Performance chp. 2

t - MIPS - FLOPS

wall-clock = elapsed time, زمان بین شروع تا پایان کار

1) Response time = Run time = Execution time ≠ CPU time (2)

(CPU time + I/O wait + ... = elapsed Time = Response)

vs. Throughput = تعداد کار / واحد زمان = توان عملیاتی

System Performance $\propto \frac{1}{x}$ vs. CPU Performance (Performance default) $\propto \frac{1}{x}$

$f_{(Hz)} = \frac{1}{T_{(sec)}}$ → دوره تناوب کلاک (clock Cycle Time)

$f = 100\,Hz$ → ۱۰۰ کلاک در ثانیه

$T_{(sec)} \times \#of\,clock_{(-)} = t_{(sec)}$

3) CPI = Clock Per Instruction (average) — متوسط کلاک برای هر دستور

K	3	m
M	6	μ
G	9	n
T		p

4) SpeedUp = $\frac{t_A}{t_B}$ (تسریع B نسبت A)

4) MIPS = Million Instruction Per Second

$$MIPS\left(\frac{M}{sec}\right) = \frac{\#I}{t_{(s)} \cdot 10^6} = \frac{\#}{1\mu s} = \frac{f_{(Hz)} \times 10^{-6}}{CPI} = \dots$$

($t = T \times clk$، بر حسب CPI و f)

$\frac{f}{CPI} \to \frac{\#}{1s \times 10^6}$

5) مثال اصل ۷۲ پوران

5) کلی هر چی متوسط کلاک لازم دستورات (CPI) بیشتر باشه ← ↓MIPS ⇏ ↓Perfor (احتمالاً) (۳) طبق !! نه همیشه

برنامه ۱۰۰ کلاک لازم داره، $f = 50\,H$ → t_{Run} → $t = \frac{100}{50} = 2_{sec}$

But ولی اگه ۲ تا کد واسه یک برنامه اجرایی تولید شه :

مثال
A : کلی دستورات زیاد ولی شامل دستورات کلاک پایین
B : اندکی دستورات کم ولی شامل دستورات کلاک بالا

$MIPS_A > MIPS_B \not\Rightarrow t_A < t_B$ (نه لزوماً) $\xrightarrow{\text{dic:}}$ ممکنه دستورات A ∞ تا باشه و هیچوقت اجرای A تموم نشه !!

5) اشکالات MIPS
1) به برنامه بستگی داره . ۲ تا برنامه روی یه PC ، ۲ تا MIPS متفاوت می‌ده
2) قابلیت دستورات را در نظر نمی‌گیره . (با CPI کار داره که متوسط این همه دستورات است) ← برای مقایسه ۲ تا PC با حجم دستورات مختلف چه ساختی !
3) ممکن است MIPS و کارایی عکس هم بشن

$\{$ MFLOPS = Million Floating Point Per Second (× ÷ + −)

G ... = میلیارد

* تعداد اعمال ممیز شناور یک برنامه خاص روی ۲ تا PC مختلف ثابت می‌مونه ⟸ MFLOPS بهتر از MIPS !

⊖ باز هم به برنامه بستگی داره ⊖ برای PC های دانشمندان کاربرد داره صرفاً (برنامه‌های معمولی معمولاً MFLOPS=0)

⊕ [در محاسبات ریاضی کاربرد دارد ... دانش بازنشسته ... ماتریس تعداد عملیات ممیز شناور ... معلوم چیه]

$$Speed\text{-}up = \frac{t_{old}}{t_{new}} \rightarrow S = \frac{12}{8} \times 100 = 150\%$$ (نسبت به قبل)

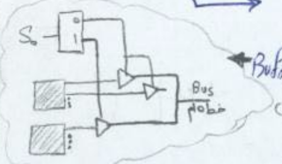
$$= \frac{t_{old} - t_{new}}{t_{new}} \times 100\ \% \rightarrow S = \frac{12-8}{8} \times 100 = \%50$$ (تغییرات)

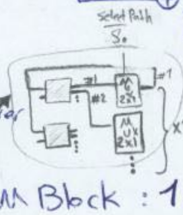
10^9 بیلیون
10^6 میلیون M
10^3 کیلو K
10^{-3} میلی m
10^{-6} میکرو μ

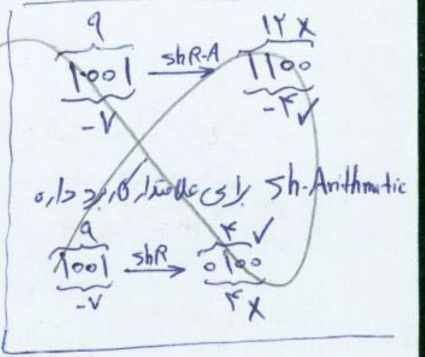
chp3

Arch
- Instruction Set → whatever binary-Programmer need to know [AMD = Intel]
- Hardware → Programmer don't care [AMD ≠ Intel]

HW Path
- Point-to-Point
  - (−) too Much wire
  - (+) { simultinous, stronger }
- Bus
  - Buffer
  - Multiplexer



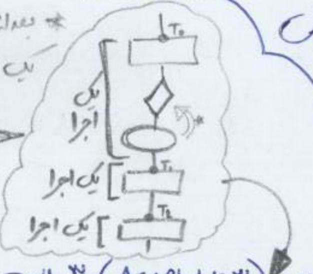




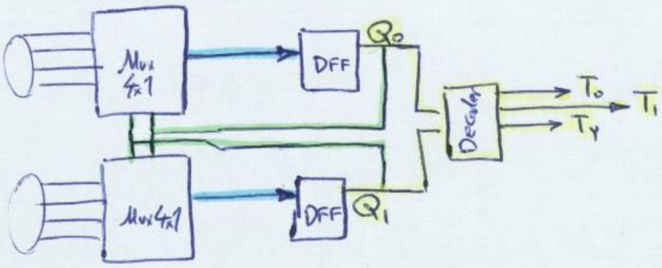
شیفت حسابی رو برای علامت‌دار استفاده کن
و شیفت معمولی رو برای بی علامت
تا ÷2 و ×2 درست کار کنه!

ASM chart
- ASM Block : 1 clock (بمانه که توجه حالت برای جنبه یا ویژه نیست)
- واحد کنترل —activate→ [ASM Block (by $T_0, T_1, \ldots$) / یعنی 1 state]

طراحی واحد کنترل
- Decoder + DFF (I)
- one-shot : one DFF per state (II)



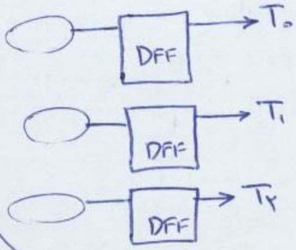
مثال → ۳ حالت (ASM Block تا ۳)



(I) ۳ حالت ← ۲ تا DFF

فعلی $Q_1 Q_0$	بعدی $Q_1 Q_0$	شرط
[بر اساس ASM chart]		[بر اساس ASM chart]

مثال در 8.9 یوران



(II) ۳ حالت ← ۳ تا DFF

دستور RTL درست

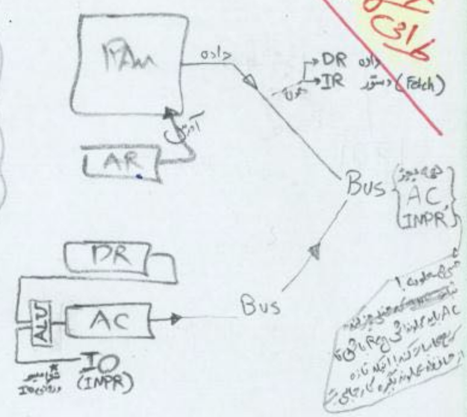
P+9: $R_1 \leftarrow R_1 + R_2$, $R_2 \leftarrow R_1$

صفحه 8.9 یوران

← می شه همزمان که از R می‌خونی، توشم چیزی بنویسی ✓

۴ بیتی Buffer →

Chp4 طراحی کامپیوتر پایه



F { T_0) AR ← PC

  { T_1) IR ← M[PC] , PC++   (AR)

D { T_2 : I ← IR(15) , D_{7-0} ← IR(14-13-12), AR ← IR(0-11)   (تجزیه IR لود شده)

(آدرس موثر) I T_3 : AR ← M[AR]

Effective Addr — دستورات حافظه‌ای (اگر I=1 بود و نبود T_4 ...)

Execute — دستورات شاخی و IO

1/2

ساخت کامپیوتر پایه (6 رجیستر - RAM) ← 1) Bus (Decoder 3x8) 2) 16 تا Mux 8x1

دیتا بیس — "AC ← M[AR]" دیتا عمل نیست

چون در یک کلاک قابل انجام نیست (۲ کلاک)

AC ← DR ← M[AR]

Memory (T_4 به بعد)

$D_7 \neq 1$

AND | * DR ← M[AR]
2 | AC ← AC ∧ DR ; SC ← 0 ✓Done

ADD (2: AC ← AC + M[AR]) | STA (1: M[AR] = AC)

LDA (2: AC ← M[AR]) | BUN : 1 | PC ← AR

BSA : 1 | M[AR] ← PC ; AR++ ;
2 | PC ← AR ; SC = 0 ;

Branch & Save Return Addr

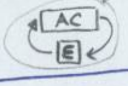
ISZ : | * DR ← M[AR]
Increment & skip if zero — 3 | DR++
M[AR] ← DR ; if(DR==0) PC++ ;

Reg

$D_7 = 1$, $I = 0$

(همه در T_3 انجام و تمام می‌شوند)

CLA (AC)	CMA (AC, مکمل)	CIR : (Circulate (shift))	INC (AC)	SPA (skip if +)	SZA (skip if zero)	HLT
CLE	CME	CIL		SNA (if −)	SZE	



2/2

IO

$D_7 = 1$, $I = 1$

(همه در T_3 انجام و تمام می‌شوند)

INP : AC(0-7) ⇐ INPR (Reg) ; FGI = 0 ;

OUT : OUTR ⇐ AC(0-7) ; FGO = 0 ;

Flag

SKI : if(FGI=1) PC++   (Input Flag) | ION : IEN = 1

SKO : if(FGO=1) PC++ | IOF : IEN = 0 (Intrupt off)

INTRUPT

RT_0 { AR = 0 ; TR ← PC }

RT_1 { M[AR] ← TR ; PC = 0 }

RT_2 { PC++ ; IEN = 0 ; R = 0 ; SC = 0 }

Pipeline | chp 6

کلاس تست

* مقایسه اجرای یک برنامه با چندین CPU بجای یکی

$$t' = t\left(f_1 + \frac{f_2}{P}\right)$$

t': جدید (هنگام ...)

f_2: درصد کد قابل موازی سازی

P: تعداد پردازنده

$$t' = t\left(0/1 + \frac{0/9}{3}\right)$$

۹۰٪ کد — ۱۰٪ کد سریال

ایده‌آل → $t' = \frac{t}{P}$

$P \to \infty$ → $t' = tf_1 \Longrightarrow S = \frac{t}{tf_1} = \frac{1}{f_1} > 1$ ($0 < f_1 < 1$)

$$E = \frac{S}{P}$$

E: کارایی — S: Speedup کل — P: تعداد کل P پردازنده

منظور حس: کارایی P پردازندگی شدن نسبت به حس ... پردازنده

اگه S=P یعنی P برابر کردن پردازنده‌ها ۱۰۰٪ داره جواب می‌ده و واقعاً سرعت سیستم P برابر شده! ولی معمولاً Speedup ما P برابر نمی‌شه!

$$S = \frac{t_{\text{قدیم}}}{t_{\text{جدید}}} > 1$$

Speed Up تسریع — معمولاً

$$\text{کارایی خط لوله} = \frac{S}{K}$$

K: تعداد بند

تعریف حس مشابه — ارزش تست ندیدم کلا!

T_P = Clock cycle time ($T_{Pipeline}$ = Max)

* معمولی به پایپلاین

(K=3) | 10 | 30 | 20 | — تأخیر مدار، تأخیر کل، تأخیر Latch

$$t_p = KT_P + (n-1)T_P$$

(KT_P: دستور اول — $(n-1)T_P$: بقیه) — Pipeline

$$t_{NP} = \left(\sum t_i\right) \times n$$

NoPipline

A: | 20 |

B: | 10 | 10 |

C: | 5 | 5 | 5 | 5 |

* C هر ۵ ثانیه یه کاری ده بیرون ولی B هر ۱۰ ثانیه

⇐ سرعت پردازش C بیشتر از B

اجرای ۵ دستور:

A → 5×20 = 100

B → 2×10 + 4×10 = 60

C → 4×5 + 4+5 = 40 — $n \to \infty$ (تعداد کدها) → $\frac{100}{4} = 25$

* خط لوله در حالت ایده‌آل سرعت اجرا را K برابر می‌کنه

A: | ... | 10 | (50)

B: | ... | 5 | (70)

* اجرای دستورات (کلی) در B سریعتر (هر ۵ ثانیه)

ولی اجرای یک تک دستور در A سریعتر (A: 50 < B: 70)

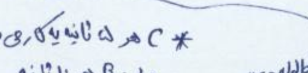
* اگر توی کد Branch زیاد باشه، A سریعتر

(پس هواره نمی‌شه گفت کدوم سریعتره)

Super Scaler: چند خط لوله موازی

اجرای موازی داخلی ×

Super Pipeline: هر Stage، خط لوله Pipeline خود یک



* اگر f درصد از کدها، دستورات پرش باشند:

I) دو هر پرش خط لوله تا اتمام اجرای پرش قابل استفاده نباشه

$$t_{\text{کل}} = KT_P + (n-1)T_P + n \times f \times [\text{Penalty}]$$

(KT_P: دستور اول — $n \times f$: میزان کد پرش)

$$\text{Penalty} = (K-1) \times T_P$$

به ازای هر پرش تو کد تا K-1 دستور NoOp می‌ذاریم بعدش

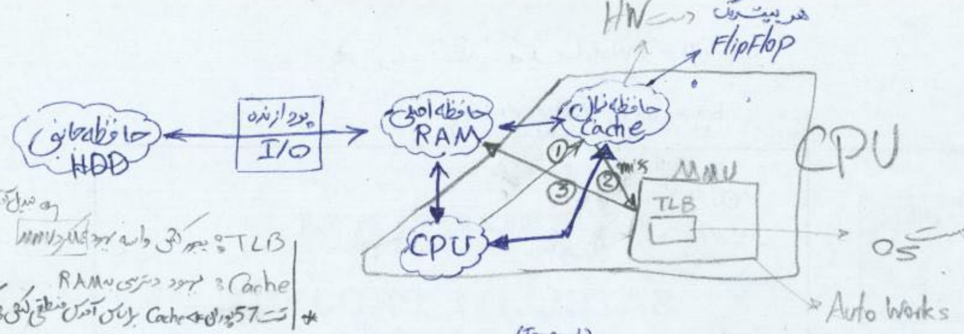
* تذکر: اگر آخرین دستور، پرش خط آخر هست، (K-1) ای NoOP آخر لازم نیست!

II) ...

Mi

$$\text{توان عملیاتی} = \frac{\#\text{کار انجام شده}}{\text{زمان}}$$

chp 7 — حافظه / سازمان:



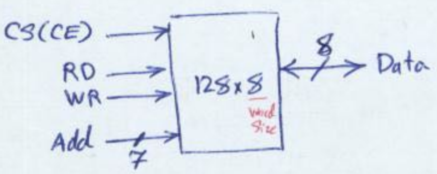
TLB: بهره‌گیری واسه بهبود MMU
Cache: بهبود دسترسی به RAM
* تست 57: جدول Cache ← براساس آدرس منطقی کش می‌کنه!! :)
به جدول آدرس منطقی به فیزیکی

اصل "محلی بودن مراجعات":

← برنامه‌ها تمایل به استفاده از چند ناحیه خاص و محدود در حافظه هستند.
- زمانی (Temporal): بعداز یه مدتی به یک آیتم دوباره رجوع می‌شه
- مکانی (Spatial): اگه به یک آیتم رجوع شه، به اطرافش هم رجوع می‌شه

حافظه اصلی
- ROM: دستیابی تصادفی R، دائم
- RAM: دستیابی تصادفی R/W، فرار
  - Static (→SRAM): FlipFlop، سرعت بیشتر
  - Dynamic (→DRAM): در خازن (← نیاز به refresh) (← فضای بیشتر)، مصرف کمتر + ظرفیت بیشتر

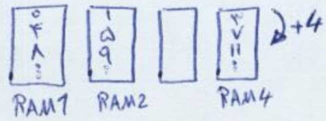
SRAM ← سرعت (Speed)



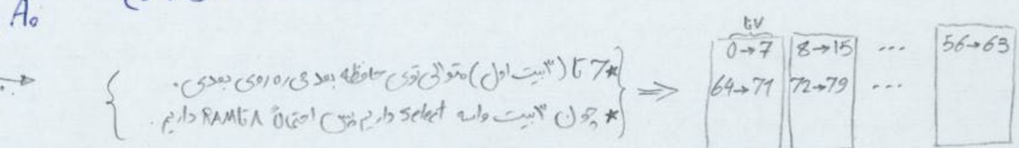
CS	RD	WR	وضعیت تراشه (Data)
0	X	X	High Apdance (—)
1	1	X	Read
1	0	1	Write
1	0	0	HA(—)

* R/W با هم نمی‌شه → Read ✓

برگ‌برگ سازی ("بزرگ کردن" حافظه)
- decoder: A_7, A_8 → CS1 … CS4
- Multiplexer: آدرس به همه وصل ولی خروجی رو براساس A_7 A_8 انتخاب می‌کنیم
- high Order: آدرس [انتخاب RAM | آدرس در RAM] → تحمل پذیری ↑ ، موازی خوانی X
- low order: اگه $\overline{A_1 A_0}$ واسه انتخاب ⇐ مناسب برای موازی سازی، اصل "محلی بودن مراجعات" دسترسی
- etc: طراحی lowordr: شکل صفحه ۱۹۵ جزوه (ماتریس 8x8)



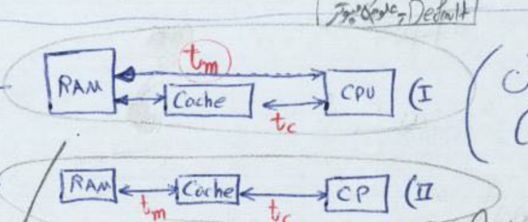
$A_8 A_7 A_6$ [$A_5 A_4 A_3$] $A_2 A_1 A_0$ → یعنی:
- * 7 تا (۳ بیت اول) متوالی توی حافظه بعدی ۰/۱ روی بعدی.
- * چون ۳ بیت واسه select داریم پس احتمالاً ۸ تا RAM داریم.
- * دسترسی جدول مربوط به دقیق هست که بیت‌های آدرس (بجز A_3 و A_5 و A_4) یه مقدار بگیرند مثلاً: 000XXX000



حافظه نهان Cache

(I
$$t_{avg} = h\,t_c + (1-h)\,t_m$$

(II Default ✓ (Default در صورت عدم ذکر)
$$t_{avg} = t_c + (1-h)\,t_m = h\,t_c + (1-h)(t_c + t_m)$$
... قابل بسط



$h = \frac{\text{نرخ hit}}{\text{تعداد کل مراجعات}}$ ← hit ratio