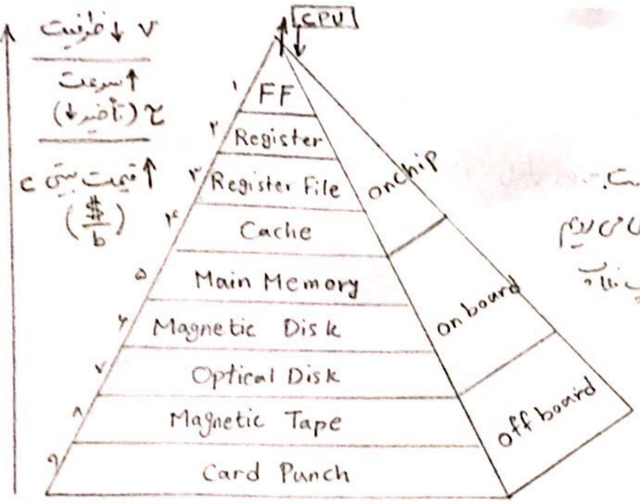
تمرین درس معماری کامپیوتر - شماره ۱
مهدی نیک نژاد - ۹۶۳۰۰۳۹



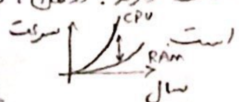
$\tau_1 < \tau_2 < \cdots < \tau_9$
$v_1 < v_2 < \cdots < v_9$
$c_1 > c_2 > \cdots > c_9$

۱- الف) سلسله مراتب حافظه →

ما می‌خواهیم CPU طراحی کنیم. طراحی CPU، بدون استفاده از RAM کار اشتباهی است. چون عملیات داخل CPU نیاز به فضایی برای ذخیره دارد. پس به سراغ انواع معروفی می‌رویم تا با آنها آشنا شویم. حافظه هایی که ما تا الان یاد گرفتیم: D / T / RS / JK فلیپ فلاپ

چون X و Z تولید می‌کند ↓ — مناسب نیستند — مناسب نیست.

پس فقط D-FF مناسب کار ماست و با وصل کردن چندین D-FF هم register بدست می‌آید. پس ما فقط با register آشنایی داریم. حافظه ها تکنولوژی های متفاوتی در طبقات مختلف دارند. در ضمن، سرعت رشد حافظه ها به نسبت CPU بسیار کندتر است.



* دسته بندی: حافظه ها
Off board ← حافظه هایی که می‌خریم و از بیرون connect می‌شوند: Hard External/USB (هزینه کمتر)
Onboard ← روی board سوار می‌شوند، plug: RAM / HARD
Onchip ← روی chip قرار دارند (سرعت بالاتر ← علت: تکنولوژی ترانزیستورها
(اندازه کوچک ← سیم مفروض کمتر، تأخیر کمتر ← سرعت بیشتر) ~nm
offchip

برای طراحی CPU، حافظه زیادی نیاز نداریم. CPU باید یک سری محاسبات انجام دهد. برای اینکه صحت CPU را تست کنیم، روی آن OS نصب می‌کنیم و می‌بینیم بالا می‌آید یا نه. هدف از ایجاد CPU ← برنامه ما را انجام دهد. توجه کنید که CPU ای که ما قرار است طراحی کنیم باید عام منظوره باشد.

برای ما که کاربر عام هستیم (desktop processor) ۳ عامل مهم است: ۱) سرعت (تأخیر) / ۲) قیمت / ۳) حجم (ظرفیت)

متأسفانه این ۳ عامل در تضاد با هم هستند: سرعت ↑ ~ قیمت ↑
حجم ↑ ~ قیمت ↑
حجم ↑ ~ سرعت ↓

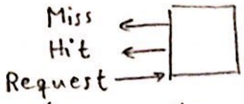
ایده‌آل: حجم بالا، سرعت بالا، قیمت پائین

* کاری که ما با حافظه داریم: read / write

FF را که می‌شناسیم. register، مجموعه ای از FF ها هستند. registerfile، آرایه ای از register ها هستند.

* Magnetic Disk ← هارد، فلاپی
Optical Disk ← CD
Tape ← ضعف: مکانیکی بودن و سرعت کم
فایده: بی‌انتها بودن حجم این حافظه ها

این طور به نظر می‌آید که ما بایستی از هر نوع حافظه درصدی را در کامپیوتر خود داشته باشیم. در طراحی یک کامپیوتر تا آنجا که بتوانیم داده ها را در سطح بالاتر نگه می‌داریم و در صورتی که به داده ای نیاز داشته باشیم به سطح پائین می‌آییم. شمای کلی کار با حافظه:



اتصال سری حافظه ها به یکدیگر مناسب است. حافظه سریع تر را به CPU نزدیکتر قرار می‌دهیم تا اگر جواب را داد که هیچ (hit) و اگر نه (Miss) به سراغ بعدی می‌رویم:

CPU → M_1 → M_2 → ⋯ → M_n
(حافظه سریع ... حافظه کند)

← به حجم بستگی دارد! $\uparrow h \sim \uparrow v$

یک پارامتر دیگر تعریف می‌کنیم: احتمال وجود داده درخواست شده $= h$ (hit) ← $0 \leq h \leq 1$
نرخ برخورد hit rate

$h_1 < h_2 < \cdots < h_9$

$h_n = 1$
$h_1 \simeq 0$

* نحوه بدست آوردن hit rate: $\frac{\# hit}{\# hit + \# miss}$ ← آنقدر سؤال می‌پرسیم اگر جواب گرفتیم (hit) و اگر جواب نگرفتیم (miss)، کل: $\# hit + \# miss$

(I): تأخیر $= h_1 \tau_1 + (1-h_1)\left(h_2 \tau_2 + (1-h_2)(\cdots)\right)$
(II): تأخیر $= h_1 \tau_1 + (1-h_1)\left(\tau_1 + h_2 \tau_2 + (1-h_2)(\tau_2 + \cdots)\right)$
(III): تأخیر $= \tau_1 + (1-h_1)\left(\tau_2 + (1-h_2)(\cdots)\right)$ سری

(II) = (III) ≃ (I)

Scanned with CamScanner

ب) دسترسی به تمام داده‌ها/ حافظه‌های کاربردی‌تر را register بنامیم ← فواید: سرعت بیشتر
ضرر: هزینه بیشتر / حجم کمتر

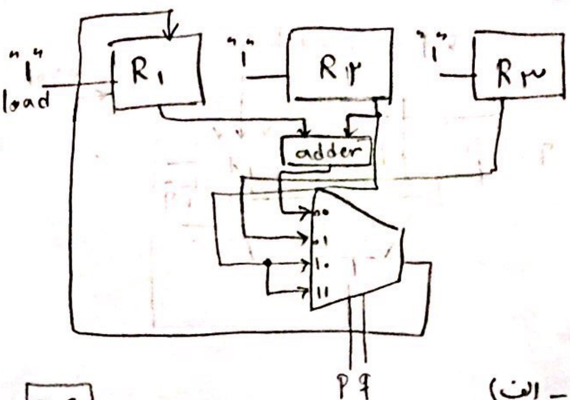
ج) تفاوت DRAM (حافظه پویا) و SRAM (حافظه ایستا)

SRAM	DRAM
* توان مصرفی کم	* توان مصرفی بالا
* نیاز به Refresh ندارد	* نیاز به Refresh دارد ← چون خازن دشارژ می‌شود و باید با رفرش شارژ شوند
* گران	* ارزان
* تأخیر کم	* تأخیر زیاد (سرعت کم)
* مساحت زیاد	* مساحت کم
* چگالی بیتی کم	* چگالی بیتی زیاد ← چگالی بیتی = تعداد بیت / مساحت اشغالی
* کاربرد: Cache	* کاربرد: RAM های لپ‌تاپ‌ها

۲ـ

if $(p=1)$ then $(R_1 \leftarrow R_2)$
else if $(Q=1)$ then $(R_1 \leftarrow R_3)$
else $(R_1 \leftarrow R_1 + R_2)$

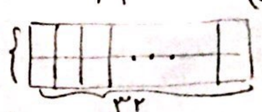
$PQ + PQ'$
$= P: R_1 \leftarrow R_2$
$\bar{P}Q: R_1 \leftarrow R_3$
$\bar{P}\bar{Q}: R_1 \leftarrow R_1 + R_2$



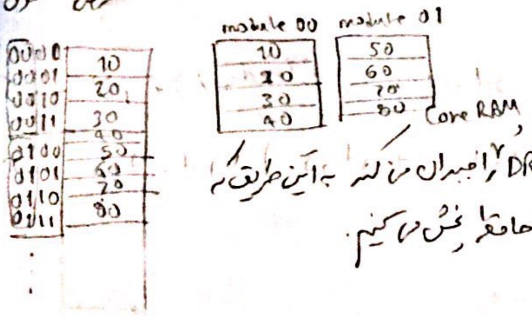
۳ـ الف)

$128 * 8 \longrightarrow 128$ Bytes
$4096 * 16 \longrightarrow 4096 * 2 * 8 \longrightarrow 8192$ Bytes
عرض طول

$\frac{8192}{128} = \boxed{64} \longrightarrow 32 \times 2$



ب) Interleaved Memory ← طراحی ماژولار RAM
سرعت بیشتر



در محاسبات، حافظه‌ها باز (بزرگ بزرگ شده) طراحی است که سرعت کند DRAM را جبران می‌کند به این طریق که آدرس این حافظه‌ها را با توزیع یکسان و دوباره در سراسر بانک‌های حافظه پخش می‌کنیم.
حافظه اصلی را به چند حافظه کوچکتر تقسیم می‌کنیم. (modularization)

۴ـ

طراحی الف باب: به نظر این طور است که ۲ تا ۳۲x۳۲ حافظه داریم که با استفاده از تراشه‌های اضافی (ترجیحاً decoder) قابل پیاده‌سازی است.
۲۰بیت

سری:

$t_1 + (1-h_1)(t_2 + (1-h_2)t_3) \longrightarrow 10 + (1-0.7)(20 + 0.1 \times 30) = \boxed{16.9}$

موازی:

$t_1 + (1-h_1)\left(t'_2 + (1-h_2)(t'_3)\right) \longrightarrow 10 + 0.3\left((10+20) + 0.1 \times (10+20+30)\right)$

$t'_2 = t_1 + t_2$, $t'_3 = t_1 + t_2 + t_3$

$\boxed{20.8}$

$S: R_C \leftarrow 0,\ S \leftarrow 0, E \leftarrow 0, F_0 \leftarrow 1, R_1 \leftarrow R_0$

$F_0: R_1 \leftarrow \sim R_1,\ F_0 \leftarrow 0,\ F_1 \leftarrow 1$

$F_1: R_1 \leftarrow R_1 + 1,\ F_1 \leftarrow 0,\ F_2 \leftarrow 1$

$F_2: R_2 \leftarrow R_2 + R_1,\ F_2 \leftarrow 0,\ F_3 \leftarrow 1$

$F_3: F_3 \leftarrow 0$, if $(R_2 < 0)$ then $[R_2 \rightarrow R_2 + R_0,\ E \leftarrow 1]$ else $[R_3 \leftarrow R_3 + 1,\ F_2 \leftarrow 1]$

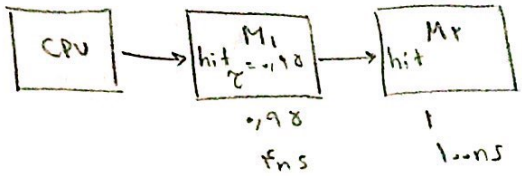
(R خارج قسمت، باقی‌مانده)

RTL → Continuous Assignment

اگر $S=1$ باشد، صفر را در R_3، صفر را در S، صفر را در E و 1 را در R_0، F_0 را در R_1 می‌ریزیم. حال F_0 صفر می‌شود و دیگر این خط اجرا نمی‌شود. به خط دوم می‌رویم. اگر F_0 یک باشد، R_1 را در R_1 می‌ریزیم و 0 را در F_0 و 1 را در F_1 می‌ریزیم. حال F_0 را می‌ریزیم. به خط بعد می‌رویم چون F_1 یک شده است. این خط بعد اجرا می‌شود. R_1 را یکی زیاد می‌کند و در R_1 می‌ریزد. 0 را در F_1 و 1 را در F_2 می‌ریزیم. حال در این خط F_1 صفر شده است و این خط دیگر اجرا نمی‌شود. به خط بعد می‌رویم. F_2 اگر یک باشد، R_1 را با R_2 جمع می‌کنیم و در R_2 می‌ریزیم و صفر را در F_2 و 1 را در F_3 می‌ریزیم. F_2 در خط بالا صفر می‌شود و دیگر (فعلاً) اجرا نمی‌شود. به سراغ خط پایین‌تر می‌رویم. اگر F_3 یک شود، صفر را در F_3 می‌ریزیم. حال اگر $R_2 < 0$ باشد، $R_2 + R_0$ را در R_2 می‌ریزیم و E هم 1 می‌شود (با در غیر این صورت)، R_3 را یکی زیاد می‌کنیم در R_3 می‌ریزیم و F_2 را یک می‌کنیم. این بار چون F_2 یک می‌شود، برمی‌گردیم به خط بالاتر و این چنین لوپ (حلقه) می‌سازیم. البته در صورتی که $R_2 > 0$ باشد، else اجرا می‌شود و F_3 برابر با یک می‌شود و در loop نمی‌افتیم. در غیر این صورت در loop می‌افتیم.

خارج قسمت R_2 بر R_0 تقسیم می‌شود، R_2 خارج قسمت، R_0 باقی‌مانده

این مثل یک عملیات تقسیم است با تفریق‌های متوالی! ←



$0.95 \times 4 + 0.05(1 \times 100 + 4) = 9$

$0.97 \times 5 + 0.03(1 \times 100 + 5) = 8$

$9 - 8 = 1\text{ns}$ اختلاف