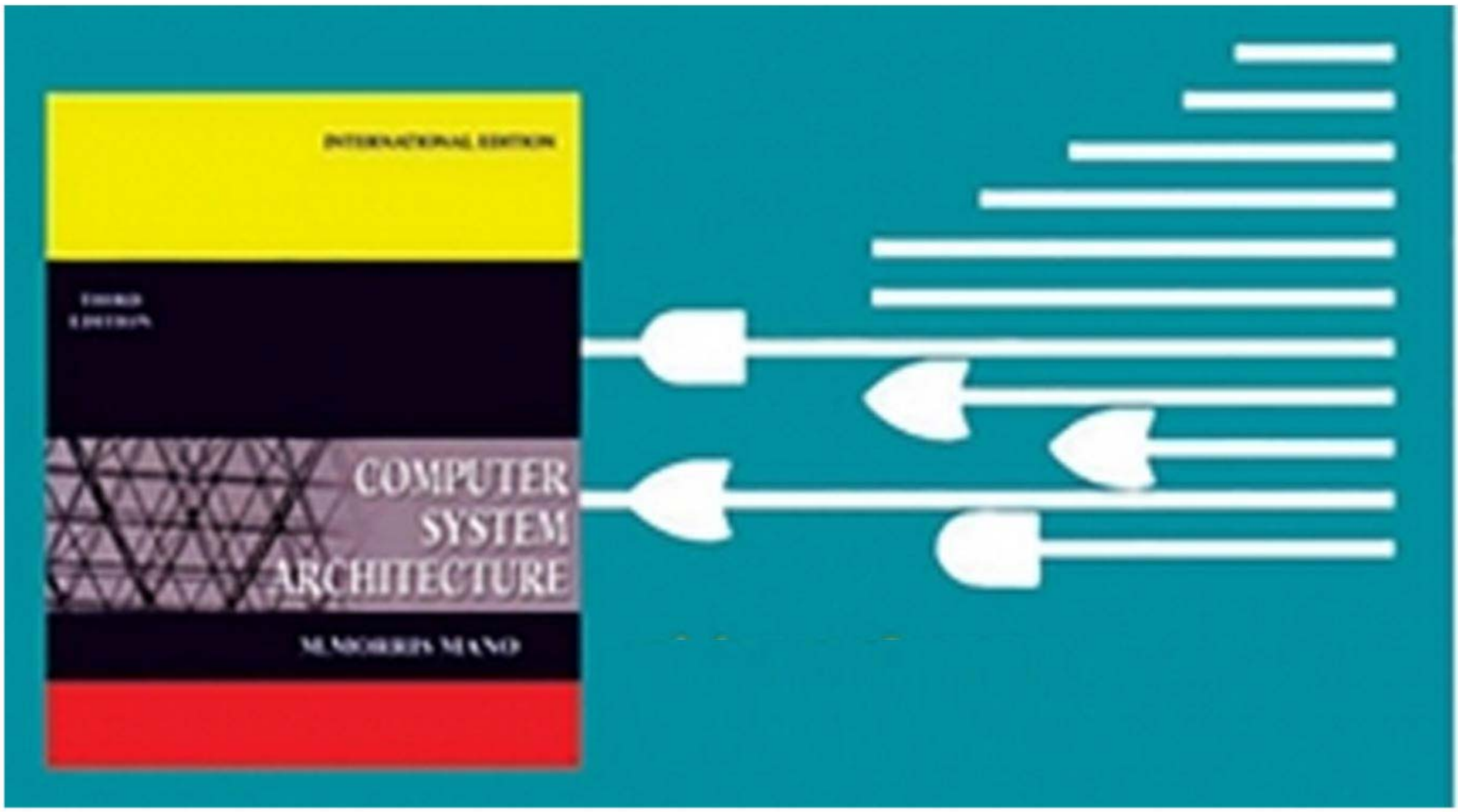
بسمه تعالی

جزوه درس معماری کامپیوتر

مدرس: نوبهاری



Subject:

Year, Month, Date, ( )

دسته بندی کامپیوترها :

(۱) کامپیوترهای رومیزی (Desktop computer)

(۲) کامپیوترهای سرور (Server computer)

(۳) کامپیوترهای تعبیه شده (Embedded computer) مثل : گوشی موبایل که برای انجام خدمات خاصی است - تلویزیون و ...

درواقع کامپیوترهای تخصصی قابل برنامه نویسی نیستند .

سطوح Abstraction : سطوحی که لایه های پایین تر را از لایه های بالاتر جدا می سازد و از جمله مهمترین Instruction Set Arthitecture (ISA) است.

که ISA خود شامل چند قسمت است:

(۱) دستورالعمل ها (۲) مجموعه رجیسترها

(۳) مدهای آدرس دهی (۴) معماری حافظه

+ برنامه نویسی از طریق ISA با سخت افزار در ارتباط است .

+ از دیگر سطوح Abstraction می توان زبان C را هم نام برد

+ Performance : $\text{Performance}_X = \frac{1}{\text{Execution time}_X}$

* هرچه بیشتر بهتر *

Execution Time : مدت زمان اجرای یک برنامه خاص

- برای مقایسه Performance دو سخت افزار روی یک job instruction از رابطه ی $\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execute time}_Y}{\text{Execute time}_X}$ استفاده می شود.

$\text{Performance}_X = n\,\text{Performance}_Y$ ← نسبت عکس مدت زمان اجرا

+ CPU Time : * هرچه کمتر بهتر *

$\text{CPU Time} = \#\text{Clock cycle} \times \text{clock cycle time}$

← تعداد کلاک

$\text{Clock cycle time} = \frac{1}{\text{Clock Rate}}$: که

← دوره تناوب هر کلاک ← فرکانس

- اگر #Clock cycle را داشتیم می‌توانستیم با داشتن CPI، تعداد دستورالعمل‌ها را بدست آوریم.

$$\#\text{Clock cycles} = \text{Cycles Per Instruction} \times \#\text{instruction}$$

$$\text{CPU time} = \frac{\#\text{Instruction} \times \text{CPI}}{\text{Clock Rate}}$$ پس فرمول نهایی CPU time بصورت روبرو است

در واقع CPI: بین هر instruction چند کلاک طول می‌کشد

هر دستور 2 کلاک طول می‌کشد ⇐ CPI = 2.0

* برای 2 برنامه منظور از Same ISA (یکسان ISA) این است که تعداد دستورالعمل‌ها

$\#\text{Instruction}_A = \#\text{Instruction}_B$ هر دو برنامه مشابه است.

- این فرمول‌ها را برای instruction خصوصی‌سازی کردیم؛ در صورتیکه دستورهای مختلف با CPI مختلف داشته باشیم آنگاه:

$$\#\text{Clock cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \#\text{instruction}_i)$$

که از این فرمول CPI به این صورت

$$\text{CPI} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \frac{\#\text{Inst}_i}{\#\text{inst}}\right)$$

* تعداد سیکل کمتر ⇐ #Cycle کمتر

* CPI کمتر ⇐ CPI_{ave} کمتر

MIPS برای مقایسه دو ISA متفاوت نمی‌تواند مناسب باشد!!!

Millions Instruction Per Seconds

$$\text{MIPS} = \frac{\#\text{instruction}}{10^6 \times \text{Execution Time}} = \frac{\text{Clock Rate}}{\text{CPI} \times 10^6}$$

در صورتی مقایسه با MIPS انجام می‌شود که دو برنامه روی دو کامپایلر روی یک سخت‌افزار اجرا شوند.

(Same ISA)

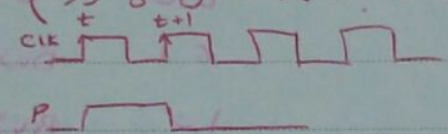
* بیشتر MIPS ⇐ #بهتر

Subject:

Year. Month. Date. ( )

وقتی که تعداد رجیسترها و تعداد رجیسترها در یک کامپیوتر زیاد می‌شود نیاز به یک زبان داریم که اجزای داخلی را منظم مرتب کند. "Register Language Transfer" RTL

نشانه	توضیح	مثال
حروف بزرگ و اعداد	مشخص کردن ثبات ها	R1, MAR, PC, IR AR, SR, TR, ...
:	برای شرطی ها (کنترل P تولید شده توسط CPU شرط انجام عملیات است و به Load رجیستر R1 وصل می شود که در صورت یک بودن مقدار R2 به R1 منتقل می شود)	P: R1 ← R2 (در واقع این دستور را مشخص می کند که: if(P==1) then R1 ← R2
,	چند عملیات همزمان و تحت یک شرط انجام گیرند	P: R1 ← R2, R2 ← R1
( )	برای مشخص نمودن بخشی از ثبات	IR(2-0)
→	برای عملیات انتقال ثبات	R1 ← R2
[ ]	آدرس یا فضای حافظه و محتوای درون کروشه خانه حافظه را مشخص می کند.	DR ← M[100]



و عملیات در انتهای کلاک یعنی t+1 انجام می‌شود.

(Edge Trigger)

انواع عملیات: MOP عملیات هایی هستند که روی ثبات ها و حافظه (ورودی) می‌آیند می‌شوند و خروجی یا ثبات ها نشان داده می‌شوند و در یک کلاک انجام شوند.

Subject

Year. Month. Date. ( )

در زبان RTL : - ترتیب مهم نیست.

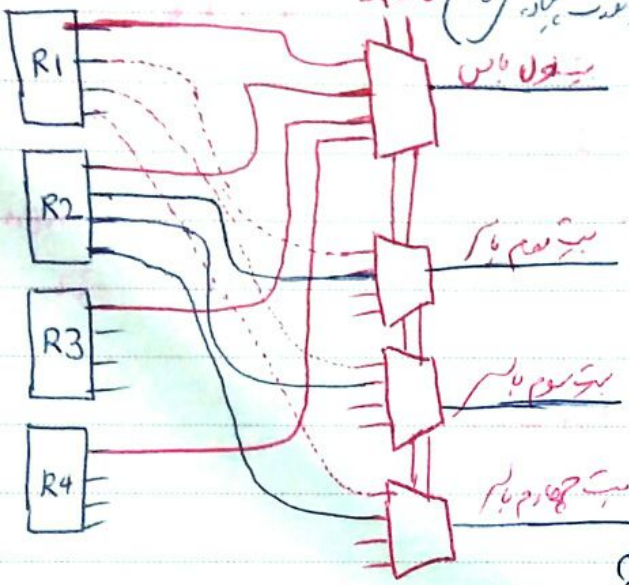
- همه عملیات با کلاک هماهنگام هستند.

- همه عملیات موازی اجرا می‌شوند.

در ریز عمل ها و انتقال ها نیاز است که رجیترها با هم در ارتباط باشند لذا مثلاً برای 3 رجیتر باید n^2-n سیم باید بین این سه رجیتر وصل باشد بنابراین از BUS استفاده می‌شود.

طراحی BUS : (تعداد سلکتور دارد که به ورودیت پیاده سازی می‌شود S_1S_0)

۱- با استفاده از مالتی پلکسر: BUS



S_1S_0	
0 0	BUS ← R1
0 1	BUS ← R2
1 0	BUS ← R3
1 1	BUS ← R4

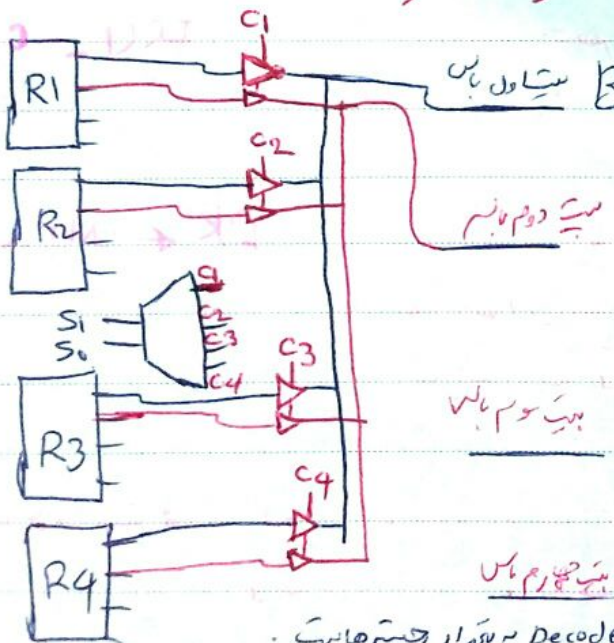
نکته ۱: به تعداد بیت های هر رجیتر MUX داریم

نکته ۲: به تعداد رجیتر برای ورودی MUX، Pin داریم

در این مثال رجیترها 4 بیتی اند و BUS نیز 4 بیتی است.

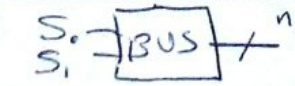
۲- با استفاده از باز:

دو به دو سیم ها به هم وصل می‌شوند برای جدا سازی از دیگر هم استفاده می‌شود.



S_1 S_0	
0 0	BUS ← R1
0 1	BUS ← R2
1 0	BUS ← R3
1 1	BUS ← R4

نکته ۱: خروجی Decoder به تعداد رجیترهاست.



نکته ۱: به تعداد رجیترها دسته باز داریم و هر کدام به تعداد بیت‌های رجیترهاست

ASEMAN

Subject :

Year. Month. Date. ( )

ریز عمل ها : مربوط به BUS و حافظه :

AR ← M[AR] : Read

M[AR] ← M[DR] : write

Bus ← R1

ریز عملیات حسابی :

جمع	R3 ← R1 + R2
تفریق	R3 ← R1 − R2
NOT کردن	R2 ← $\bar{R2}$
منفی کردن	R2 ← $\bar{R2}$ + 1
تفریق کردن در 2's comp	R3 ← R1 + $\bar{R2}$ + 1
افزایش دادن	R1 ← R1 + 1
کاهش دادن	R1 ← R1 − 1

ریز عملیات جهت انتقالی قطعات ها :

مثل R1 ← R2

ریز عملیات Shift :

شیفت ثباتR به چپ	R ← shl R
شیفت ثبات R به راست	R ← shr R
شیفت چرخشی R به چپ	R ← cil R
شیفت چرخشی R به راست	R ← cir R
شیفت حسابی R به چپ	R ← ashl R
شیفت حسابی R به راست	R ← ashr R

در شیفت حسابی به چپ مقدار R_0 برابر صفر شده و بقیه بیت ها یکی به چپ انتقال می یابند و بیت آخری R_{n-1} از بین می رود و در شیفت راست $R_{n-1} = R_{n-2}$

سرریز زمانی رخ می دهد که در شیفت حسابی به چپ قبل از انجام شیفت داشته باشیم :

$V_S = R_{n-1} \oplus R_{n-2}$ حین اگر R_{n-1} و R_{n-2} مساوی نباشند سرریز رخ میدهد

ریز عملیات منطقی :

F ← A ^ B ... F ← 0

عملیاتی که روی داده های یک رجیستر انجام میگیرد (موقع بیت به بیت)

XOR (ی شود)

Subject :

Year. Month. Date. ( )

نکات : شیفت منطقی به راست : R_0 حذف شده و R_{n-1} جدید 0 می‌شود

شیفت منطقی به چپ : R_{n-1} حذف شده و R_{n-2} به جای آن می‌نشیند و 0 به جای R_0 جدید

شیفت چرخشی به راست : R_0 به R_{n-1} می‌رود

شیفت چرخشی به چپ : R_{n-1} به R_0 می‌رود

شیفت حسابی به راست : همان شیفت منطقی است ولی R_{n-1} جدید با 1 پر می‌شود.

شیفت حسابی به چپ : همان شیفت منطقی به چپ است.

دو وقتی خود R_{n-1} را نگه می‌دارم

1001 ⟶ 1100

0110 ⟶ 0011

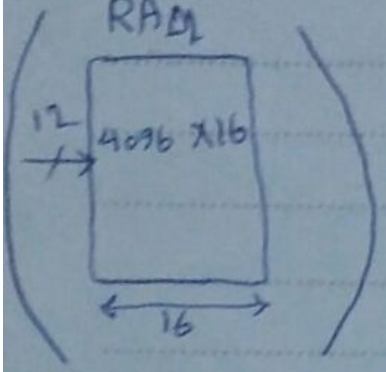
در Shift به چپ محتوای رجیستر عملاً در 2 ضرب شده

در Shift به راست محتوای رجیستر عملاً بر 2 تقسیم شده

DATE / /

SUBJECT

کامپیوتر پایه (PDP-8) شامل:



حافظه

AC ← (علایم فلاپی که حالت ذخیره و Store دارد)

(یک رجیستر 16 بیتی که محتویات یک Word حافظه در آن قرار می‌گیرد و محاسبات روی آن Data انجام شود)

INPR ← (برای عملیات ورودی مثلاً ارسال Data به رجیستر و 8 بیت دارد)

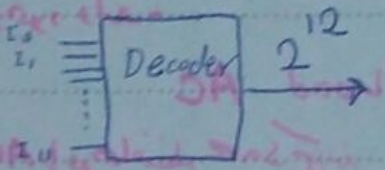
OUTR ← (برای عملیات خروجی مثلاً ارسال Data به رجیستر و 8 بیت دارد)

(برای عملیات ورودی مثلاً یک دیتا را از کیبرد بخواند و 8 بیت دارد)

IF ← (علایم فلاپی که نشان می‌دهد دستگاه ورودی یک دیتای آماده دارد)

OH ← (" " " " " خروجی " ")

IEN ← (علایم فلاپ جهت اطلاع یک چیپ دیگر)



درون RAM: 16 بیتی وجود دارد

کامپیوتر دارای m خانه حافظه‌ی p بیتی

if $2^n = m$ ⟹ n: تعداد بیت‌های آدرس

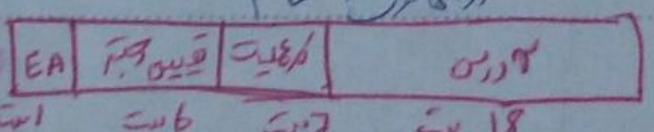
m: تعداد کلمات حافظه

p: تعداد بیت هر کلمه حافظه

اگر S تا رجیستر داشته باشیم باید حداقل از کدام بیت که S رجیستر را تعیین کند استفاده کنیم.

مثلاً ⟸ یک کامپیوتر 256k و 64 رجیستر کدبندی بصورت زیر دارد:

$2^{18} = 256k$



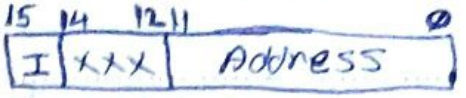
$2^6 = 64$

2^{18}

DATE

SUBJECT

IR



سه دسته دستور در حافظه داریم: ① دستوری حافظه

(I = mod (indirect OR direct))

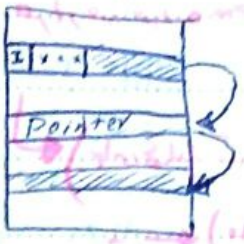
② دستورات ورودی خروجی

③

2^{12} دستور حافظه میتونه داشته باشه؟

2^{12} دستوری میتوانند داشته باشند

عملیات ثبات ها



مود های آدرس دهی: ① مستقیم ② غیر مستقیم

(I=1), (I=0)

مثلاً مود های آدرس دهی برای 8086 بیشتر از 2 تا هستند.

دستورات ورودی و کنترلی به حافظه:

Symbol	Hex code I=0	Hex code I=1	
AND	0 XXX	8 XXX	این دستورات بین AC و حافظه وجود دارد.
ADD	1 XXX	9 XXX	
LDA	2 XXX	A XXX	→ Load AC
STA	3 XXX	B XXX	→ AC را در حافظه save می کند
BUN	4 XXX	C XXX	→ بدون هیچ شرطی پرش می کند
BSA	5 XXX	D XXX	→ از یک آدرس حافظه گذشته و آدرس برگشت را save می کند.
ISZ	6 XXX	E XXX	→ یکی به آدرس حافظه اضافه و اگر صفر شد دستور بعد را اجرا نکن (بپر)

دستورات ثباتی:

CLA	Clear AC
CLE	= E
CMA	Complement AC
CME	= E
CIR	Circulate Right E, AC
CIL	Circulate Left E, AC

یعنی در صورت خالی بودن E، shift میدهم

JAHAN NAMA

DATE / /

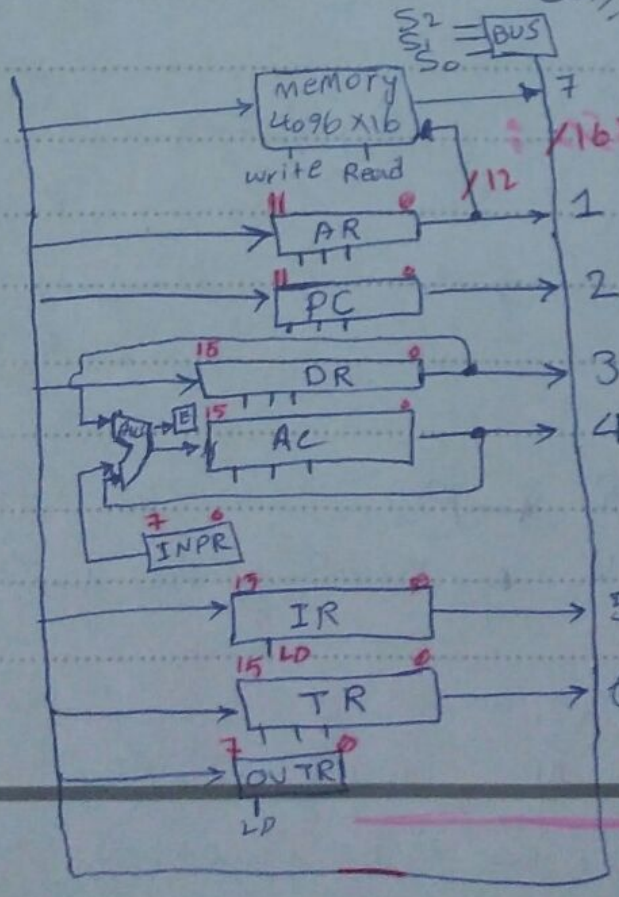
SUBJECT

INC	increament Ac (AC=1 میشه)
SPA	Skip next instruction if AC>0
SNA	" " " " AC<0
SZA	" " " " AC=0
SZE	" " " " E=0
HLT	Halt (متوقف یا پایان) ← غیرفعال شدن رجیستر ها

دستورات مربوط به I/O :

INP	input a character
OUT	output " " (8 بیت کم ارزش AC به OUTPR)
SKI	Skip next Instruction ON Input flag
SKO	" " " " Output flag
ION	Interupt ON
IOF	Interupt off

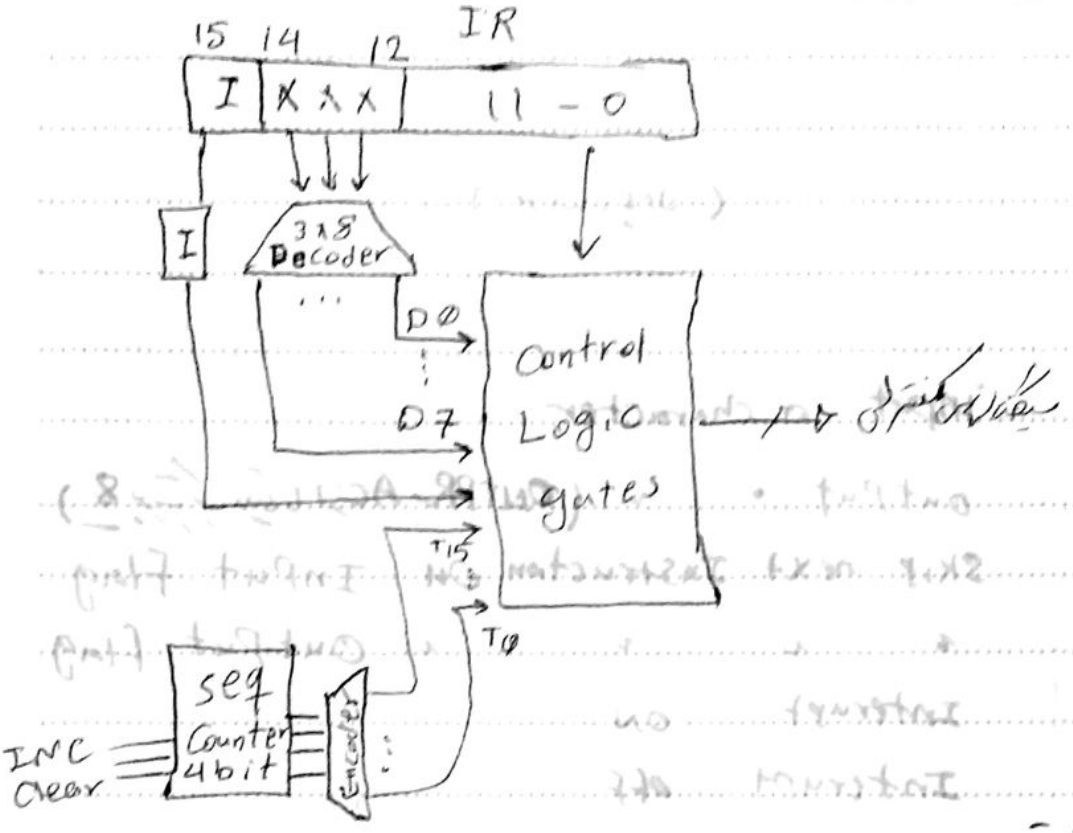
نکته : Data Path نرم افزار.



* AR برای ارتباط با حافظه است
* بدلیل اینکه ۸ بیت در AC انجام می شود پس با ALU در ارتباط است.
* TR یک رجیستر کمکی است.
* IR برای نگه داری دستورالعمل
* آدرس حافظه متصل به AR وصل است. پس برای دستیابی به حافظه ابتدا باید آدرس را در AR قرار داد.
* JAHA INRM دارد

DATE
SUBJECT

ساختار داخلی C.U بصورت زیر است.
(کار کنترل یونیت تولید سیگنال های کنترلی درون DataPath وجود دارد)
* ورودی C.U ، IR است.



اجرای دستورات :
جهت اجرای دستورات در کامپیوتر باید چرخه دستورالعمل را رفت.
۱- Fetch (واکشی) : دستورالعمل از حافظه خوانده می شود و در IR نوشته می شود
۲- دیکد کردن
۳- محاسبه ی آدرس مؤثر (مستقیم و غیر مستقیم بودن)
۴- اجرا

fetch : T0 : AR ← PC
T1 : IR ← M[AR] , PC ← PC+1

توضیح : یعنی دیتای روی PC (PC حاوی دستوری است که میخواهد اجرا شود) را ابتدا روی BUS قرار داده و از طریق AR به حافظه رفت و با این آدرس محتوای ~~AR~~ IR قرار می دهیم و همزمان (موازی) سیستم را برای انجام دستور بعدی آماده می کنیم (PC+1)

DATE / /

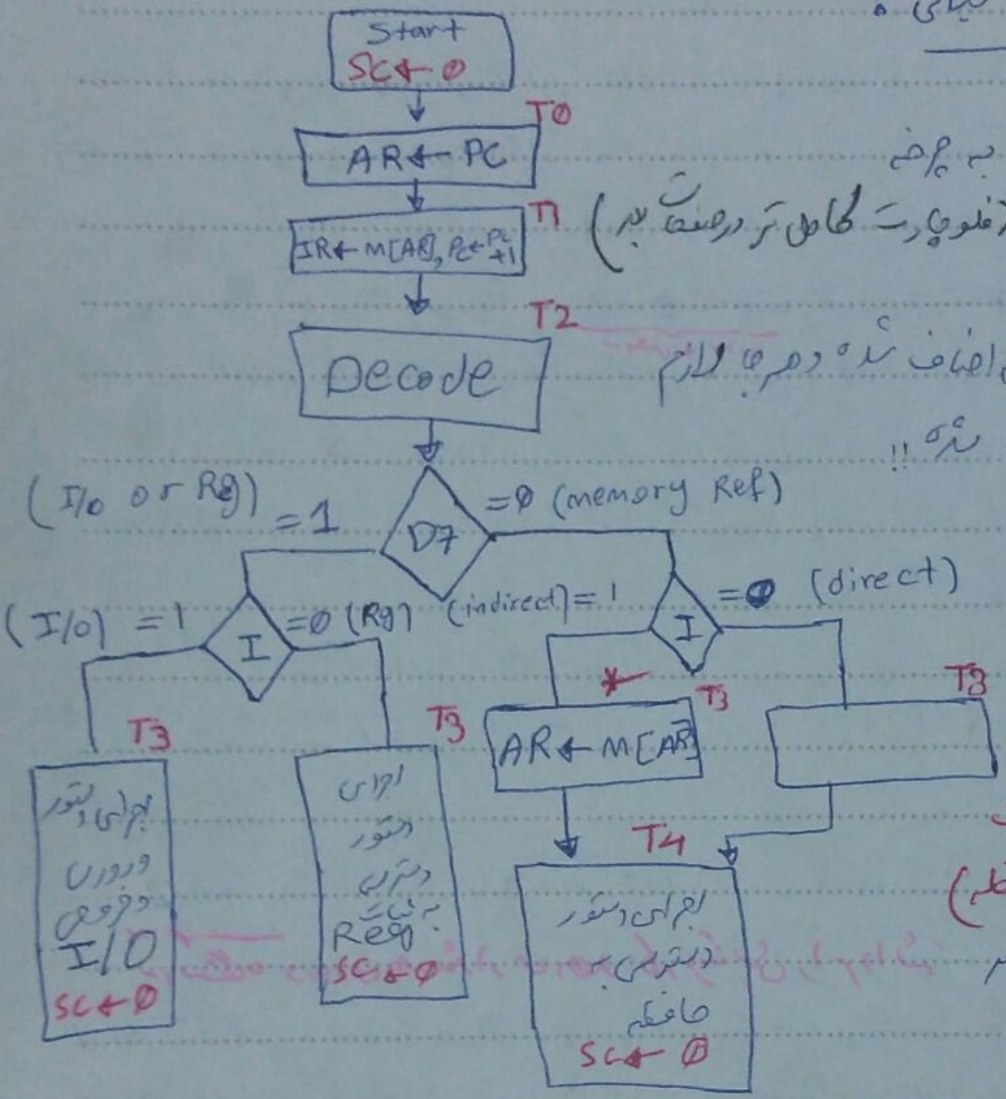
SUBJECT

decode : T2 : I ← IR(15), D7 ... D0 ← IR(14-12), AR ← IR(11-0)

D'7 T3 I : AR ← M[AR]

اجرا مربوط به دستورات ثباتی :

نکته: فلوچارت مربوط به چرخه دستورالعمل: (فلوچارت کامل تر در صفحات بعد)

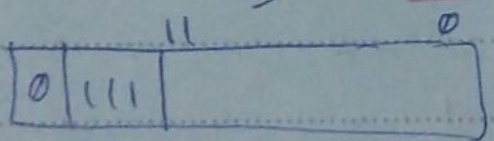


نکته: SC در هر مرحله اضافه شده و هر بار لازم بود صفر شود تنها نوشته شده !!

* D7 I درصورت مقدار محتوای AR در مرحله Decode را در AR قرار میدهیم زیرا آدرس غیر مستقیم است (یعنی مراجعه دوباره به حافظه) تا آدرس مورد نظر

در کلیه دستورات مربوط به ثبات ها بصورت زیر بنده

نکته: اگر هر بیت از بیت های 0 تا 11 یکی یک باشد و بقیه صفر آنگاه تنها 12 دستور مربوط به ثبات ها داریم



CIA : 0 111 1000000000000

CLE : 0 111 0100000000000

⋮ ⋮

HLT: 0 111 0000000000001

یعنی :

DATE / /

SUBJECT

if $D_7 I' T_3 = r$, $B_i = IR(i)$

* B_{11} : CLA ; rB_{11} : $AC \leftarrow 0$

B_{10} CLE rB_{10} : $E \leftarrow 0$

B_9 CMA rB_9 : $AC \leftarrow \overline{AC}$

B_8 CME rB_8 : $E \leftarrow \overline{E}$

B_7 CIR rB_7 : $AC \leftarrow shr\ AC$, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$

B_6 CIL rB_6 : $AC \leftarrow shl\ AC$, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$

B_5 INC rB_5 : $AC \leftarrow AC + 1$

B_4 SPA rB_4 : if $(AC(15) == 0)$ $PC \leftarrow PC + 1$ (مثبت یا صفر)

B_3 SNA rB_3 : if $(AC(15) == 1)$ then $PC \leftarrow PC + 1$

B_2 SZA rB_2 : if $(AC == 0)$ then $PC \leftarrow PC + 1$

B_1 SZE rB_1 : if $(E == 0)$ then $PC \leftarrow PC + 1$

B_0 HLT rB_0 : $S \leftarrow 0$

* کلمه : حافظه، پردازنده، PD-8 (کامپیوتر آموزشی) برای BUS، 000 تا 111، 000 تا 777 شماره‌گذاری شده است.

if $P = D_7 I T_3$, $B_i = IR(i)$: I/O دستورات مربوط به اجزای ورودی

INP : PB_{11} : $AC(0-7) \leftarrow INPR$, $FGI \leftarrow 0$

- انتقال داده‌ای که دستگاه ورودی می‌فرستد به رجیستر داده

OUT : PB_{10} ; $OUTR \leftarrow AC(0-7)$, $FGO \leftarrow 0$

SKI : PB_9 : if $(FGI == 1)$ then $PC \leftarrow PC + 1$

SKO : PB_8 : if $(FGO == 1)$ then $PC \leftarrow PC + 1$

ION : PB_7 : $IEN \leftarrow 1$

IOF : PB_6 : $IEN \leftarrow 0$

FGI : یک فلیپ فلاپ که هرگاه 1 شود بیانگر داده ورودی روی AC قرار داده شده

FGO : یک ″ ″ ″ ″ ″ 1 ″ ″ ″ داده خروجی آماده است و باید پردازد

DATE...../...../.....

SUBJECT.....

اجرای دستورات دسترسی به حافظه :

دستور	شرط پیمایش از رجیستر	دستور انتقال ثبات	
AND	D0	AC ← AC ∧ M[AR]	Cout
ADD	D1	AC ← AC + M[AR], E ← Carry out	
LDA	D2	AC ← M[AR]	
STA	D3	M[AR] ← AC	(عملیات معکوس)
BUN	D4	PC ← AR	
BSA	D5	M[AR] ← PC , PC ← AR + 1	
ISZ	D6	M[AR] ← M[AR] + 1 , if(M[AR] == 0) then PC ← PC+1	

همانند حالا در یک کلاک انجام نمی شود و فقط دستورات شرح داده شده بنابراین باید

این توضیحات به ازای عملیاتی بشکونیم . پس داریم :

AND : چون برای AND کردن باید در ALU این کار را انجام دهیم

و ALU تنها با DR و AC در ارتباط است پس ابتدا باید M[AR] را به DR برگ

سپس AND کرده پس داریم :

$$\begin{cases} D_0T_4 : DR \leftarrow M[AR] \\ D_0T_5 : AC \leftarrow AC \wedge DR,\ SC \leftarrow 0 \end{cases}$$

ADD :
$$\begin{cases} D_1T_4 : DR \leftarrow M[AR] \\ D_1T_5 : AC \leftarrow AC + DR,\ E \leftarrow Cout,\ SC \leftarrow 0 \end{cases}$$

LDA : چون برای Load کردن تنها میرا از ALU نمی شتن است

پس باید ابتدا به DR برود.

$$\begin{cases} D_2T_4 : DR \leftarrow M[AR] \\ D_2T_5 : AC \leftarrow DR,\ SC \leftarrow 0 \end{cases}$$

DATE

SUBJECT

STA : D_3T_4 : $M[AR] \leftarrow AC$, $SC \leftarrow 0$

BUN : D_4T_4 : $PC \leftarrow AR$, $SC \leftarrow 0$

BSA :
D_5T_4 : $M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
D_5T_5 : $PC \leftarrow AR$, $SC \leftarrow 0$

ISZ :
D_6T_4 : $DR \leftarrow M[AR]$
D_6T_5 : $DR \leftarrow DR + 1$
D_6T_6 : $M[AR] \leftarrow DR$,
if $(DR == 0)$ $PC \leftarrow PC + 1$, $SC \leftarrow 0$

* برای اینکه M[AR] را بخواند، کنترل LD دارد (memory) و با DR, LD کار کند.

نکته مهم : جمع کردن / تفریق کردن / ضرب کردن / از مسیر ALU در شیفت / دقت به کردن با صفر تنها باید از ALU بگذرد و سپس باید داده به DR برود.

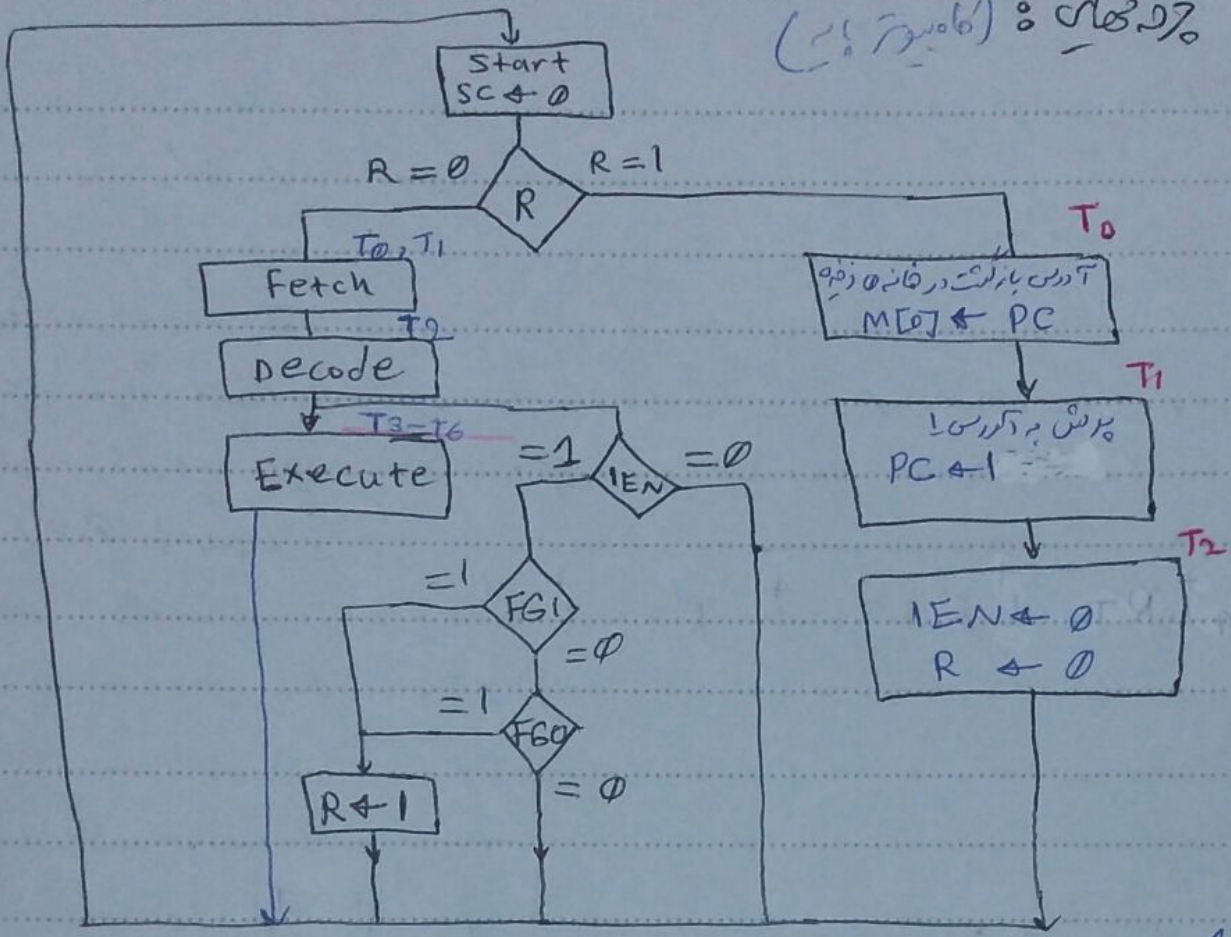
جریان وقفه : گاهی در میان اجرای دستورات در یک CPU به توقف و رخ به وقفه INTerupt نیاز است. (INTERUPT)

در این جا یک flip-flop با نام R معرفی می‌کنیم که شانی دهد اگر $R = 1$ باشد یعنی اینکه یک IEN آمده و باید به چرخه وقفه برویم و اگر $R = 0$ باشد باید همان سیکل اجرای دستورات به روند عادی ادامه یابد.

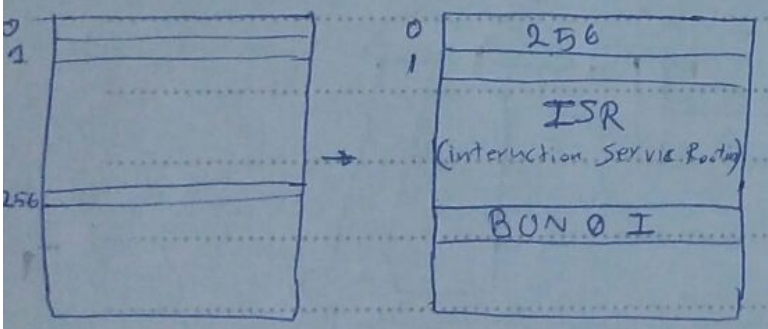
DATE / /

SUBJECT

چرخه‌ی دستورالعمل (کامپیوتر پایه)



نکته: بررسی وقفه در حین اجرا است.



وقتی درخواست وقفه در آدرس ۱ باشد به خانه‌ی ۲۵۶ میرویم.

وقتی درخواست وقفه در آدرس صفر باشد به خانه‌ی صفر میرویم.

وقتی در خانه‌ی ۲۵۶ هستیم: ریزعمل‌های مربوط به interrupt

$T_0' T_1' T_2 (IEN)(FG1 + FG0) : R \leftarrow 1$ (زیرا ممکن است در T0 دستور جدید به حافظه فرستاده شود)

نیاز به وقفه با شرایط فوق NOT میشود (در T0، T1، T2 این کار را انجام میدهیم)

$R\,T0 : AR \leftarrow 0 , TR \leftarrow PC$

$R\,T1 : M[AR] \leftarrow TR , PC \leftarrow 0$

$R\,T2 : PC \leftarrow PC+1 , IEN \leftarrow 0 , R \leftarrow 0 , SC \leftarrow 0$

DATE
SUBJECT

* بدلیل اضافه شدن R در چرخه دستورالعمل در شرط‌های زمانی اولین دستورات باید R' اضافه شود

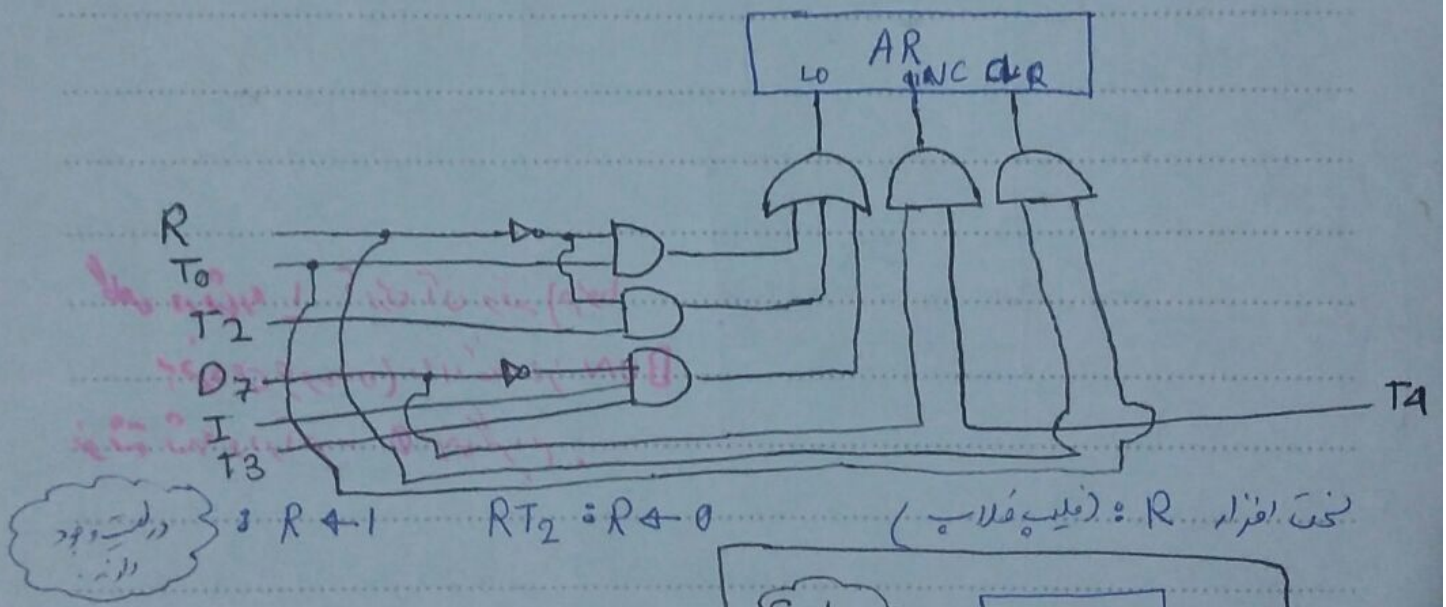
طراحی واحد کنترل: (سخت افزاری)

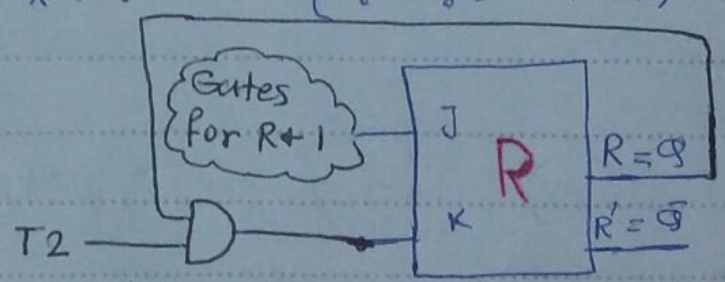
مثلاً با استفاده از لیست دستورات (میکرو اپریشن) یا ریزعمل‌های دستورات مراجعه به حافظه، I/O، ثبات‌ها کنترل‌های CLR، INC، LD مربوط به رجیستر AR را طراحی می‌کنیم.

برای کنترل LD: $D_7 I T_3 + R' T_2 + R' T_0$

$D_7 T_4$: INC ″ ″

$R T_0$: CLR ″ ″





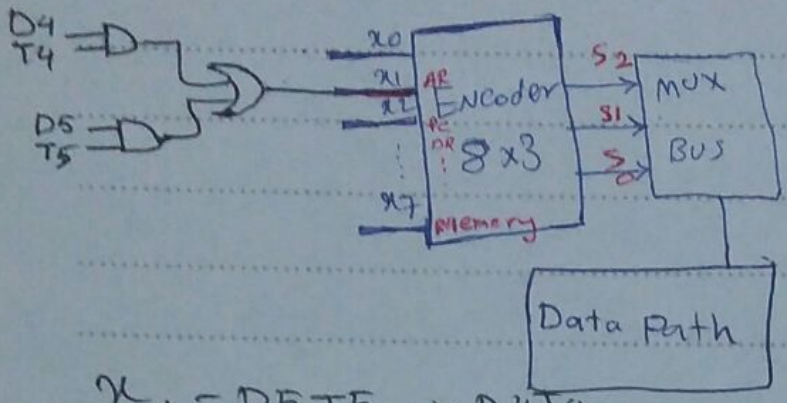
J آنهایی که قرار است R را 1 کنند

K ″ ″ ″ ″ ″ ″ ″ 0 ″

DATE ../../..

SUBJECT .....

ساخت افزار BUS :



$x_1 = D5T5 + D4T4$

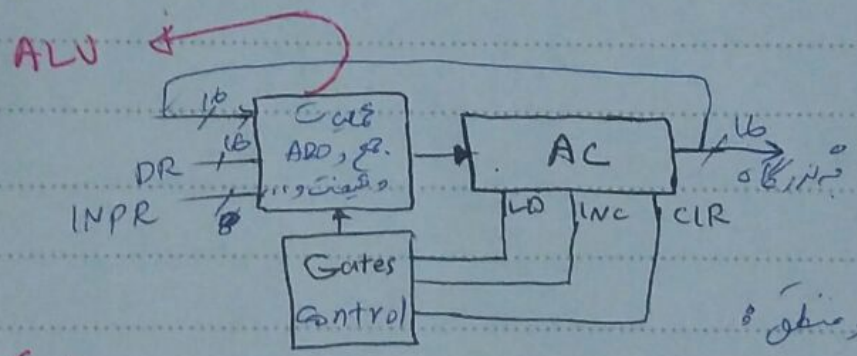
مثلاً برای زمانی که AR می خواهد روی BUS رود باید شرط فوق و شرط عمل های مربوط به آن را بنویسیم و طراحی کنیم.

(یعنی: D5T5 یا D4T4)

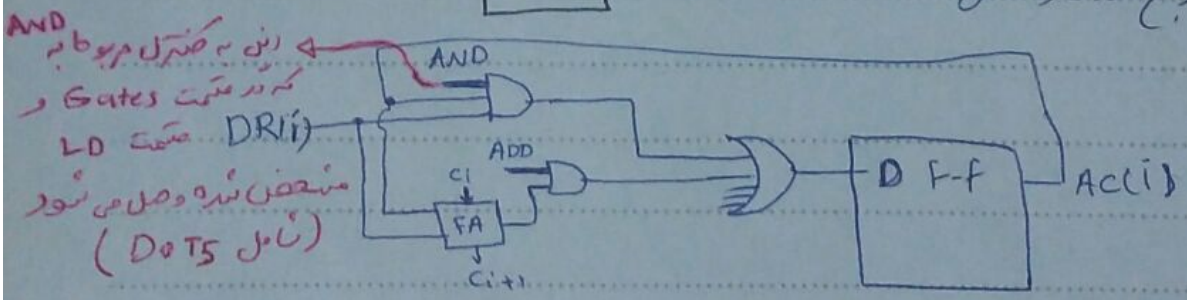
یا برای x7 (memory) باید آن شرط هم بصورت M[AR] ← است را بگیریم.

* نکته: ساخت افزار کنترل Read حافظه ای با x7 یکی است.

ساخت افزار AC :



یک طبقه از مدار جمع کننده و منطقی ALU



مثلاً برای AND داشتیم:

$D_0T_5 : AC \leftarrow AC \wedge DR$

یا برای ADD داشتیم:

$D_1T_5 : AC \leftarrow AC + DR$

DATE ..../..../....

SUBJECT .....

* برای نوشتن برنامه بصورت اسمبلی علاوه بر دستورات برنامه به ثابت ها و I/O، (دستورهایی) نیز نیاز داریم. (دستورالعمل : END, HEX, DEC, ORG)

فرم کلی هر سطر برنامه :

Label	دستور	پارامتر	I
↓			
~	~	~	~

نکته : شبه دستورها :

1) ORG 100 ⇝ مثلاً اجرا را از نقطه یا آدرس 100 شروع کن

2) DEC 83 ⇝ برای داده های در مبنای 10

3) HEX FF ⇝ برای داده های در مبنای 16

4) END ⇝ انتهای خطوط برنامه

دستورات برنامه نویسی شامل :

1 - حسابی
2 - LOOP
3 - فراخوانی زیر برنامه

0 مثال
01 حسابی

مثال ساده برنامه تفریق دو عدد : M - S

Label	دستورات	پارامتر	I	توضیح
	ORG 100			یعنی اجرا را از آدرس گاه ۱۰۰ شروع کن
	LDA S			یعنی آدرس S را در AC، بارگذاری کن
	CMA			یعنی AC را کامپلیمنت کن
	INC			یعنی AC را (با ۱ جمع کن) (2's complement)
	ADD M			یعنی ADD کن M را با محتویات AC
	STA D			یعنی AC را در آدرس D، ذخیره کن
	HLT			یعنی به برنامه خاتمه بده
M,	DEC 83			تعیین محتویات آدرس M در مبنای 10
S,	DEC -23			~ ~ ~ S ~ ~ ~
D,	HEX 0			16 ~ ~ D ~ ~ ~
	END			پایان خطوط برنامه نویسی اسمبلی

پس یعنی M - S را به صورت M + S' + 1 می‌کردیم

M - S
M + (-S)
M + S' + 1
(-S)

DATE ... / ... / ...

SUBJECT .....

LOOP : یک Label در برنامه نویسی اسمبلی است که بعنوان حلقه تکرار کاربرد دارد و معمولاً با ISZ (اگر آدرس حافظه را یکی اضافه کردیم و صفر شد دستور بعدی را اجرا نکن) همراه است.

و برای برگشتن به نقطه شروع LOP از BUN LOP استفاده می‌شود

مثال 82 صفحه

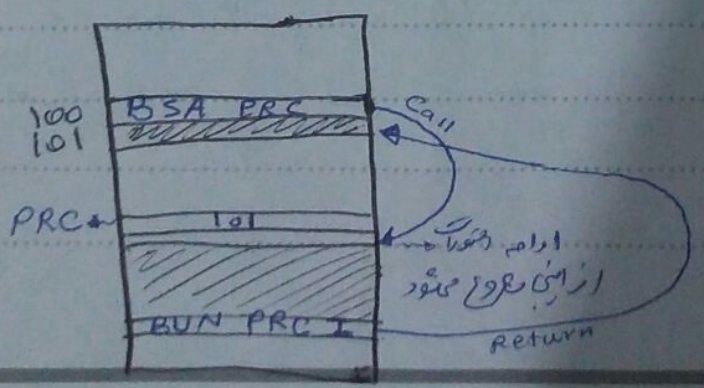
دستور مربوط به یک آرایه ۱۰۰ تایی LOOP : (جمع ۱۰۰ عدد)

```
        ORG   100
        LDA   ADS
        STA   PTR
        LDA   NBR
        STA   CTR
        CIA
LOP,    ADD   PTR  I
        ISZ   PTR
        ISZ   CTR
        BUN   LOP
        STA   SUM
        HLT
ADS,    HEX   150
PTR,    HEX   0
NBR,    DEC   -100
CTR,    HEX   0
SUM,    HEX   0
        END
```

یعنی به آدرس CTR اضافه کردن اگر صفر شد دستور بعدی که BUN است نگذرد و چون مقدارش CTR برابر ۱۰۰- (یک بایت ۱۰۰ مرتبه این حلقه تکرار شود و به LOP برگردد (یعنی ۱۰۰ مرتبه BUN LOP داریم)

فراخوانی زیر برنامه: یعنی ابتدا و حین اینکه دستورات اجرا می‌شوند نیاز باشد یک زیر برنامه‌ای اجرا شود (Call) سپس به برنامه اصلی برگردیم (Return)

که معمولاً از BSA (پرش کند و آدرس بازگشت را ذخیره کند) و BUN در آخر استفاده می‌شود.



DATE / /

SUBJECT

مثال: ۵3 سید از فراگیری زیر برنامه

```
    ORG 100
    LDA X
    BSA SH4
    STA X
    LDA Y
    BSA SH4
    STA Y
    HLT
X,  HEX 1234
Y,  HEX 4321
SH4, HEX 0
    CIL
    CIL
    CIL
    CIL
    AND MSK
    BUN SH4 I
MSK, HEX FFF0
    END
```

یعنی 2 تا فراگیری زیر برنامه داریم

در اینجا X را چهار مرتبه چرخش به چپ داده سپس با AND: FFF0 کرده که باعث می‌شود چهار بیت سمت راست نیز صفر شوند.

بصورت غیر مستقیم به آدرس اولیه برمی‌گردد

برنامه ورود یک کاراکتر ورودی و

```
CIF, SKI
    BUN CIF
    INP
    OUT
    STA CHR
    HLT
CHR, HEX 0
    END
```

* CIF همان loop است.

پرچم کی است پس دریافت ورودی /

دستور چاپ /

توضیح: در واقع تا زمانی که flag ورودی 0 است همیشه دستور اول اجرا می‌شود و هرزمان 1 شد BUN اجرا نشده و حلقه‌ای اجرا می‌شود پس INP خوانده شده و با دستور OUT چاپ می‌شود.

DATE / /

SUBJECT

برنامه خروجی یک کاراکتر :

```
        LDA   CHR
Cof,    BKO
        BUN   Cof
        OUT
        HLT
CHR,    HEX   —
```

DATE / /

SUBJECT

واحد کنترل ← به صورت سخت افزاری (سیم بندی)

به صورت برنامه ای

مشخصه CPU از نظر برنامه نویس: ۱) مجموعه دستورات ۲) مدهای آدرس دهی ۳) مجموعه ثبات ها ۴) مشخصات CPU

۵) مجموعه دستورات ۶) اندازه واژه بر حسب بیت

معماری های مختلف CPU: ۱) AC ماشین ها (که آکومولاتور دارند یا به عبارتی که دستورات مانند کامپیوتر PDP-8)

۲) General Register (مثل MIPS, Alpha که با رجیستر حافظه)

۳) ماشین های Stack (java)

Stack: به معنای پشته ← صدا زدن وقفه های نرم افزاری (نوع نرم افزار)

پیاده سازی Stack ← با استفاده از حافظه اصلی

← با استفاده از رجیستر ها

نمونه یک استک 64 بیتی: رجیستر 64 بیتی

SP: برای اشاره به رجیستر مورد نظر که پشته در آن کار می کند ← 6 بیتی

DR: 8 بیتی و داده یا آدرس مورد نظر را نگهداری می کند

دو فلیپ فلاپ EMTY و FULL جهت کنترل خالی یا پر بودن استک

$$EMTY = \begin{cases} 0 & \text{خالی نیست (پر است)} \\ 1 & \text{خالی است} \end{cases} \qquad FULL = \begin{cases} 0 & \text{پر نیست (خالی است)} \\ 1 & \text{پر است} \end{cases}$$

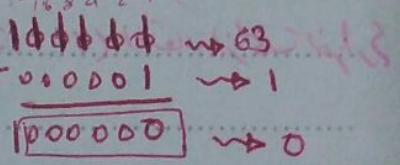
DATE ... / ... / ...

SUBJECT .....

پشته با داده می‌گیرد یا می‌دهد به DR :

Push (گرفتن) (SP=0 , EMTY=1 , full=0 مقدار اولیه)

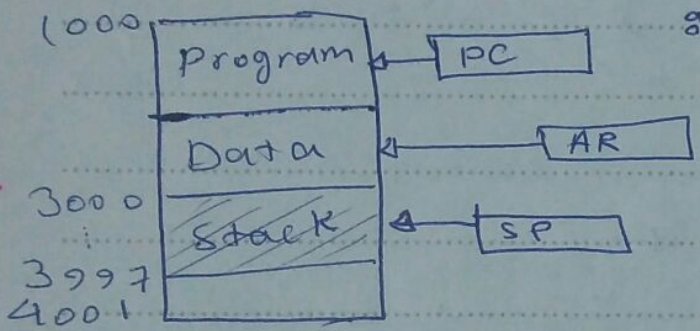
- SP ← SP+1
- M[SP] ← DR , EMTY ← 0
- if (SP = 0) then (full ← 1)



POP (برداشتن)

- DR ← M[SP] , full ← 0
- SP ← SP-1
- if (SP=0) then (EMTY ← 1)

پیاده سازی پشته از RAM :



Push: SP ← SP-1
M[SP] ← DR

POP: DR ← M[SP]
SP ← SP+1

کاربرد استک : انجام حالت (عملیات منطقی)

* انواع شکل‌های دستورات حسابی :

infix : A+B

prefix : +AB

postFix : AB+ (لهستانی)

در Stack عملیات لهستانی قابل انجام است. (زبان خاصی دارد)

که تبدیل به postFix به عهده کامپایلرها هستند.

مثال : دستور جمع AB+ در زبان استک : A در خانه 1 مرد گرفته

Push A → A در خانه اول (پشته) ←

Push B → B در خانه دوم (پشته) ←

ADD → جمع می‌شوند و حاصل در خانه 1 ←

POP → حاصل فرستاده شد به DR

Subject :

Year. Month. Date. ( )

- سازمان پشته برای ارزیابی عبارات حسابی بسیار مفید است.
- جهت انجام محاسبات ریاضی باید فرم میانوندی را به پیوندی تغییر داد.

پیش نمایش :

پیشوندی : عملگر قبل از معلوند قرار می‌گیرد.

پسوندی : عملگر بعد از معلوند قرار می‌گیرد. ← (RPN)

مثال از نمایش پسوندی : $A*B+C*D = AB*CD*+$

و برای حالت برعکس همین از نمایش به میانوندی : عبارت را از چپ به راست دنبال کرده و وقتی به عملگری رسیدیم آنرا بروی دو معلوند قبلی اعمال کرده سپس دو معلوند را حذف و آنها را با حاصل جایگزین کرده و این عمل را ادامه داده تا به عملگری برخورد نشود.

* در تبدیل به نمایش ابتدا اولویت به پرانتزها و سپس ضرب ها و سپس جمع هاست.

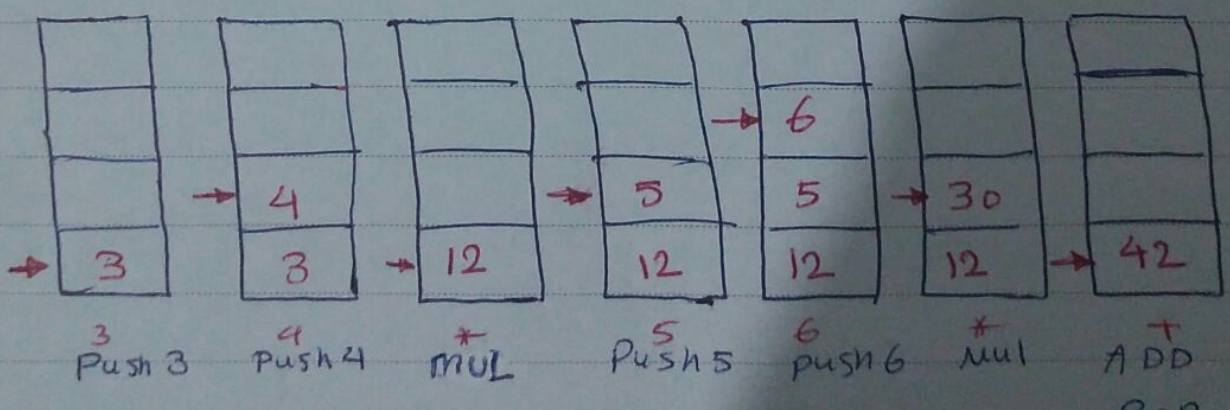
$$(A+B)*[C*(D+E)+F] = AB+DE+C*F+**$$

مثال عملی در استک :

می‌خواهیم $(3*4)+(5*6)$ را در استک انجام دهیم.

فرم نمایش = $34*56*+$

عمل ها بترتیب زیر هستند :



Subject :

Year. Month. Date. ( )

- مثال فرم دستورات 3 آدرسه :

```
ADD   R1, A, B      R1 ← M[A] + M[B]
MUL   X, R1, R2     M[X] ← R1 * R2
```

- مثال از فرم ۲ آدرسه :

```
MOV   R1, A     R1 ← M[A]
ADD   R1, B     R1 ← R1 + M[B]
MUL   R1, R2    R1 ← R1 * R2
```

(یکطرف همان خروجی است)

- مثال از فرم ۱ آدرسه :

```
Load  A     AC ← M[A]
ADD   B     AC ← AC + M[B]
Store X     M[X] ← AC
```

(یکطرف AC است)

- مثال دستور صفر آدرسه :

```
Push  A     TOS ← A
ADD         TOS ← (A+B)
MUL         TOS ← (A+B) * (C+D)
Pop   X     M[X] ← TOS
```

(بالای پشته) (یکطرف بالای پشته است)

Subject :

Year. Month. Date. ( )

خدمت دستورات :

دستورات ۳ آدرسه : هزینه : کوتاهی برنامه در ازای بی نیازی محاسباتی

مثلا : ADD R1, A, B

یعنی تعداد زیادی بیت (48 بیت) برای انجام محاسبه ی بالا تخصیص داده است.

دستورات دو آدرسه : مثل : ADD R1, B

دستورات یک آدرسه : در این نمونه فرض بر این است که یک طرف محاسبات AC

آکومولاتور است مثل :

LOAD A

ADD B

دستورات صفر آدرسه : نمونه یک کامپیوتر سازمان یافته با Stack پایشته است یعنی

تنها دستورات Push و POP برای تعیین عملوندی که با پشته تبادل اطلاعات

می کند نیاز به آدرس دارند. مثلاً :

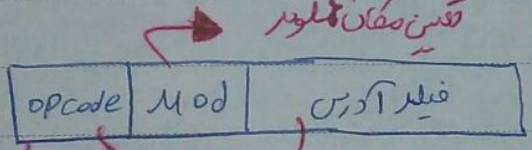
Push A

Push B

ADD

در واقع بدلیل عدم وجود میدان آدرس در این نوع کامپیوتر

به آنها صفر آدرسه گویند.



Mod (روش ، شیوه) آدرس دهی :

۱- ضمنی : عملوند در خود عملوند (تعریف خود عملوند) مستتر است مثل : CMA

آدرس دهی یا دستورات Stack از نوع ضمنی هستند

۲- بلافصل (فوری) : مقدار عملوند در خود دستورالعمل است یا به عبارت دیگر یک دستورالعمل

از این نوع به جای آدرس دارای عملوند است. مثل : MOV CX, 1024

عملوند

۳- ثباتی : فیلد آدرس یک رجیستر یا ثبات را مشخص می کند و در آن ثبات عملوند وجود دارد.

ADD CX, AX ; (CX ← CX+AX)

۴- ثباتی غیر مستقیم : فیلد آدرس یک رجیستر را مشخص می کند که محتوای آن رجیستر یک

عملوند در حافظه می باشد.

MOV BX, [SI]

Subject :

Year. Month. Date. ( )

۵- روش خود افزایش یا خودکاهش :

مانند حالت ۴ است با این تفاوت که پس از هر بار استفاده از آدرس حافظه، مقدار آن در ثبات بطور خودکار افزایش یا کاهش می‌یابد.

افزایش : پس از اجرا، محتوای رجیستر یکی افزایش

کاهش : قبل از اجرا محتوای رجیستر یکی کاهش

نکته : آدرس موثر : آدرس آن word حافظه که عملوند در آن قرار دارد.

۶- مود آدرس دهی مستقیم : عملوند واقع در حافظه است که فیلد آدرس مستقیم

MOV AX, 3000 : AX ← M[3...]

مشخص می‌کند

۷- روش آدرس دهی غیر مستقیم : آدرس آدرس عملوند در فیلد آدرس دستورالعمل است.

* روش‌های زیر از نوع آدرسی هستند که :

آدرس موثر = (فیلد) بخش آدرس دستور + محتویات ثبات خاصی در CPU

۸- مود آدرس دهی نسبی :

EA = PC + Address field

12bit 12bit

۹- آدرس دهی با ثبات پایه :

EA = BX + Address field

12bit 12bit

BX ثابت و آدرس تغییر می‌یابد

← برای جابجایی برنامه هاست.

۱۰- مد آدرس دهی شاخص :

EA = IX + Add field

← استفاده برای آرایه ها که محل شروع آرایه در حافظه مشخص در دستورالعمل مشخص می‌شود و IX مشخص می‌کند کدام عنصر آرایه مشخص شود.

IX متغیر ولی آدرس ثابت است "

* هر کامپیوتر تعداد خاصی از این مودها را دارد مثلاً

کامپیوتر ۱ : مستقیم و غیرمستقیم (I=0 / I=1)

پردازنده Berkeley : بلافصل - نسبی - ثباتی

Subject:
Year. Month. Date. ( )

مجموعه دستورات کامپیوتر: مجموعه دستوراتی که برای ایجاد ارتباط در کامپیوتر برای انجام محاسباتی در اختیار کاربر قرار می دهد

— در اینجا دستورات مشترک بین تقریباً همه کامپیوترها بررسی می شود.

۱) دستورالعمل های انتقال داده:

نوع انتقال داده بین حافظه و ... می شود

سمبل	نام
IN	ورودی
OUT	خروجی
PUSH	فشردن در استک
POP	برداشتن از استک

سمبل	نام
LD	لود کردن
ST	ذخیره
MOV	انتقال
XCH	تبادل

* نکته * معمولاً دستورات انتقال داده، mod های آدرس دهی همراه اند مثلاً برای دستور LD ، 8 مود آدرس دهی وجود دارد.

— جدول سمبل های مود آدرس دهی: (برای LD)

روش	قرارداد اسمبلی	انتقال ثباتی
آدرس دهی مستقیم	LD ADR (ADR: بدون آدرس)	AC ← M[ADR]
آدرس غیرمستقیم	LD @ADR	AC ← M[M[ADR]]
آدرس نسبی	LD $ADR	AC ← M[PC+ADR]
عملوند بلافصل	LD #NBR	AC ← NBR
آدرس دهی شاخص دار	LD ADR(X)	AC ← M[ADR+XR]
آدرس دهی ثباتی	LD R1	AC ← R1
ثباتی غیرمستقیم	LD(R1)	AC ← M[R1]
خود افزایش	LD(R1)+	AC ← M[R1], R1 ← R1+1

۲) دستورالعمل های دستکاری داده ها: شامل

- دستورالعمل های حسابی
- " " منطقی و دستکاری بیت ها
- " " شیفت

Subject :

Year. Month. Date. ( )

✓ دستورالعمل های حسابی :

نام	سمبل	نام	سمبل
افزایش	INC	جمع	ADD
کاهش	DEC	تفریق	SUB
جمع با نقلی	ADDC	ضرب	MVL
تفریق با قرض	SUBB	تقسیم	DIV
متمم 2	NEG		

✓ دستورالعمل های منطقی و دیگر جهت دهی :

نام	سمبل
پاک کردن	CLR
متمم کردن	COM
XOR یا AND یا OR	AND / OR / XOR
فعال کردن وقفه	EI
غیرفعال کردن وقفه	DI
صفر کردن (۱) نقلی	CLRC / SETC

✓ دستورالعمل های شیفت :

نام	سمبل
شیفت منطقی به راست	SHR
" " " چپ	SHL Arithmatic
شیفت حسابی به راست	SHRA
" " " چپ	SHLA
چرخش به راست / چپ	RoR / RoL
چرخش به چپ با نقلی	RoLC

(۳) دستورات کنترل برنامه :

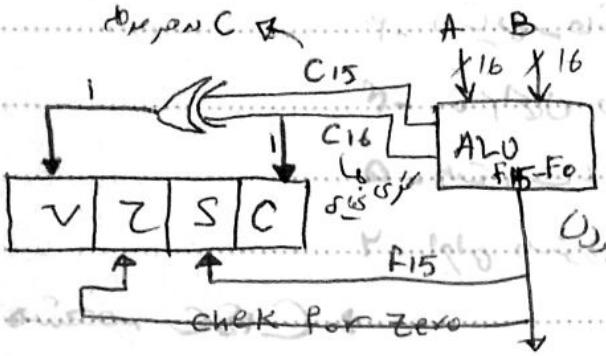
نام	سمبل
انشعاب	BR
پرش	JMP
گذر	SKP
فراخوانی	CALL
بازگشت	RET
مقایسه	CMP
آزمون (تست)	TST

DATE ../../..

SUBJECT .....

سخت افزار مربوط به رجیستر (ثبات) بیت های وضعیت ؟

بیت های وضعیت ، وضعیت درون CPU را مشخص می کند.



C: Carry out .

S: Sign (علامت)

Z: بررسی خروجی از نظر صفر بودن یا نبودن

V: سرریز (در واقع اگر خروجی بیش از 127 یا کمتر از -128 برای ALU هشت بیتی شد) هرگاه دو عدد هم علامت (مثبت) با هم جمع شود سرریز داریم ($V=1$)

مثال:

$$\begin{array}{r} 01 \\ 01111010 \\ + \quad 00011110 \\ \hline 10010000 \end{array}$$

$V=0\oplus1=1$, $C=0$, $S=1$, $Z=0$

انواع وقفه ؟

خارجی (I/O) : توسط یک وسیله خارجی (پرینتر، منبع تغذیه و...)

به CPU اعلان می دارد سه با ایجاد سیگنال

داخلی: در اثر استفاده غلط یا نابجا (مثل تقسیم بر صفر شدن، سرریز شدن stack، دستور العمل غلط و...) سه با ایجاد سیگنال به CPU اعلان می دهد.

نرم افزاری: نوع خاصی از صدا زدن حفره برنامه فرکس است برای انجام عملیات خاصی (مانند: اینکه نرم افزار برای اینکه کاربر تغییراتی در کد باکیبورد)

تفاوت فراخوانی زیر برنامه و وقفه ؟ ① در زیر برنامه تنها آدرس بازگشت ذخیره می شود. و در وقفه علاوه بر این اطلاعاتی مثل وضعیت بیت های وضعیت CPU در رجیستر ها منتقل می شود

② فراخوانی زیر برنامه در خط خاصی از برنامه رخ می دهد و لی وقفه هر خطی رخ می دهد.

DATE

SUBJECT

کامپیوترهای RISC :

مشخصه ها:

۱- دستورات نسبتاً کم (تعداد) ۲- روش های آدرس دهی محدود

۳- دسترسی به حافظه منحصر است به دستورات LD و ST

۴- کار انجام در داخل ثبات های CPU انجام می شود

۵- دستورات با طول ثابت مبادرت می شود

۶- اجرای دستورات در یک سیکل Pipeline ۷

مشخصه CISC :

۱- تعداد دستورات زیاد معمولاً بین ۱۰۰ تا ۲۵۰ دستور

۲- دستورالعمل هایی که کار خاصی انجام می دهند ولی بندرت کاربرد دارند

3- انواع متنوعی از آدرس دهی معمولاً بین 5 الی ۲۰ روش

4- طول دستورات با طول متغیر

5- دستوراتی که محاسبات را در حافظه انجام می دهند

Benkeley Risk : نمونه یک کامپیوتر نوع Risk است.

① دارای ۳۲ مود آدرس دهی : بلافصل - مستقیم و ثباتی است

② دارای 31 دستور ③ طول دستورات 32 بیتی است ④ آدرس دهی مبتنی جهت انشعاب

⑤ 138 ثبات دارد ⑥ سه فرمت دستور دارد ⑦ R0 مقدار 0 را دارد.

برنامه اسمبلی این کامپیوتر:

ADD R22, R21, R23 R23 ← R22 + R21

ADD R0, #150, R22 R22 ← 150 + 0

LDL (R22), #150, R5 R5 ← M[R22] + 150

SLL : شیفت به چپ (منطقی) SLA : (شیفت حسابی به چپ)

SLL R22, #2, R23 R23 ← R22 shifted by 2

JMP : پرش شرطی JMP Z, #3200, R1 PC ← R1 + 3200

پرش نسبی شرطی

JMPR : پرش نسبی JMPR Z, 200 PC ← 200 + PC

به شرط Z=1

Subject :

Year. Month. Date. ( )

MIPS : Microprocessor without interlocked pipline Stages معماری پردازنده

پردازنده‌ای که حفاظ‌گونه دارد و بین طبقات حفاظ‌گونه

اینترلاک (قفل) وجود ندارد یا بسیاری Pipline می‌توانند بدون هیچ مشکل به کار خود

ادامه دهد. توسط Henecy ارائه شد.

عملکرد Performance نهایی یک کامپیوتر بصورت ضرب تعداد دستورالعمل‌ها (متناسب با ISA است)

کامپایلر که هرچه بهتر باشد تعداد دستور کمتر خواهد بود و همچنین متناسب با CPI در cycle time سپس بنابراین مدت زمان اجرای کل برنامه به CPI

و cycle time وابسته می‌کنیم و این را به CPU (سخت‌افزار) مربوط‌اند

در اینجا نمونه ساده‌ای از پردازنده MIPS را بررسی می‌کنیم.

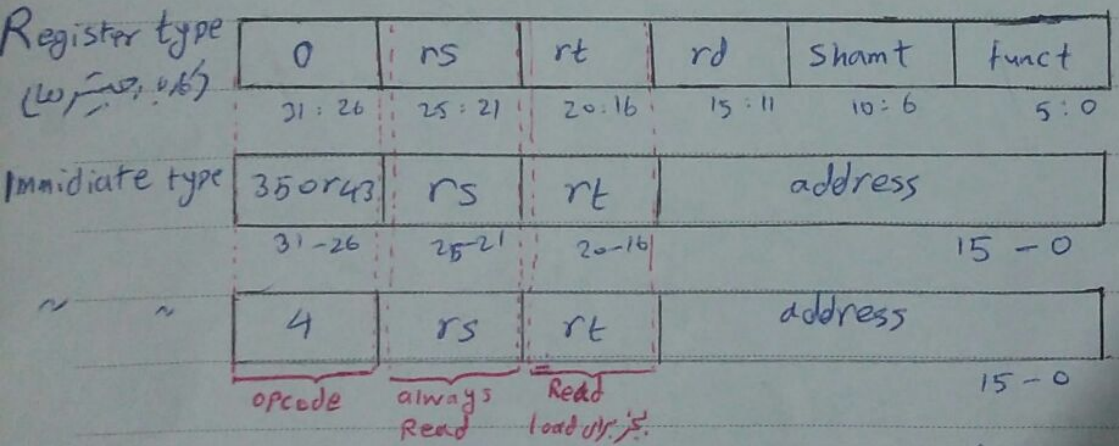
مطالعه 3 دسته دستور :

1. Memory Reference : lw , sw
2. Arithmatic/Logic : add , sub , and , or , slt
3. Control transfer : beq , j

* انجام دستورات ، اجرا در یک Clock در Mips انجام می‌شود

در طراحی ابتدا ۱ فرمت دستور را بررسی می‌کنیم (مانند کامپیوتر PDP-8)

در MIPS سه نوع فرمت دستور داریم : ① R ② I ③ J (بعداً بررسی می‌شود)



R-type ⊖

⊕ در R-type بدلیل اینکه کار با رجیسترها را داریم مثلاً برای دستور جمع کردن نیاز به سه رجیستر داریم (۲ تا برای جمع و یکی برای نتیجه) سپس rs , rt برای جمع ، rd برای نتیجه

⊕ فیلدهای funct , 0 , shamt عملیاتی که قرار است انجام دهیم را مشخص می‌کنند

⊕ در این پردازنده 32 رجیستر داریم لذا برای انتخاب این‌ها 5 بیت لازم داریم.

rs , rt , rd همه‌ی بیتی‌اند

ASEMAN

⊕ funct در کنار 0 (opcode) می‌تواند عملیات بیشتری را مشخص کند.

Subject :

Year. Month. Date. ( )

⊞ shamt برای shift استفاده می‌شود.

⊟ I-type

⊟ load/store

⊞ opcode برای 43 ، 35

⊞ load برای rs ، store برای rt ، load است و store برای write

⊞ چون آدرس 32 بیتی نیاز است علاوه rs (16 بیتی) 16 بیتی آدرس

آدرس را مشخص می‌کند.

⊟ Branch

⊞ rs ، rt را باهم مقایسه می‌کند و اگر با هم برابر بودند آنگاه Branch می‌کند

(آدرس Branch = در صفحه بعد !!!)

طراحی :

نکته ① : در طراحی ALU باید کارهای زیر را انجام دهد :

1. Arithmetic result / 2. memory address for load and store / 3. Branch target address

نکته ② : PC باید بعد از انجام عملیات فراخوانی باید و برای فراخوانی PC در Mips

PC ← PC + 4 (0100) (یعنی PC چهار تا چهار تا فراخوانی می‌شود)

و باهم اگر دستور Branch، PC ← target address نتیجه

نکته ③ در طراحی ۲ قسمت داریم : 1. instruction MEM

2. Data MEM

همچنین Register file داریم که 32 رجیستر در آن قرار دارند.

ASEMAN

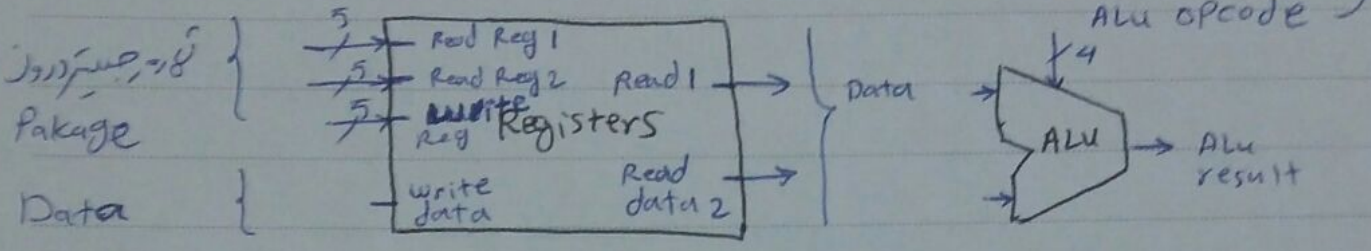
Subject:

Year. Month. Date. ( )

طراحی مسیر داده‌ها : R-type : شامل

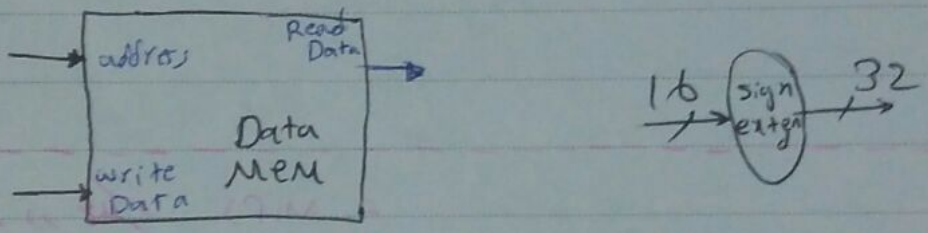
a. Read two register operands
b. Perform arithmetic/logical operation
c. write register result

یعنی ALU در ۱۶ حالت مختلف می‌تواند عمل کند.



گروه حافظه‌ها
Pakage
Data

طراحی مسیر Load/Store : i-type مربوط به دستورات



چون آدرس باید 32 بیتی باشد لذا برای تبدیل کردن آدرس و ورودی Data Mem باید از سخت افزار Sign extend داریم.

بدین گونه که ابتدا 16 بیت آدرس را Sign extend کرده سپس با rs جمع می‌کنیم.

کوه کارمی : S-E : تبدیل 16 به 32 انجام می‌شود مثلاً :

(عدد +) 7000 → 00007000
HEX
(عدد منفی) 8000 → FFFF8000

طراحی مسیر Branch : i-type مربوط به دستورات بین

در واقع rs و rt را مقایسه می‌کند اگر برابر بودند برنچ می‌کند و خروجی صفر فعال می‌شود.

target address (آدرس برنچ) :

✓ Sign extend
✓ Shift left 2 Places
✓ PC ← PC + 4

Shift left به این دلیل است که هر دستور باید دوبیت پایین صفر باشد.

Subject :

Year. Month. Date. ( )

سومین نوع فرمت دستور Jump است.

JUMP

2	address
31-26	25 - 0

j-type ⊟

⊞ به آدرس مشخص شده توسط فیلد می‌رود (منتها با قبل از همون 32 بیتی کردن آدرس) ← PC را به روز می‌کند.

⊞ update PC : 2 بیت به چپ ← 26 bit ✓ 2 بیت به سمت چپ shift می‌شود

✓ 4 بیت دیگر با استفاده از 4 بیت بالای PC می‌آید و به 4 بیت بالای PC آپدیت شده می‌پیوندد (با استفاده از سیم بندی)

کمی حالت بد در عملکرد MIPS :

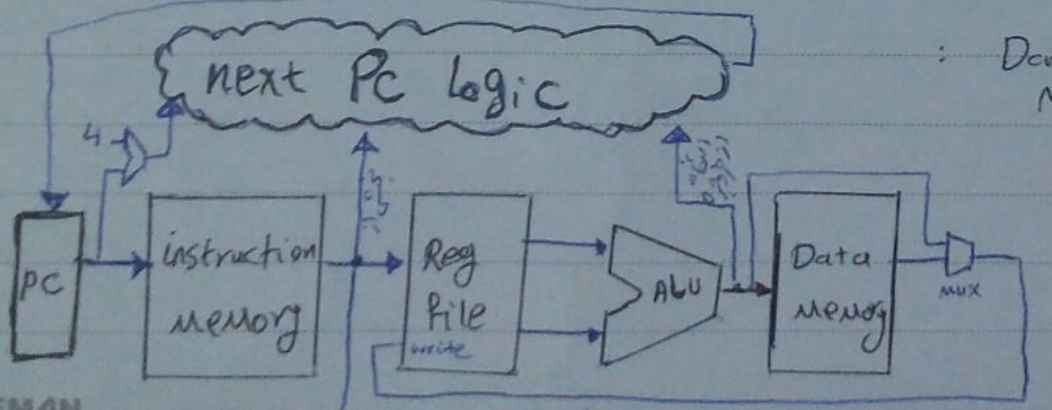
طولانی شدن یک دستور :

مثلاً دستور load این مراحل را طی می‌کند.

instruction mem (200 ps) → register file (100 ps) → ALU (200 ps) → data MEM (200 ps) → register file (100 ps) = 800 ps

و این باعث طولانی شدن clock می‌شود (زیرا clock هر دستور در این پردازنده با طولانی‌ترین clock ، ست می‌شود)

← برای حل این مشکل از Pipeline استفاده می‌شود. (200 ps)



شکل مسیر Data Path : MIPS

Control unit ، next PC با استفاده از دو شکل طرح شده دیده می‌شوند!!

ASEMAN

instruction Fetch (IF) | Control Unit | instruction decode (ID) | Execute (EX) | Memory (MEM) | write Back (WB)

Subject :

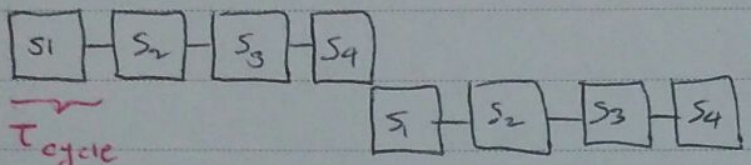
Year. Month. Date. ( )

"Pipe line"

Pipeline برای شکستن گام‌های یک عمل ترتیبی !!

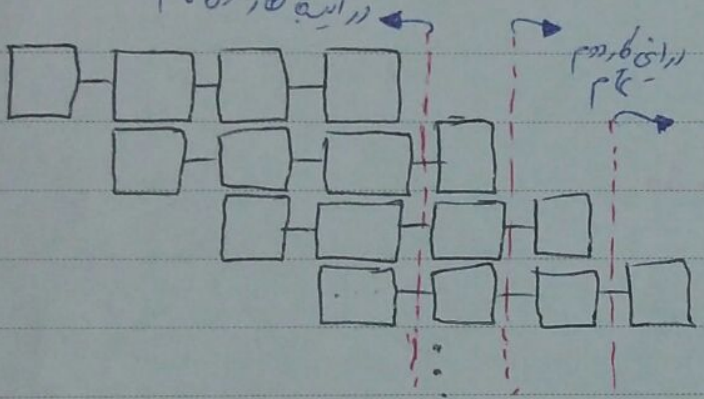
برای مثال کار 4 مرحله‌ای را در نظر بگیرید اگر بخواهیم n تا کار k مرحله‌ای را انجام دهیم داریم :

بدون Pipeline :



$T_{non-pipeline} = n \times k \times T_{cycle}$ | in here $k=4$, $n=2$ $= 2 \times 4 \times T_c$

با Pipeline :



کار اول در k کلاک

$T_{Pipeline} = \overbrace{k \times T_c}^{} + (n-1) \times 1 \times T_c$

بقیه (n-1) کار بعدی در 1 کلاک انجام می‌شوند

$S_{Read\ up} = \frac{T_{nonPipe}}{T_{Pipe}} = \frac{nk t_p}{[k+(n-1)]t_p}$, $S_{max} = \frac{t_n}{t_p}$, $t_n = k \cdot t_p$

نمونه انجام برون خط روی

اگر تعداد زیادی Task داشته باشیم :

$\lim_{n \to \infty} S = \frac{nk}{n} = k$

* یعنی اگر Pipe line استفاده کنیم k (تعداد مرحله) برابر شدن سرعت انجام برنامه‌ها را داریم .

* t_p زمان انجام هر Stage ضرب با حاصلترین ترین Stage است.

- در Pipe line بین Stage's از رجیستر استفاده می‌کنیم برای اینکه داده‌ها روی همدیگر نیفتند. (رجیسترهای بزرگ) سه مثلاً چند ده بیتی اند.
- خواندن ها در نیمه دوم کلاک و نوشتن ها در نیمه اول یک کلاک انجام می‌شوند.
- رجیسترهای بین طبقات برای : ① ذخیره کردن داده‌های هر Stage از یک Task
② ذخیره کردن شماره رجیسترهای که قرار است در آن Write انجام شود با خود حمل می‌کنیم در هر مرحله)
③ ذخیره کردن سیگنال‌های کنترلی برای هر task

Subject :

Year. Month. Date. ( )

خاطرات pipeline برای پردازنده MIPS

۱- ساختاری ۲- داده ای ۳- کنترلی

Structure Hazard : ← (با اضافه کردن سخت افزار رفع می شود)

مثلاً دستوری از یک منبع (مثل Memory) در اجرا بصورت pipeline و باعث متوقف شدن pipeline می شود. (توقف برنامه دیگر) ← ایجاد حباب Bubble

هم Read و هم Write نیاز داریم ← لذا برای رفع مشکل Mem ها را جدا می کنیم

در پردازنده MIPS نیز Data Mem , inst Mem را جدا کرده اند

Data Hazard :

دستور اول یک داده ای را تولید می کند که دستور دوم از این دستور می خواهد استفاده کند.

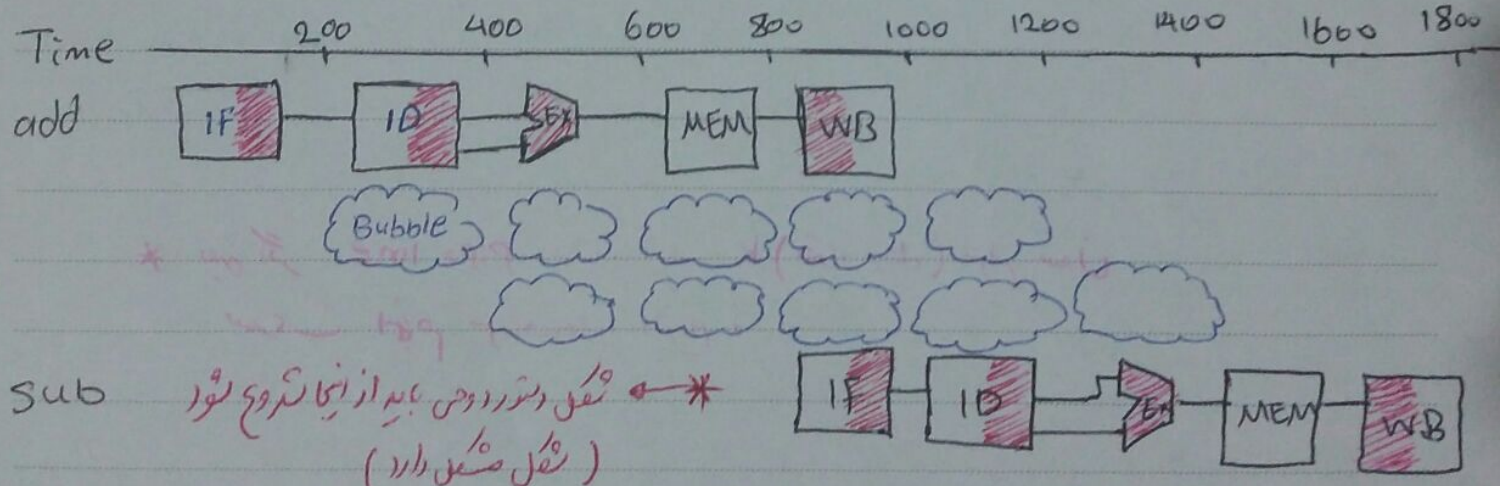
(اولی پیش نیاز دومی)

مثلاً :

```
add.  $s0 , $t0 , $t1
sub   $t2 , $s0 , $t3
```

هنوز در اجرای دستور sub به $s0 نیاز داریم و این هنوز تولید نشده است.

add : جمع Reg t0 , Reg t1 و قرار دادن نتیجه در s0



شکل دستور دومی باید از اینجا شروع شود (شکل مشکل دارد)

- دستور دوم (sub) می تواند در ردیف دوم اجرا شود به دلیل Data Hazard به این ترتیب خط لوله متوقف می شود تا وقتی که مقدار مورد نیاز $s0 آماده شود (s0 در 600ps آماده می شود ، از مموری به مدت 200ps استفاده می شود و نهایتاً در 900ps در رجیستر قرار داده می شود و آماده استفاده است) و خط لوله آماده کار می شود و دستور sub می تواند اجرا شود.

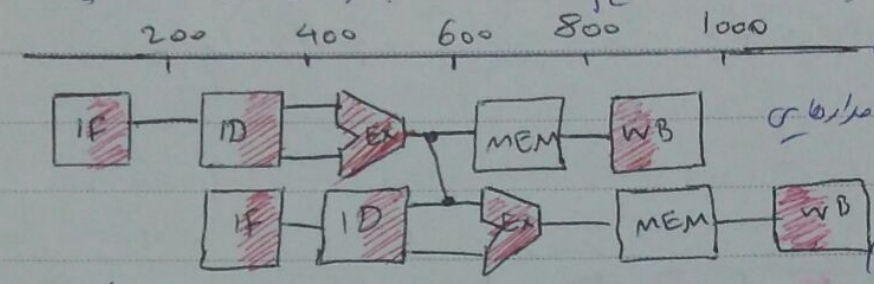
- در این مثال از MEM استفاده نمی شود لذا این حالت سیستم بی کار است

ASEMAN

Subject :

Year. Month. Date. ( )

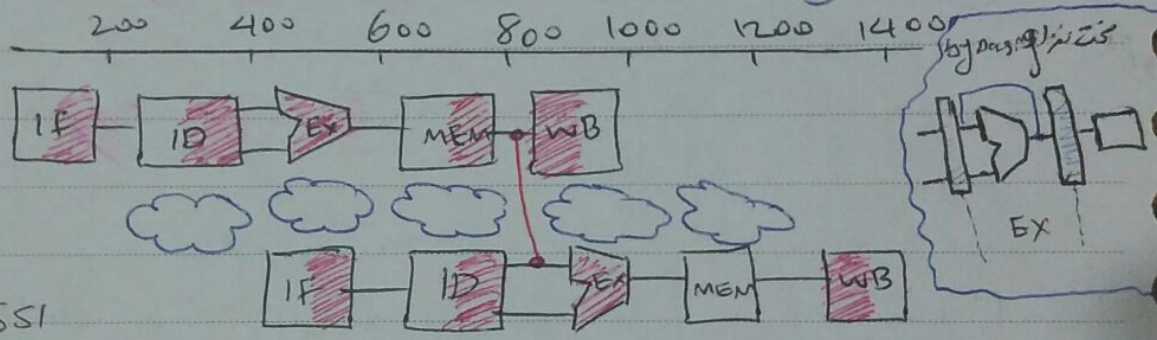
نباید این در این مدل حتی یک interlike ایجاد شده و نیستم در interlike, Mips وجود ندارد و برای رفع این مشکل از تکنیک bypassing استفاده می‌شود تا حباب‌ها موجود از بین بروند. پس در واقع Forwarding به این معنی است که ساختار پردازنده باید بگونه‌ای باشد که



این کار لازم است طبق مدارهای از خروجی ALU به ورودی ALU مثل بالم

* همچنین در برخی موارد می‌توان از bypassing استفاده کرد ولی باز هم حباب داریم.

مثلاً برای دستور load از حافظه



- نکته: پس می‌توانیم نتیجه بگیریم بعد از هر دستور lw یک ردیف حباب وجود دارد لذا می‌توانیم از آن ردیف حباب برای استفاده‌ی دستور دیگری استفاده ببریم.

مثال: C code for A = B + E ; C = B + F

$t3 ← $t1 ← $t2 ← $t5 ← $t4

```
lw  $t1, 0($t0)
lw  $t2, 4($t0)
add $t3, $t1, $t2
sw  $t3, 12($t0)
lw  $t4, 8($t0)
add $t5, $t1, $t4
sw  $t5, 16($t0)
```

حباب ← حباب

با استفاده از forwarding حباب‌ها کاهش یا از بین می‌روند

```
lw  $t1, 0($t0)
lw  $t2, 4($t0)
lw  $t4, 8($t0)
add $t3, $t1, $t2
sw  $t3, 12($t0)
add $t5, $t1, $t4
sw  $t5, 16($t0)
```

ASEM4N

5+6+2 = 13 cycle

حباب دستورات هم دستور اول

5+6 = 11 cycle

دستورات بدون استور اول

Subject :

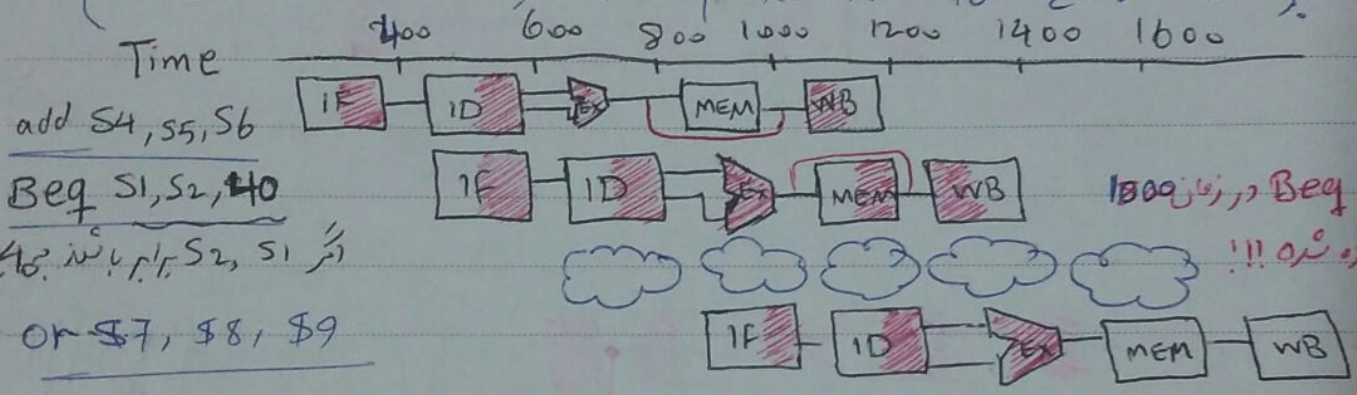
Year. Month. Date. ( )

همین کامپایلرها دستورات را بگونه‌ای مرتب می‌کنند که حباب‌ها از بین روند و در اینصورت مدار سیکل‌ها کاهش (در این مثال حباب‌ها صفر شده‌اند) می‌یابد.

- بنابراین مشکل حافظه داده‌ای را می‌توان با bypassing و جابجا کردن دستورات (با کامپایلر) حل کرد.

Control Hazard: ← این مخاطرات باعث می‌شود PC تغییر کند مثل انشعاب

دستورات Branch یا Jump شرطی باید کاملاً انجام شوند سپس دستور بعدی اجرا شود. (زمانی حاصل beq در پایان مرحله چهارم MEM توسط Zero bit آماده و مشخص می‌شود)



* نتیجه Beq در زمان 1000 مشخص شده!!

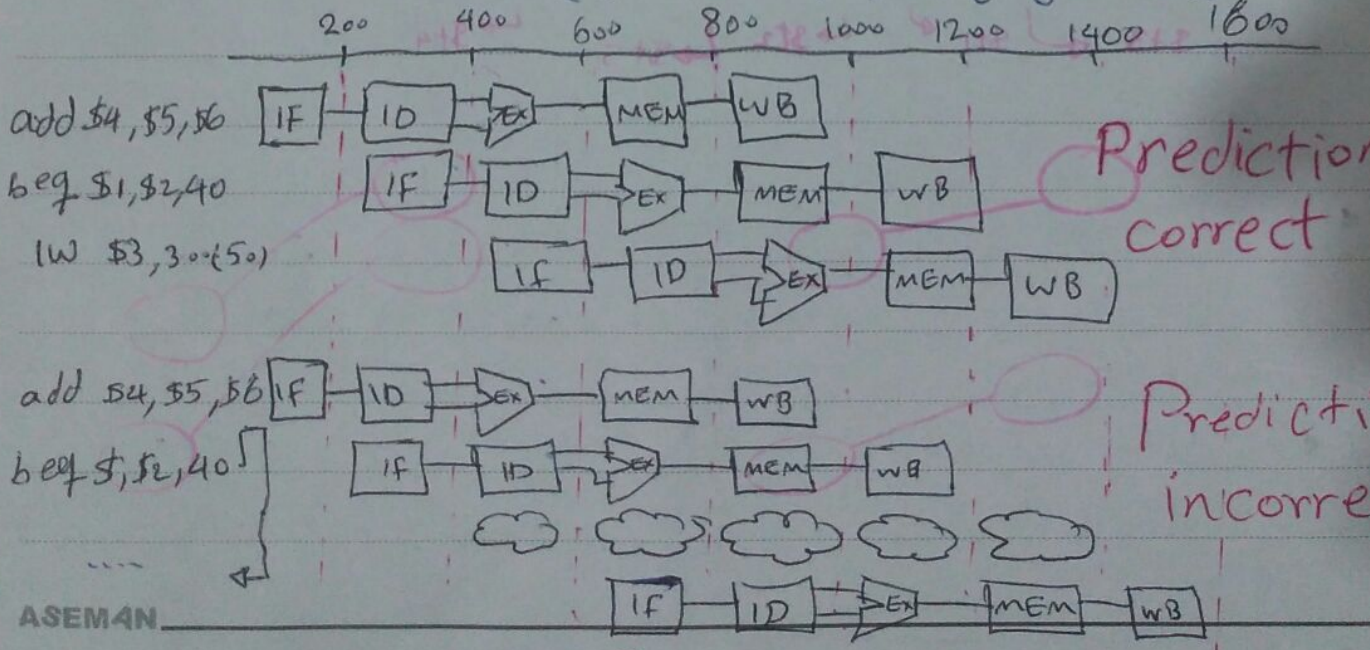
راه حل برای این مشکل: (برای Branch)

1. Prediction (پیشبینی) برای taken بودن یا نبودن انجام می‌شود

- taken (Branch رخ می‌دهد) ← دستورات loop for.....end مانند ← در نظر گرفتن taken
- NOT Taken (Branch رخ نمی‌دهد) ← دستورات if ← در نظر گرفتن NOT Taken

در واقع پیشبینی می‌کنیم که شرط رخ می‌دهد لذا به آدرس مورد نظر رفته و ادامه می‌دهیم

* در دستورات حلقی رخ می‌دهد (loop)



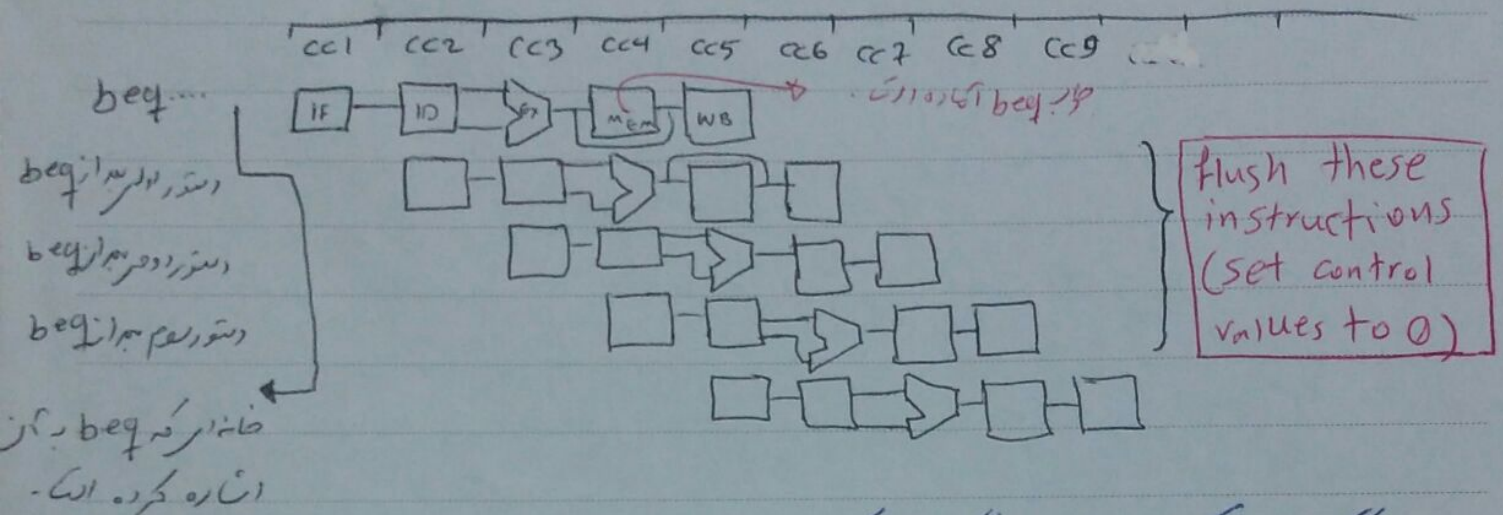
درصورت خبر صحیح بودن همان حباب‌ها را ایجاد می‌کند.

Subject

Year. Month. Date. ( )

2. Delay Branch :

این کار پیچیده‌ی کامپایلر است؛ به این طریق که ابتدا همه دستورات بعد از Beq اجرا می‌شوند و در واقع دستور مربوط به Beq بعد از ۳ دستور بعد از آن نیز اجرا می‌شود لذا کامپایلر در صورتی که مقدار بوده Branch رخ دهد مقدار No operation می‌گذارد (مقدار NOP را در رجیستر IF/ID گذاشته و باعث رفتن این جیب تاریخ می‌شود)



اگر بتوان یک سخت افزار جبرانی که در مرحله ID باشد اضافه کرد که بتواند تشخیص دهد که آیا دو مقدار برابرند یا نه؛ توانسته‌ایم تنها 1 حباب (در صورت عدم نداشتن) به جای ۳ حباب داشته باشیم ← به این دلیل سخت افزار را به طبقه ID محدود کرده‌ایم که در این مرحله می‌توان به رجیسترها نیز دسترسی داشت.

Subject:

Year. Month. Date. ( )

خط لوله 4 طبقه ای شامل 4 مرحله برای هر Task یا دستور:

FI: Fetch Instruction

DA: Decode Address

FO: Fetch operation

EX: Execute

خط لوله 3 طبقه ای:

I: دریافت دستور

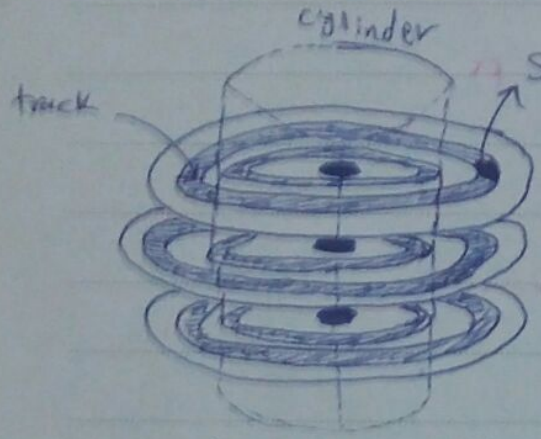
A: عمل ALU

E: اجرای دستور

Subject:
Year: Month: Date: ( )

I/O (دستگاه های ورودی و خروجی)

Disk :



در Hard disk معمولاً سه تا دیسک داریم که برگشتا این ها Head وجود دارد.
(هر یک قطعه مکانیکی است که روی داده در حال مورد نظر قرار می گیرد) و همچنین یک کنترلر نیز دارد.

- track : دایره های هم مرکز
- cylinder : track های هم مرز را سیلندر می گویند و روی دیسک های مختلف با شند
- Sector : هر track از تقسیم یکتور تشکیل شده است و کوچکترین جزء هر دیسک و اطلاعات روی یکتور روی SectorID ذخیره می شود - دیسک ها معمولاً از 512 بایت تا 4k روی یکتور نوشته می شود

زمان دسترسی به داده : شامل زمان های : ① زمان دسترسی به track مورد نظر (Seek time) ② دسترسی به یکتور (با چرخش صورت می گیرد) ③ زمان انتقال ④ زمان تأخیر کنترلر

مشخصه های I/O : ۱. قابلیت اطمینان ۲. قابلیت دسترسی ۳. قابلیت انگا

Flash Storage :

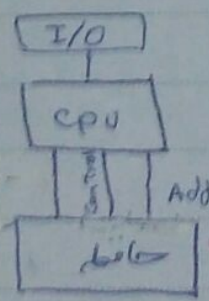
معمولاً دو نوع فلش مموری داریم :

- NOR flash : فلش مموری های با سرعت بالا هستند و معمولاً برای حافظه یک وسیله بکار میروند.
- NAND flash : از دسته فلش هایی که ما استفاده می کنیم و معمولاً نسبت به NOR کم سرعت تر

درون فلش ها یک چیپ کنترل و یکی حافظه وجود دارد.

- انواع باس (Bus) :

Bus control, Bus Address, Bus Data



هر I/O یک واحد واسط دارد که آدرس و کنترل دریافتی از I/O را دیکد کرده سپس آنها را برای وسایل جانبی تفسیر و سیگنال های لازم را برای کنترل کننده ی وسایل جانبی فراهم می آورد

دو جز برای ارتباط با همدیگر نیاز به هماهنگ سازی دارند.

روش های هماهنگ سازی : ① سنکرون : با استفاده از کلک ⊓⊓⊓ و باعث می شود اجزا با سرعت مشابه با هم در ارتباط باشند (مثل ارتباط حافظه با CPU)

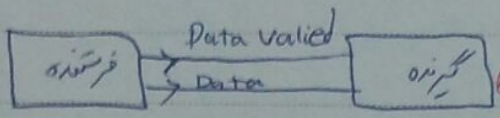
② آسنکرون : تنها برای ارتباط I/O با CPU برقرار می شود، و به CLK نیاز ندارد

Subject:

Year. Month. Date. ( )

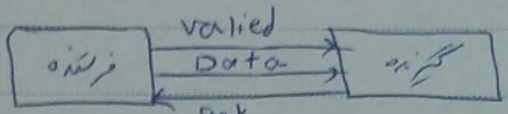
نحوه های همگام در روش آسنکرون :

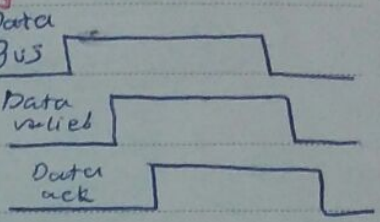
a دست تکانی : مبدأ با ارسال یک سیگنال Valied به مقصد می گوید که سیگنال های من قابل استفاده اند (فرض بر این است که Data به گیرنده می رسد)



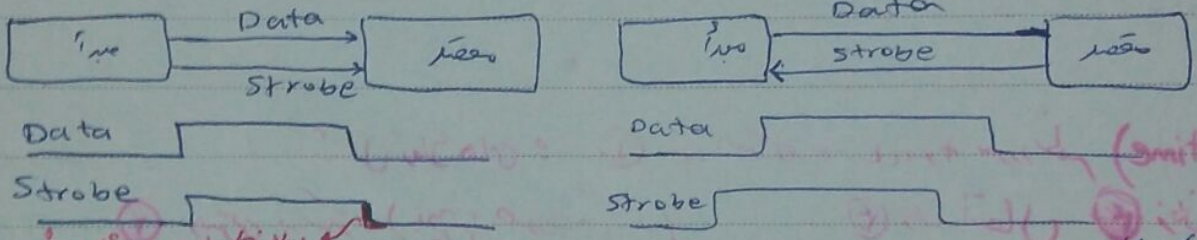
Bus جهت مشترک کردن داده ها Valied :

+ برای اینکه اطمینان رسیدن داده 100 درصد شود گیرنده یک سیگنال ack به فرستنده می فرستد.





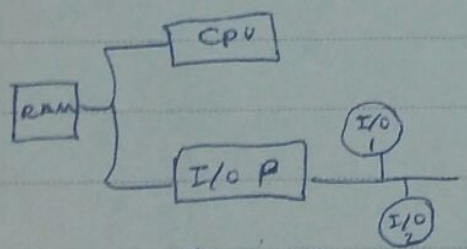
b Strobe : داده بهمراه سیگنال فرستاده می شود.



در لبه پایین داده برداشت می شود.

انواع گذرگاه های مربوط به I/O که باحافظه در ارتباطند :

الف- گذرگاه I/O و گذرگاه حافظه جدا از هم باشند ← در اینصورت I/O Processor تعریف می شود برای سرویس دادن به I/O



در اینجا CPU فرمان می دهد (master) و حافظه ای را می خواند (slave)

ب - Memory Map I/O (بخشی از حافظه را به آدرس ها اختصاص می دهد) از کل فضای حافظه جهت آدرس دهی اشکاله ای I/O استفاده می شود یعنی همه خطوط آدرس - داده و کنترل مشترک اند پس برای کار با حافظه و I/O دستور یکسان داریم.

ج - Isolated I/O :

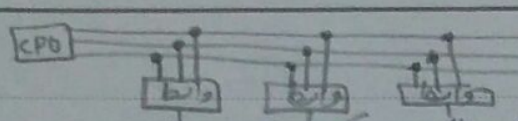
خطوط آدرس و داده مشترک ولی خطوط کنترل متفاوت اند پس دستورات متفاوتی برای کار با حافظه و I/O داریم. دستور حافظه ← LD

دستور I/O → { in / out }

ASEMAN

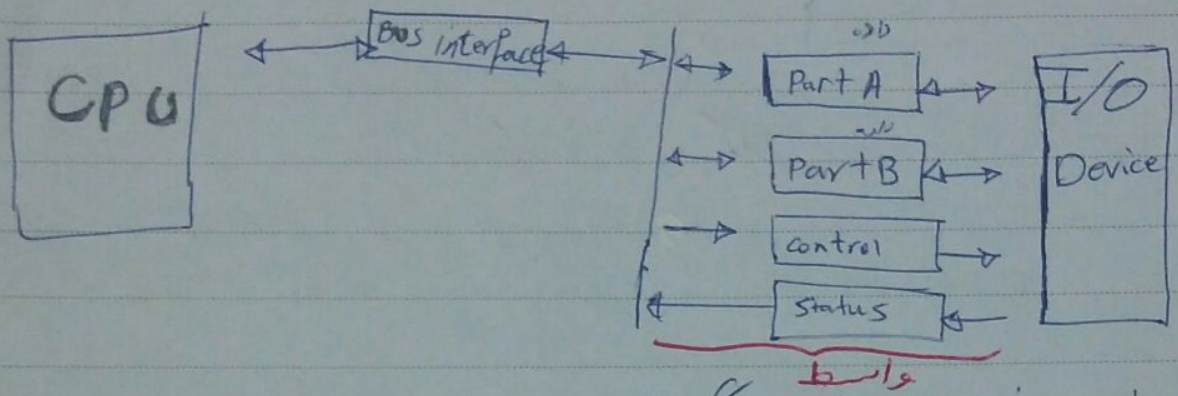
Subject :

Year. Month. Date. ( )



جهت تبادل اطلاعات با I/O

واسط بین I/O و CPU از مدار آدرس دیکدر و تعدادی رجیستر (ثبات داده - ثبات وضعیت و ثبات کنترل) تشکیل شده است.



روش های خبردار شدن از وضعیت دستگاه I/O و ورودی - خروجی :

(مودهای انتقال اطلاعات) : اطلاعات دریافتی از I/O برای پردازش بعدی معمولاً در حافظه ذخیره میگردند

و برای این کار دو روش وجود دارد :

① روش برنامه ریزی شده (polling) :

مانند کامپیوتر پایه CPU مدام دستگاه I/O را چک می کند (FG0, FG1 چک می کند) که در واقع برنامه ای بصورت نرم افزاری نوشته می شود.

② روش وقفه (interupt) :

هروقت I/O آماده انتقال اطلاعات شد توسط سیگنالی به CPU اعلام می کند (CPU دارای پین INT است)

سیگنال تخت افزاری IRQ به INT وارد می شود.

و این کار به دو صورت انجام می پذیرد (پیدا کردن آدرس سرویس وقفه)

الف) برداری : هر دستگاه I/O آدرس خود را در یک آرایه رجیستر اعلام می کند و برنامه وقفه را آدرس انتقاب وسیله I/O گویند.

ب) غیر برداری : آدرس تابعی از سطر برنامه می شود.

درصورتیکه چند دستگاه I/O داشته باشیم روش ها بصورت زیر (جهت اولویت دادن) :

الف) سرکشی (شرح داده شد) : منظور I/O که اولویت بالاتر دارد اول چک می شود و زمان زیادی را

ب) زنجیره ای (Daisy-Ching) : سرکشی بصورت سخت افزاری

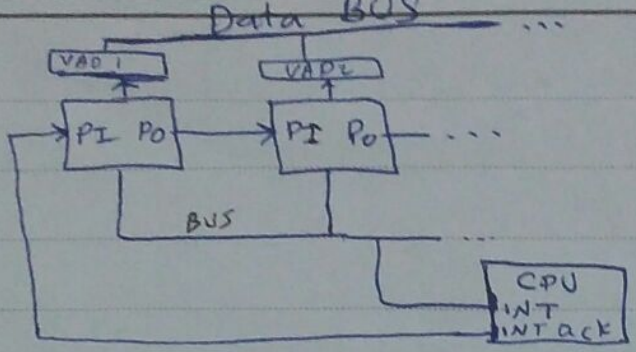
اولویت

انجام می شود و بصورت سری به I/O ها مرتبط می شود و استفاده

که اولویت بالاتر دارد سرکشی می کند.

Subject :

Year. Month. Date. ( )



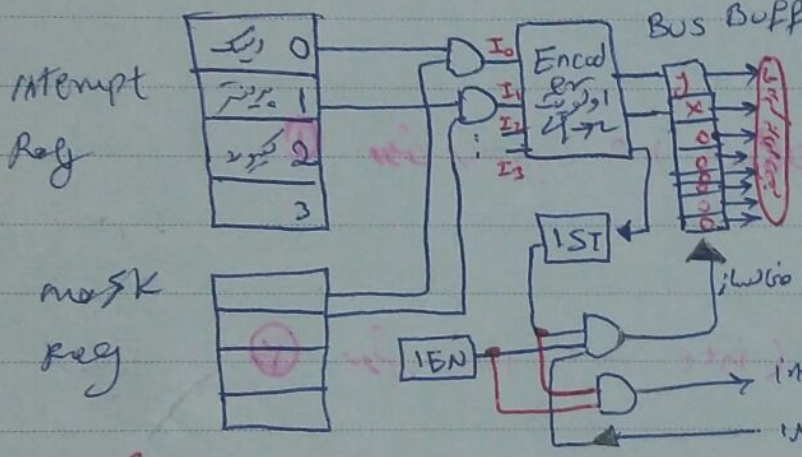
وقفه

2) روش نظرسنجی : (اولویت دار)

در این روش 2 جیتر داریم و از آنکه را اولویت بیشتر اولویت دهی استفاده می شود.

جیتر Interupt : هر بیت این جیتر یک I/O را مشخص می کند

جیتر mask : وقتی جواب می دهد که Mask منتظر آن را باشد (بشرطی بیت ها برابر 1 هستند)



IEN : کنترل فعالسازی عملیات وقفه

VAD to CPU : خط فلاپ وضعیت وقفه

IST : خط فلاپ وضعیت

$IST = I_0 + I_1 + I_2 + I_3$

تفاوت دو جیتر بالا : Interupt Req : توسط نرم افزار I/O تنظیم می شود.

Mask Req : بصورت نرم افزاری و با دستورات IN می شود.

* در صورتیکه تمام ورودی های Encoder برابر صفر باشد IST=0 می شود و این بدین معنی است که آدرس بردار به CPU منتقل نمی شود.

* خروجی انکدر بعنوان بخشی از آدرس بردار هر منبع وقفه بکار می رود.

* جیتر پوشش (mask) را می توان بعنوان غیرفعال کننده وقفه های پایین تر بکار برد.

- CPU checks IEN & IST

- if IEN * IST = 1, CPU → interrupt cycle : در حین تکمیل وقفه CPU یک سری اعمال را انجام می دهد:

SP ← SP - 1 : آماده گرفتن پشته را کاهش بده

M[SP] ← PC : پوش کردن PC به پشته

INTAck ← 1 : فعال کردن تصدیق وقفه

PC ← VAD : منتقل کردن بردار آدرس به PC

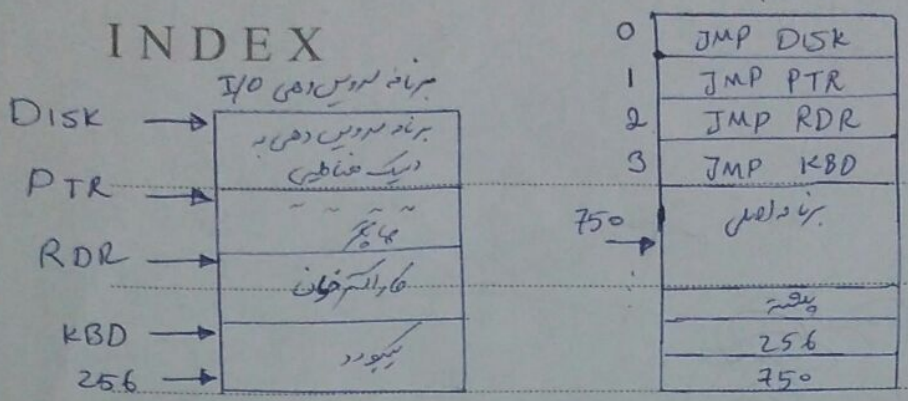
IEN ← 0 : غیرفعال کردن وقفه بعدی

وقتی سرویس دستور الهی بدی

ASEMAN

* IEN : وقتی صفر شود یعنی اینکه تقاضای وقفه ای که از IST می رسد توسط CPU نادیده گرفته می شود. و وقتی ۱ می شود یعنی اینکه در حین اجرای برنامه جاری از قابلیت وقفه می توان استفاده کرد.

* IST : وقتی صفر است یعنی اینکه آدرس برنامه به CPU متصل نمی شود.



* اگر هر دو تقاضا با هم یا با فاصله به CPU وارد شوند یکی وقفه می شود.

روال های نرم افزاری :

مثلاً دستورالعمل JMP KBD جهت سرویس دادن به وقفه که با فشردن کیبورد ایجاد می شود انجام می شود.

برنامه اصلی در 749 قرار می شود، آدرس 750 در پشته ذخیره می شود.

عملیات هایی که در ابتدای سرویس وقفه انجام می گیرد و عملیات های انتهای سرویس وقفه :

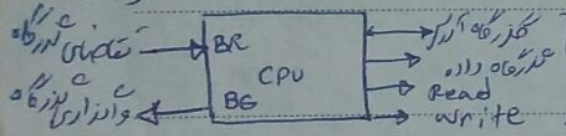
[1] IEN ← 0
[2] Restore cpu reg (PC)
[3] clear the bit in the interupt
[4] Set lower level Mask Reg (1)
[5] Restore return address, IEN ← 1

[1] clear lower level mask Reg bits
[2] IST ← 0
[3] Save content of CPU Reg (PC)
[4] IEN ← 1 (یعنی وقفه ای در حال اجرا نیست)
[5] go to interupt Service

ابتدا ۱۰ اینکه سطح می تواند بصورت نرم افزاری یا سخت افزاری طراحی شود (سخت افزاری مانند 8259)

DMA : Direct Memory access (دسترسی مستقیم به حافظه)

DMA یک کنترلر هست که بین حافظه و I/O قرار گرفته و باعث می شود CPU را در حالت استراحت قرار دهد و کنترل BUS را بر عهده بگیرد.



DMA از طریق BR تقاضای دریافت گذرگاه را می دهد وقتی BG=1 باشد یعنی CPU گذرگاه های خود را در حالت High impedance قرار داده و کنترلی بر آن ها ندارد. (گذرگاه آدرس - گذرگاه داده - کنترل)

* DMA برای دیسک های مغناطیسی ضروری است (به دلیل سرعت بالا)

* بین های Read و write برای DMA دوطرفه اند به این خاطر که : وقتی CPU با DMA ارتباط برقرار می کند این خط ها بعنوان ورودی از CPU به DMA استفاده می شود و وقتی که DMA با RAM ارتباط دارد این خط ها بعنوان خروجی از DMA به RAM عمل می کنند.

+ فایده DMA ثبت به وقفه : در وقفه دستورالعمل جاری متوقف می شود و CPU عملیات fetch را انجام می دهد و دوباره دستورالعمل جاری را ادامه می دهد ولی در DMA : CPU در دستورالعمل جاری را ادامه می دهد و DMA هم در حافظه دستورالعمل خود را انجام می دهد و هر دو دستورالعمل در یک زمان انجام می گیرند.

ASEMAN

INDEX

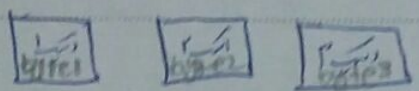
فهرست

یکی از راه های افزایش Performance دیسک های جدید از خروجی : هماهنگ سازی

(یعنی مثلاً در یک دیسک به جای اینکه دیتا را روی یک دیسک بگذاریم هر کدام از داده ها را روی یک دیسک می گذاریم) ← byte اول روی دیسک اول byte دوم دیسک دوم و...

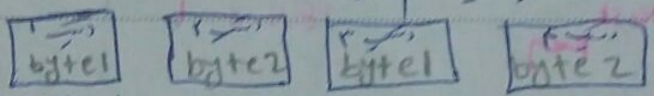
RAID : Redundant Array of inexpensive Disks

RAID 0 : (N disk) هیچ گونه افزونگی (Redundant) در این نوع RAID نیست و در واقع می توان به آن AID 1 نیز گفت.



از هر سه استفاده می شود. ←

RAD 1 : ← هر دیسک یک آینه دارد و به این معنی که هر داده ای که قبلاً روی دیسک ۱ نوشته می شود روی دیسک ۲ هم نوشته می شود

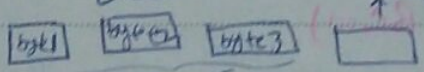


RAID 2 : N+E disks E disk : Error correcting

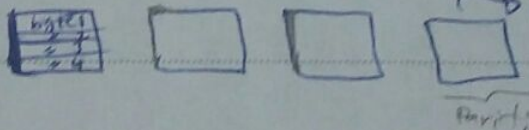
به تعداد دیسک های E می توان ، میکرو های خراب را تشخیص داد ،
تنها ۱ خرابی را تشخیص می دهد. E=1

RAID 3 : N+1 disk ← به ازای N بیت دیتا یک بیت برای تشخیص خطا روی دیسک اضافی است.

Ecc ← به اصلی



RAID 4 : دیتا را روی یک از N تا قرار می دهد و با دیسک Ecc دارد



RAID 5 : N+1 disk : Parity های RAID 4 است ولی بلوک های

صورت توزیع شده در دیسک های اصلی موجود دارند

این کید از بایت یک خود داری می کند.

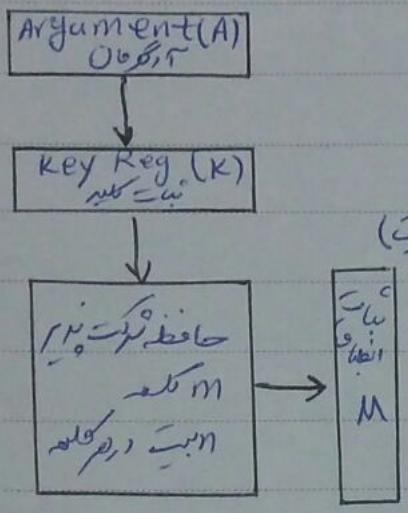
BASHGAN N+2 disks (گلوگاه)

Subject :

Year. Month. Date. ( )

حافظه ترکت پذیر (Associative Memory)

در حافظه اصلی Main Memory آدرس داده می شود و Data آن برداشت می آوریم ولی در حافظه شرکت پذیر خود داده مورد جستجو می شود و در حافظه جستجو می شود که آیا این داده در حافظه وجود دارد یا نه و یا در کجا قرار دارد ؟



عمل خواندن با مراجعه متوالی به کلماتی از حافظه که بیت های متناظر آنها در ثبات M ، ۱ شده اند انجام می شود .

ثبات A : ثباتی که محتوای آن می خواهد سرچ شود

ثبات K : برای انتخاب یک میدان خاص یا کلید آرگومان مراهم می آورد (n بیت)

if k کل آرگومان با هر ⇒ ۱ = کلیت های یک از کلمات حافظه مقایسه می شود .

سطوری فقط بیت هایی در آرگومان سرچ می شوند که K متناظر با آنها برابر ۱ باشد .

ثبات M : در صورتیکه بیت های آن ۱ می شوند که نشان می دهد بیت های مقایسه شده برابر همان بیت جستجو شده بوده است .

نکته : اگر بین همه ی بیت های پوشش یافته آرگومان و بیت های کلمه تطابق وجود داشته باشد بیت متناظر در ثبات انطباق یعنی M_i برابر ۱ است و اگر یک یا چند بیت پوشش یافته با کلمه حطابقت نداشته باشد M_i برابر ۰ خواهد بود .

Subject:
Year. Month. Date. ( )

دسته بندی انواع حافظه :
SRAM : (static Random access Memory)
DRAM : (Dynamic Random access Memory)
Magnetic disk :

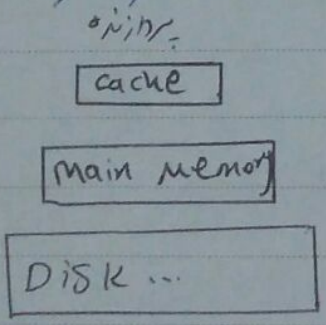
SRAM ← سریعترین حافظه است و کمتر از این نوع است.
DRAM ← سرعت آن از SRAM کمتر است و حافظه اصلی از این است.
disk ← های مغناطیسی سرعت کم، حجم زیاد و قیمت کم دارند.

بنابراین ما برنبال یک حافظه ایده آل هستیم؛ یعنی؛ حافظه ای که :
① access time (زمان دسترسی) آن شبیه SRAM باشد.
② قیمت و حجم آن شبیه دیسک های مغناطیسی باشد.

سلسله مراتب حافظه :

دلیل اصلی محلی بودن دسترسی ها نیاز به معرفی سلسله مراتب حافظه داریم ؛
اصل محلی بودن : زمانی ؛ اگر به داده ای دسترسی پیدا کنیم به احتمال زیاد در آینده ای نزدیک به آن احتیاج پیدا می کنیم (مثل دستورات حلقه)
مکانی ؛ اگر به حافظه ای دسترسی پیدا کردیم به حافظه های اطراف آن نیز نیاز خواهیم داشت (مثل داده های یک آرایه)

داده ها روی دیسک قرار می گیرند و به دلیل اصل محلی بودن داده های دیسک روی حافظه ای نزدیک تر به پردازنده مانند Main Memory یا Cache قرار می گیرند.



معرفی چند اصطلاح :
Hit ؛ اگر پردازنده به کش مراجعه و به آنجای مورد نظر دسترسی پیدا کند گوییم Hit رخ داده است

Subject :

Year. Month. Date. ( )

Miss : اگر پردازنده به کش دسترسی پیدا کند ولی داده‌ای مورد نظر را پیدا نکند گوئیم Miss رخ داده است.

جهت تعیین کارایی کش دو نسبت معرفی می‌شود :

$$\text{Hit ratio} = \frac{\#\text{Hits}}{\#\text{total accesses}}$$

$$\text{Miss ratio} = \frac{\#\text{Misses}}{\#\text{total accesses}}$$

ارتباط بین دو رابطه بالا :

$$\text{Miss ratio} = 1 - \text{Hit ratio}$$

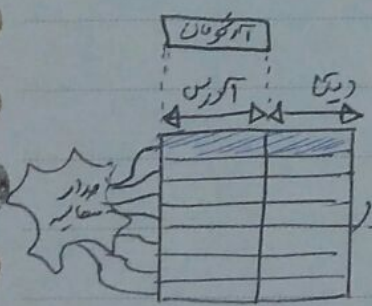
Miss Penalty : جریمه‌ی Miss را گوئیم به این معنی که در صورتیکه Miss رخ دهد ناچاریم به Disk یا Main-m دسترسی پیدا کنیم و این خود چند صد سیکل طول می‌کشد که به جریمه‌ی miss معروف است.

برای کاهش آن از Multilevel cache استفاده می‌شود یعنی استفاده از کش دیگر بین ای برای کش قبلی

$$T_e = T_c + (1 - \text{Hit ratio})\, T_m$$

زمان دسترسی به کش ← زمان متوسط دسترسی ← زمان دسترسی به ممور

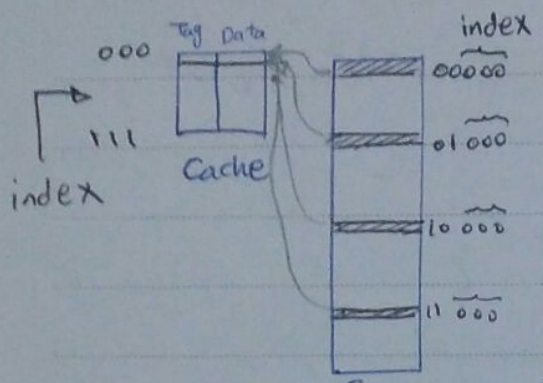
روش های دسترسی به کش (روش های پرکردن کش) :



۱- روش شرکت پذیری کامل (Fully associative) :
آدرس و دیتا در هر خط کش (بلاک) ذخیره می‌شود و بصورت موازی یک مقایسه بر اساس آرگومان داده شده انجام گرفته و شماره ردیف مورد نظر را می‌دهد.
پیاده سازی این نوع دسترسی از طریق حافظه شرکت پذیر است.

دیگر های این روش : ① هر بلاک کش می‌تواند هر سطری از حافظه را ذخیره کند (انعطاف پذیر)
② گران بودن (به دلیل اینکه مدار های مقایسه و تطبیق در این روش وجود دارد)
③ استفاده از سیاست FIFO (هرگاه داده مورد نظر جدیدی بخواهد در کش قرار گیرد اولین داده و آدرس وارد شده در کش خارج شود)

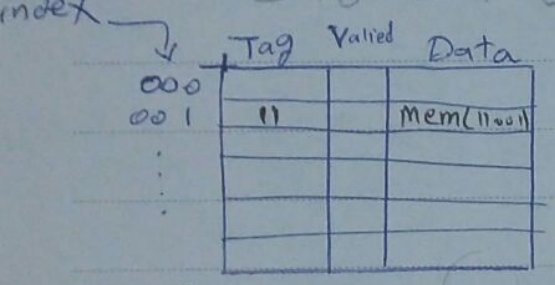
Subject:
Year. Month. Date. ( )



۲- روش نگاشت مستقیم :

در این روش قسمتی از آدرس های حافظه را به Cache ، نگاشت می کنیم ،
در این روش براساس index به کش دسترسی پیدا می کنیم و دیتا را در کش پیدا می کنیم
در واقع حافظه را به قسمت های مساوی به اندازه کش تقسیم می کنیم .
خانه های با index های یکسان به خانه متناظر کش نگاشت می شوند .

هر آدرس بصورت : Tag + index می باشد که index مشخص کننده خانه ی Cache است (بلوک) و Tag نیز ذخیره می شود ، همچنین یک بیت Valid جهت معتبر شدن یا نبودن بلاک کش است .



مثال ← اگر 11001 در کش وجود داشته باشد و برای بار دوم بخواهیم به آدرس 10000 دسترسی پیدا کنیم بدلیل اینکه وجود ندارد Miss رخ می دهد و در نهایت 10000 در کش ذخیره می شود ← همیشه آخرین چیزی که استفاده می شود در کش قرار می گیرد .

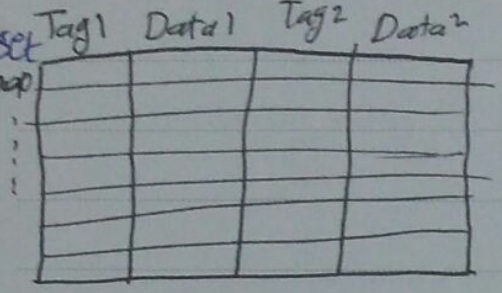
در روش نگاشت مستقیم آدرس بصورت : Tag + index + byte offset
↳ برای این است که هر خانه کش (هر بلاک) از چند word یا byte تشکیل شده است و این مشخص کننده ورد مورد نظر است .

عیب این روش : دو کلمه با index مشخص و Tag متفاوت نمی توانند در حافظه کش قرار گیرند
مثلاً اگر 11001 در کش باشد و 10001 خوانده شود دیتا جدید جای اولی را می گیرد .

Subject :

Year. Month. Date. ( )

۳- روش شرکت پذیری مجموعه‌ای (Set associative) :



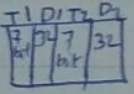
two-way cache

در واقع این روش تصحیح شده روش نگاشت مستقیم می باشد (ایجاد Tag های کمیاب در حافظه کش)

هر بار که داده همراه با Tag آن ذخیره می شود و تعداد اعلام تگ - داده در هر بلاک از حافظه کش یک مجموعه نامیده می شود ، هر بلاک کش از یک یا چند کلمه تشکیل شده و طول هر بلاک برابر : $n \times (\#\text{Tag bits} + \#\text{Data bits})$

اندازه حافظه کش = اندازه مجموعه × طول هر بلاک ← مثال 512 × 36 ← تعداد مجموعه

در کل می توان $n \times 2$ کلمه از حافظه اصلی را در کش قرار داد . (اندازه مجموعه)

مثال ← 

اندازه این کش : $1024 \times 78 = 1024 \times 2(7+32)$

ناحیه تعدادی این روش بصورت : n-Way می باشد و معمولاً 1-Way داریم

و 2-way که one-way همان نگاشت مستقیم است .

و n-Way نگاشت Full associative است .

- به هر ردیف کش در این روش Set گویند .

- افزایش n ، Miss rate را کاهش می دهد ، زیرا کلمات بیشتری با index یکسان و Tag متفاوت داریم .

← وقتی که Miss نرخ دهد در این روش و مجموعه پر باشد لازم است تا به جای یکی از اعلام نشانه ، داده (تگ - داده) مقدار جدید جایگزین شود .

پس الگوریتم های جایگزینی تعریف می شود :

① LRU : آن ردیف کش بیرون می رود که اخیراً کمتر مورد استفاده قرار گرفته است .

ASEMAN

Subject:
Year. Month. Date. ( )

Random ①
FIFO ②

← که FIFO و LRU را می توان با افزایش چند بیت در هر بلاک کش پیاده سازی کرد.

خواندن از کش: وقتی CPU در حین عمل خواندن است (خواندن کلمه ای در کش) آگاه حافظه اصلی در عمل انتقال دخالت نمی کند.

سیاست های نوشتن در کش:

در نوشتن حافظه در عمل انتقال دخالت می کند (زیرا CPU مستقیماً با M.M ارتباط ندارد)

الف- سیاست نوشتن کامل (Write-through):

در این مورد درست همان خانه ای که در کش نوشته شود در حافظه اصلی هم نوشته می شود ← همیشه حافظه بهنگام است.

مزیت ← حافظه BUS را اشغال می کند ← راه حل استفاده از Write-Buffer

عیب ← یک قطعه کمت افزاری است که عملیات نوشتن را به آن واگذار می کند و خود کار خودش را انجام می دهد.

ب- سیاست پس نویسی (Write-Back):

وقتی داده از کش بیرون رفت آگاه در حافظه اصلی نوشته می شود ←
توسط یک بیت dirty bit تشخیص داده می شود که در M.M نوشته شود یا نه تا هنگامی که از داده استفاده نشود در صورتی ذخیره نشده و در واقع سیاست خوبی است زیرا ممکن است چندین مرتبه تغییر یابد

وقتی Miss rate در نوشتن رخ می دهد دو سیاست داریم: (بیاریش توکش یا نه؟)
Write allocate:
وقتی سیاست نوشتن write-through است و miss رخ دهد در کش نوشته می شود به این امید که مورد استفاده قرار می گیرد.

Subject :

Year, Month, Date, ( )

Write around : وقتی حیث نوشتن Write-Back است.
این روش و درکش نوشته نمی شود و در واقع به این معنی که ابتدا مورد استفاده قرار می گیرد

AMAT = Hit rate + Miss rate X miss Penalty : فرمول
(Average Memory Access Time)

حافظه مجازی (Virtual Memory)

در واقع یک نوع حافظه است که باعث می شود حافظه کمکی را به حافظه اصلی نگاشت دهد (RAM بدون یک کش برای دیسک است)

ویژگی : - هر برنامه ای روی کامپیوتر فضای حافظه را در اختیار خود می بیند
- برنامه نویسی را ساده تر می کند

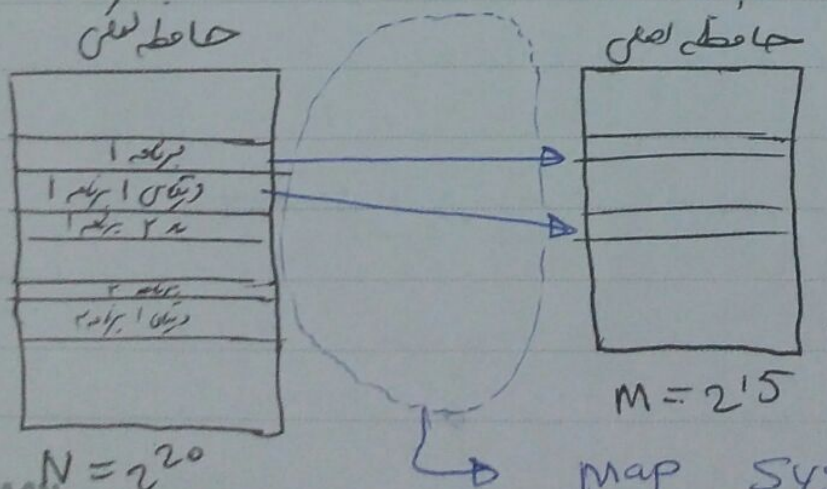
در واقع یک تکنیک است که از حافظه اصلی بعنوان کش برای حافظه کمکی استفاده می کند

نگاشت حافظه به دو صورت انجام می گیرد : (نگاشت آدرس مجازی به آدرس فیزیکی)
الف) قطعه بندی Segmentation ب) صفحه بندی Paging

* تعریف *

فضای آدرس : آدرسی که به وسیله برنامه نویس مورد استفاده قرار می گیرد و آدرس مجازی و مجموعه این آدرس ها فضای آدرس نامیده میشه

آدرس فیزیکی : یک آدرس در حافظه اصلی ، مکان یا آدرس فیزیکی نامیده میشه .

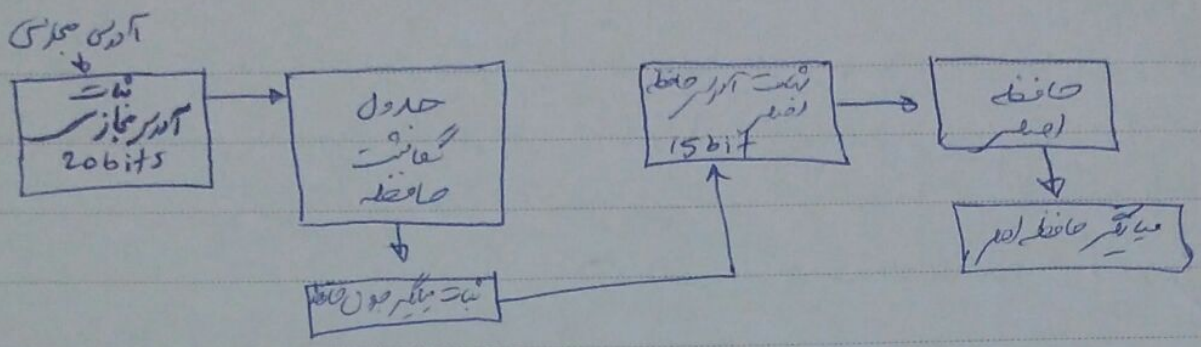


با استفاده از سیستم نگاشت این حافظه مرتبا آدرس مجازی به آدرس فیزیکی تبدیل می شود.

ASEMAN

Subject :

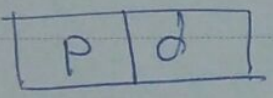
Year. Month. Date. ( )



که جدول نگاشت برای هر برنامه، بافر برنامه وجود دارد.

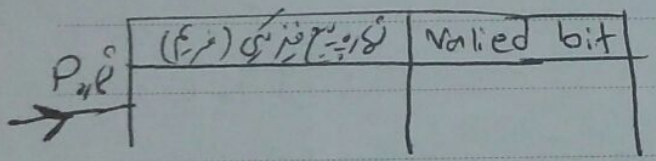
روش صفحه بندی :

حافظه کمکی و حافظه اصلی به صفحاتی تقسیم می شوند که اندازه یکسان دارند.
نام قطعات تقسیم شده حافظه اصلی را frame نیز می نامند.
و CPU و سیستم عامل کار ترجمه آدرس را انجام می دهند
+ طور کلی آدرس مجازی در روش صفحه بندی بصورت

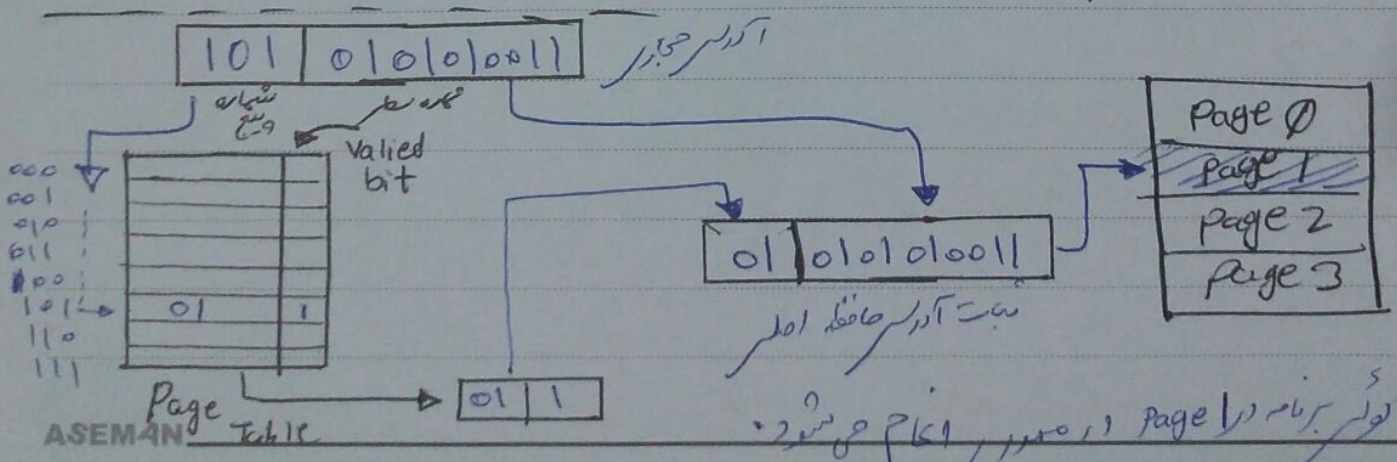


است

که P شماره بیج را در حافظه کمکی مشخص می کند و باید map شود و d آفست درون بیج است و word درون بیج را مشخص می کند.



مثال ← شماره بیج داده شده
و از طریق جدول بیج برنامه ۱
مثلاً در Page3 حافظه
اصلی قرار می گیرد و offset شماره word مربوط به آن در Page3 را مشخص می کند



گونه برنامه در Page ها به صورت انجام می شود.

ASEMAN

Subject :

Year. Month. Date. ( )

- درگیری Miss رخ می‌دهد و در اینجا Page Fault !!! و باعث Page Fault نامیدیم داده را به physical Memory ببریم (در آنجا به کش می‌بردیم) زیرا Ph.M مقدار کمی کش برای حافظه کمکی است. و این کار میلیون‌ها سیکل زمان می‌برد.

برای کاهش Page Fault :
① استفاده از حالت full associative برای Ph. Memory
② روش‌هایی مانند LRU استفاده کردن

روش قطعه بندی (Segmentation) :

در این روش جهت به‌دست آوردن آدرس فیزیکی (آدرسی که RAM آنرا می‌خواند) از آدرس مجازی (آدرس حافظه کمکی) لازم است که حافظه‌ها را به قطعه‌هایی که اصطلاحاً segment نام دارند تقسیم کرده که اینها دارای طول‌های متفاوت و دلخواهی هستند.

در این روش نیز نیاز است یک جدول تبدیل تعریف کنیم که Segment Table معروف است که آدرس شروع سگمنت و حد قطعه را مشخص می‌کند

مثلاً در 8086 ، CS ، DS ، SS داریم

TLB :

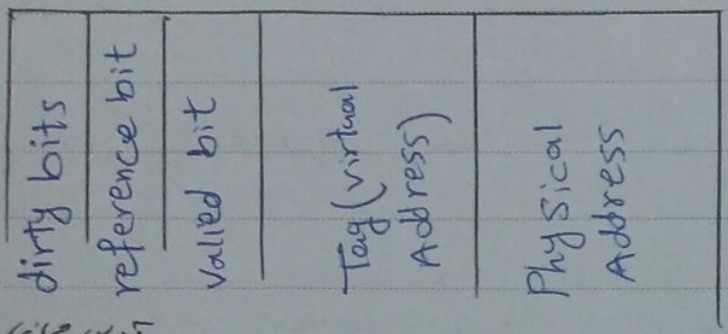
Working set : به آن تعداد صفحاتی که در یک بازه زمانی برنامه با آن کار دارد و دیگر صفحات را براساس نیاز حافظه اصلی می‌آید گویند.

از آنجایی که ارجاعات محلی وجود دارد لذا می‌توان آدرس‌های Working set را در یک جدول که TLB نام دارد (Translation Lookaside Buffer) و بصورت شرکت‌پذیر (full associative) الک به این دلیل که سرچ در این نوع بیشتر است و درواقع تگ‌ها بصورت موازی باهم مقایسه می‌شوند و اگر Hit رخ دهد سریع تفکیک می‌شود، و همچنین دارای 2 بیت کنترلی آدرس Page های مجازی و دیگری فیزیکی است.

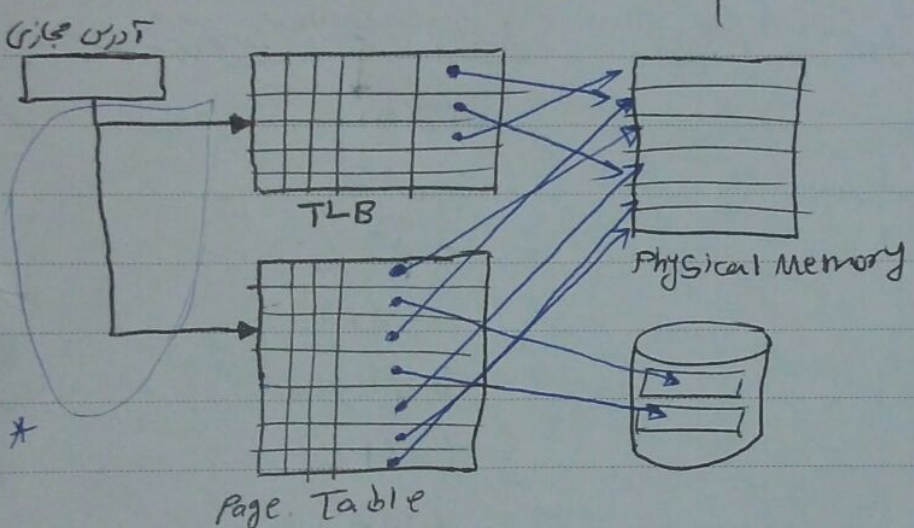
ASEMAN

Subject :
Year. Month. Date. ( )

- TLB در واقع عنوان یک کش برای Page Table می باشد.
- در TLB، Miss وجود دارد.
- در جدول TLB زیر در Tag همان آدرس مجازی نوشته می شود که این تگ ها مقایسه می شوند در حالت شرکت پذیری.



یک مثال:



سیاست های جایگزینی (Replacement):

وقتی یک Page Fault رخ دهد و یک بیج برای خروج باید انتخاب شود، جهت این کار سیاست LRU استفاده می شود، به اینصورت که در همان Page Table یک بیت به عنوان reference قرار می دهند و هر وقت به page مراجعه شد این بیت 1 می شود، وقتی می خواد بیرون ببره از اون هایی که صفره رو بیرون میبره و جایگزین میشه و این کار توسط سیستم کامل (OS) انجام می گیرد.

سیاست های Write آن:

جهت نوشتن هم دیگه زیاد الکی چند میلیون سیکل طول نکشه (به دلیل حجم زیاد دیتا ها) لذا از سیاست write back استفاده می شود.

Subject :

Year. Month. Date. ( )

که اگر اگر Write through استفاده شود باید بعد از هر نوشتن در حافظه اصلی باید روی دیسک هم نوشته شود و این زمان بر است پس معمولا استفاده نمی‌گردد.

و برای سیاست Write back نیز کد بیت dirty جهت اینکه آیا در دیسک نوشته شده یا نه !!

یک مثال از سیاست های replacement !

Page alighs : 4 2 0 1 2 6 1 4 0 1 0 2 3 5 7

8 have 8 page in auxiliary Mem & 4 page in ph. Mem

اولین بیت که وارد میشود اولین صفحه → FIFO

قدر در بیت 4 بایت بیشتر 4 Page را نمی‌تواند داشته باشد

LRU → بیت آخری که مورد استفاده قرار گرفته

	FIFO	LRU
initial	4201	4201
2	4201 → بدون تغییر	4012
6	2016	0126
1	2016	0261
4	0164	2614
0	0164	6140
1	0164	6401
0 ..	0164	6410
2	1642	4102
3	6423	1023
5	4235	0235
7	2357	2357