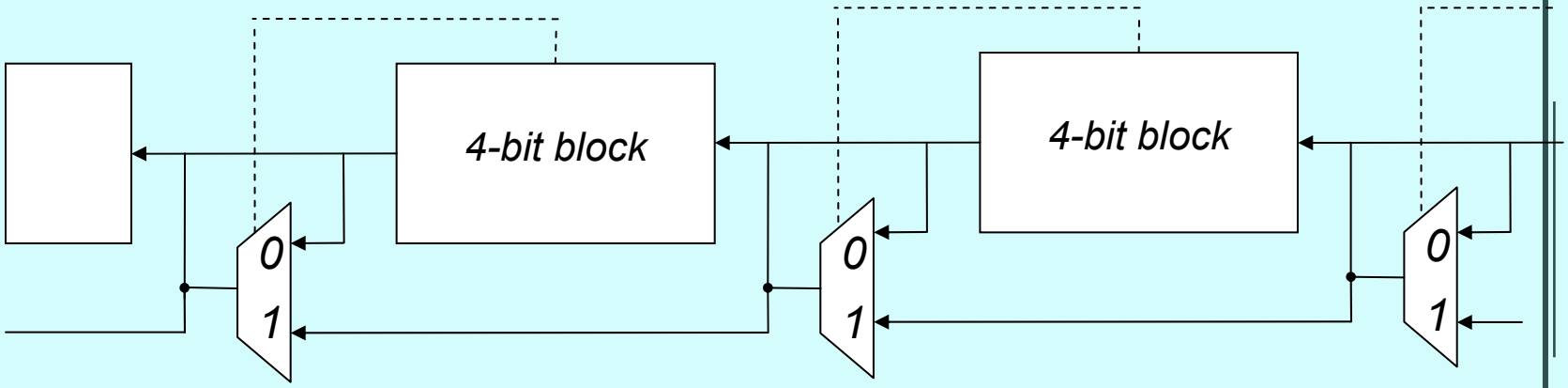


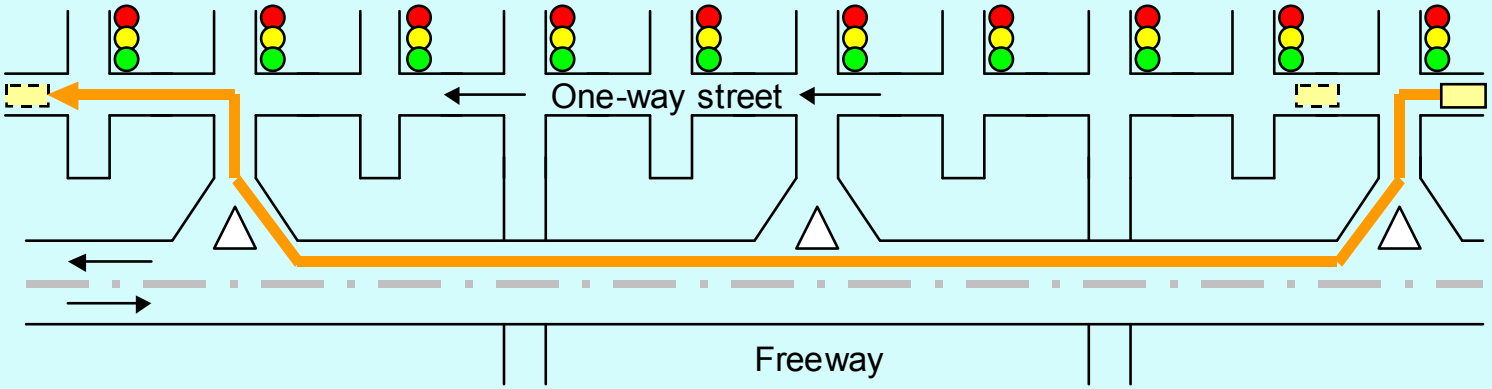
Carry Skip


g_{4j+3}
p_{4j+3}
g_{4j+2}
p_{4j+2}
g_{4j+1}
p_{4j+1}
g_{4j}
p_{4j}
c_{4j+4}
c_{4j+3}
c_{4j+2}
c_{4j+1}
c_{4j}

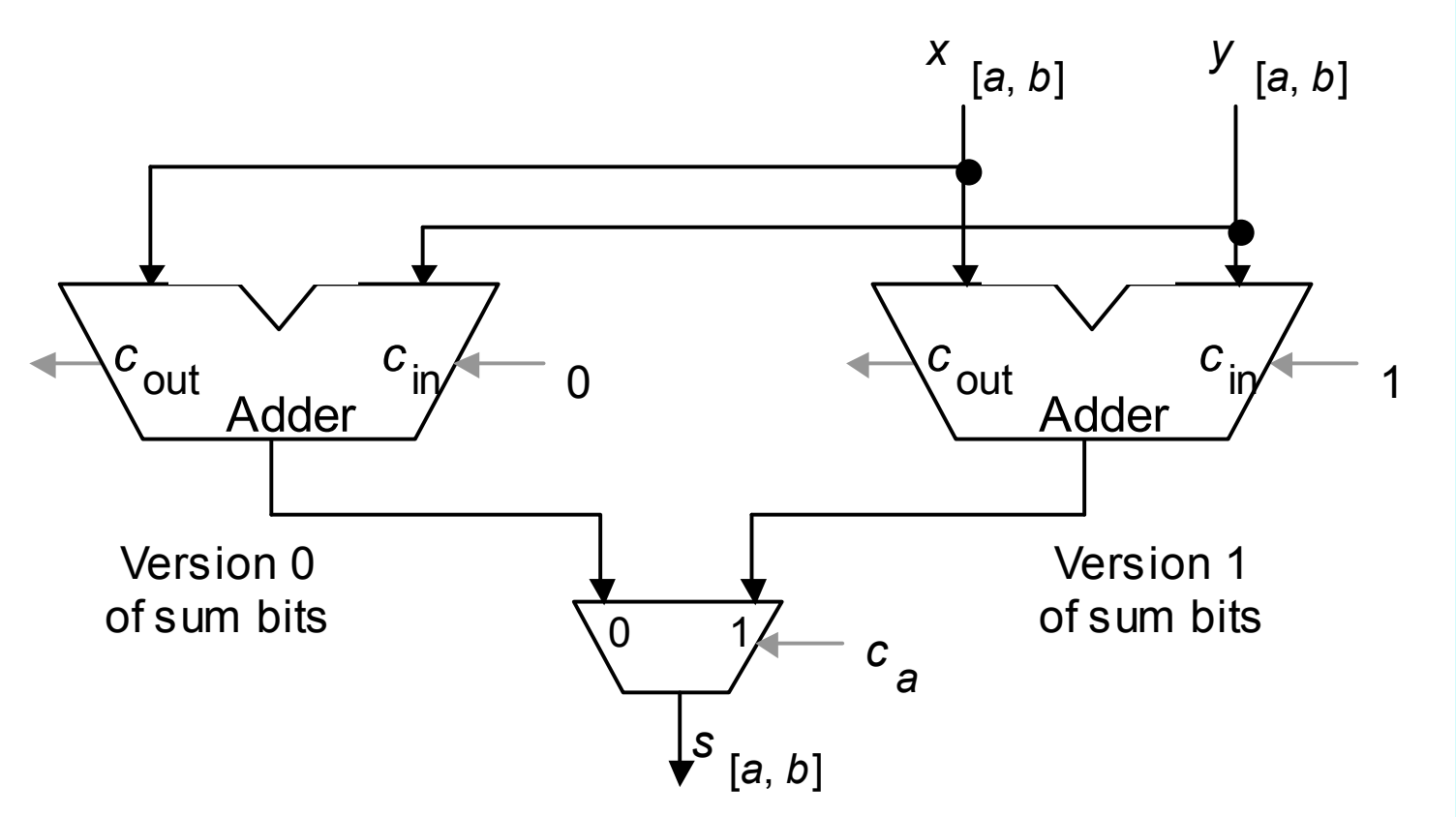

4-bit block
4-bit block
0
1
0
1
0
1

Carry Skip





Carry Select



معماری کامپیوتر (۱۳۹۱-۱۱-۱۳)
جلسه‌ی هشتم

دانشگاه شهید بهشتی
دانشکده‌ی مهندسی برق و کامپیوتر
بهار ۱۳۹۱
احمد محمودی ازناوه

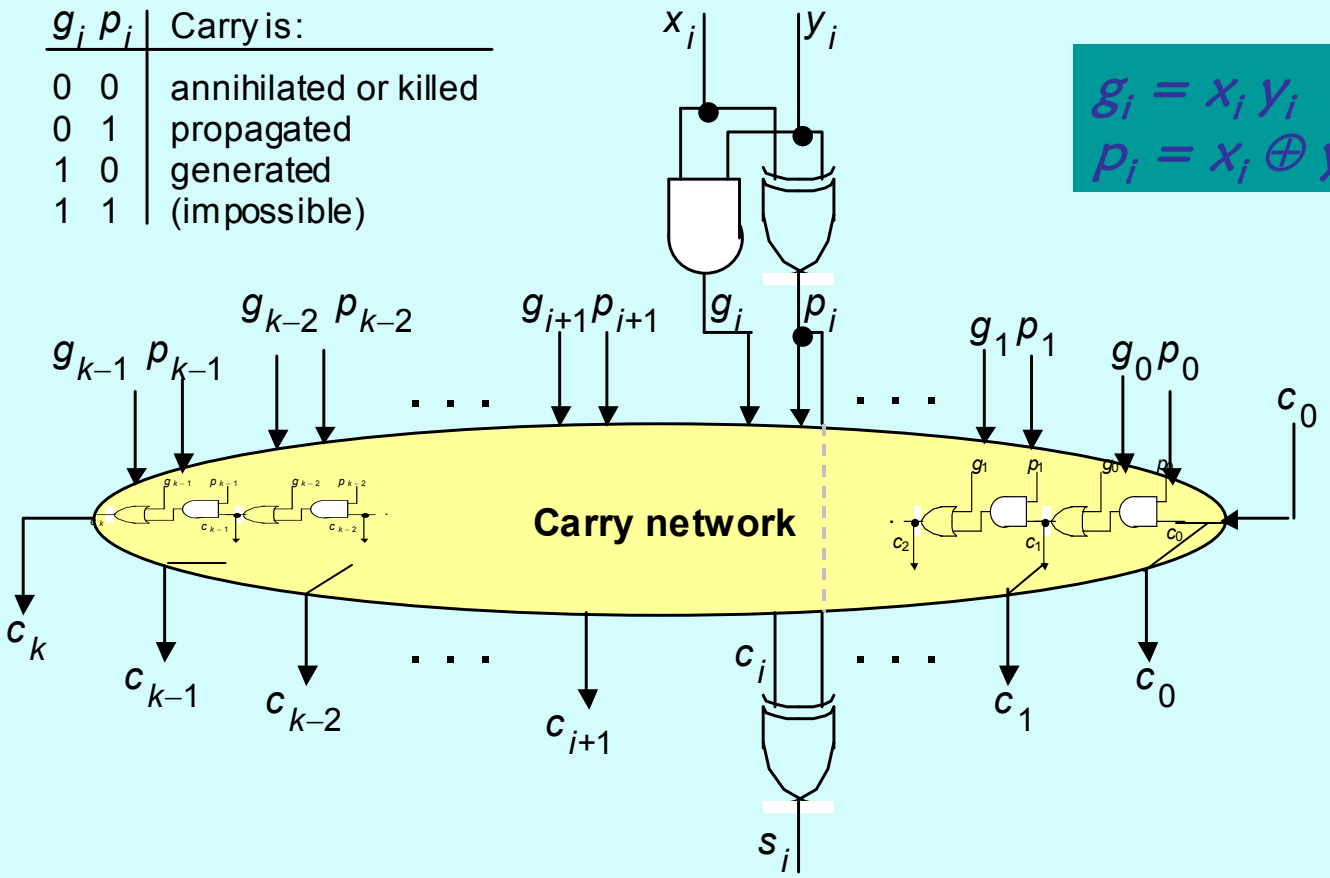
فهرست مطالب

- جمع‌کننده‌ها
  - جمع‌کننده‌های سریع
- جمع ده‌دهی

زنجیره‌ی انتقال رقم نقلی (ادامه...)

g_i	p_i	Carry is:
0	0	annihilated or killed
0	1	propagated
1	0	generated
1	1	(impossible)

$$g_i = x_i\, y_i$$
$$p_i = x_i \oplus y_i$$



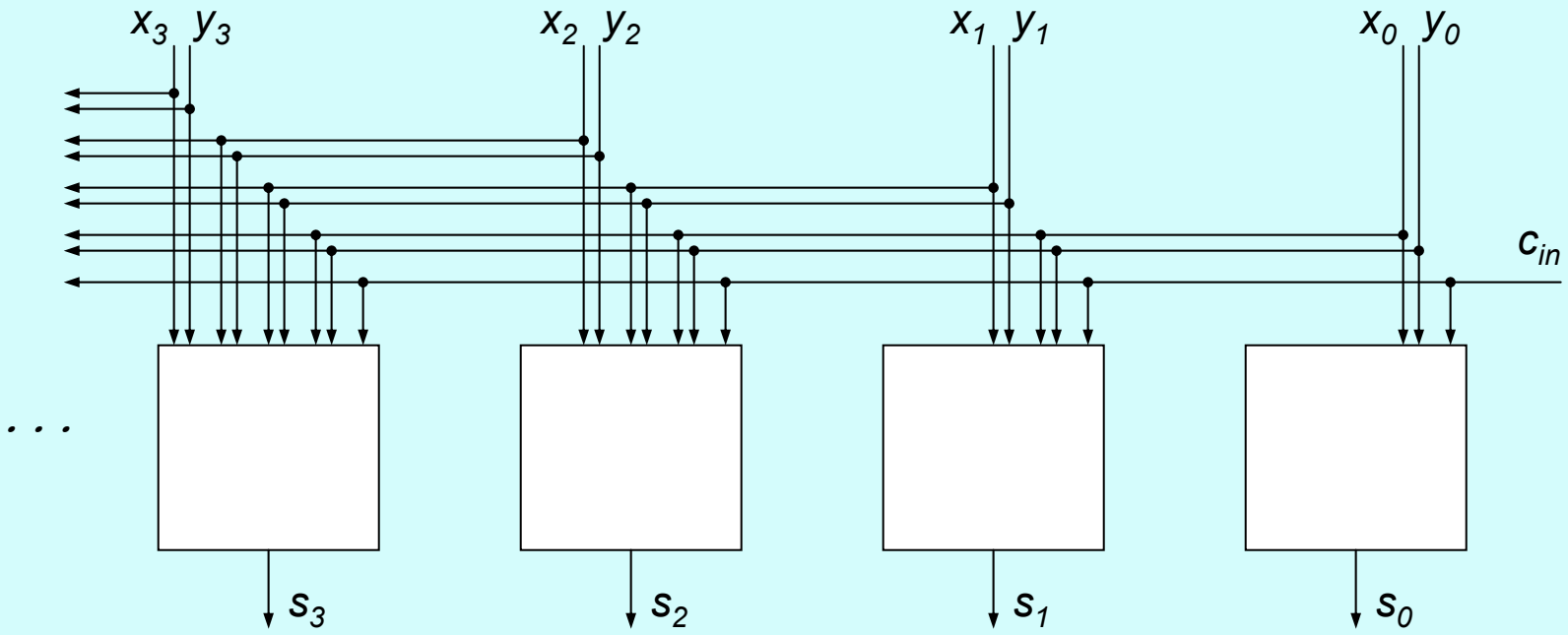
پیش‌بینی رقم نقلی (ادامه...)

$$c_1 = g_0 + p_0 c_0$$

$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$$

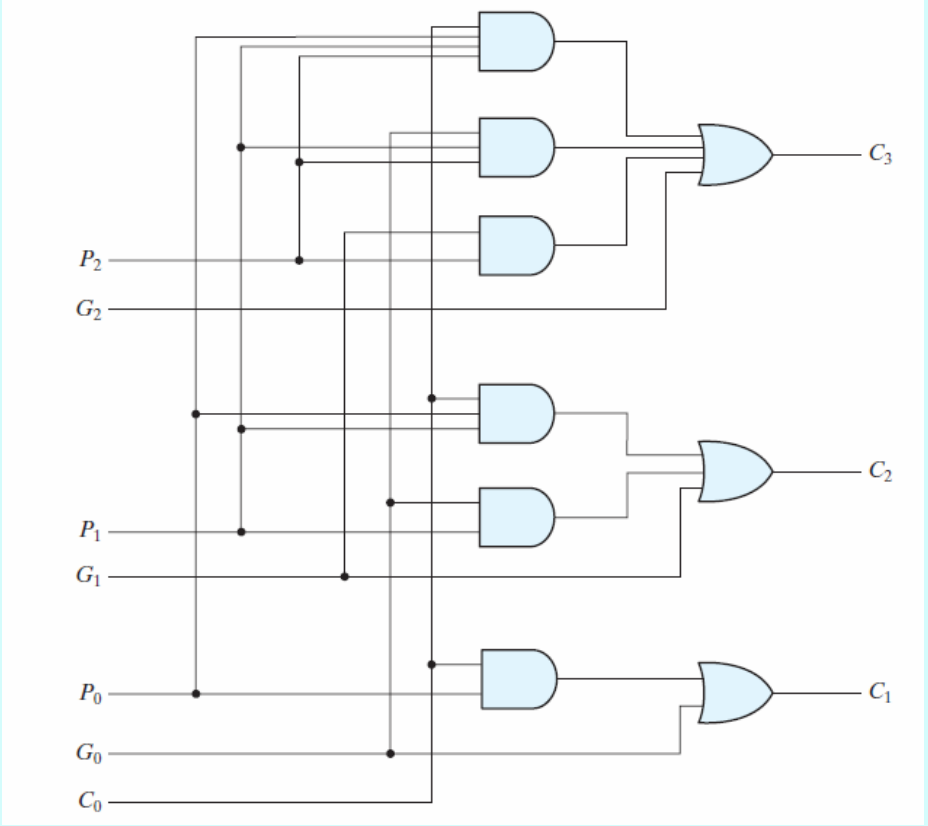
$$c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$$

$$c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$$

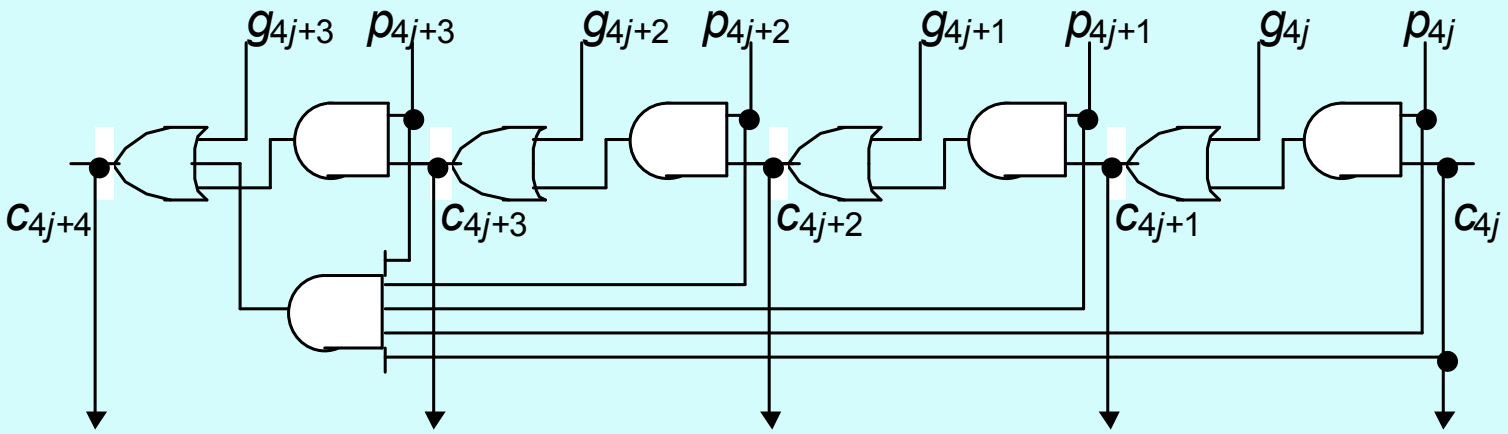


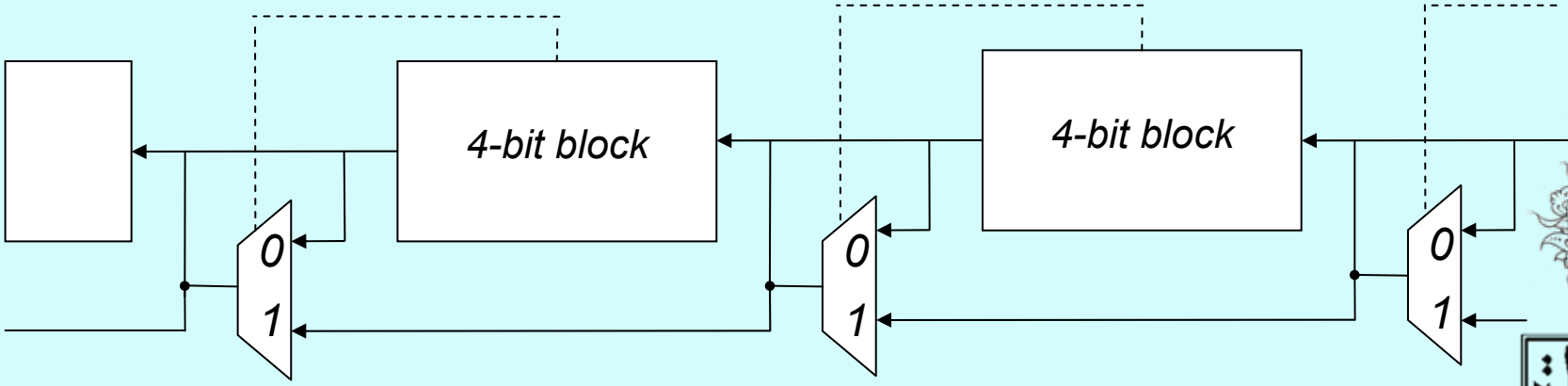
پیش‌بینی رقم نقلی (ادامه...)

$$c_1 = g_0 + c_0 p_0$$
$$c_2 = g_1 + g_0 p_1 + c_0 p_0 p_1$$
$$c_3 = g_2 + g_1 p_2 + g_0 p_1 p_2 + c_0 p_0 p_1 p_2$$
$$c_4 = g_3 + g_2 p_3 + g_1 p_2 p_3 + g_0 p_1 p_2 p_3 + c_0 p_0 p_1 p_2 p_3$$

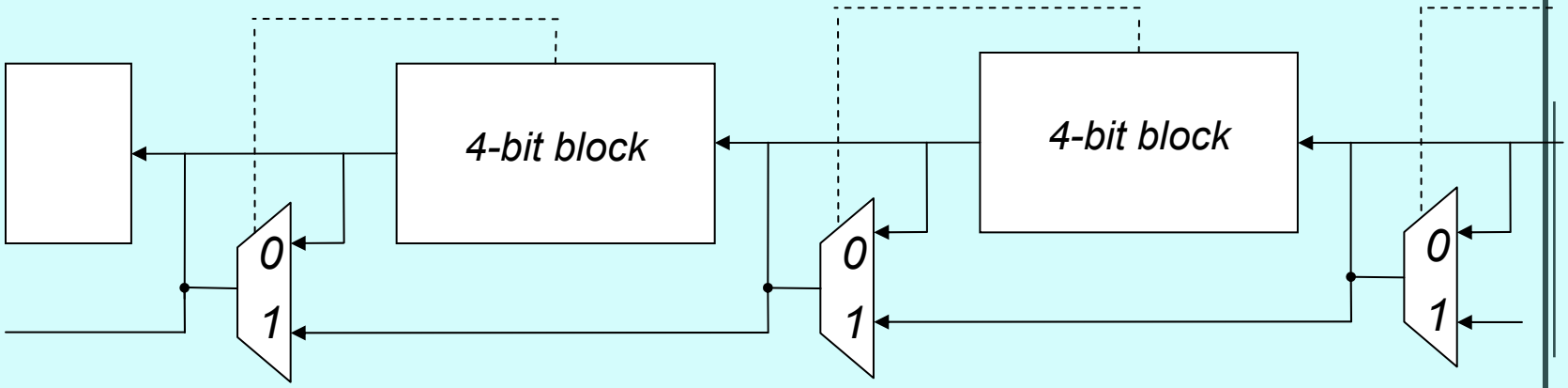


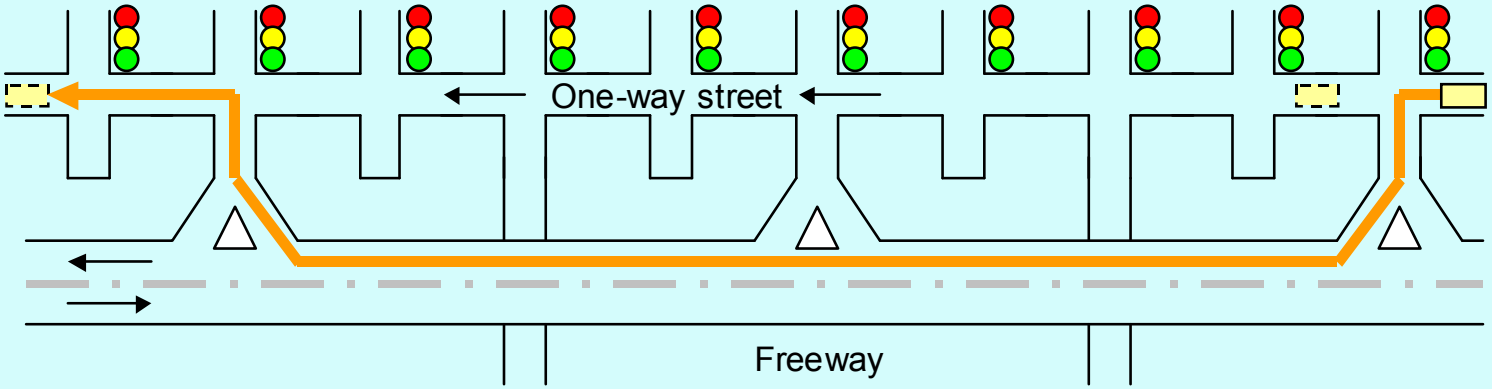
Carry Skip


g_{4j+3}
p_{4j+3}
g_{4j+2}
p_{4j+2}
g_{4j+1}
p_{4j+1}
g_{4j}
p_{4j}
c_{4j+4}
c_{4j+3}
c_{4j+2}
c_{4j+1}
c_{4j}

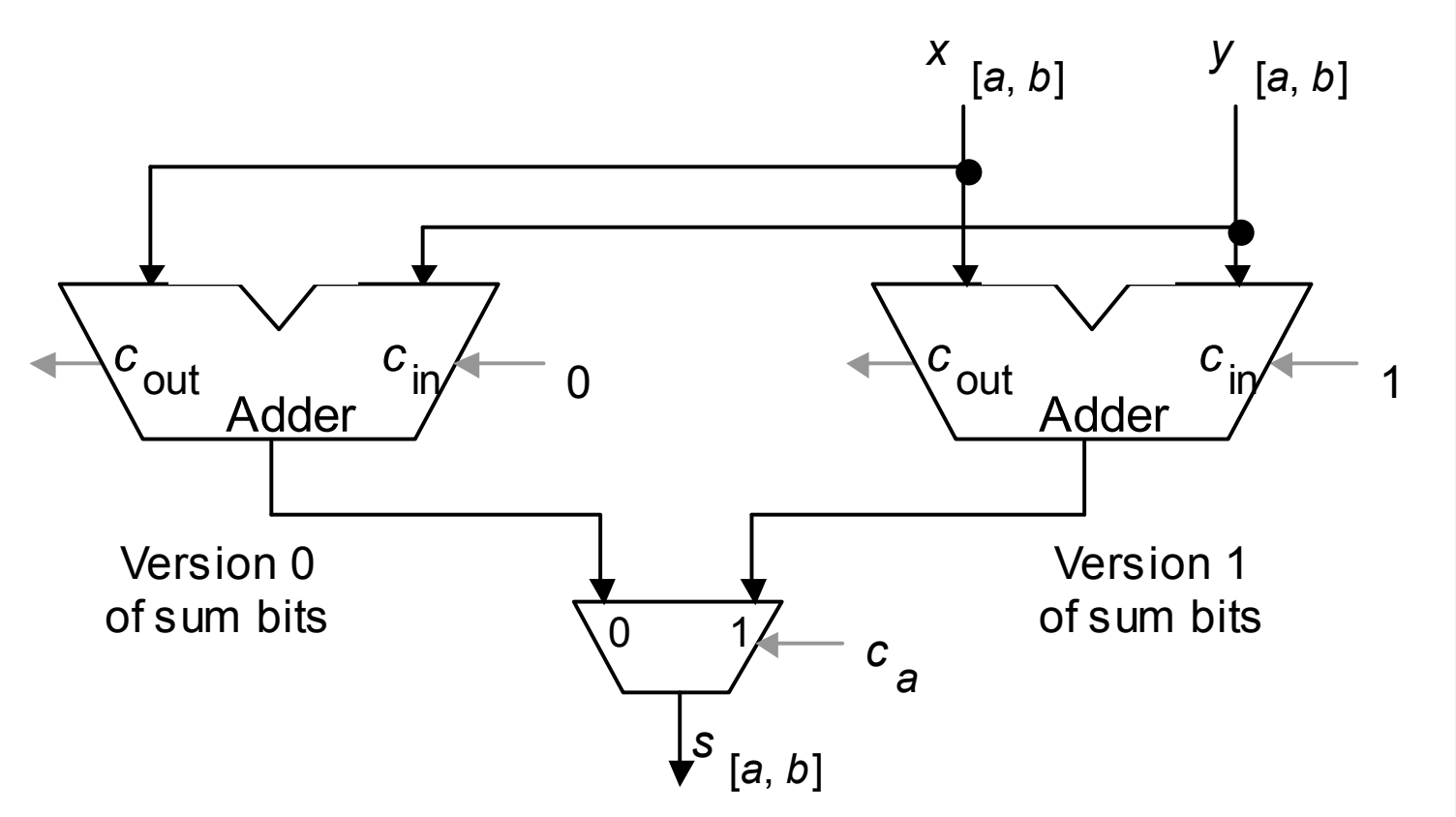

4-bit block
4-bit block
0
1
0
1
0
1

Carry Skip





Carry Select



- برخی زبان‌های برنامه‌نویسی، مانند C بروز سرریز را نادیده می‌انگارد.
  addu, addui, subu
- در برخی زبان‌ها، مانند fortran بروز سریز باعث ایجاد استثنا می‌شود.
- استثنا (یا وقفه)، پیشامدی برنامه‌ریزی نشده است که بروز آن باعث توقف اجرای روند عادی برنامه می‌شود.
  exception
  - به استثناهایی که منشا خارجی دارد، وقفه گفته می‌شود.
    interrupt
  add, addi, sub

نمایش BCD

- در برخی کاربردها دسترسی به ارقام دهدهی به صورت مجزا اهمیت دارد.
- با توجه به این که برخی اعداد در مبنای ده نمایش دقیق دارند، قابل نمایش به صورت دودویی نیستند، لازم است برای نمایش دقیق آن‌ها تدبیری اندیشیده شود.
- در این نمایش هر رقم دهدهی با
- چهار بیت نمایش داده می‌شوند.

Decimal	BCD Code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

BCD (ادامه...)

$(12)_{10}$

1100

نمایش در مبنای ۲

00010010

استفاده از شیوه‌ی نمایش BCD

$$\begin{array}{r} 9 \\ +\ 3 \\ \hline 12 \end{array}$$

1

carry

جمع BCD

Number	C	S8	S4	S2	S1
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1

جمع BCD (ادامه...)

Number	C	S8	S4	S2	S1
10	1	0	0	0	0
11	1	0	0	0	1
12	1	0	0	1	0
13	1	0	0	1	1
14	1	0	1	0	0
15	1	0	1	0	1
16	1	0	1	1	0
17	1	0	1	1	1
18	1	1	0	0	0
19	1	1	0	0	1

جمع BCD (ادامه...)

BCD adder sum

Number	C	S8	S4	S2	S1
10	1	0	0	0	0
11	1	0	0	0	1
12	1	0	0	1	0
13	1	0	0	1	1
14	1	0	1	0	0
15	1	0	1	0	1
16	1	0	1	1	0
17	1	0	1	1	1
18	1	1	0	0	0
19	1	1	0	0	1

+6

Binary sum

K	Z8	Z4	Z2	Z1
0	1	0	1	0
0	1	0	1	1
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1

الگوریتم جمع BCD

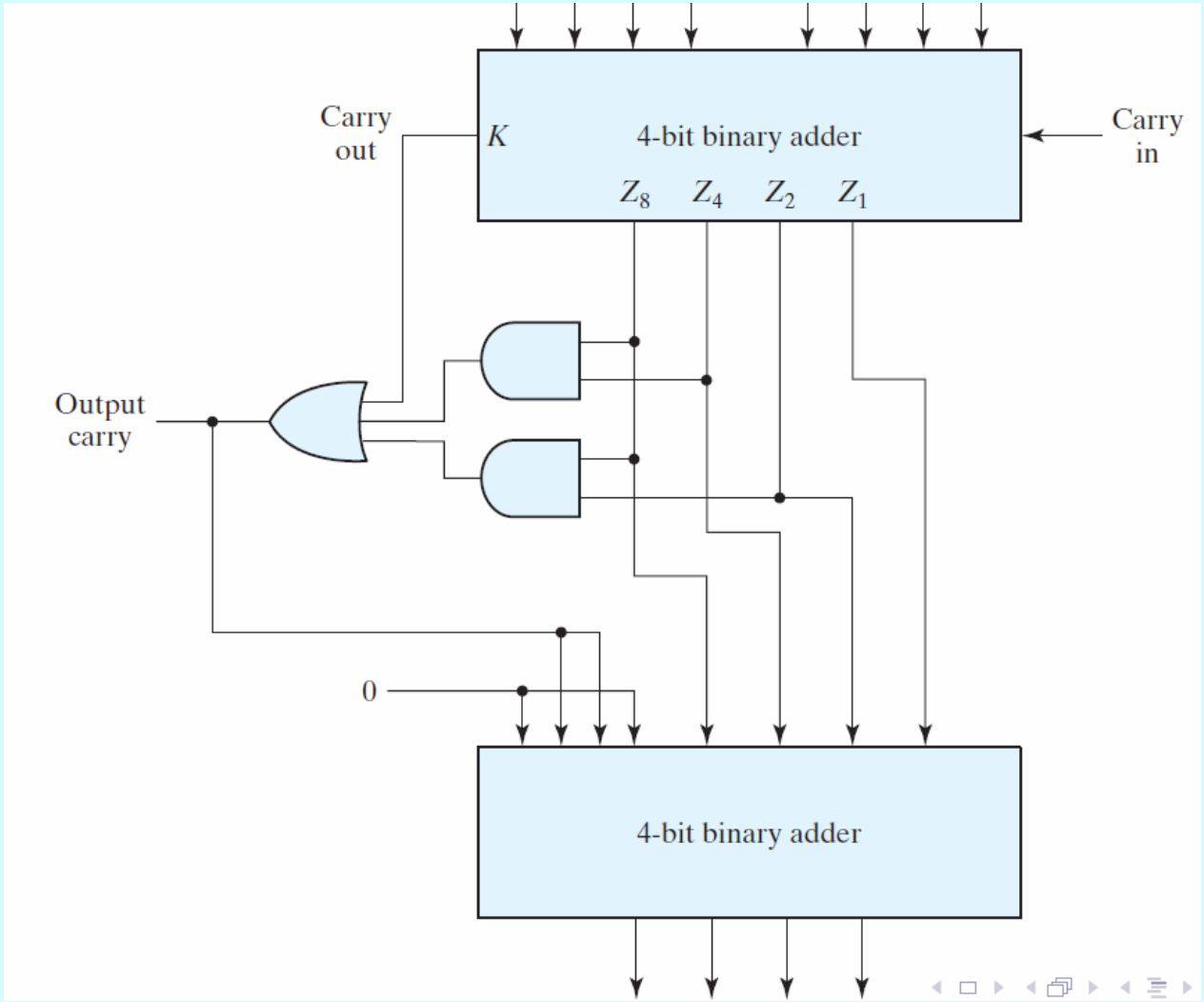
- If sum is up to 9
  - Use the regular Adder.
- If the sum > 9
  - Use the regular adder and add 6 to the result

اصلاحات مورد نیاز

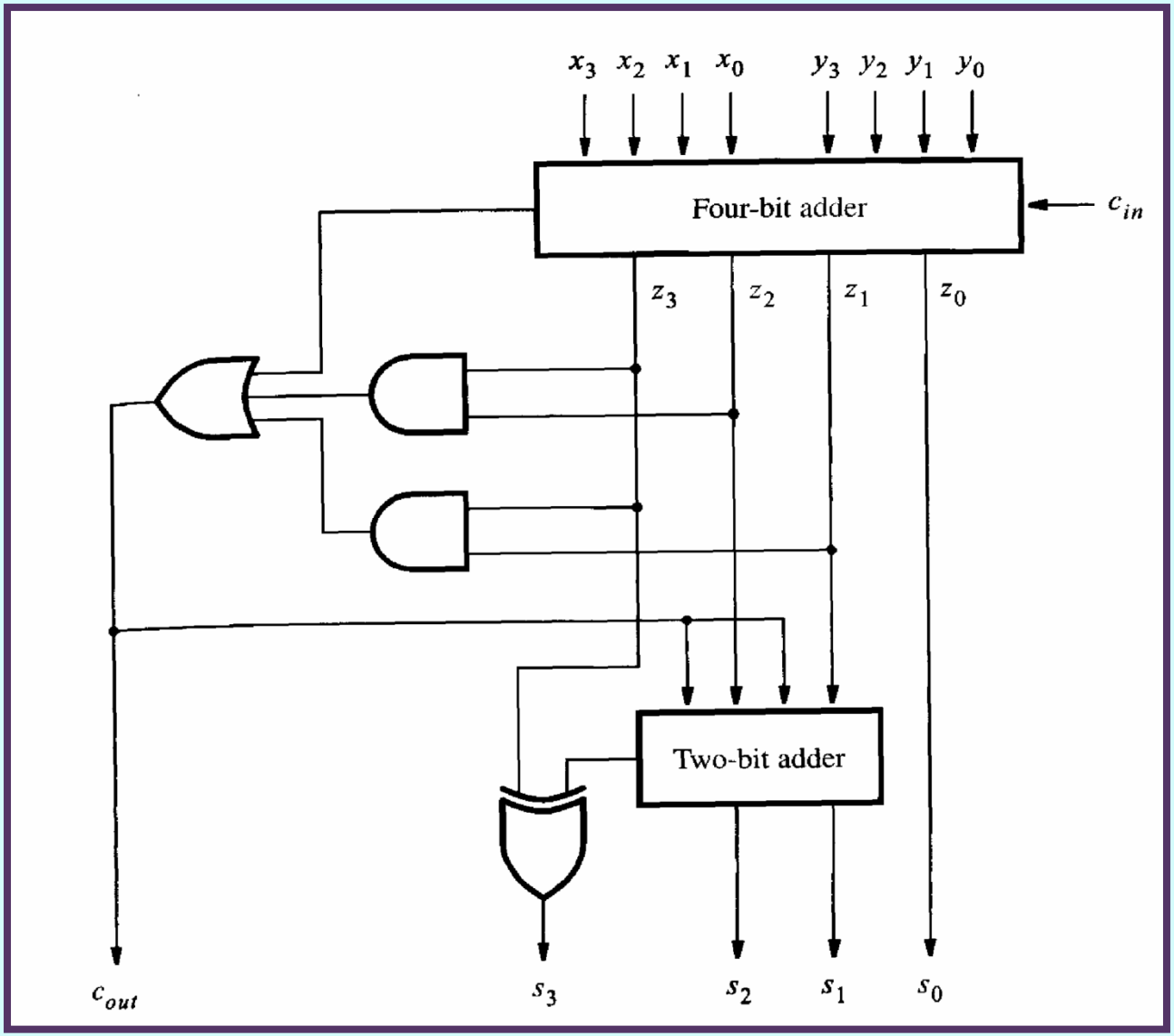
Number	K	Z8	Z4	Z2	Z1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0
19	1	0	0	1	1

$$C = K + Z_8.Z_4 + Z_8.Z_2$$

جمع BCD (ادامه...)



جمع BCD (ادامه...)



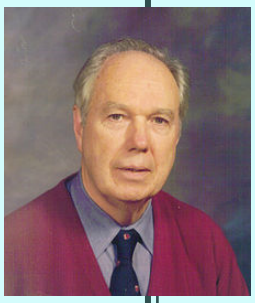
مدارهای حساب برای چندرسانه‌ای

- در کاربردهای چند رسانه‌ای، معمولا عملگر یکسانی روی بردارهایی شامل داده‌های هشت یا شانزده‌بیتی اعمال می‌شود.
- در صورتی که یک جمع‌کننده‌ی شصت‌وچهار بیتی در اختیار داشته باشیم،

  در عمل می‌توان هشت جمع هشت‌بیتی انجام داد.

  – هزینه‌ی چنین کاری شکستن زنجیره‌ی انتشار رقم نقلی است. در واقع یک دستورالعمل بر روی چند داده، اجرا می‌شود. این نوع اجرای موازی به SIMD معروف است.

طبقه‌بندی Michael J. Flynn

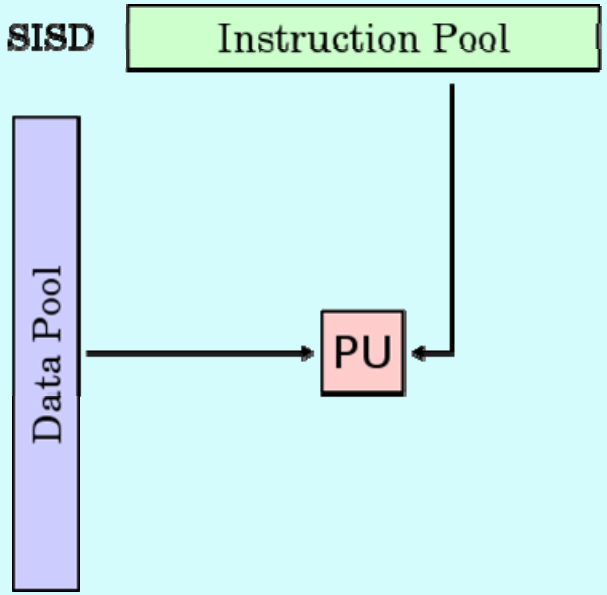


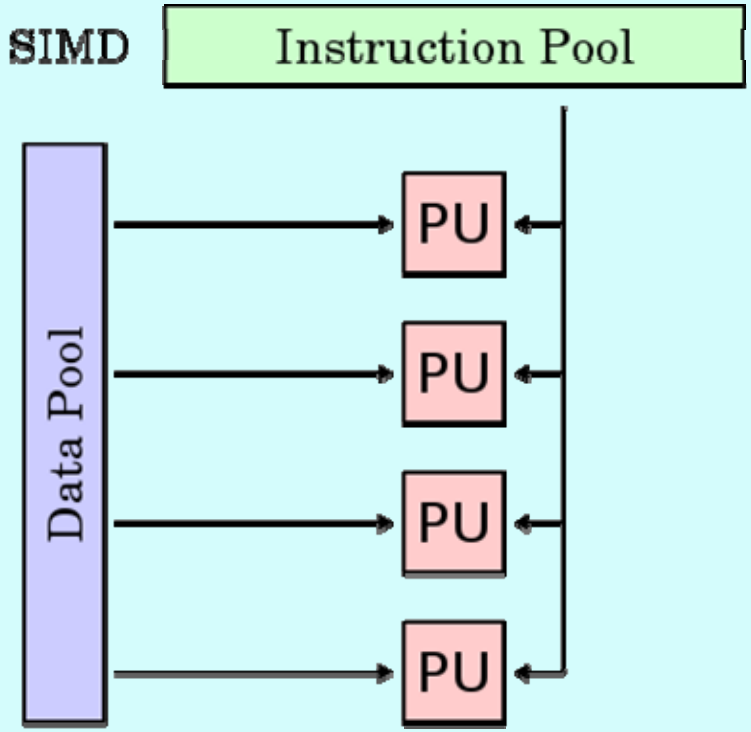
این طبقه‌بندی برای انواع معماری در سال ۱۹۹۶ توسط آقای Flynn پیشنهاد شده است، بر اساس این دسته‌بندی چهار نوع معماری وجود دارد:

- Single Instruction, Single Data stream (SISD)
- Single Instruction, Multiple Data streams (SIMD)
- Multiple Instruction, Single Data stream (MISD)
- Multiple Instruction, Multiple Data streams (MIMD)

طبقه‌بندی Michael J. Flynn (ادامه...)

کامپیوترهای تک پردازنده‌ی معمولی (کامپیوترهای قدیمی)

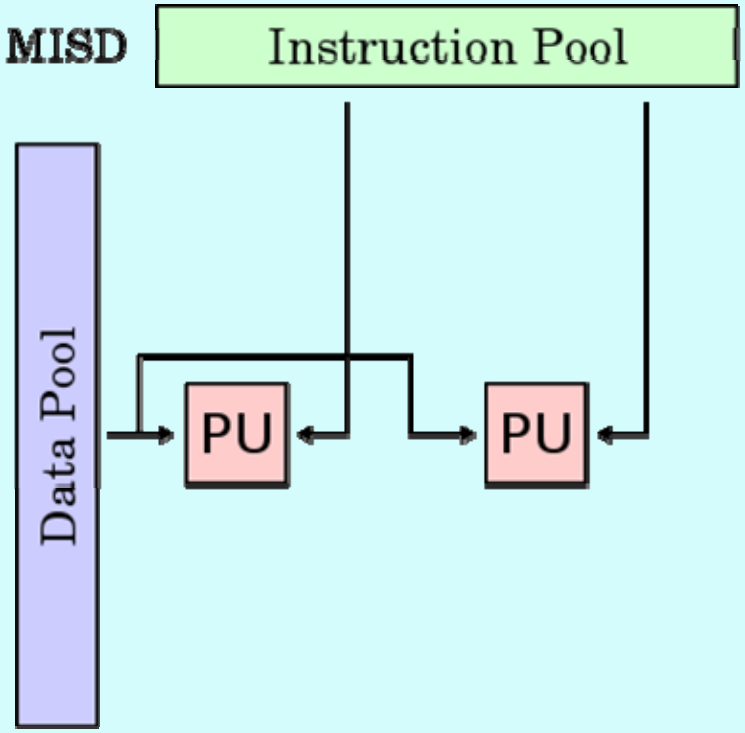


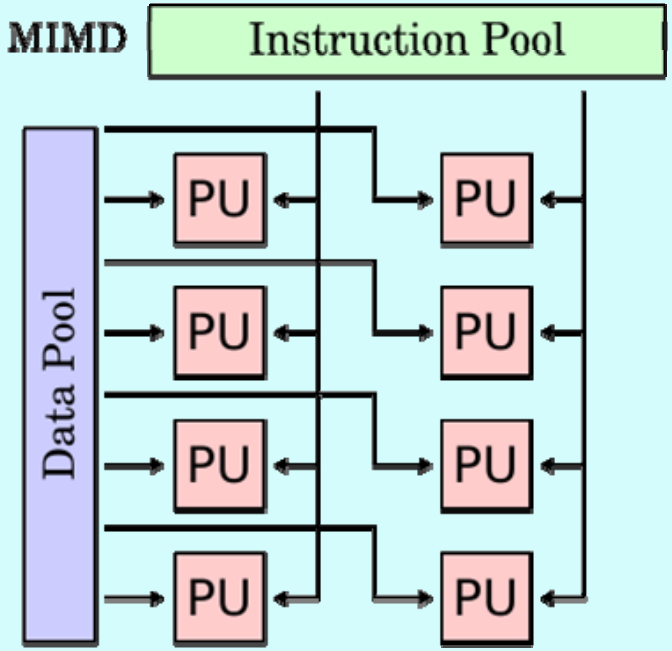


پردازنده‌های آرایه‌ای و GPU

طبقه‌بندی Michael J. Flynn (ادامه...)

معماری رایجی نیست، برخی منابع مطرح شدن چنین دسته‌ای را به دلیل کامل بودن طبقه‌بندی بیان نموده‌اند.
با این حال سیستم‌های تحمل‌پذیر در برابر خطا و حتی خط لوله از نمونه‌های این نوع معماری بیان شده است





سیستم‌های توزیع شده، نمونه‌ای از این معماری است

Saturating Operations

- **عملگرهای اشباع‌کننده**
  - در صورت سریز، به جای جمع پیمانه‌ای، حاصل به بزرگ‌ترین عدد مثبت و یا کوچک‌ترین عدد منفی تبدیل می‌شود.

Instruction category	Operands
Unsigned add/subtract	Eight 8-bit or Four 16-bit
Saturating add/subtract	Eight 8-bit or Four 16-bit
Max/min/minimum	Eight 8-bit or Four 16-bit
Average	Eight 8-bit or Four 16-bit
Shift right/left	Eight 8-bit or Four 16-bit

ضرب

مضروب

multiplicand → 1000

multiplier → × 1001

مضروب فیه ضرب کننده

1000

0000

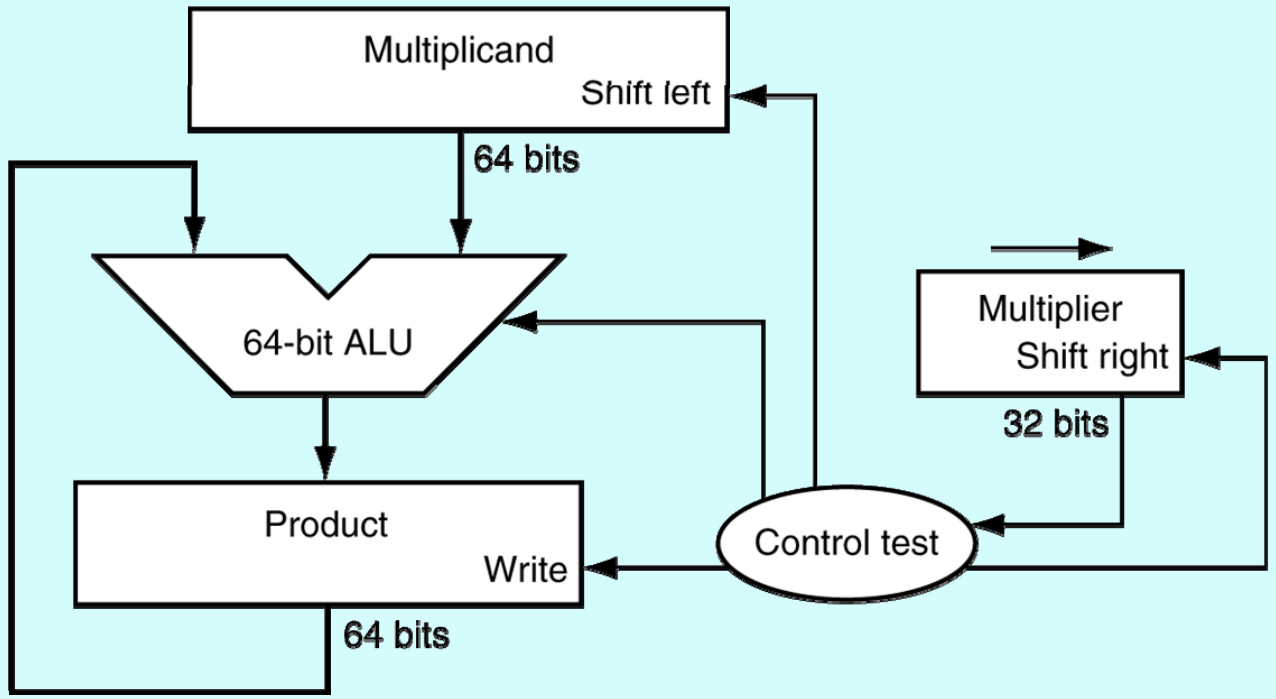
0000

1000

product → 1001000

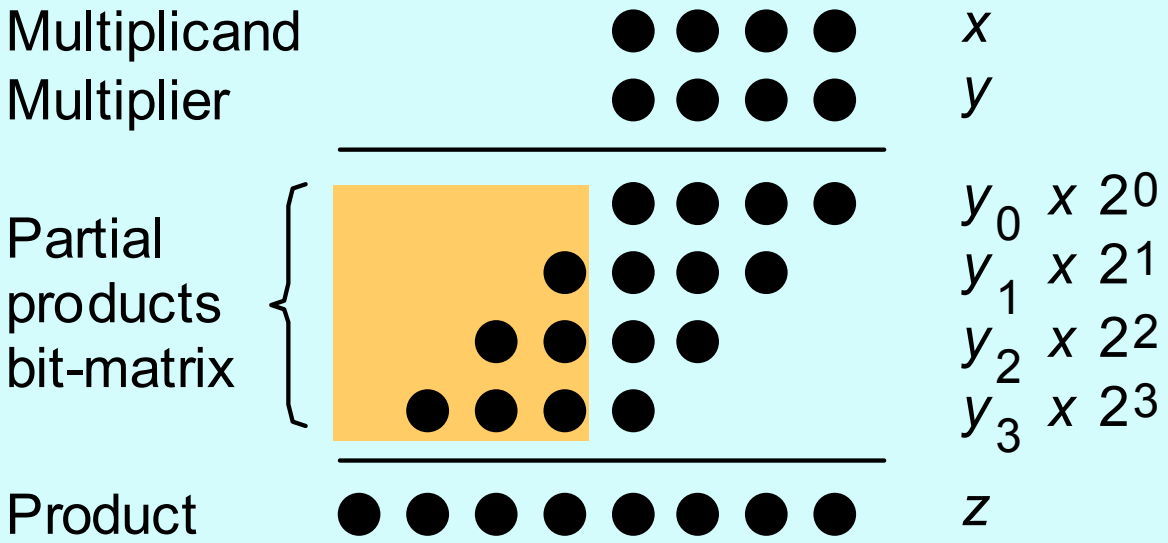
حاصل ضرب

طول حاصلضرب برابر است با جمع طول عملوندها



Dot Notation

ضرب (ادامه...)

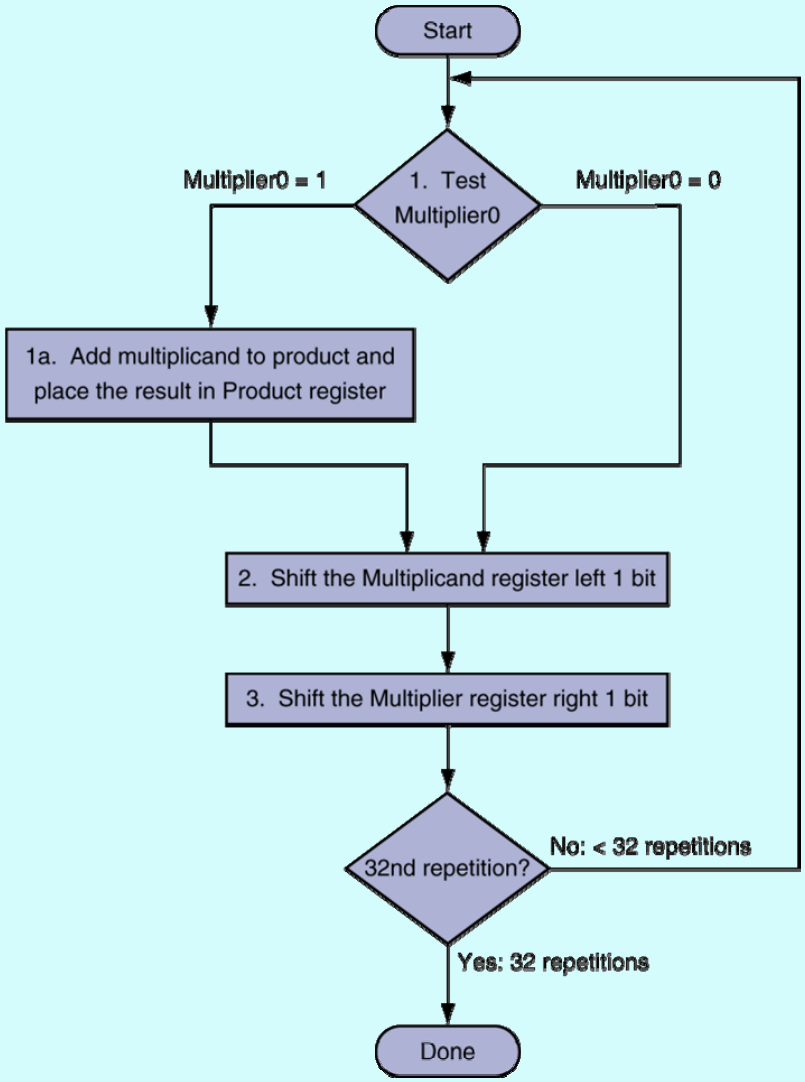


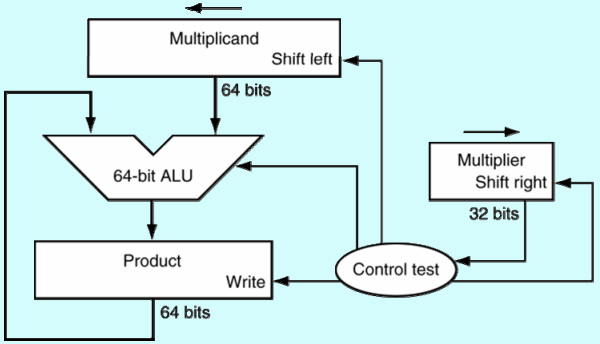
$$z^{(j+1)} = (z^{(j)} + y_j \times 2^k)\, 2^{-1} \quad \text{with } z^{(0)} = 0 \text{ and } z^{(k)} = z$$

$|$—— *add* ——$|$

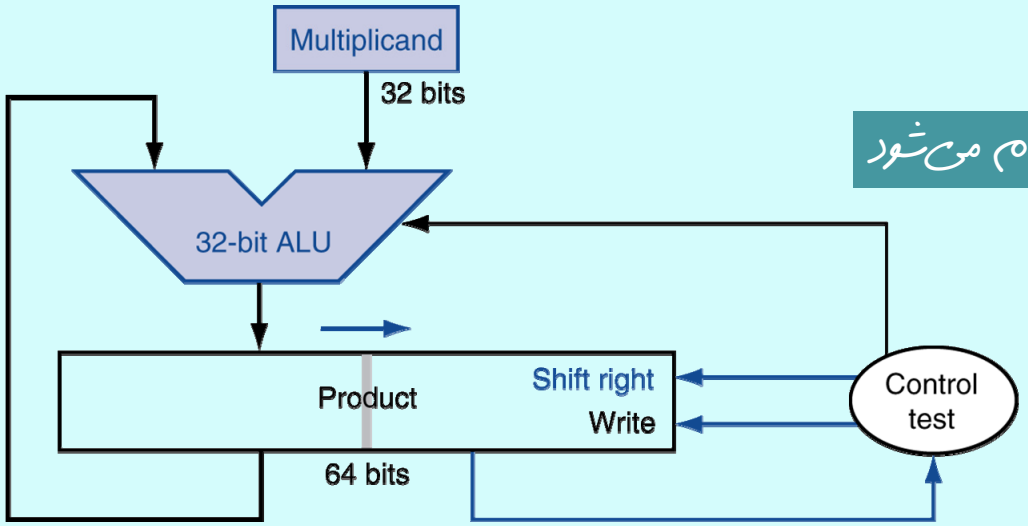
$|$— *shift right* —$|$

سخت افزار ضرب



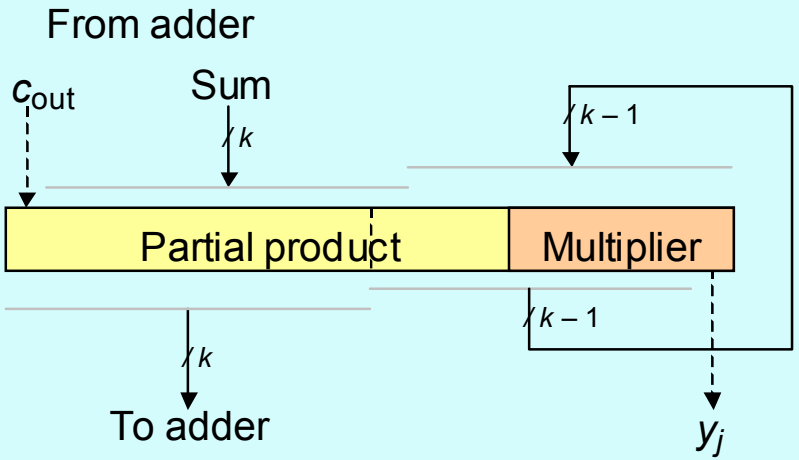


ضرب بهینه‌سازی‌شده

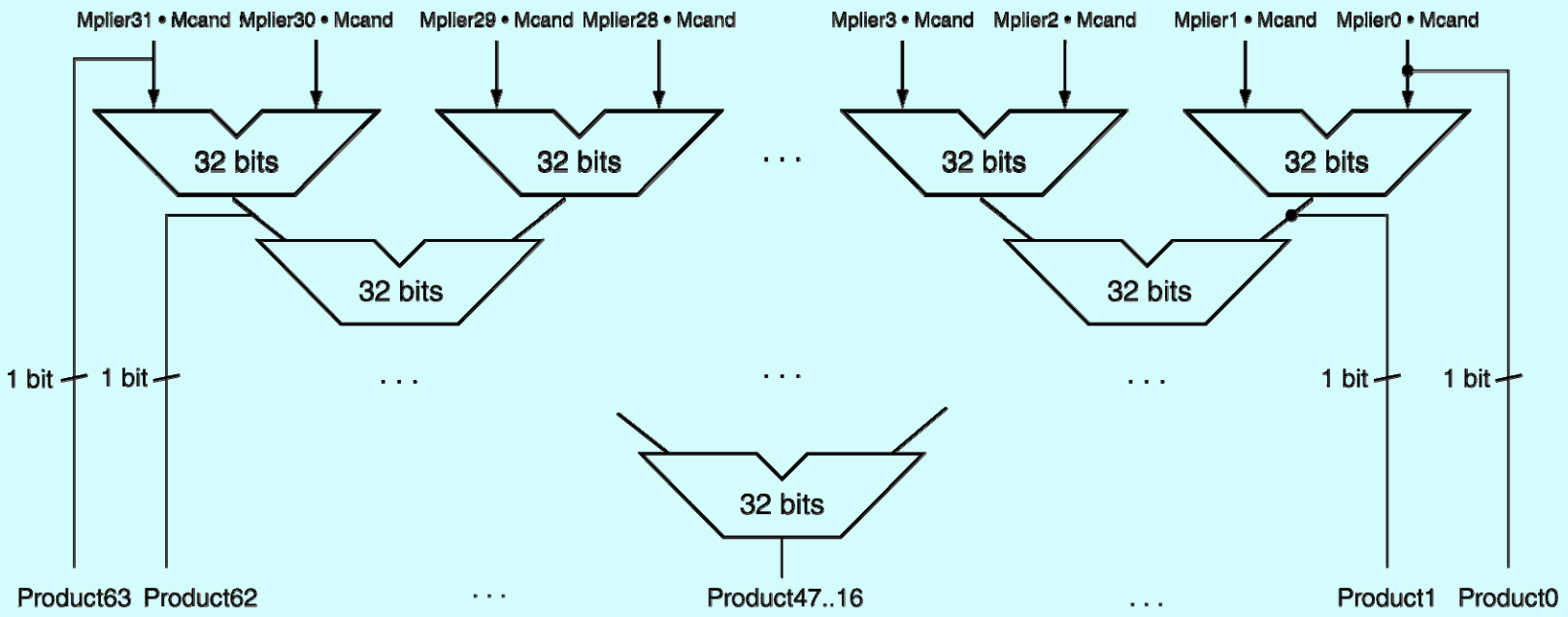


جمع و ضرب به صورت سری انجام می‌شود

در صورتی که تعداد عملیات ضرب کم باشد، چنین مداری کفایت می‌کند.



ضرب‌کننده‌های سریع



به صورت خط لوله قابل استفاده می‌باشد